UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

Sistema Automático para Geração de Modelos Base de Regressão

Leandro Giusti Mugnaini

São Carlos – SP

Sistema Automático para Geração de Modelos Base de Regressão

Leandro Giusti Mugnaini

Orientador: **Prof. Dr. Fernando Santos Osório**

Coorientador: **Prof. Dr. Diego Renan Bruno**

Monografia final de conclusão de curso apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação.
Área de Concentração: Aprendizagem de Máquina

USP – São Carlos
Novembro de 2021

Mugnaini, Leandro Giusti
Sistema Automático para Geração de Modelos Base de Regressão / Leandro Giusti Mugnaini. - São Carlos - SP, 2021.
56 p.; 29,7 cm.

Orientador: Fernando Santos Osório.
Coorientador: Diego Renan Bruno.
Monografia (Graduação) - Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos - SP, 2021.

1. Aprendizagem de Máquina. 2. Regressão. 3. Pré-Processamento. 4. Modelagem. 5. *Big Data*. I. Osório, Fernando Santos. II. Instituto de Ciências Matemáticas e de Computação (ICMC/USP). III. Título.

Este trabalho é dedicado à todas as pessoas que fizeram parte da minha história na longa jornada da vida.
O que eu me tornei hoje nada mais é do que a soma das contribuições individuais de cada um de vocês!!!

AGRADECIMENTOS

Gostaria de agradecer primeiramente à minha mãe Isabel e meu pai Eden pelo apoio emocional e suporte dados durante toda a minha trajetória, desde a infância até os dias de hoje, sempre me incentivando a estar em constante evolução. À minha irmã Aline, agradeço pelo companheirismo, apoio e bons momentos vividos durante todos esses anos. Essa graduação não teria sido concluída sem a ajuda de vocês três, muito obrigado por tudo!!!

Aos meus amigos Alyson (Xeros), Bruno (Adele), João Marco (Yaya), Leonardo (Leo) e Matheus (Barney), agradeço pelos bons momentos vividos durante a graduação (e fora dela) e pelo apoio dado durante todos esses anos. Sem dúvidas a trajetória se tornou mais simples e agradável quando nós nos tornamos amigos. Espero que nossa amizade dure para a vida toda!

Agradeço também aos meus companheiros de moradia, Gabriel e Fernando, que sempre me deram apoio nos momentos difíceis e me ensinaram lições valiosas que levarei para a vida.

Ao meu orientador Prof. Dr. Diego Renan Bruno, muito obrigado por ter aceitado a orientação e por toda a ajuda fornecida durante a elaboração deste projeto. Agradecimentos especiais ao Prof. Dr. Fernando Santos Osório por ter apoiado a ideia e permitido que o projeto fosse realizado.

Por fim, mas não menos importante, agradeço à toda população do Estado de São Paulo que, através do ICMS, custeia as atividades realizadas nas Universidades Estaduais de SP, tornando possível a formação de excelentes profissionais que impactarão de forma positiva a sociedade.

“O jogo não acaba aqui e o
mais importante ainda está por vir!”
(Prof. Dr. Clóvis de Barros Filho)

RESUMO

MUGNAINI, L. G.. **Sistema Automático para Geração de Modelos Base de Regressão**. 2021. 56 f. Monografia (Graduação) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Com o avanço da inteligência artificial e a ascensão da era da *Big Data*, o volume, variedade e velocidade dos dados que são coletados e que podem ser analisados cresce exponencialmente a cada dia. Neste contexto, surge a necessidade de otimizar o tempo dos profissionais que trabalham manipulando grandes conjuntos de dados - os cientistas de dados - para que eles possam focar em tarefas menos repetitivas e de maior valor agregado. Este trabalho apresenta as decisões de projeto e detalhes de implementação de um sistema gerador de modelos base para problemas de regressão, que pode ser utilizado no primeiro contato do cientista com o conjunto de dados que deve ser manipulado, gerando *insights* e direcionamentos sobre qual caminho deve ser tomado na etapa seguinte. O sistema apresenta um módulo de pré-processamento, que realizará manipulações genéricas nos dados, e um módulo de modelagem, que realizará a definição, treinamento e avaliação de modelos de regressão clássicos. O sistema pode ser facilmente customizado de acordo com a necessidade do usuário e sua saída estará disponível para consulta em um endereço Web devido a utilização do *MLFLow*, software de gerenciamento de modelos de aprendizagem de máquina.

Palavras-chave: Aprendizagem de Máquina, Regressão, Pré-Processamento, Modelagem, *Big Data*.

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Motivação e Contextualização

Por meio da revolução científica e tecnológica que vem ocorrendo nas últimas décadas, proporcionada principalmente pela evolução das técnicas de *software* e *hardware*, sistemas e soluções cada vez mais complexos estão sendo construídos, proporcionando mudanças radicais na forma que as pessoas vivem em sociedade (MAKRIDAKIS, 2017). Tais mudanças afetam aspectos em diferentes âmbitos, influenciando hábitos de consumo, meios de comunicação, formas de trabalho, relações pessoais e até políticas públicas de nações. Neste cenário, a inteligência artificial surge como grande protagonista, estando presente nos mais diversos setores da sociedade.

Com o advento da inteligência artificial e o início da era do *Big Data*, no qual os dados são coletados em volume e variedade cada vez maiores, em velocidades surpreendentes, surge também uma crescente demanda de profissionais que trabalham com dados. O cientista de dados é um dos profissionais que atuam neste cenário, manipulando dados, criando e utilizando modelos de aprendizagem de máquina, além de utilizar conhecimentos de computação, estatística e probabilidade para auxiliar na tomada de decisões.

Quando os cientistas de dados estão lidando com um problema pela primeira vez, uma quantidade de tempo considerável é empreendida buscando entender e preparar os dados, além de realizar análises com o objetivo de determinar qual o próximo passo que deve ser dado. No entanto, muitas dessas tarefas e análises são comuns a diferentes projetos, se tratando portanto de um processo que pode ser automatizado. Existem atualmente técnicas que realizam a automação de processos de *machine learning*, conceito chamado de *AutoML* (HE; ZHAO; CHU, 2021), porém muitas dessas técnicas ainda são apenas protótipos. As técnicas que estão mais avançadas (e tornaram-se inclusive soluções comerciais) são de difícil acesso devido ao alto custo das licenças comerciais.

A motivação deste projeto é de criar um sistema capaz de automatizar e agilizar a etapa inicial do trabalho do cientista de dados, focando o trabalho para problemas de regressão, tornando possível que o cientista foque seus esforços em tarefas que agregam mais valor ao projeto. Como se trata de um sistema regressor, o objetivo do sistema é utilizar as variáveis de entrada e através de um processo de modelagem, predizer um valor real para a variável alvo. Os problemas de regressão são caracterizados como aprendizado supervisionado e podem

assumir complexidades diversas, sendo utilizados em uma infinidade de cenários que vão desde o mercado financeiro, na predição de fraudes bancárias, até áreas de *E-commerce*, na precificação de produtos em plataformas *online* por exemplo.

