Universidade Federal da Paraíba
Centro de Ciências Exatas e da Natureza
Departamento de Estatística

Relatório de estágio supervisionado II

Gabriel de Jesus Pereira

abril, 2025

Gabriel de Jesus Pereira

Relatório de estágio supervisionado II

Relatório exigido como requisito básico para a conclusão do Estágio Supervisionado II do curso de Estatística, do Centro de Ciências Exatas e da Natureza da Universidade Federal da Paraíba.

Orientador: Prof. Dr. Ulisses Umbelino dos Anjos

João Pessoa
abril, 2025

Sumário

Lista de Figuras

Lista de Algoritmos

1 Introdução

(FALAR SOBRE PARA QUE FOI O ESTÁGIO E FALAR BREVEMENTE SOBRE A EMPRESA)

2 Recursos Computacionais

Nesta seção, serão apresentados os recursos computacionais utilizados no desenvolvimento do projeto realizado durante o estágio. As ferramentas selecionadas incluem a linguagem de programação Python, amplamente conhecida, e o sistema de banco de dados **Denodo**, utilizado por toda a empresa para consulta à base de dados. Dessa forma, a primeira etapa consiste em descrever a tecnologia empregada na coleta dos dados utilizados no projeto.

2.1 Recurso para coleta de dados

Para a coleta dos dados utilizados no projeto, foi adotado o **Denodo**, uma plataforma de virtualização de dados amplamente utilizada pela empresa. Essa ferramenta permite o acesso, integração e consulta a múltiplas fontes de dados, estruturadas ou não estruturadas, sem a necessidade de mover ou replicar os dados fisicamente.

A partir do **Denodo**, é possível extrair dados da quantidade de novos associados obtidos pelas cooperativas e agências, os lucros obtidos da empresa em relação a produtos de crédito, previdência, despesas e receitas, entre muitos outros. No caso do projeto realizado durante o estágio, foram coletados dados de receitas, despesas e quantidade de novos associados em cada uma das agências sediadas no nordeste.

A utilização do Denodo proporcionou maior agilidade no processo de extração e consulta das informações necessárias, além de garantir padronização e segurança no acesso aos dados corporativos. Por meio de sua interface intuitiva, suporte à linguagem SQL e sua fácil conexão com a linguagem **Python**, foi possível realizar consultas eficientes à base de dados, atendendo às demandas específicas do projeto desenvolvido durante o estágio.

Com os dados coletados, o próximo passo foi fazer toda a sua organização para finalmente realizar a modelagem e poder utilizar esses dados na modelagem final. Portanto, a seção a seguir irá descrever quais ferramentas foram utilizadas para realizar a limpeza, organização e modelagem de cada um dos produtos utilizados.

2.2 Recursos utilizados para modelagem e visualização dos resultados

Com os dados coletados, a etapa de modelagem dos produtos financeiros torna-se essencial para converter essas informações em entendimentos relevantes e aplicáveis. Para chegar a esses resultados, foram utilizadas bibliotecas desenvolvidas em Python, como **scikit-learn**, **StatsForecast**, **pandas** (TEAM, 2020), empregada para a manipulação das bases de dados utilizadas neste trabalho, **numpy** (HARRIS et al., 2020), voltada para computação numérica, entre outras.

A biblioteca **StatsForecast** foi utilizada especificamente para a aplicação de modelos de

séries temporais, com o objetivo de prever a evolução dos diferentes produtos financeiros ao longo do tempo. Essa ferramenta oferece implementações eficientes de modelos clássicos de previsão, como ARIMA, ETS (Exponential Smoothing), AutoETS e AutoARIMA, além de suportar geração de previsões em larga escala. A StatsForecast é especialmente otimizada para performance, com implementações em **Numba**, o que permite processar grandes volumes de séries temporais de forma rápida e escalável. Essas funcionalidades foram fundamentais para lidar com a diversidade e a quantidade de produtos analisados neste estudo.

O **scikit-learn** é uma das bibliotecas de aprendizado de máquina mais populares em **Python**. Ela oferece uma extensa coleção de algoritmos para tarefas como classificação, regressão, clusterização, além de ferramentas para pré-processamento de dados, validação cruzada e seleção de modelos. O projeto teve início no Google Summer of Code como uma iniciativa do engenheiro francês David Cournapeau. O **scikit-learn** foi criado como uma ferramenta baseada na biblioteca **SciPy** (VIRTANEN et al., 2020), que é voltada para computação científica e cálculo numérico em **Python**.

Uma das funcionalidades mais úteis do **scikit-learn** é o uso de pipelines para transformar dados. As pipelines permitem combinar etapas de pré-processamento e ajuste de modelos em um único fluxo organizado. Isso não apenas simplifica o código, mas também assegura que as transformações aplicadas aos dados de treinamento sejam automaticamente replicadas nos dados de teste ou em novos dados. Além de serem úteis para tarefas mais simples, como normalização de variáveis, as pipelines permitem a inclusão de etapas personalizadas para lidar com cenários mais complexos, como tratamentos avançados de dados ou integração com ferramentas externas.

O exemplo abaixo ilustra uma pipeline para pré-processamento e modelagem. As variáveis numéricas passam por padronização, enquanto as variáveis categóricas são transformadas em variáveis binárias utilizando codificação one-hot. Por fim, os dados processados alimentam um algoritmo de Random Forest para regressão:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.ensemble import RandomForestRegressor

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)

numerical_features = X.select_dtypes(include=np.number).columns
categorical_features = X.select_dtypes(include=object).columns

pipeline = Pipeline([
```

```
    ('preprocessor', ColumnTransformer([
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(), categorical_features)
    ])),
    ('model', RandomForestRegressor(
        n_estimators=100,
        random_state=42
    ))
])

pipeline.fit(X_train, y_train)
```

Outra etapa fundamental no processo de aprendizado de máquina, além do pré-processamento dos dados e do ajuste do algoritmo, é a otimização dos hiperparâmetros. O **scikit-learn** oferece algumas ferramentas para essa tarefa, mas elas possuem funcionalidades mais básicas e podem ser limitadas para cenários mais complexos. Dessa forma, de forma complementar, esse trabalho utilizou a biblioteca **Optuna** (AKIBA et al., 2019), que é uma biblioteca que contém uma grande quantidade de algoritmos para otimização, como, por exemplo, métodos de otimização bayesiana.

Além das funcionalidades de otimização, o **Optuna** também oferece ferramentas para analisar o comportamento da função objetivo durante o processo de otimização e identificar os hiperparâmetros mais relevantes. Uma dessas ferramentas é a função `plot_param_importances`, que gera um gráfico destacando a importância relativa de cada hiperparâmetro. Além disso, como o `plot_param_importances` utiliza a biblioteca **Matplotlib** (HUNTER, 2007), os gráficos gerados podem ser personalizados pelo usuário.

Conclui-se, assim, a apresentação das ferramentas empregadas na modelagem e visualização dos resultados. A seguir, serão descritas as tecnologias utilizadas no desenvolvimento da aplicação final deste trabalho. Vale destacar que o **scikit-learn** continuou desempenhando um papel relevante na aplicação, com sua pipeline sendo serializada no formato `.pkl` por meio da biblioteca **pickle**. Essa biblioteca permitiu a reutilização da pipeline no back-end[1] da aplicação, garantindo a consistência, eficiência do processamento dos dados e predição do modelo para novos dados. Segue, então, a descrição das ferramentas empregadas na construção da aplicação final.

2.3 Recursos utilizados para a criação da aplicação final

A aplicação final foi desenvolvida com o objetivo de realizar previsões dos produtos para que se tivesse uma comparação da agência com sua linha de porte e de cluster. A partir disso, seria possível saber se o planejamento das agências estavam sendo muito abaixo ou acima do seu porte ou cluster e o porquê. Além de integrar a pipeline de machine learning criada durante a etapa de modelagem, a aplicação também dispõe de funcionalidades interativas que permitem aos usuários explorar os dados e visualizar os resultados de forma clara e intuitiva. Para alcançar esses objetivos, foram empregadas tecnologias para

[1] Back-end é a parte de um sistema ou aplicação responsável pelo processamento de dados, lógica e comunicação com bancos de dados e servidores, funcionando nos bastidores da interface com o usuário.

o desenvolvimento do front-end[2] e do back-end, que foram desenvolvidos em **Python**.

O front-end da aplicação foi desenvolvido utilizando a biblioteca **Dash** (HOSSAIN, 2019), uma ferramenta voltada para a criação de aplicativos web em **Python**. Desenvolvida pela **Plotly** (INC., 2015), a **Dash** permite construir interfaces gráficas completas sem a necessidade de conhecimentos avançados em desenvolvimento web, integrando tecnologias como `HTML`, `CSS` e **JavaScript** diretamente no ambiente **Python**. Entre suas principais características, destacam-se a facilidade de criar gráficos interativos, a integração com bibliotecas populares como **Plotly** e a capacidade de atualizar componentes de forma dinâmica por meio de funções suas funções `callback`. A seguir é apresentado um exemplo do uso da função `callback`:

