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 Tabelas

Lista de Algoritmos

1 Introdução

2 Sobre a empresa

3 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.

3.1 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.

3.2 Recursos utilizados para modelagem e visualização dos resultados

Com os dados coletados, a etapa de modelagem dos valores de imóveis torna-se essencial para converter essas informações em entendimentos relevantes e aplicáveis. Essa etapa permite identificar padrões de comportamento entre as variáveis que influenciam os valores imobiliários, além de determinar quais fatores exercem maior impacto sobre esses valores. Para chegar a esses resultados, foram utilizadas bibliotecas desenvolvidas em **Python**, como **scikit-learn** (PEDREGOSA, F. et al., 2011), **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.

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.

Vale destacar que bibliotecas como o **Matplotlib** e o **seaborn** (WASKOM, 2021) desempenharam um papel fundamental na geração dos gráficos e na visualização dos dados apresentados neste trabalho. Essas ferramentas foram indispensáveis para a análise exploratória e a apresentação dos resultados de forma clara e compreensível. Além dessas bibliotecas, também foram utilizados recursos voltados para a análise do impacto e das relações entre as variáveis e as previsões realizadas pelos modelos. Nesse contexto, destacam-se o **SHAP** (LUNDBERG; LEE, 2017), que fornece explicações interpretáveis para os modelos de aprendizado de máquina, e o próprio **scikit-learn**, que foi essencial para a criação dos gráficos de ICE (Individual Conditional Expectation). Esses métodos serão detalhados na seção de metodologia deste trabalho.

Vale ressaltar que algumas bibliotecas externas, baseadas na API do **scikit-learn**, também foram utilizadas neste trabalho. Entre elas, destacam-se a **LightGBM** (KE et al., 2017) e a **XGBoost** (CHEN, T.; GUESTRIN, 2016), empregadas para a implementação de algoritmos de aprendizado de máquina. Essas bibliotecas fornecem, respectivamente, os algoritmos de gradient boosting LightGBM e Extreme Gradient Boosting, reconhecidos por sua eficiência computacional e desempenho em tarefas de modelagem preditiva.

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.

[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.

3.3 Recursos utilizados para a criação da aplicação final

A aplicação final foi desenvolvida com o objetivo de realizar previsões de valores de imóveis e oferecer uma análise detalhada do mercado imobiliário da cidade de João Pessoa. 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 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')
```

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

```
)
def update_graph(selected_category):
    filtered_df = df[df['Categoria'] == selected_category]
    fig = px.bar(
      filtered_df,
      x='Categoria',
      y='Valores',
      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 valores imobiliários diretamente na aplicação. Não obstante, os mapas da cidade de João Pessoa, criados com a biblioteca **Folium** (PYTHON-VISUALIZATION, 2020), foram servidos pela API, que expôs seus arquivos HTML para consumo no front-end.

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.

Com as tecnologias mencionadas anteriormente e o modelo final obtido, foi possível criar uma aplicação capaz de realizar previsões para os imóveis da cidade de João Pessoa, além de proporcionar uma análise de seu setor imobiliário. No entanto, o desenvolvimento de todo o código da aplicação exigiu o uso de ferramentas que facilitassem sua escrita, desenvolvimento e organização. Assim, a seguir serão apresentadas as ferramentas utilizadas tanto para a implementação do código deste trabalho quanto para a elaboração do documento final de texto que o descreve.

3.4 Recursos utilizados para escrita de código e de documento

O desenvolvimento deste trabalho foi realizado em um computador equipado com processador AMD Ryzen 7 5800H (16 núcleos), 8 GB de memória RAM, placa de vídeo GeForce GTX 1650 e um SSD NVMe de 256 GB, operando sob o sistema Pop!_OS 22.04 LTS. Embora a máquina ofereça um desempenho geral satisfatório, a quantidade limitada de memória RAM apresentou desafios em tarefas mais intensivas, como a otimização de hiperparâmetros dos modelos. Essas tarefas frequentemente demandavam até dois dias para sua conclusão e, em alguns casos, falhavam próximo ao término devido ao alto consumo computacional. Dessa forma, a escolha das ferramentas utilizadas para a escrita do código foram as que impactassem o mínimo possível no desempenho do sistema.

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.

4 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.

4.1 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[1]. 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 4.1.

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

[1] Uma árvore de regressão é um caso específico da árvore de decisão, mas para regressão.