1.2 Objetivos

O objetivo principal deste projeto é de automatizar a etapa inicial da análise de dados e modelos de aprendizagem de máquina, gerando resultados que possam ser conferidos rapidamente pelo cientista de dados, permitindo que ele prossiga com um desenvolvimento mais aprofundado e direcionado, em um menor tempo.

1.3 Organização

No Capítulo 2 são apresentados os métodos, técnicas e tecnologias utilizados para o desenvolvimento do trabalho, bem como referências bibliográficas que adicionam embasamento científico ao projeto. No Capítulo 3, apresentam-se a estrutura geral do sistema, detalhes das atividades realizadas, resultados obtidos e dificuldades/limitações encontradas. No Capítulo 4, encontram-se as conclusões sobre o projeto desenvolvido. Por fim, no Apêndice A, encontram-se detalhes sobre a instalação e execução do sistema.

Capítulo 2

MÉTODOS, TÉCNICAS E TECNOLOGIAS UTILIZADAS

2.1 Tecnologias Utilizadas

Para realizar o desenvolvimento do projeto, a linguagem escolhida foi a linguagem Python (ROSSUM; DRAKE, 2009). A escolha da linguagem deve-se a grande diversidade de pacotes e bibliotecas disponíveis, o que a tornou uma das principais ferramentas dos cientistas de dados (junto com a linguagem R). Os pacotes principais utilizados para manipulação, análise e modelagem de dados foram:

- Matplotlib: trata-se de uma biblioteca utilizada para criar visualizações estáticas, animadas e interativas, através da utilização de dados tabulares (HUNTER, 2007);
- scikit-learn: utilizado nas etapas de pré-processamento dos dados e na etapa de modelagem (PEDREGOSA; VAROQUAUX *et al.*, 2011);
- pandas: utilizado para leitura dos arquivos *csv* e manipulação dos dados (MCKINNEY *et al.*, 2010);
- xgboost: utilizado para fornecimento do modelo *XGBRegressor* (CHEN T.; GUESTRIN, 2016).
- SHAP (SHapley Additive exPlanations): utilizado para traçar visualizações que explicam a saída dos modelos de aprendizado de máquina utilizados (LUNDBERG; LEE, 2017b).

Para instalar os pacotes e executar o sistema, optou-se pela utilização de um ambiente virtual, utilizando o módulo *venv* da linguagem Python, de forma a isolar a instância Python do sistema operacional da instância Python do regressor, permitindo que o regressor e suas dependências sejam instalados e removidos com mais facilidade. Os pacotes utilizados devem ser instalados através do gerenciador de pacotes *pip*.

Além disso, utilizou-se o pacote *MLFLow* da linguagem Python. O *MLFlow* trata-se de uma plataforma de código aberto, utilizada para gerenciar os ciclos de vida de projetos de aprendizado de máquina. Neste projeto, o *MLFlow* será utilizado para facilitar a visualização e

registro dos resultados e métricas dos modelos testados, melhorando a experiência do usuário do sistema.

O *MLFlow* utilizará uma base de dados para registrar os artefatos, que no caso deste projeto serão as métricas de desempenho, modelos e visualizações. Para gerenciar essa base de dados, optou-se por utilizar o PostgreSQL. A utilização de uma base de dados é importante, pois permite que o usuário armazene as informações em um *cluster*, um servidor remoto ou até na máquina local, conforme a necessidade.

Mais detalhes sobre a instalação e configuração do sistema estão presentes na Seção 3. Além disso, as documentações de cada um dos pacotes e bibliotecas estão presentes no Apêndice A.

O sistema de regressão automático, depois de configurado, possui basicamente dois módulos principais: o módulo de pré-processamento e o módulo de modelagem. Os módulos foram desenvolvidos utilizando o paradigma de orientação a objetos, com o objetivo de permitir que as classes sejam herdadas e os métodos sobrescritos, caso alguma modificação pontual necessite ser feita para tratar de algum problema específico. Cada um dos módulos possui métodos que realizam tarefas genéricas distintas, que englobam manipulação, análise e modelagem dos dados.

Algumas etapas do módulo de pré-processamento e de modelagem são customizáveis e devem ser definidas pelo usuário através do preenchimento de um arquivo de configuração *conf.ini*. A escolha do formato *.ini* se deve pela simplicidade de sua estrutura, sendo composta por seções e propriedades. Um exemplo de arquivo .ini pode ser conferido no Código-fonte 1.

Código-fonte 1: Exemplo de arquivo .ini

```
1  [SEÇÃO1]
2  # Definindo o valor das chaves da SEÇÃO1
3
4  ChaveExemplo1=6090
5
6  ChaveExemplo2=10
7
8  ChaveExemplo3=/home/ubuntu/tcc/exemplo/
9
10
11 [SEÇÃO2]
12 # Definindo o valor das chaves da SEÇÃO2
13
14 ChaveExemplo1=1
15
16 ChaveExemplo2=/opt/ecs/mvuser/
```

Com o arquivo de configuração preenchido, basta o usuário ativar o ambiente virtual, ativar o MLFlow e executar os módulos de pré-processamento e modelagem. Desta forma, a execução será automaticamente detectada pelo *MLFlow* e os resultados estarão disponíveis para consulta.

2.2 Detalhamento do módulo: Pré-Processamento

Como apontado na Seção 2.1, os módulos de pré-processamento e modelagem possuem métodos que realizam manipulações, análises e modelagens genéricas nos dados, podendo ou não serem personalizadas pelo usuário.

No caso do módulo de pré-processamento, as seguintes tarefas são realizadas:

- Criação de um histograma com a distribuição dos valores da variável alvo;
- Criação de uma tabela contendo a correlação entre diferentes *features*;
- Remoção de colunas altamente correlacionadas, que excedam um limiar definido pelo usuário, deixando apenas uma das colunas no *dataset*;
- Remoção de colunas com valores nulos, que excedam um limiar definido pelo usuário;
- Remoção de colunas com variância nula;
- Conversão de colunas não numéricas para colunas numéricas;
- Imputação de valores faltantes;
- Normalização dos dados;
- Estandardização dos dados.

Cada uma das etapas são explicadas nas sub-seções a seguir.

2.2.1 Criação de histograma com a variável alvo e tabela de correlações

Para a criação do histograma com a distribuição dos valores da variável alvo, é necessário utilizar uma abordagem que leve em consideração a distribuição e a quantidade de valores no conjunto de dados analisado. Dessa forma, optou-se por definir a largura das barras do histograma de forma dinâmica, utilizando regra Freedman-Diaconi (FREEDMAN; DIACONIS, 1981), dada pela expressão:

$$Largura = 2\frac{IIQ(x)}{\sqrt[3]{n}} \tag{2.1}$$

Onde $IIQ(x)$ representa o intervalo interquartil dos dados e n representa o número de observações na amostra x.

Com esta regra, objetiva-se definir larguras de barras que sejam robustas a *outliers*, já que apenas a dispersão dos dados e o tamanho da amostra são levados em consideração.