```
from dash import Dash, dcc, html, Output, Input
import plotly.express as px
import pandas as pd

df = pd.DataFrame({
    'Categoria': ['A', 'B', 'C'],
    'Valores': [10, 20, 30]
})

app = Dash(__name__)

app.layout = html.Div([
    dcc.Dropdown(
        id='dropdown',
        options=[
          {'label': cat, 'value': cat}
          for cat in df['Categoria']
        ],
        value='A',
        clearable=False
    ),
    dcc.Graph(id='graph')
])

@app.callback(
    Output('graph', 'figure'),
    Input('dropdown', 'value')
)
def update_graph(selected_category):
    filtered_df = df[df['Categoria'] == selected_category]
    fig = px.bar(
      filtered_df,
      x='Categoria',
      y='Valores',
```

[2]Front-end é a parte de um sistema ou aplicação responsável pela interface gráfica e pela interação com o usuário.

```
        title=f'Valores da Categoria {selected_category}')

    return fig

if __name__ == '__main__':
    app.run_server(debug=True)
```

Neste exemplo, o código permite que o usuário selecione um valor em um menu suspenso, criado a partir da classe `Dropdown`, e exiba o valor escolhido em um gráfico. Primeiramente, é criado um pequeno conjunto de dados utilizando a biblioteca **pandas**, contendo categorias (A, B e C) e valores associados a elas. Em seguida, define-se o layout da aplicação (`app.layout`), que consiste em um `Dropdown` para selecionar uma categoria e um componente `Graph` para exibir o gráfico correspondente. A lógica para atualizar o gráfico conforme a seleção do usuário é implementada por meio da função `callback`. O decorador `@app.callback` conecta o valor selecionado no `Dropdown` (definido pela classe `Input`) ao gráfico (definido pela classe `Output`). A função associada ao decorador `callback`, chamada `update_graph`, recebe como argumento o valor escolhido no `Dropdown`, filtra o DataFrame com base na categoria selecionada e gera um gráfico de barras utilizando a biblioteca **Plotly**. Este gráfico é então retornado para ser exibido no componente `Graph`.

Para o desenvolvimento do back-end da aplicação foi utilizado o framework **FastAPI**, uma ferramenta moderna e eficiente para a criação de APIs em **Python**. O **FastAPI** é conhecido por sua alta performance, graças ao uso de tipagem estática e a sua facilidade de suporte assíncrono, além de permitir a criação de APIs de maneira rápida e simples. Além disso, integra-se facilmente com bibliotecas como **Pydantic** para validação de dados e **SQLAlchemy** (BAYER, 2012) para interação e conexão com bancos de dados.

No projeto, o **SQLAlchemy** foi utilizado para gerenciar a conexão com o banco de dados, implementado com **PostgresSQL**, um sistema de gerenciamento de banco de dados relacional amplamente reconhecido por sua robustez e alto desempenho. Isso permitiu que os dados fossem expostos na API e consumidos no front-end. Além disso, a API foi responsável por expor a pipeline do modelo, possibilitando a realização de previsões de dos produtos diretamente na aplicação.

O **Docker** foi utilizado para facilitar a execução e integração dos componentes da aplicação, permitindo a criação de ambientes isolados e consistentes para o front-end e o back-end. Por meio de containers, foi possível iniciar e conectar ambos os serviços de forma simplificada, garantindo que a aplicação funcionasse corretamente independentemente do ambiente em que fosse executada. Com o uso do **Docker**, a configuração do ambiente tornou-se reprodutível, escalável e de fácil manutenção. Além disso, o banco de dados **PostgresSQL** foi criado a partir do **Docker**.

O **Docker** é uma plataforma que permite empacotar aplicações e suas dependências em unidades chamadas containers, que podem ser executadas de maneira padronizada em qualquer sistema que possua o **Docker** instalado. Isso elimina problemas comuns relacionados a diferenças de ambiente e facilita o processo de desenvolvimento, teste e implantação de aplicações.

2.4 Recursos utilizados para escrita de código e de documento

O desenvolvimento deste trabalho foi realizado em um computador equipado com processador Intel Core i5 de nova geração, 16 GB de memória RAM, placa de vídeo GeForce GTX 1650 e um SSD NVMe de 256 GB, operando sob o sistema Windows 11. A máquina apresentou desempenho satisfatório em todas as etapas do projeto, incluindo tarefas mais intensivas como a otimização de hiperparâmetros dos modelos. Não foram observados problemas durante o treinamento, o que permitiu a execução das rotinas de forma eficiente e contínua. Dessa forma, a escolha das ferramentas utilizadas para a escrita do código priorizou a praticidade e a compatibilidade com o ambiente de desenvolvimento adotado.

O **Visual Studio Code (VSCode)** foi a principal ferramenta utilizada para a escrita do código neste trabalho. O **VSCode** é um editor bastante leve e oferece suporte a diversas linguagens de programação e permite integração com inúmeras tecnologias por meio de suas extensões. Entre as extensões utilizadas, destaca-se o **vscodevim**, que incorpora os key mappings do editor de texto Vim, originalmente criado por Bram Moolenaar e lançado em 2 de novembro de 1991. Além disso, foram empregadas extensões específicas para as linguagens **Python** e **R**, tendo como principal objetivo uma melhor formatação de código, execução e identificação de possíveis erros.

Por exemplo, uma das extensões utilizadas para análise estática de código em **Python** foi o **Pyright**, desenvolvido pela Microsoft. Essa ferramenta é projetada para detectar erros durante o desenvolvimento e verificar a tipagem de código em **Python**, contribuindo para a melhoria da qualidade e da manutenção do programa. Outra ferramenta empregada foi o **Black Formatter**, um formatador de código **Python**. Ele automatiza a padronização do estilo de código, garantindo consistência e legibilidade. Além disso, foi utilizado o **Flake8**, que analisa o código em busca de erros de sintaxe, problemas de estilo com base no padrão **PEP 8**, e outras questões, como redundâncias ou importações desnecessárias. Por fim, também foi utilizada uma extensão mais geral de suporte à linguagem **Python**, que oferece diversas funcionalidades, como correção de código por meio da extensão **PythonDebugger**, que utiliza a biblioteca debugpy para suporte ao processo de debugging.

Para a elaboração deste documento, foi utilizado o **Quarto** (ALLAIRE; DERVIEUX, 2024), uma plataforma de publicação científica desenvolvida pela empresa Posit. O **Quarto** é uma evolução do RMarkdown e se destaca por sua capacidade de criar documentos de alta qualidade que integram texto e código. Compatível com diversas linguagens de programação, como **R**, **Python**, e outras, essa ferramenta é extremamente versátil para análise de dados e geração de relatórios. Com o **Quarto**, é possível produzir relatórios, artigos, livros, apresentações e até sites. Ele é amplamente adotado na comunidade científica, especialmente entre usuários de **R**, e oferece suporte a Markdown e LaTeX, o que facilita a inclusão de fórmulas matemáticas, gráficos, tabelas e outros elementos visuais. Além disso, os documentos gerados podem ser exportados para diversos formatos, como HTML, PDF, MS Word, entre outros. O **Quarto** também foi utilizado através do **VSCode**, com a sua extensão disponível em https://quarto.org/docs/tools/vscode.html.