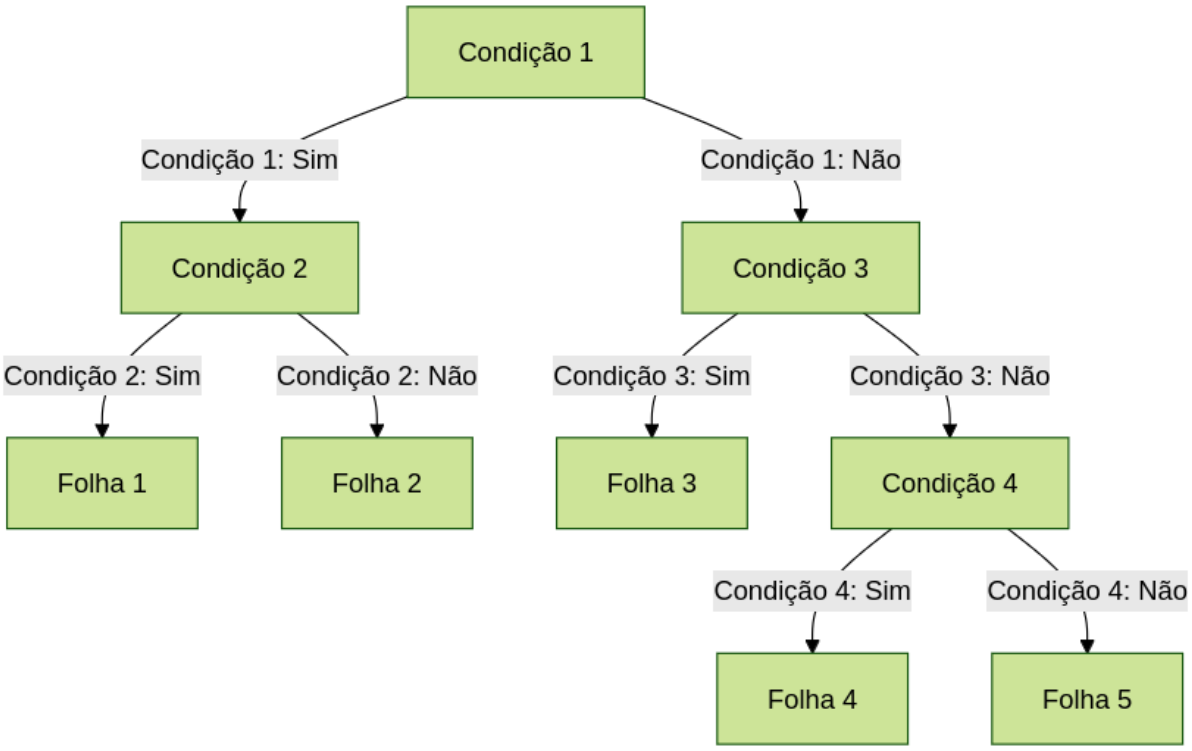

Figura 4.1: Exemplo de estrutura de árvore de regressão. A árvore tem cinco folhas e quatro nós internos.

$$\frac{\partial}{\partial c_j} \sum_{x_i \in R_j} (y_i - c_j)^2 = -2 \sum_{x_i \in R_j} (y_i - c_j) \tag{4.1}$$

e, ao igualar a Equação 4.1 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:

$$\sum_{x_i \in R_j} y_i - \hat{c}_j N_j = 0 \Rightarrow \hat{c}_j = \frac{1}{N_j} \sum_{x_i \in R_j} y_i. \tag{4.2}$$

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 4.1.

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:

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

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 4.1 exemplifica o processo de crescimento de uma árvore de regressão:

Algoritmo 4.1 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 4.1: 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.

4.2 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.

4.2.1 Random Forest

Algoritmo 4.2 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 4.2: Fonte: HASTIE et al. (2009, p. 588).

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.

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).

(FALAR SOBRE O EXTRA TREES)

4.3 Modelos de séries temporais

4.3.1 Suavização exponencial sazonal de Holt-Winters

O método de suavização exponencial de Holt-Winters (HW) é uma extensão da suavização exponencial para séries que contém tendência e sazonalidade. Esse método é baseado em três equações com constantes de suavização diferentes, que são associadas a cada uma das componentes do padrão da série: nível, tendência e sazonalidade (MORETTIN; TOLOI, 2022).

Considerando uma série sazonal com período s, a variante mais usual do método considera o fator sazonal F_t como sendo multiplicativo, enquanto a tendência permanece aditiva, isto é,

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

As três equações de suavização são dadas por

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

e representam estimativas do fator sazonal, do nível e da tendência, respectivamente. A,

C e D são as constantes de suavização. Por fim, para esse caso, a previsão dos valores futures se dá 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 procedimento anterior pode ser modificado para

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

Nesse caso, as estimativas do fator sazonal, nível e tendência da série são dadas por

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

respectivamente. As constantes A, C e D são as constantes de suavização. Por fim, para esse caso a previsão final é dada por

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

4.3.2 Naive sazonal

O naive sazonal é um método bastante simples. Ele utiliza a última observação conhecida do mesmo período para realizar as previsões. Dessa forma, embora seja simples, consegue capturar variações sazonais presentes na série.

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

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

em que m é o período sazonal e k é a parte inteira de $(h-1)/m$, isto é, o número de anos completos na previsão do período anterior ao tempo $t+h$. Por exemplo, com dados mensais, as previsões para todos futuros meses de fevereiro é igual ao último mês de fevereiro observado. Com dados trimestrais, a previsão para o segundo trimestre é igual ao último segundo trimestre observado. Essa regra se aplica a outros períodos sazonais (HYNDMAN; ATHANASOPOULOS, 2018).

4.3.3 ARIMA

4.3.4 LSTM

4.4 Elastic-Net

5 Metodologia

6 Produto final

7 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.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. [S.l.]: [s.n.], 2016. p. 785–794.

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.

KE, G. et al. Lightgbm: A highly efficient gradient boosting decision tree. **Advances in neural information processing systems**, 2017. v. 30.

LUNDBERG, S. M.; LEE, S.-I. A Unified Approach to Interpreting Model Predictions. *Em*: GUYON, I. et al. (Org.). **Advances in Neural Information Processing Systems 30**. [S.l.]: Curran Associates, Inc., 2017, p. 4765–4774.

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

PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python. **Journal of Machine Learning Research**, 2011. v. 12, p. 2825–2830.

PYTHON-VISUALIZATION. **Folium**. Disponível em: <https://python-visualization.github.io/folium/>.

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.

WASKOM, M. L. seaborn: statistical data visualization. **Journal of Open Source Software**, 2021. v. 6, n. 60, p. 3021. Disponível em: <https://doi.org/10.21105/joss.03021>.