Para a análise de correlação, utilizou-se o Coeficiente de Correlação de Pearson como métrica para avaliar a correlação entre diferentes *features*. O coeficiente de Pearson é definido pela seguinte expressão:

$$\rho = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2.2}$$

O coeficiente de correlação ρ está contido no intervalo $[-1, 1]$. Valores positivos indicam que as variáveis são positivamente correlacionadas, ou seja, se uma variável aumenta de valor a outra também aumenta. Já valores negativos indicam uma correlação negativa, indicando que quando uma variável aumenta, a outra diminui. Se $|\rho| \simeq 1$, as variáveis são altamente correlacionadas e se $\rho \simeq 0$, as variáveis são fracamente correlacionadas. No projeto, um arquivo *.csv* é automaticamente salvo, contendo a correlação entre todas as variáveis.

2.2.2 Remoção de variáveis altamente correlacionadas, com variância nula e com valores faltantes

Utilizando a análise de correlação, é possível remover variáveis altamente correlacionadas entre si. Com a era da Big Data, conjuntos de dados cada vez maiores estão sendo utilizados, fazendo com que os modelos demorem mais tempo para serem treinados, testados e validados. Dessa forma, a remoção de variáveis que são irrelevantes/redundantes ao problema pode trazer economias de tempo, poder computacional e em alguns casos até melhorar a capacidade de generalização do algoritmo (DASH; LIU, 1997). Assim, adicionou-se ao projeto a possibilidade do usuário definir um limiar de correlação, e caso variáveis superem esse limiar, apenas uma das duas variáveis será mantida.

Além da remoção de variáveis altamente correlacionadas, a remoção de variáveis com muitos atributos ausentes é outra técnica utilizada no pré-processamento de dados, sendo geralmente empregada quando não há risco de perda de informações importantes. Dessa forma, cabe ao usuário definir um limiar aceitável de valores nulos para as variáveis. Caso o usuário defina por exemplo, o valor 0.9, variáveis que possuam 90% ou mais valores nulos serão removidas. É importante ressaltar que essa técnica não deve ser utilizada quando o conjunto de dados possui poucas variáveis, já que neste caso qualquer perda de informações pode impactar no resultado do modelo.

A remoção de variáveis com variância nula também é uma técnica utilizada para diminuir o tamanho do conjunto de dados e reduzir a dimensionalidade dos dados. A variância (σ^2) da

população y_i, onde $i = 1,2,...,N$, é definida pela seguinte equação:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mu)^2 \tag{2.3}$$

Onde μ é a média da população y_i. Já no caso de variáveis Booleanas, que seguem uma distribuição de Bernoulli, a variância é dada por:

$$\sigma^2 = p(1-p) \tag{2.4}$$

Onde p representa a probabilidade de sucesso ($1 =$ sucesso, $0 =$ fracasso).

Variáveis com variância nula não fornecem melhorias de performance ao modelo, pois os modelos precisam identificar padrões distintos entre os dados para realizar predições e como todas as observações de determinada variável são idênticas, não é possível identificar um padrão que diferencie as observações, por isso é possível removê-las sem perdas na performance.

2.2.3 Conversão de variáveis para tipos de dados válidos e imputação de valores faltantes

Outro desafio enfrentado no pré-processamento, é garantir que os dados estejam com tipos de dados válidos. Nos problemas de regressão, os modelos aceitam tipos de dados numéricos (*int*, *float*, *bool*), dessa forma é necessário converter tipos de dados não numéricos (categóricos) para numéricos. Neste projeto, realizou-se o procedimento de *One-Hot Encoding*, que transformará as variáveis categóricas em novas variáveis binárias, que são aceitas como entrada pelos modelos. Um exemplo de transformação utilizando a técnica de *One-Hot Encoding* pode ser conferido nas Tabelas 1 e 2:

Tabela 1 – Exemplo de variável categórica antes da transformação *One-Hot Encoding*

Cor
Amarelo
Amarelo
Verde
Azul

Fonte: Elaborada pelo autor.

Outra técnica para tratamento de valores ausentes, é a imputação. Neste projeto, utilizou-se a imputação através do algoritmo KNN (*k-Nearest Neighbors*). Neste método, K vizinhos são escolhidos com base em alguma medida de distância e a média dos valores dos k vizinhos é usada como uma estimativa de imputação (TROYANSKAYA *et al.*, 2001). Desta forma, os valores faltantes são substituídos por valores prováveis que a variável realmente teria, apresentando um

Tabela 2 – Variável categórica transformada em variáveis numéricas depois da transformação *One-Hot Encoding*

Amarelo	Verde	Azul
1	0	0
1	0	0
0	1	0
0	0	1

Fonte: Elaborada pelo autor.

resultado melhor do que outras técnicas, como por exemplo uma simples substituição pela média da variável.

2.2.4 Estandardização e normalização dos dados

Por fim, para finalizar os detalhes do módulo de pré-processamento, é possível realizar a normalização ou estandardização dos dados (SOLA; SEVILLA, 1997). A técnica de normalização utilizada é a *Min-Max Scaling*, que utiliza os valores máximos e mínimos das variáveis para garantir que todos os dados estejam no intervalo $[0,1]$. A normalização Min-Max é dada por:

$$x_{norm} = \frac{x - min(x)}{max(x) - min(x)} \tag{2.5}$$

A normalização evita que o algoritmo fique enviesado para as variáveis com maior ordem de grandeza (evitando o *overfitting*), trazendo ganhos de performance no processo de modelagem. Já a estandardização (ou normalização $Z-score$), consiste em fazer com que os dados tenham média 0 e desvio padrão igual a 1. Para realizar a normalização, a seguinte expressão é utilizada:

$$z = \frac{x - \mu}{\sigma} \tag{2.6}$$

A escolha entre normalização ou estandardização deve ser realizada analisando a distribuição dos dados. Caso a distribuição não seja Gaussiana ou o desvio padrão pequeno, é recomendável a utilização da normalização. Caso contrário, a estandardização é uma boa técnica a ser utilizada. Neste projeto, o usuário deve escolher qual das duas técnicas ele deseja utilizar, mudando entre as técnicas de acordo com o conjunto de dados que está sendo analisado.

2.3 Detalhamento do módulo: Modelagem

Para o módulo de modelagem, as seguintes etapas são realizadas:

- Separação dos dados em treino e teste;
- Escolha dos modelos de aprendizagem de máquina desejados;

- Escolha das métricas para cálculo do desempenho;
- Criação dos gráficos de importância de variáveis;
- Registro dos resultados finais no MLFlow.

Cada uma das etapas é explicada nas sub-seções a seguir.