No contexto deste trabalho, o **Quarto** foi utilizado para a produção de todo o texto, garantindo conformidade com as normas da ABNT. Sua capacidade de integrar texto, código e gráficos de maneira organizada foi essencial para facilitar o desenvolvimento do

documento.

3 Modelos utilizados no projeto

Neste capítulo, serão descritos os algoritmos de aprendizado de máquina e estatísticos utilizados no projeto. Alguns dos métodos utilizados podem fazer uso de diversos algoritmos, modelos estatísticos, regressão com regularização. Entre os algoritmos de aprendizagem de máquinas utilizados, estão aqueles que são baseados em árvores e uma rede neural conhecida como LSTM (Long Short-Term Memory). Dessa forma, esse capítulo começará explicando os algoritmos baseados em árvores utilizados no projeto, fundamentando primeiramente as árvores de decisão.

3.1 Árvores de decisão

Árvores de decisão podem ser utilizadas tanto para regressão quanto para classificação. Elas servem de base para os modelos baseados em árvores empregados neste trabalho, focando particularmente nas árvores de regressão. O processo de construção de uma árvore se baseia no particionamento recursivo do espaço dos preditores, onde cada particionamento é chamado de nó e o resultado final é chamado de folha ou nó terminal. Em cada nó, é definida uma condição e, caso essa condição seja satisfeita, o resultado será uma das folhas desse nó. Caso contrário, o processo segue para o próximo nó e verifica a próxima condição, podendo gerar uma folha ou outro nó. Veja um exemplo na Figura 3.1.

O espaço dos preditores é dividido em J regiões distintas e disjuntas denotadas por $R_1, R_2, \ldots, R_J$. Essas regiões são construídas em formato de caixa de forma a minimizar a soma dos quadrados dos resíduos. Dessa forma, pode-se modelar a variável resposta como uma constante c_j em cada região R_j:

$$f(x) = \sum_{j=1}^{J} c_j I\left(x \in R_j\right).$$

O estimador para a constante c_j é encontrado pelo método de mínimos quadrados. Assim, deve-se minimizar $\sum_{x_i \in R_j} \left[y_i - f(x_i)\right]^2$. No entanto, perceba que $f(x_i)$ está sendo avaliado somente em um ponto específico x_i, o que reduzirá $f(x_i)$ para uma constante c_j. É fácil de se chegar ao resultado se for observada a definição da função indicadora $I\left(x \in R_j\right)$:

$$I_{R_j}(x_i) = \begin{cases} 1, & \text{se } x_i \in R_j \\ 0, & \text{se } x_i \notin R_j \end{cases}.$$

Como as regiões são disjuntas, x_i não pode estar simultaneamente em duas regiões. Assim, para um ponto específico x_i, apenas um dos casos da função indicadora será diferente de 0. Portanto, $f(x_i) = c_j$. Agora, derivando $\sum_{x_i \in R_j} \left(yi - c_j\right)^2$ em relação a c_j

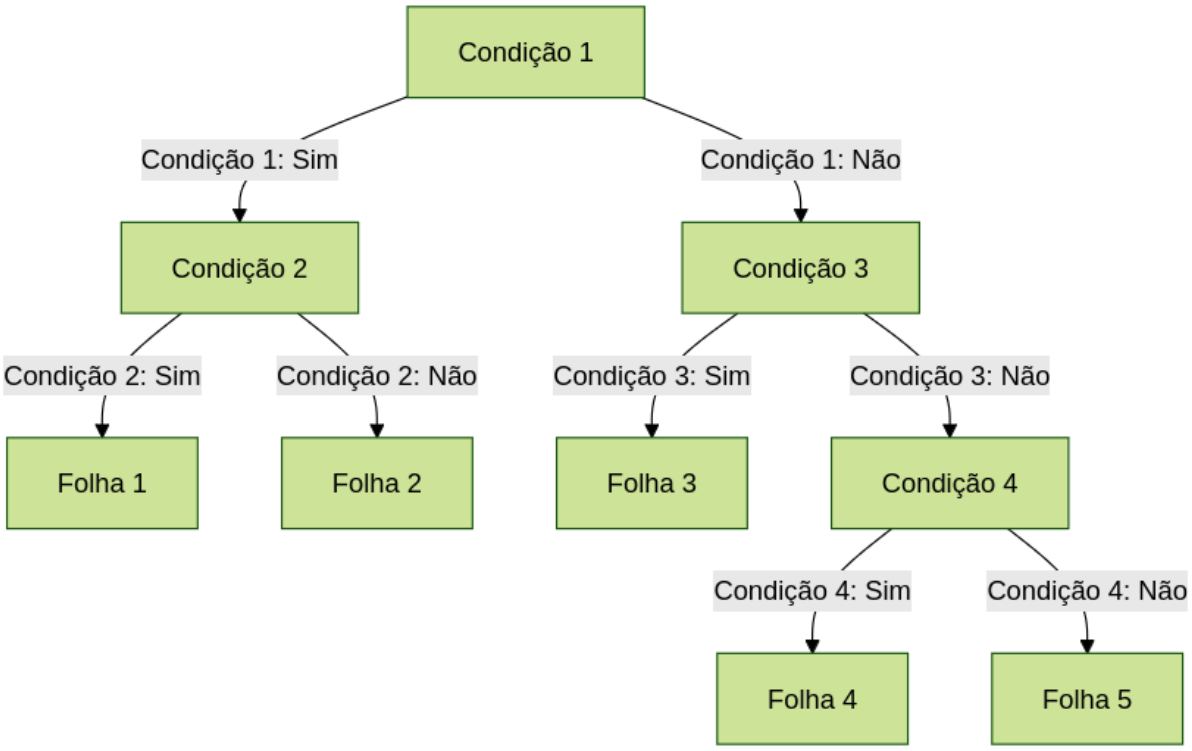

Figura 3.1: Exemplo de estrutura de árvore de regressão. A árvore tem cinco folhas e quatro nós internos.

$$\frac{\partial}{\partial c_j} \sum_{x_i \in R_j} (y_i - c_j)^2 = -2 \sum_{x_i \in R_j} (y_i - c_j) \tag{3.1}$$

e, ao igualar a Equação 3.1 a 0, tem-se a seguinte igualdade:

$$\sum_{x_i \in R_j} (y_i - \hat{c}_j) = 0.$$

Expandindo o somatório e dividindo pelo número total de pontos N_j na região R_j, concluí-se que o estimador de c_j, denotado por $\hat{c}_j$, é simplesmente a média dos valores observados y_i dentro da região R_j:

$$\sum_{x_i \in R_j} y_i - \hat{c}_j N_j = 0 \Rightarrow \hat{c}_j = \frac{1}{N_j} \sum_{x_i \in R_j} y_i. \tag{3.2}$$

No entanto, JAMES et al. (2013) caracteriza como inviável considerar todas as possíveis partições do espaço das variáveis em J caixas devido ao alto custo computacional. Dessa forma, a abordagem a ser adotada é uma divisão binária recursiva. O processo começa no topo da árvore de regressão, o ponto em que contém todas as observações, e continua sucessivamente dividindo o espaço dos preditores. As divisões são indicadas como dois novos ramos na árvore, como pode ser visto na Figura 3.1.

Para executar a divisão binária recursiva, deve-se primeiramente selecionar a variável independente X_j e o ponto de corte s tal que a divisão do espaço dos preditores conduza

a maior redução possível na soma dos quadrados dos resíduos. Dessa forma, definimos dois semi-planos:

$$R_1(j,s) = \{X|X_j \leq s\} \text{ e } R_2(j,s) = \{X|X_j > s\},$$

e procuramos a divisão da variável j e o ponto de corte s que minimizem a seguinte expressão:

$$\min_{j,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2 + \min_{c_2}\sum_{x_i\in R_2(j,s)}(y_i-c_2)^2\right],$$

em que c_1 e c_2 é a média da variável dependente para as observações nas regiões $R_1(j,s)$ e $R_2(j,s)$, respectivamente. Após determinar a melhor divisão, os dados são particionados nessas duas regiões, e o processo é repetido recursivamente para todas as sub-regiões resultantes.

O tamanho da árvore pode ser considerado um hiperparâmetro para regular a complexidade do modelo, pois uma árvore muito grande pode causar sobreajuste aos dados de treinamento, capturando não apenas os padrões relevantes, mas também o ruído. Como resultado, o modelo pode apresentar bom desempenho nos dados de treinamento, mas falhar ao lidar com novos dados devido à sua incapacidade de generalização. Por outro lado, uma árvore muito pequena pode não captar padrões, relações e estruturas importantes presentes nos dados. Dessa forma, a estratégia adotada para selecionar o tamanho da árvore consiste em crescer uma grande árvore T_0, interrompendo o processo de divisão apenas ao atingir um tamanho mínimo de nós. Posteriormente, a árvore T_0 é podada utilizando o critério de custo complexidade, que será definido a seguir.

Para o processo de poda da árvore, definimos uma árvore qualquer T que pode ser obtida através do processo da poda de T_0, de modo que $T \subset T_0$. Assim, sendo N_j a quantidade de pontos na região R_j, seja

$$Q_j(T) = \frac{1}{N_j}\sum_{x_i\in R_j}\left(y_i - \hat{c}_j\right)^2$$

uma medida de impureza do nó pelo erro quadrático médio. Assim, define-se o critério de custo complexidade:

$$C_\alpha(T) = \sum_{j=1}^{|T|} N_j Q_j(T) + \alpha|T|,$$

em que $|T|$ denota a quantidade total de folhas, e $\alpha \geq 0$ é um hiperparâmetro que equilibra o tamanho da árvore e a adequação aos dados. A ideia é encontrar, para cada α, a árvore $T_\alpha \subset T_0$ que minimiza $C_\alpha(T)$. Valores grandes de α resultam em árvores menores, enquanto valores menores resultam em árvores maiores, e $\alpha = 0$ resulta na própria árvore T_0. A busca por T_α envolve colapsar sucessivamente o nó interno que provoca o menor aumento em $\sum_j N_j Q_j(T)$, continuando o processo até produzir uma árvore com um único nó. Esse processo gera uma sequência de subárvores, na qual existe uma única subárvore menor que, para cada α, minimiza $C_\alpha(T)$.

A estimação de α pode ser realizada por validação cruzada com cinco ou dez folds, sendo