2.3.1 Separação dos dados em treino e teste

A separação dos dados em treino e teste é personalizada pelo usuário através de um parâmetro no arquivo *conf.ini*. O conjunto de treino passará por um processo de *k-fold Cross-Validation* (REFAEILZADEH; TANG; LIU, 2016), que separa o conjunto de treino em k diferentes *folds*, utilizando $k-1$ *folds* para treinamento e 1 *fold* para teste. O processo é repetido k vezes, de forma que todo o conjunto de treino é utilizado para treinamento e para teste. Após o treinamento e definição dos melhores parâmetros, o conjunto original de testes é utilizado para validar o processo de modelagem. Na Figura 1 é possível conferir o funcionamento deste processo, para o caso de $k = 5$.

O número de *folds* também é um parâmetro que deve ser definido pelo usuário, logo é possível configurar tanto o tamanho que o conjunto de treinamento terá, quanto o número de *folds* que ele será dividido, sendo possível ajustá-los de acordo com o problema que está sendo analisado e o tamanho do conjunto de dados.

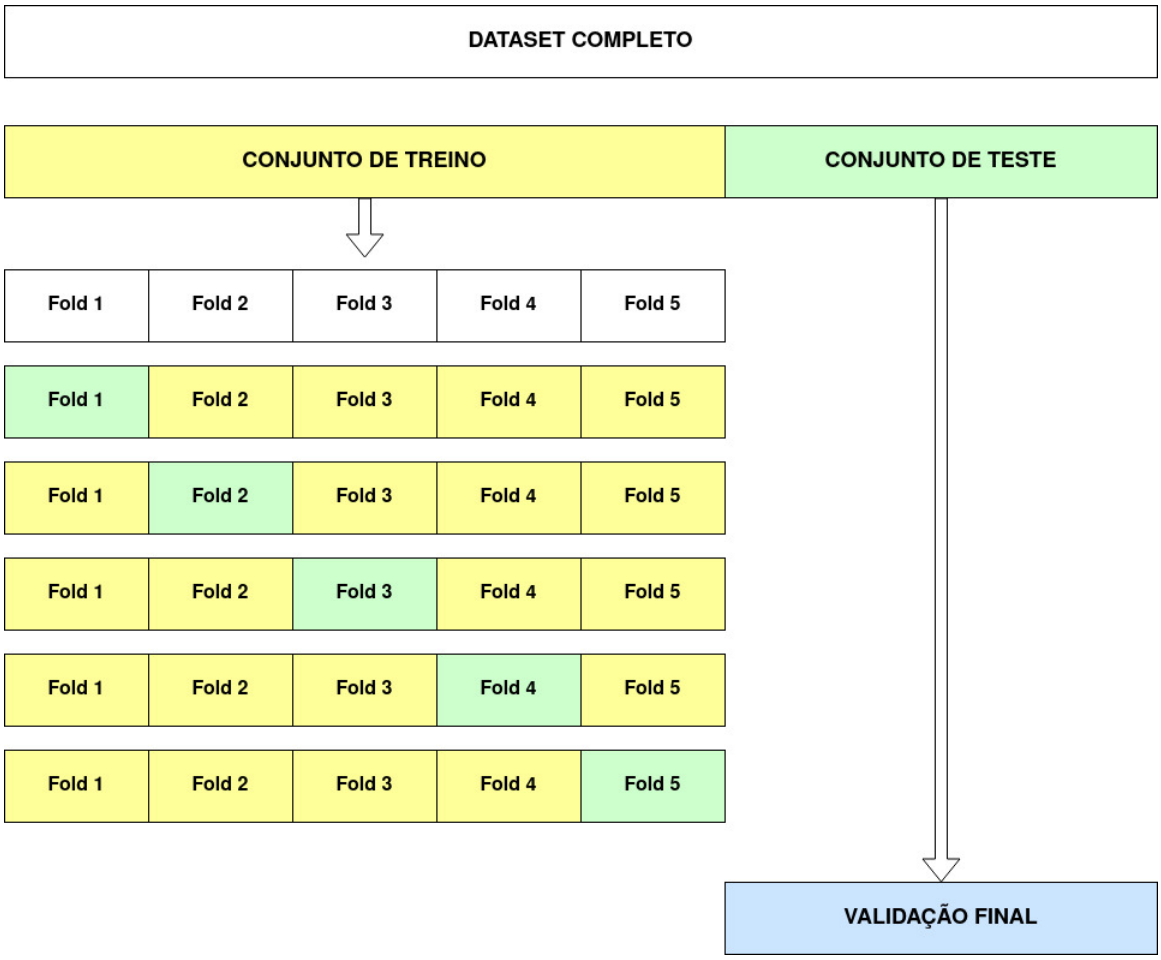
2.3.2 Escolha dos modelos de aprendizagem de máquina

Em relação aos modelos disponíveis, existem cinco modelos à disposição para o usuário:

- *K-Neighbors Regressor* (KNN);
- *XGBoost*;
- Perceptron Multicamadas (*Multilayer Perceptron* - MLP);
- *Lasso Regression*;
- *Random Forest Regressor*.

Tais modelos foram escolhidos com o objetivo de fornecer diferentes estratégias para solução dos problemas de regressão. Como o desempenho dos modelos dependem muito de como estão estruturados os dados de entrada, o teste de modelos que adotam diferentes abordagens pode fazer com que resultados melhores sejam obtidos. A seguir é possível conferir uma descrição breve do funcionamento de cada algoritmo e o motivo da escolha para compor o sistema.

Figura 1 – Processo de *K-Fold Cross-Validation* para $k = 5$



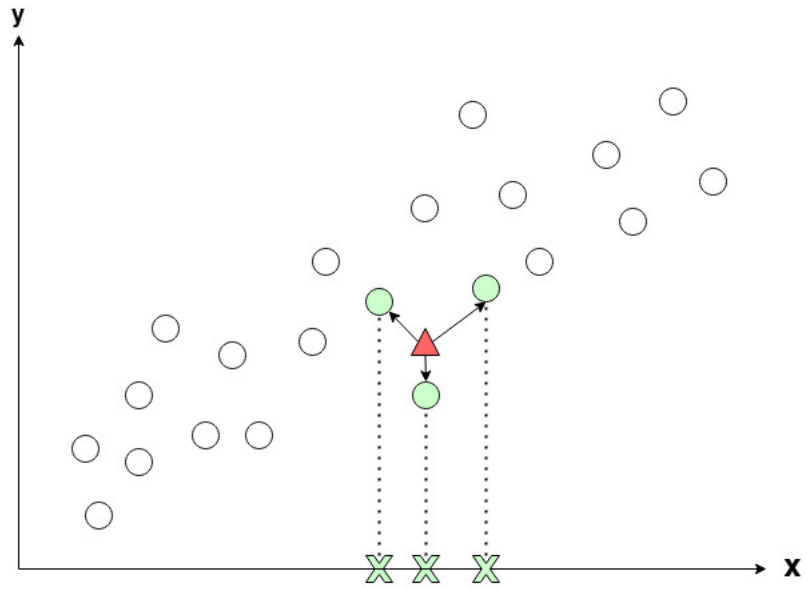
Fonte: Elaborada pelo autor.

O algoritmo *KNN*, apesar de simples, pode apresentar resultados interessantes dependendo da complexidade do problema. O algoritmo consiste em calcular uma métrica de distância entre os dados, utilizando-a para encontrar os K vizinhos mais próximos. Neste projeto, optou-se pela utilização da distância Euclidiana e um conjunto de valores $k = \{3,5,7\}$. Para os problemas de regressão, que possuem o objetivo de predizer um valor real, utiliza-se a média das instâncias dos K vizinhos mais próximos como valor predito.

Na Figura 2 é possível conferir como o algoritmo KNN funciona, no caso de $k = 3$. O triângulo vermelho representa a instância que se deseja realizar a predição, os círculos verdes representam os três vizinhos mais próximos e os "X" em verde representam os valores de cada um dos vizinhos. Neste caso o valor predito pelo algoritmo será a média dos três valores "X".

O *XGBoost* é um algoritmo de *gradient boosting* que implementa várias melhorias em relação aos algoritmos de *gradient boosting* clássicos, como regularização, criação de uma função de custo personalizada e paralelização do processamento. A ideia do *XGBoost* é utilizar uma série de árvores de decisão como preditores fracos, onde cada árvore aprende com os erros da árvore anterior, gerando assim um preditor forte no fim da execução do algoritmo. Para realizar

Figura 2 – Exemplo de funcionamento do algoritmo *KNN* para $k = 3$

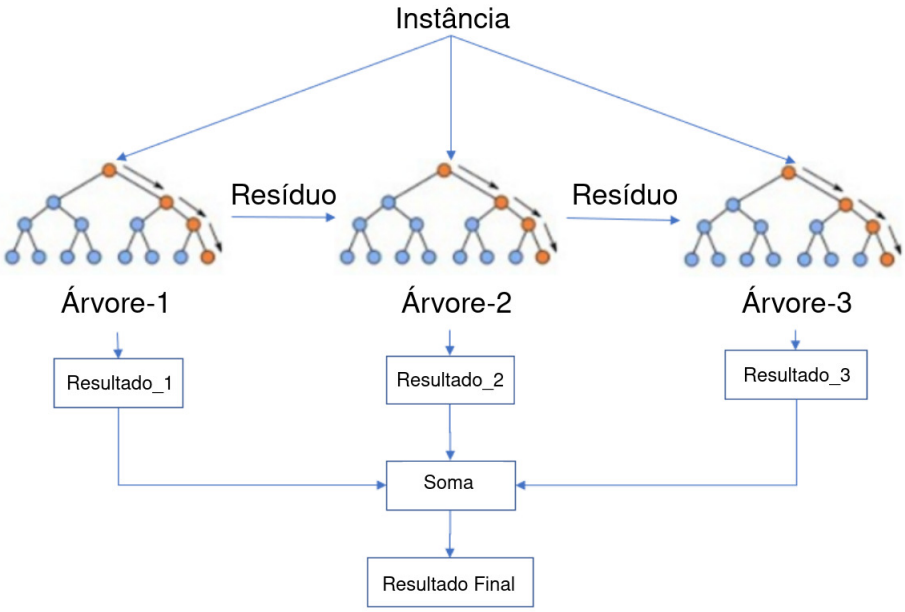


Fonte: Elaborada pelo autor.

o processo de melhoria do modelo, utiliza-se a técnica do gradiente descendente, objetivando assim reduzir o valor da função de custo. A escolha do *XGBoost* para compor um dos modelos disponíveis se deve ao alto desempenho proporcionado pelo algoritmo, sendo um dos modelos mais utilizados por cientistas de dados para predizer dados tabulares em problemas de regressão e classificação. Na Figura 3, é possível conferir o funcionamento do algoritmo *XGBoost*. Destaca-se a passagem dos resíduos de uma árvore para outra, uma das etapas principais dos algoritmos de *boosting*.

As redes neurais são utilizadas em uma variedade enorme de aplicações e têm promovido resultados excelentes em diversas áreas como visão computacional, reconhecimento de fala e processamento de linguagem natural, por exemplo. O MLP nada mais é que um modelo de rede neural que apresenta mais de uma camada de neurônios, sendo basicamente composto por uma camada de entrada, uma camada de saída e uma ou mais camadas intermediárias, chamadas de camadas escondidas (*hidden layers*). Cada camada é composta por vários nós (neurônios), sendo que cada um desses nós possui um peso associado. A função de ativação (que neste projeto é a ReLU (*Rectified Linear Unit)*), é responsável pela ativação ou não desses pesos de acordo com a variável de entrada, passando para os nós seguinte uma saída. Na última camada é então calculada a função de perda (*loss*), que compara a saída da rede com os dados reais do conjunto de treino, calculando o erro entre eles. Caso o erro seja alto, o processo de *backpropagation* entra em ação, atuando com o algoritmo de gradiente descendente de modo a atualizar os pesos das funções de todos os nós, otimizando assim o resultado da rede. Devido à ampla utilização das redes neurais nos processos de modelagem de dados, decidiu-se por utilizá-las também neste projeto. Na Figura 4 é possível conferir uma estrutura de rede neural contendo três camadas: 1

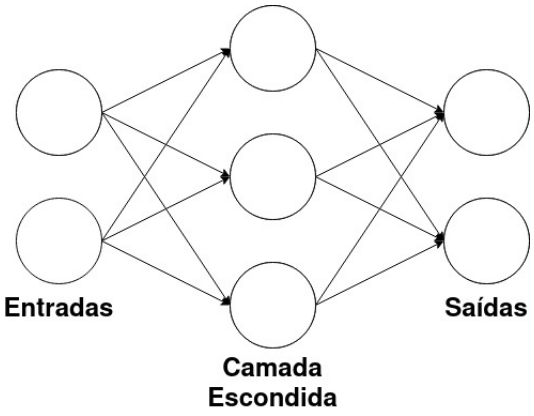
Figura 3 – Exemplo de funcionamento do algoritmo *XGBoost* com três árvores de decisão



Fonte: Adaptada de Wang, Chakraborty e Chakraborty (2021, 5).

camada de entrada, 1 camada escondida e 1 camada de saída.

Figura 4 – Exemplo de estrutura de uma rede neural com três camadas



Fonte: Elaborada pelo autor.

O modelo de regressão por *Lasso* (*Least Absolute Shrinkage and Selection Operator* nada mais é que um modelo de regressão linear tradicional, adicionado de um fator de regularização L1 (ou *Lasso*). Um modelo de regressão linear múltipla, tenta ajustar uma função linear aos dados com a forma:

$$\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}$$

Onde $\hat{y}_i$ é a variável dependente, x_i são as variáveis independentes, β_0 é o coeficiente de interceptação e β_p são os pesos atribuídos a cada variável independente.

O procedimento de ajuste ocorre através da escolha dos coeficientes β, de forma a minimizar a função custo, dada por:

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - \sum_{j=0}^{p}\beta_j x_{ij})^2$$

Onde n é o número de instâncias do dado e p é o número de *features*.