$\hat{\alpha}$ escolhido para minimizar a soma dos quadrados dos resíduos durante o processo de validação cruzada. Assim, a árvore final será $T_{\hat{\alpha}}$. O Algoritmo 3.1 exemplifica o processo de crescimento de uma árvore de regressão:

Algoritmo 3.1 Algoritmo para crescer uma árvore de regressão.

1. Use a divisão binária recursiva para crescer uma árvore grande T_0 nos dados de treinamento, parando apenas quando cada folha tiver menos do que um número mínimo de observações.

2. Aplique o critério custo de complexidade à árvore grande T_0 para obter uma sequência de melhores subárvores T_α, em função de α.

3. Use validação cruzada K-fold para escolher α. Isto é, divida as observações de treinamento em K folds. Para cada $k = 1, \ldots, K$:

(a) Repita os Passos 1 e 2 em todos os folds, exceto no k-ésimo fold dos dados de treinamento.

(b) Avalie o erro quadrático médio da previsão no k-ésimo fold deixado de fora, em função de α.

Faça a média dos resultados para cada valor de α e escolha α que minimize o erro médio.

4. Retorne a subárvore $T_{\hat{\alpha}}$ do Passo 2 que corresponde ao valor estimado de α.

Algoritmo 3.1: Fonte: JAMES et al. (2013, p. 337).

No caso de uma árvore de decisão para classificação, a principal diferença está no critério de divisão dos nós e na poda da árvore. Para a classificação, a previsão em um nó j, correspondente a uma região R_j com N_j observações, será simplesmente a classe majoritária. Assim, tem-se:

$$\hat{p}_{jk} = \frac{1}{N_j} \sum_{x_i \in R_j} I\left(y_i = k\right),$$

como sendo a proporção de observações da classe k no nó j. Dessa forma, as observações no nó j são classificadas na classe $k\left(j\right) = \arg\max_k \hat{p}_{jk}$, que é a moda no nó j.

Para a divisão dos nós no caso da regressão, foi utilizado o erro quadrático médio como medida de impureza. Para a classificação, algumas medidas comuns para $Q_j\left(T\right)$ são o erro de classificação, o índice de Gini ou a entropia cruzada.

3.2 Métodos Ensemble

As árvores de decisão são conhecidas por sua alta interpretabilidade, mas geralmente apresentam um desempenho preditivo inferior em comparação com outros modelos e algoritmos. No entanto, é possível superar essa limitação construindo um modelo preditivo que combina a força de uma coleção de estimadores base, um processo conhecido como aprendizado em conjunto (Ensemble Learning). De acordo com HASTIE et al. (2009), o aprendizado em conjunto pode ser dividido em duas etapas principais: a primeira etapa

consiste em desenvolver uma população de algoritmos de aprendizado base a partir dos dados de treinamento, e a segunda etapa envolve a combinação desses algoritmos para formar um estimador agregado. Portanto, nesta seção, serão definidos os métodos de aprendizado em conjunto utilizados neste trabalho.

3.2.1 Random Forest

O algoritmo Random Forest é uma técnica derivada do método de Bagging, mas com modificações específicas na construção das árvores. O objetivo é melhorar a redução da variância ao diminuir a correlação entre as árvores, sem aumentar significativamente a variabilidade. Isso é alcançado durante o processo de crescimento das árvores por meio da seleção aleatória de variáveis independentes.

Algoritmo 3.2 Algoritmo de uma Random Forest para regressão ou classificação.

1. Para b = 1 até B:

(a) Construa amostras bootstrap $\mathcal{L}^*$ de tamanho N dos dados de treinamento.

(b) Faça crescer uma árvore de floresta aleatória T_b para os dados bootstrap, repetindo recursivamente os seguintes passos para cada folha da árvore, até que o tamanho mínimo do nó n_{min} seja atingido:

i. Selecione m variáveis aleatoriamente entre as p variáveis.
ii. Escolha a melhor variável entre as m.
iii. Divida o nó em dois subnós.

2. Por fim, o conjunto de árvores $\{T_b\}_1^B$ é construído.

No caso da regressão, para fazer uma predição em um novo ponto x, temos a seguinte função:

$$\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$$

Para a classificação é utilizado o voto majoritário. Assim, seja $\hat{C}_b(x)$ a previsão da classe da árvore de floresta aleatória b. Assim:

$$\hat{C}_{rf}^{B}(x) = \arg\max_{c} \sum_{b=1}^{B} I\left(\hat{C}_b(x) = c\right),$$

em que c representa as classes possíveis.

Algoritmo 3.2: Fonte: HASTIE et al. (2009, p. 588).

No algoritmo Random Forest, ao construir uma árvore a partir de amostras bootstrap, selecionam-se aleatoriamente $m \leq p$ das p variáveis independentes como candidatas para a divisão, antes de cada ramificação (com $m = p$ no caso do Bagging). Dessa forma, diferente do Bagging, aqui não se considera todas as p variáveis independentes para realizar

a divisão e minimizar a impureza, mas apenas m dessas p variáveis. A escolha aleatória de apenas m covariáveis como candidatas para a divisão ajuda a solucionar um dos principais problemas do algoritmo de Bagging, que tende a gerar árvores de decisão semelhantes, resultando em previsões altamente correlacionadas. O Random Forest busca diminuir esse problema ao criar oportunidades para que diferentes preditores sejam considerados. Em média, uma fração $(p-m)/p$ das divisões nem sequer incluirá o preditor mais forte como candidato, permitindo que outros preditores tenham a chance de serem selecionados (JAMES et al., 2013). Esse mecanismo reduz a correlação entre as árvores, o que, por sua vez, diminui a variabilidade das predições produzidas pelas árvores.

A quantidade de variáveis independentes m selecionadas aleatoriamente é um hiperparâmetro que pode ser estimado por meio de validação cruzada. Valores comuns para m são $m = \sqrt{p}$ com tamanho mínimo do nó igual a um para classificação, e $m = p/3$ com tamanho mínimo do nó igual a cinco para regressão (HASTIE et al., 2009). Quando o número de variáveis é grande, mas poucas são realmente relevantes, o algoritmo Random Forest pode ter um desempenho inferior com valores pequenos de m, pois isso reduz as chances de selecionar as variáveis mais importantes. No entanto, usar um valor pequeno de m pode ser vantajoso quando há muitos preditores correlacionados. Além disso, assim como no Bagging, a Random Forest não sofre de sobreajuste com o aumento da quantidade de árvores B. Portanto, é suficiente usar um B grande o bastante para que a taxa de erro se estabilize (JAMES et al., 2013).

Outro método bastante utilizado e semelhante ao algoritmo de Random Forest é o Extra Trees (ou Extremely Randomized Trees). A principal diferença entre eles está na forma como as árvores são construídas. No algoritmo Extra Trees, as divisões nos nós são feitas de maneira completamente aleatória, sem a busca exaustiva pela melhor variável ou pelo melhor ponto de corte. Essa aleatoriedade reduz significativamente o custo computacional, tornando o método mais eficiente em termos de tempo de processamento, especialmente em conjuntos de dados de grande dimensão.

3.3 Modelos de séries temporais

3.3.1 Suavização exponencial sazonal de Holt-Winters