No entanto, existe uma possibilidade do modelo se ajustar demais aos dados e aos possíveis ruídos, se tornando enviesado no conjunto de treinamento, o que causa um aumento na variância quando se realiza o teste. O fator de regularização *Lasso* objetiva diminuir o viés, piorando propositalmente a performance do modelo no conjunto de treinamento, porém resultando em uma melhor generalização no conjunto de teste. O fator de regularização atua na função custo, deixando-a da seguinte forma:

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - \sum_{j=0}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=0}^{p}|\beta_j|$$

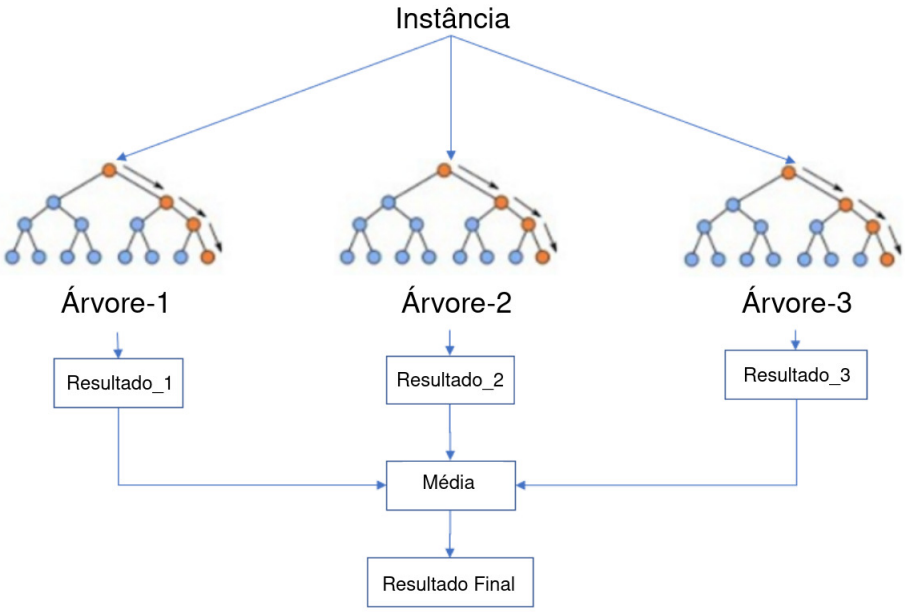
Onde λ denota o fator de encolhimento de cada peso. Um fator $\lambda = 0$ implica em um modelo de regressão linear múltipla tradicional, já um fator $\lambda = \infty$ implica que o valor de todas as variáveis são desconsideradas. Ou seja, quanto maior o valor de λ, maior o viés do modelo e quanto menor o valor de λ, maior a variância do modelo. A escolha deste modelo para compor a lista de modelos disponíveis no sistema se dá pela sua simplicidade e por ser um dos modelos base para os problemas de regressão.

Por fim, o último modelo que compõe o sistema é o *Random Forest Regressor*. Diferente do algoritmo *XGBoost*, que utiliza uma técnica de *Boosting* (onde cada árvore de decisão aprendia com os erros da árvore anterior), o modelo *Random Forest* utiliza uma técnica chamada *Bagging*. Essa técnica consiste em criar diversas amostras, escolhidas aleatoriamente com substituição, com os dados de treinamento. Cada subconjunto é usado para o treinamento de uma árvore de decisão diferente. Teremos então um conjunto de diferentes modelos, que podem ser utilizados para criar um estimador poderoso ao calcular-se a média de todos os estimadores individuais. O resultado torna-se robusto a eventuais erros, já que a média de diversas árvores tende a diminuir os erros causados por estimadores com baixa performance. Na Figura 5 é possível conferir o funcionamento de uma *Random Forest*. Nota-se que diferente do algoritmo *XGBoost*, as árvores de decisão não trocam informações entre si e o resultado final é dado pela média das diferentes árvores.

2.3.3 Escolha das métricas para cálculo do desempenho

Depois que os modelos desejados são escolhidos pelo usuário, o sistema executa o processo de modelagem, gerando métricas para cada modelo. As métricas escolhidas para

Figura 5 – Exemplo de funcionamento do algoritmo *RandomForest* com três árvores de decisão



Fonte: Adaptada de Wang, Chakraborty e Chakraborty (2021, 5).

representar o desempenho dos modelos foram: Erro Absoluto Médio (*Mean Absolute Error - MAE*), Erro Quadrático Médio (*Mean Squared Error - MSE*) e Coeficiente de Determinação (R^2).

O Erro Absoluto Médio é calculado da seguinte forma:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Já o Erro Quadrático Médio é calculado por:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Em ambos os casos, N é o número de instâncias dos dados, y_i é o valor real da instância e $\hat{y}_i$ é o valor predito pelo modelo.

O *MAE* é uma boa métrica para mostrar por quanto o modelo está errando, pois através do cálculo do valor absoluto desconsideram-se os sinais do erro, que poderiam causar uma falsa impressão de boa performance, já que erros positivos poderiam estar anulando erros negativos. O *MSE* por outro lado, faz com que erros grandes nas predições sejam evidenciados, ao elevar ao quadrado os erros de predição.

O Coeficiente de Determinação (R^2) é utilizado para mostrar por quanto o modelo treinado explica os dados melhor que a média. O valor do coeficiente na maioria dos casos está no intervalo $[0, 1]$, porém algumas vezes pode apresentar resultado negativo, neste caso indicando

que o modelo apresenta resultado pior do que se estivéssemos utilizando apenas a média dos dados como valor predito. A expressão do R^2 é dada por:

$$R^2(y,\hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

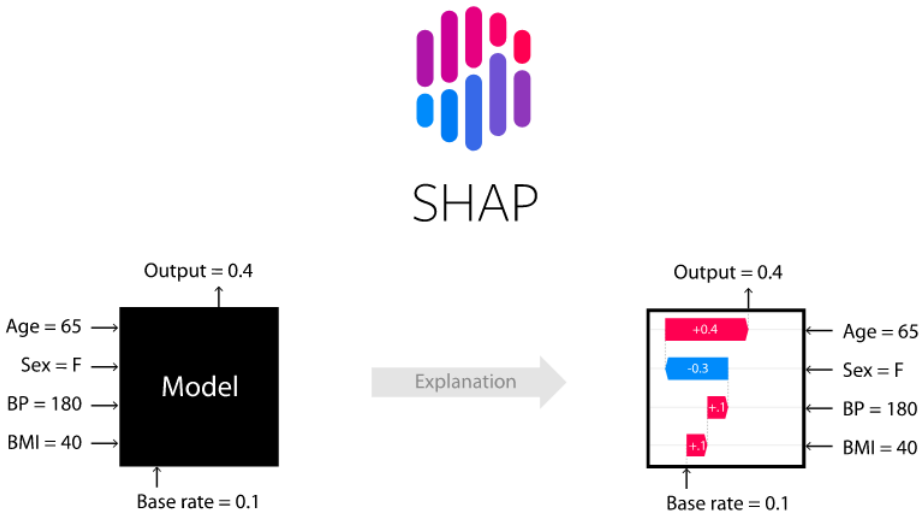
Onde $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$.