O método de suavização exponencial de Holt-Winters (HW) é uma extensão da suavização exponencial simples para séries temporais que apresentam tendência e sazonalidade. Esse método baseia-se em três equações, cada uma com uma constante de suavização distinta, associadas às três componentes fundamentais da série: nível, tendência e sazonalidade (MORETTIN; TOLOI, 2022).

Considerando uma série com sazonalidade de período s, a forma mais comum do método assume um componente sazonal multiplicativo e uma tendência aditiva. Nesse caso, a série é modelada por:

$$Z_t = \mu_t F_t + T_t + a_t, t = 1, \cdots, N.$$

As três equações de suavização associadas ao fator sazonal, ao nível e à tendência são, respectivamente:

$$\hat{F}_t = D\left(\frac{Z_t}{\bar{Z}_t}\right) + (1-D)\,\hat{F}_{t-s},\ 0 < D < 1,\ t = s+1, \cdots, N,$$
$$\bar{Z}_t = A\left(\frac{Z_t}{\hat{F}_{t-s}}\right) + (1-A)\left(\hat{Z}_{t-1} + \hat{T}_{t-1}\right),\ 0 < A < 1,\ t = s+1, \cdots, N,$$
$$\hat{T}_t = C\left(\bar{Z}_t - \bar{Z}_{t-1}\right) + (1-C)\,\hat{T}_{t-1},\ 0 < C < 1,\ t = s+1, \cdots, N$$

As constantes A, C e D representam os parâmetros de suavização do nível, da tendência e da sazonalidade, respectivamente. A previsão futura dos valores, nesse caso, é dada por:

$$\hat{Z}_{t+1}(h-1) = \left(\bar{Z}_{t+1} + (h-1)\,\hat{T}_{t+1}\right)\hat{F}_{t+1+h-s},\ h = 1, 2, \cdots, s+1$$

No caso em que o fator sazonal é aditivo, o modelo assume a forma:

$$Z_t = \mu_t + T_t + F_t + a_t.$$

As equações de suavização para o componente sazonal, o nível e a tendência, respectivamente, tornam-se:

$$\hat{F}_t = D\left(Z_t - \bar{Z}_t\right) + (1-D)\,\hat{F}_{t-s},\ 0 < D < 1,$$
$$\bar{Z}_t = A\left(Z_t - \hat{F}_{t-s}\right) + (1-A)\left(\bar{Z}_{t-1} + \hat{T}_{t-1}\right),\ 0 < A < 1,$$
$$\hat{T}_t = C\left(\bar{Z}_t - \bar{Z}_{t-1}\right) + (1-C)\,\hat{T}_{t-1},\ 0 < C < 1,$$

Por fim, a previsão dos valores futuros para o caso aditivo é dada por:

$$\hat{Z}_{t+1}(h-1) = \bar{Z}_{t+1} + (h-1)\,\hat{T}_{t+1} + \hat{F}_{t+1+h-s},\ h = 1, \cdots, s+1.$$

3.3.2 Naive sazonal

O método naive sazonal é uma abordagem bastante simples, porém eficaz. Ele utiliza a última observação conhecida do mesmo período sazonal para realizar as previsões. Assim, mesmo com sua simplicidade, consegue capturar variações sazonais presentes na série temporal.

Formalmente, a previsão para o tempo $t+h$ pode ser escrita como:

$$\hat{Z}_{t+h|t} = Z_{t+h-m(k+1)},$$

em que m representa o período da sazonalidade e k é a parte inteira de $(h-1)/m$, ou seja, o número de ciclos sazonais completos entre t e $t+h$.

Por exemplo, em séries mensais, a previsão para todos os futuros meses de fevereiro será igual ao valor observado no último mês de fevereiro disponível. Em séries trimestrais, a previsão para o segundo trimestre corresponderá ao último segundo trimestre observado. Essa regra se aplica analogamente a outras frequências sazonais (HYNDMAN; ATHANASOPOULOS, 2018).

3.3.3 LSTM

O modelo LSTM (Long Short-Term Memory) é uma arquitetura de rede neural recorrente (RNN) (GOODFELLOW et al., 2016) projetada para modelar sequências de dados com dependências temporais de longo prazo, superando limitações das RNNs tradicionais, que tendem a sofrer com o problema do desvanecimento ou explosão do gradiente durante o treinamento.

A principal inovação do LSTM está em sua estrutura interna, composta por uma célula de memória e três portas (gates) que controlam o fluxo de informações ao longo do tempo: a porta de entrada, a porta de esquecimento e a porta de saída.

A porta de entrada é responsável por definir quais informações da entrada atual serão armazenadas na célula de memória. A porta de esquecimento tem a função de decidir quais dados previamente armazenados devem ser descartados. Já a porta de saída determina quais informações contidas na célula de memória serão utilizadas como saída naquele momento.

Essas portas são implementadas por meio de funções sigmoides e multiplicações ponto a ponto, o que permite ao modelo aprender, de forma adaptativa, quais informações manter ou esquecer ao longo da sequência.

De maneira geral, o LSTM é capaz de capturar padrões complexos em séries temporais, sendo amplamente utilizado em tarefas como previsão de séries temporais, processamento de linguagem natural, reconhecimento de voz e análise de sentimentos. Sua capacidade de armazenar e manipular informações por longos intervalos de tempo o torna particularmente adequado para contextos em que eventos passados distantes são relevantes para a predição futura.

3.3.4 ARIMA

Os modelos de séries temporais da classe ARIMA estão entre os mais utilizados na modelagem de dados temporais. Seu principal objetivo é descrever as autocorrelações presentes nos dados.

Os modelos ARIMA são compostos por dois componentes fundamentais: o modelo autorregressivo (AR) e o modelo de médias móveis (MA). O modelo autorregressivo utiliza uma combinação linear de valores passados da variável. Assim, um modelo AR de ordem p pode ser escrito como:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t,$$

em que ϵ_t representa o ruído branco. O modelo autorregressivo de ordem p é denotado por $\mathrm{AR}\,(p)$.

Por sua vez, o modelo de médias móveis de ordem q, denotado por $\mathrm{MA}\,(q)$, utiliza erros passados para prever os valores futuros da série. Ele é expresso por:

$$y_t = c + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}.$$

A partir da combinação dos modelos AR e MA, tem-se o modelo ARIMA, cuja estrutura

incorpora também a diferenciação da série temporal para torná-la estacionária. O modelo ARIMA é dado por:

$$y_t^{'} = c + \phi_1 y_{t-1}^{'} + \cdots + \phi_p y_{t-p}^{'} + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} + \epsilon_t,$$

em que $y_t^{'}$ representa a série temporal após ter sido diferenciada (possivelmente mais de uma vez). O modelo é denotado por ARIMA (p, d, q), em que p é a ordem da componente autorregressiva, d é o número de diferenciações aplicadas e q é a ordem da componente de médias móveis.

O modelo ARIMA apresentado anteriormente aplica-se ao caso em que não há sazonalidade. No entanto, os modelos ARIMA também podem ser estendidos para séries temporais com padrão sazonal. Nessa situação, são incluídas componentes sazonais adicionais, resultando no modelo sazonal ARIMA, denotado por:

$$\text{ARIMA}\,(p, d, q)\,(P, D, Q)_m$$

em que m representa o período da sazonalidade (por exemplo, o número de observações por ano), e P, D e Q correspondem, respectivamente, às ordens das componentes autorregressiva, de diferenciação e de médias móveis para a parte sazonal do modelo.

3.4 Elastic-Net

O modelo de Elastic-Net combina dois tipos de regressão com penalização, o Lasso e o Ridge, que servem para reduzir os coeficientes de regressão linear.

A regressão Ridge é um tipo de regressão que busca controlar os coeficientes impondo uma penalidade aos seus valores. Quando existem múltiplas variáveis correlacionadas em um modelo de regressão linear, os coeficientes podem exibir alta variância. Um coeficiente altamente positivo em uma variável pode ser cancelado por um altamente negativo em outra variável correlacionada. O objetivo da regressão Ridge é mitigar esse problema.