Um valor de R^2 igual a 0 mostra que o modelo não conseguiu capturar nada além de um modelo que simplesmente pega a média dos dados. Já um valor igual a 0.95, por exemplo, mostra que o modelo conseguiu explicar 95% da variância dos dados, apresentando uma excelente performance.

2.3.4 Criação dos gráficos de importância de variáveis

Para criar os gráficos de importância de variáveis, utilizou-se a técnica *SHAP* (*SHapley Additive exPlanations*), que possui o objetivo de explicar as saídas dos modelos de *Machine Learning*, que são muitas vezes utilizados como caixas-pretas. A Figura 6 mostra a ideia do funcionamento da técnica *SHAP*.

Figura 6 – Funcionamento da técnica SHAP



Fonte: Lundberg e Lee (2017a).

O *SHAP* mede o impacto das variáveis, levando em consideração a interação com outras variáveis. Os valores de *Shapley* calculam a importância de uma variável comparando o que um modelo prevê com e sem a variável. As vantagens de se utilizar o *SHAP* para explicar o impacto das variáveis são:

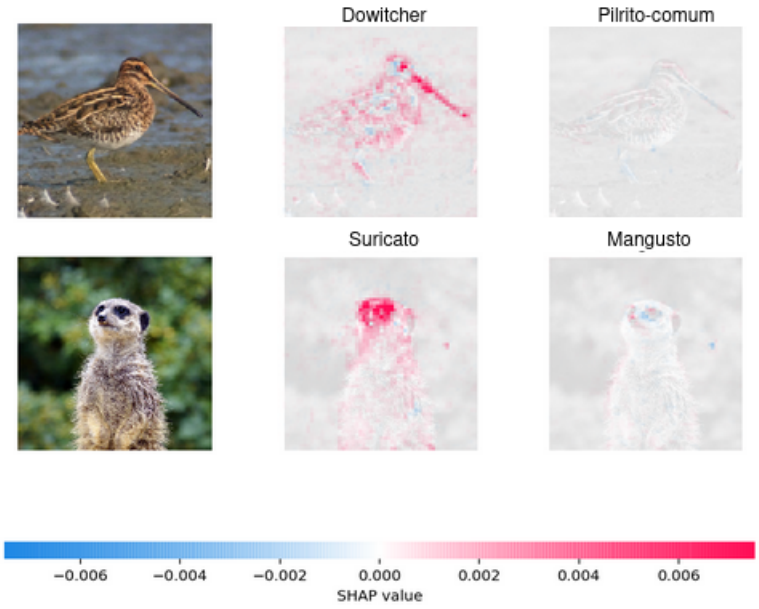
- Interpretabilidade Global: Os valores *SHAP* podem mostrar o quanto cada variável contribui, positiva ou negativamente, para a variável alvo. Funciona como um gráfico de

importância de variáveis, porém mostrando além disso se as relações com a variável alvo são positivas ou negativas;

- Interpretabilidade Local: É possível calcular os *SHAP Values* de cada observação de modo separado, aumentando assim a transparência do modelo.

Na Figura 7 é possível conferir como o *SHAP* funciona em um problema de classificação de imagens. Neste caso, pixeis vermelhos representam valores *SHAP* positivos, que aumentam a probabilidade da imagem ser daquela classe, e pixeis azuis representam valores *SHAP* negativos, que diminuem a probabilidade da imagem ser daquela classe.

Figura 7 – Exemplo de funcionamento do *SHAP Values* para classificação de imagens



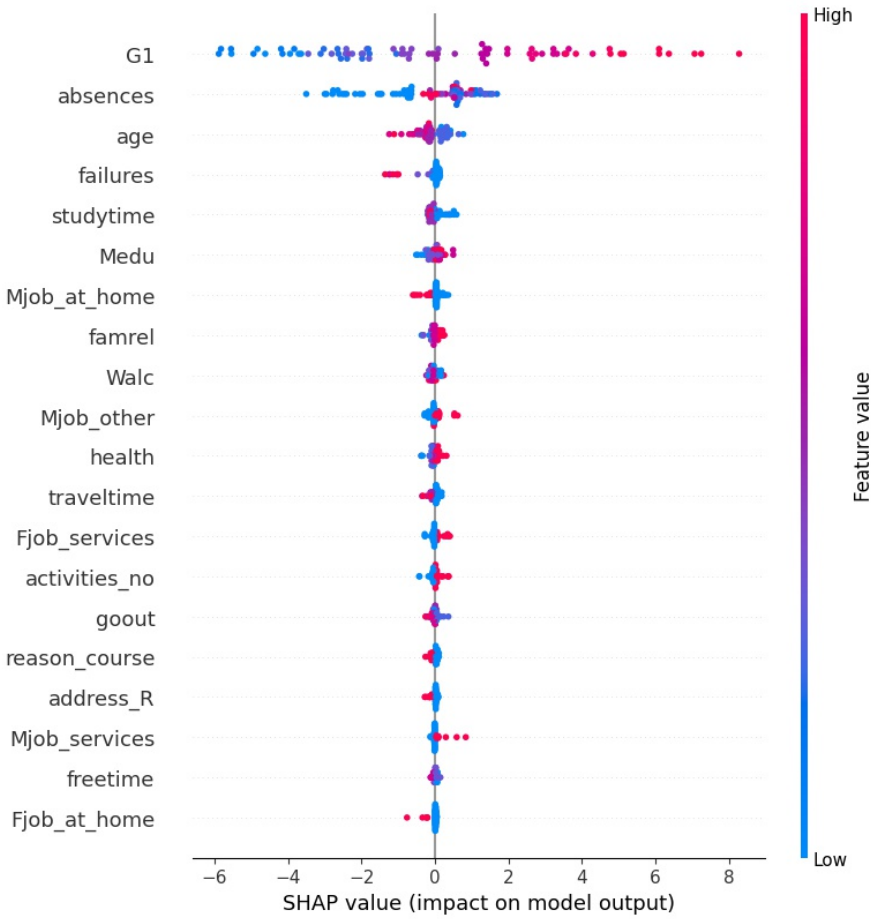
Fonte: Adaptada de Lundberg e Lee (2017a).

Para problemas de regressão, os valores *SHAP* funcionam de forma semelhante. Neste projeto, optou-se pela utilização de duas visualizações diferentes: gráfico enxame e gráfico em barras.

O gráfico enxame apresenta o valor *SHAP* para todas as amostras e variáveis, mostrando o impacto de cada amostra no valor final da predição. No exemplo da Figura 8, pode-se perceber que a variável *G1* é a variável mais importante e que valores altos dessa variável indicam um valor alto da variável alvo, como mostra a magnitude dos valores *SHAP* e a cor dos pontos vermelhos e azuis.