A regressão ridge é muito semelhante aos mínimos quadrados, exceto que os coeficientes são estimados minimizando uma quantidade ligeiramente diferente. Em particular, as estimativas do coeficiente de regressão de ridge $\widehat{\beta}^{\text{ridge}}$ são os valores que minimizam

$$\widehat{\beta}^{\text{ridge}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\},$$

em que $\lambda \geq 0$ é um parâmetro de ajuste, a ser determinado separadamente.

Assim como com os mínimos quadrados, a regressão ridge busca estimativas de coeficientes que se ajustem bem aos dados, tornando o soma dos quadrados dos erros pequeno. No entanto, o segundo termo, $\lambda \sum_{j=1}^{p} \beta_j^2$, chamado de penalidade de encolhimento (shrinkage penalty), é pequeno quando $\beta_1, \cdots, \beta_p$ estão próximos de zero e, portanto, tem o efeito de encolher as estimativas de β_j em direção a zero. O parâmetro de ajuste λ serve para controlar o impacto relativo desses dois termos nas estimativas do coeficiente de regressão.

Quando $\lambda = 0$, o termo de penalidade não tem efeito e a regressão ridge produzirá as estimativas dos mínimos quadrados. No entanto, conforme $\lambda \to \infty$, o impacto da

penalidade de encolhimento aumenta e as estimativas do coeficiente de regressão ridge se aproximarão de zero.

Na regressão lasso, a soma dos mínimos quadrados penalizada é dada por:

$$\hat{\beta}^{\text{lasso}} = \arg\min_{\beta} \left\{ \frac{1}{2} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\},$$

em que um valor de λ suficientemente grande fará com que alguns coeficientes sejam exatamente 0.

A regressão Lasso tem algumas limitações. Em problemas com p grande e n pequeno, a regressão lasso selecionará no máximo n variáveis. Além disso, em um grupo de variáveis muito correlacionadas, a lasso tende a selecionar apenas uma variável do grupo.

Para solucionar essas limitações a regressão elastic net adiciona a penalização quadrática da ridge à penalização da lasso. Dessa forma, o problema então torna-se:

$$\hat{\beta} = \arg\min_{\beta} \left\{ \frac{1}{2} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda_2 \sum_{j=1}^{p} \beta_j^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| \right\},$$

Com $\lambda_1 = 0$, temos ridge, com $\lambda_2 = 0$, temos lasso e com $\lambda_1 = \lambda_2 = 0$, temos regressão linear sem regularização.

4 Metodologia

4.1 Coleta dos dados

Os dados foram obtidos por meio do **Denodo**, plataforma utilizada pela empresa que centraliza informações de todos os setores. A partir do **Denodo**, foram coletadas 29 séries temporais distintas. Duas delas representam a quantidade de associados conquistados por cada agência a cada mês, segmentados entre pessoas físicas (PF) e pessoas jurídicas (PJ). Além dessas, foram extraídas séries relativas à quantidade de cartões emitidos mensalmente e ao rendimento obtido com investimentos em depósitos a prazo e à vista.

Além dos dados predominantemente relacionados a investimentos, associados e cartões, também foram coletadas informações sobre as despesas de cada agência. Entre essas, destacam-se as despesas operacionais, administrativas e a Provisão para Créditos de Liquidação Duvidosa (PCLD), que representa estimativas de perdas com empréstimos que provavelmente não serão quitados. Essas despesas impactam diretamente o lucro líquido da empresa. Complementando esse conjunto de dados, foram incluídas as receitas administrativas e operacionais, possibilitando a modelagem completa dessas séries temporais.

Como variáveis independentes, foram utilizados alguns indicadores econômicos apontados como relevantes por economistas da empresa. Entre eles, a taxa selic acumulada em 12 meses, que permite observar a tendência dos juros básicos da economia ao longo do tempo; a Taxa Referencial (TR), frequentemente utilizada como referência para correção de contratos e investimentos; o IPCA (Índice Nacional de Preços ao Consumidor Amplo), que mede a inflação oficial do país; e o IGP-M (Índice Geral de Preços – Mercado), amplamente utilizado para reajuste de contratos de aluguel e tarifas em geral. Esses indicadores foram considerados como potenciais explicativos para a variação dos produtos financeiros analisados. Vale destacar que esses foram os únicos dados não extraídos diretamente do Denodo, tratam-se de projeções disponibilizadas por economistas do Banco Central do Brasil (BACEN), com estimativas futuras para cada indicador até o ano de 2030.

Esses dados foram coletados para todas as agências das cooperativas que integram a Central Sicredi Nordeste. Na região Nordeste, existem 15 cooperativas, cada uma composta por diversas agências subordinadas. No total, essas cooperativas reúnem mais de 300 agências.

Devido à grande quantidade de séries temporais — consequência direta do número elevado de agências — tornou-se inviável realizar uma análise minuciosa do comportamento das previsões de cada série individualmente em relação às métricas de erro adotadas. Por isso, foi necessário adotar um processo mais automatizado para a escolha dos modelos, com base no desempenho das métricas de erro. Sendo assim, a próxima seção será dedicada à descrição dessas métricas, bem como ao processo de otimização dos hiperparâmetros do modelo selecionado para cada agência.

4.2 Construção do modelo

4.2.1 Etapas de pré-processamento

Para todas as séries temporais e variáveis independentes, foi inicialmente aplicada a transformação de Yeo-Johnson (YEO; JOHNSON, 2000), que apresenta a vantagem de ser aplicável a dados que incluem valores negativos e zero — como é o caso das séries temporais analisadas neste trabalho. Essa transformação foi aplicada com o objetivo de tornar a distribuição dos dados mais normais e estabilizar a sua variância.

A transformação de Yeo-Johnson é definida da seguinte forma:

$$\psi(\lambda, x) = \begin{cases} [(1+x)^{\lambda} - 1]/\lambda & \lambda \neq 0,\ x \geq 0 \\ \ln(1+x) & \lambda = 0,\ x \geq 0 \\ [(1-x)^{2-\lambda} - 1]/(\lambda - 2) & \lambda \neq 2,\ x < 0 \\ -\ln(1-x) & \lambda = 2,\ x < 0 \end{cases},$$

em que λ é estimado por máxima verossimilhança. Além disso, percebe-se que a transformação $\log(1+x)$ é um caso particular da transformação de Yeo-Johnson quando $\lambda = 0$ e $x \geq 0$.

No entanto, antes da aplicação da transformação de Yeo-Johnson, foi necessário lidar com a presença de valores ausentes em algumas séries temporais. Para isso, utilizou-se o método de interpolação linear, disponível na biblioteca pandas, por meio da função `pandas.DataFrame.interpolate()`.

A interpolação linear estima os valores ausentes assumindo que os pontos adjacentes apresentam uma variação linear. Em outras palavras, o valor faltante é preenchido com base em uma reta traçada entre os valores anterior e posterior ao ponto ausente. Essa abordagem é especialmente útil quando os dados apresentam continuidade e uma variação relativamente suave entre observações consecutivas.

Além das variáveis independentes mencionadas anteriormente, também foram criadas variáveis derivadas da própria série temporal. Essas variáveis foram obtidas por meio da decomposição da série utilizando o método STL (Seasonal-Trend decomposition using Loess). A partir dessa decomposição, foram extraídos os componentes de tendência e sazonalidade, que passaram a ser utilizados como variáveis explicativas adicionais no modelo. Essa abordagem visa capturar padrões estruturais intrínsecos à série, enriquecendo a modelagem e contribuindo para uma melhor capacidade preditiva.

4.2.2 Otimização de hiperparâmetros

Existem diversas técnicas para a otimização de hiperparâmetros em aprendizado de máquina. Uma das mais comuns é o Grid Search. Segundo BISCHL et al. (2023), o Grid Search divide o intervalo contínuo de valores possíveis de cada hiperparâmetro em um conjunto de valores discretos, avaliando exaustivamente o desempenho do algoritmo para todas as combinações possíveis. No entanto, como o número de combinações cresce exponencialmente com o aumento do número de hiperparâmetros, o Grid Search apresenta um custo computacional elevado.

Por essa razão, métodos de otimização mais avançados, como a otimização bayesiana, têm ganhado destaque, pois oferecem um desempenho superior ao explorar o espaço de hiperparâmetros de maneira mais eficiente. Neste trabalho, utilizamos a otimização bayesiana para realizar a otimização dos modelos e determinar a quantidade ideal de vizinhos mais próximos na classe `KNNImputer`, empregada para a imputação de valores ausentes.

A otimização bayesiana não se refere a um algoritmo específico, mas sim a uma abordagem de otimização fundamentada na inferência bayesiana, que engloba uma ampla família de algoritmos (GARNETT, 2023). Além disso, a otimização bayesiana tem alcançado benchmarks superiores em comparação com outros algoritmos em diversos problemas complexos de otimização de hiperparâmetros (SNOEK; LAROCHELLE; ADAMS, 2012).

Diferentemente de outros algoritmos de otimização de hiperparâmetros, a otimização bayesiana ajusta suas tentativas futuras com base nos resultados obtidos anteriormente (YANG; SHAMI, 2020). Para definir os pontos de avaliação futuros, utiliza-se uma função probabilística $P(c|\lambda)$, que modela a relação entre os hiperparâmetros λ e a métrica de desempenho c (BERGSTRA; YAMINS; COX, 2013). A partir dessa função, estima-se, para cada conjunto de hiperparâmetros λ, a performance esperada $\hat{c}(\lambda)$ e a incerteza associada à predição $\hat{\sigma}(\lambda)$. Com essas estimativas, a distribuição preditiva derivada da função probabilística permite a aplicação de uma função de aquisição. Essa função orienta a escolha dos próximos pontos a serem avaliados, equilibrando exploitation (explorar regiões próximas às melhores observações anteriores) e exploration (investigar áreas ainda não exploradas).

Portanto, os algoritmos de otimização bayesiana são regidos pela relação $\lambda \rightarrow c(\lambda)$ e buscam equilibrar exploitation e exploration. Isso permite identificar as regiões mais promissoras no espaço de hiperparâmetros, ao mesmo tempo em que evita negligenciar possíveis configurações melhores em áreas ainda inexploradas.

4.2.3 Tree-Structured Parzen Estimator

A função probabilística utilizada para a otimização bayesiana neste trabalho foi a Tree-Structured Parzen Estimator (TPE). O TPE define duas funções de densidade, $l(\lambda)$ e $g(\lambda)$, que são empregadas para modelar a distribuição das variáveis no domínio dos hiperparâmetros λ (YANG; SHAMI, 2020). Essas densidades são utilizadas para estimar a probabilidade condicional $p(\lambda|y)$ de observar uma combinação de hiperparâmetros λ, dado um valor de métrica de desempenho y. A definição é dada por:

$$P(\lambda|y) = \begin{cases} l(\lambda) & \text{if } y < y^* \\ g(\lambda) & \text{if } y \geq y^* \end{cases}, \tag{4.1}$$

em que $l(\lambda)$ representa a densidade associada aos valores de y menores que o limiar y^* e $g(\lambda)$ é a densidade associada aos valores de y iguais ou superiores a y^* (BERGSTRA et al., 2011). No algoritmo de TPE, o valor de y^* é definido como sendo um quantil γ dos valores observados de y, de forma que $p(y < y^*) = \gamma$.

Por padrão, o Tree-Structured Parzen Estimator (TPE) utiliza como função de aquisição o Expected Improvement (EI). O EI representa a expectativa de um modelo M, que mapeia

$f : \Lambda \to \mathbb{R}^N$, sobre a melhora esperada em relação a um limiar y^*. Formalmente, o EI é definido como:

$$EI_{y^*}(\lambda) = \int_{-\infty}^{\infty} \max(y^* - y, 0)\, p_M(y|\lambda)\, dy.$$

Se, para o valor de λ, o modelo prevê um y tal que $y > y^*$, a diferença $y^* - y$ será negativa, e o retorno será 0, o que significa que não haverá melhora. Por outro lado, se $y < y^*$, a diferença será positiva, indicando que o modelo apresenta um desempenho superior em relação ao limiar y^*.

Portanto, para o cálculo do EI utilizando as definições dadas pelo TPE e assumindo que $y^* > y$, adota-se a parametrização de $p(\lambda, y)$ como $p(y)\, p(\lambda \mid y)$, com o objetivo de simplificar os cálculos. Assim, a expressão do EI, para o TPE, se reduz a:

$$EI_{y^*}(\lambda) = \int_{-\infty}^{y^*} (y^* - y)\, p(y|\lambda)\, dy = \int_{-\infty}^{y^*} (y^* - y) \frac{p(\lambda|y)\, p(y)}{p(\lambda)} dy. \tag{4.2}$$

A partir da Equação 4.1 e da definição $p(y < y^*) = \gamma$ e $p(y \geq y^*) = 1 - \gamma$, pode-se encontrar a distribuição marginal $p(\lambda)$. Dessa forma, segue-se que:

$$\begin{aligned}
p(\lambda) &= \int_{-\infty}^{\infty} p(\lambda, y)\, dy \\
&= \int_{-\infty}^{\infty} p(\lambda|y)\, p(y)\, dy \\
&= \int_{-\infty}^{y^*} p(\lambda|y)\, p(y)\, dy + \int_{y^*}^{\infty} p(\lambda|y)\, p(y)\, dy \\
&= \int_{-\infty}^{y^*} l(\lambda)\, p(y)\, dy + \int_{y^*}^{\infty} g(\lambda)\, p(y)\, dy \\
&= \gamma l(\lambda) + (1 - \gamma)\, g(\lambda).
\end{aligned}$$

Agora, basta calcular a integral $\int_{-\infty}^{y^*} (y^* - y)\, p(\lambda|y)\, p(y)\, dy$. Tem-se, portanto:

$$\begin{aligned}
\int_{-\infty}^{y^*} (y^* - y)\, p(\lambda|y)\, p(y)\, dy &= l(\lambda) \int_{-\infty}^{y^*} (y^* - y)\, p(y)\, dy \\
&= \gamma y^* l(\lambda) - l(\lambda) \int_{-\infty}^{y^*} y p(y)\, dy.
\end{aligned}$$

Finalmente, substituindo essa última expressão encontrada e $p(\lambda)$ na Equação 4.2, chega-se a:

$$EI_{y^*}(\lambda) = \frac{\gamma y^* l(\lambda) - l(\lambda) \int_{-\infty}^{y^*} y p(y)\, dy}{\gamma l(\lambda) + (1 - \gamma)\, g(\lambda)} \propto \left[\gamma + \frac{g(\lambda)}{l(\lambda)}(1 - \gamma)\right]^{-1}.$$

Essa última expressão mostra que, para maximizar o Expected Improvement, é necessário encontrar valores de λ que apresentem alta probabilidade em $l(\lambda)$ e baixa probabilidade em $g(\lambda)$. Portanto, no TPE, maximizar o EI equivale a maximizar a razão $l(\lambda)/g(\lambda)$.

4.2.4 Otimização de hiperparâmetros com optuna

Para otimizar os hiperparâmetros dos modelos foi utilizado a bilbioteca **Optuna** da linguagem de programação **Python**. Essa biblioteca implementa diversos métodos para otimização automatizada de hiperparâmetros. Para a sua utilização, é preciso definir inicialmente uma função objetivo. Por exemplo, para otimizar os hiperparâmetros de uma random forest, é necessário definir uma função objetivo da seguinte forma:

```
import optuna
import numpy as np
import pandas as pd
from sklearn import ensemble
from sklearn.model_selection import cross_val_score, KFold

def objective(trial):
    X = train_df[variaveis_independentes]
    y = train_df.variavel_dependente

    params = dict(
        n_estimators=trial.suggest_int(
          name='n_estimators',
          low=1,
          high=1000),
        max_depth=trial.suggest_int(
          name='max_depth',
          low=20,
          high=1000),
        max_features='sqrt',
        random_state=42
    )

    model = ensemble.RandomForestRegressor(
        *params
    )
    model.fit(X=X, y=y)

    cv_scores = np.expm1(np.sqrt(-cross_val_score(
        estimator=model,
        X=X,
        y=y,
        scoring="neg_mean_squared_error",
        n_jobs=3,
        cv=KFold(n_splits=20))))

    return np.mean(cv_scores)

study = optuna.create_study()
study.optimize(objective, n_trials=100, n_jobs=-1)
```

Primeiro, define-se, em cada tentativa (trial), quais hiperparâmetros serão otimizados, especificados no objeto `params` no início da função. Após definir o espaço de busca para cada hiperparâmetro, o modelo escolhido é ajustado aos dados de treinamento. Com o modelo ajustado, realiza-se a validação cruzada em cada trial, utilizando o método K-Fold com 20 divisões (folds), conforme definido previamente. Após definir a função objetivo, inicializa-se um estudo com `optuna.create_study` e, em seguida, inicia-se a otimização com `study.optimize(objective, n_trials=100, n_jobs=-1)`. Por fim, para selecionar os melhores hiperparâmetros ao fim do último trial, basta executar `study.best_params`.

Por padrão, a biblioteca **Optuna** utiliza o Tree-Structured Parzen Estimator (TPE) para otimizar hiperparâmetros de um modelo. A técnica de otimização é escolhida por meio do argumento `sampler` no método `create_study`. Para selecionar o TPE, basta passar `optuna.samplers.TPESampler` como argumento para a criação do estudo. O método TPE é o padrão para otimização na biblioteca **Optuna**. No entanto, caso se deseje utilizar outro método de otimização da biblioteca, basta especificá-lo da mesma forma: `optuna.create_study(sampler=metodo_otimizacao)`.

A função objetivo definido no **Optuna** foi utilizada com a métrica de error MAPE (Erro Percentual Absoluto Médio), que entrega boas interpretações sobre o comportamento do modelo. Essa métrica mede, em termos percentuais, o desvio médio entre os valores estimados e os valores observados, oferecendo uma interpretação relativa ao erro.

5 Produto final

6 Conclusão

Referências

AKIBA, T. et al. Optuna: A Next-generation Hyperparameter Optimization Framework. [S.l.]: [s.n.], 2019.

ALLAIRE, J.; DERVIEUX, C. **quarto: R Interface to 'Quarto' Markdown Publishing System**. [S.l.]: [s.n.], 2024.

BAYER, M. SQLAlchemy. *Em*: BROWN, A.; WILSON, G. (Org.). **The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks**. [S.l.]: aosabook.org, 2012.

BERGSTRA, J. et al. Algorithms for hyper-parameter optimization. **Advances in neural information processing systems**, 2011. v. 24.

_____; YAMINS, D.; COX, D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. [S.l.]: PMLR, 2013. p. 115–123.

BISCHL, B. et al. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. **Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery**, 2023. v. 13, n. 2, p. e1484.

GARNETT, R. **Bayesian optimization**. [S.l.]: Cambridge University Press, 2023.

GOODFELLOW, I. et al. **Deep learning**. [S.l.]: MIT press Cambridge, 2016. V. 1.

HARRIS, C. R. et al. Array programming with NumPy. **Nature**, set. 2020. v. 585, n. 7825, p. 357–362. Disponível em: <https://doi.org/10.1038/s41586-020-2649-2>.

HASTIE, T. et al. **The elements of statistical learning: data mining, inference, and prediction**. [S.l.]: Springer, 2009. V. 2.

HOSSAIN, Shammamah. Visualization of Bioinformatics Data with Dash Bio. (Chris Calloway et al., Org.). [S.l.]: [s.n.], 2019. p. 126–133.

HUNTER, J. D. Matplotlib: A 2D graphics environment. **Computing in Science & Engineering**, 2007. v. 9, n. 3, p. 90–95.

HYNDMAN, R. J.; ATHANASOPOULOS, G. **Forecasting: principles and practice**. [S.l.]: OTexts, 2018.

INC., P. T. Collaborative data science. 2015. Disponível em: <https://plot.ly>.

JAMES, G. et al. **An introduction to statistical learning**. [S.l.]: Springer, 2013. V. 112.

MORETTIN, P. A.; TOLOI, C. M. De C. Análise de séries temporais. 2022.

SNOEK, J.; LAROCHELLE, H.; ADAMS, R. P. Practical bayesian optimization of machine learning algorithms. **Advances in neural information processing systems**, 2012. v. 25.

TEAM, T. Pandas Development. **pandas-dev/pandas: Pandas**. Zenodo. Disponível em: <https://doi.org/10.5281/zenodo.3509134>.

VIRTANEN, P. et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. **Nature Methods**, 2020. v. 17, p. 261–272.

YANG, L.; SHAMI, A. On hyperparameter optimization of machine learning algorithms: Theory and practice. **Neurocomputing**, 2020. v. 415, p. 295–316.

YEO, I.-K.; JOHNSON, R. A. A New Family of Power Transformations to Improve Normality or Symmetry. **Biometrika**, 2000. v. 87, n. 4, p. 954–959. Disponível em: <http://www.jstor.org/stable/2673623>. Acesso em: 25 out. 2023.