Já o gráfico em barras apresenta a média do valor absoluto dos valores *SHAP*, apresentando nesse caso uma medida mais generalizada da importância das variáveis. No exemplo da Figura 9, pode-se conferir que a variável *G1* é muito importante para a predição da variável alvo.

Figura 8 – Exemplo de gráfico de enxame



Fonte: Elaborada pelo autor.

2.3.5 Registro dos resultados no MLFlow

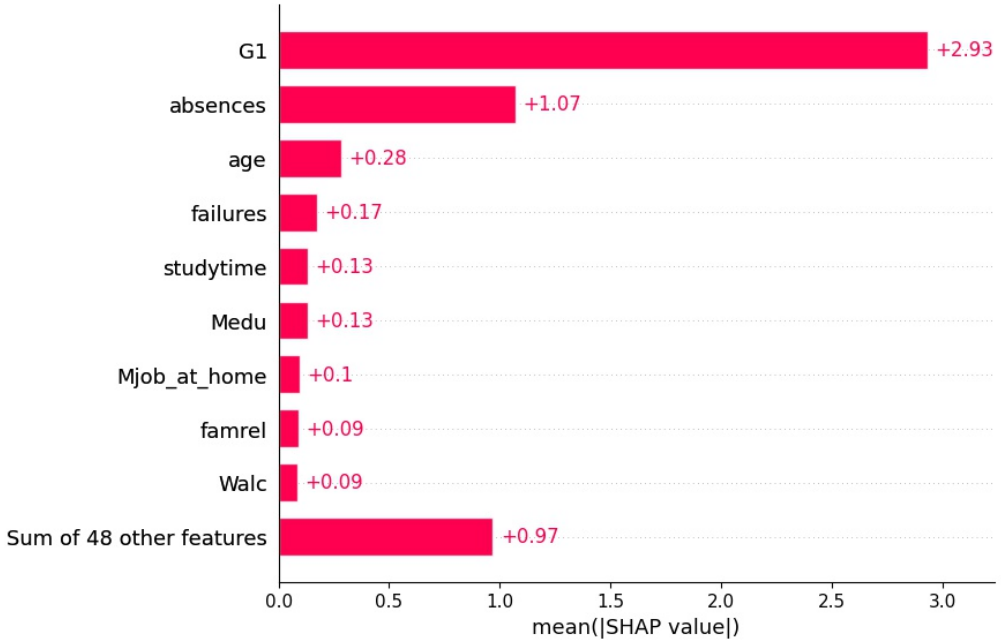
Por fim, os resultados são registrados no *MLFlow*. Na Figura 10 possível conferir um exemplo da interface do *MLFlow*.

2.4 Dados utilizados para teste do sistema

Para realizar o teste do sistema e garantir que todas as funcionalidades estão funcionando corretamente, escolheu-se três conjuntos de dados. Os conjuntos de dados foram os seguintes:

- *The Boston Housing Dataset* (HARRISON; RUBINFELD, 1978)
- *Student Performance Dataset* (CORTEZ; SILVA, 2008)
- *Wine Quality Dataset* (CORTEZ *et al.*, 2009)

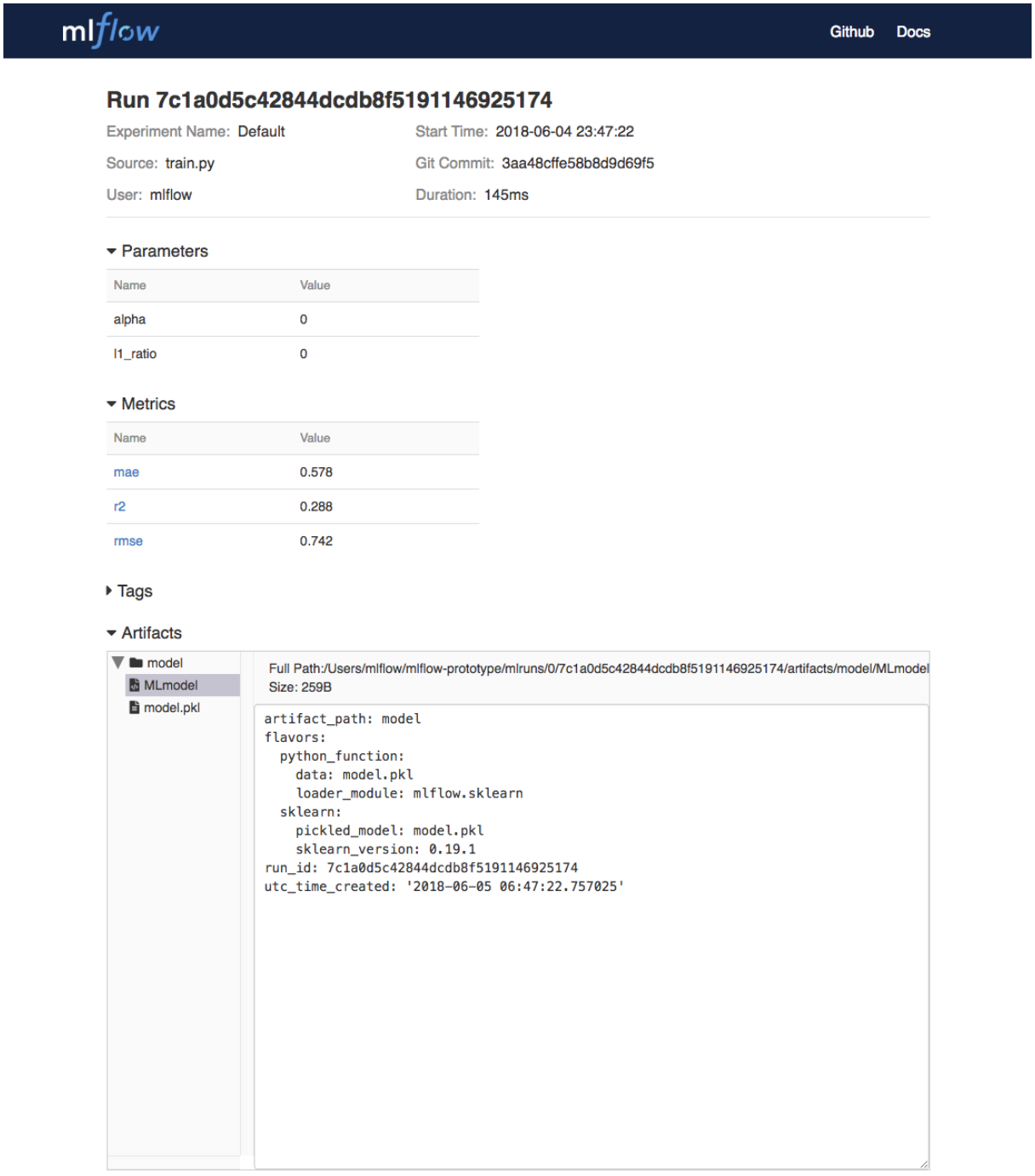
Figura 9 – Exemplo de gráfico de barras



Fonte: Elaborada pelo autor.

A escolha desses três *datasets* ocorreu pela possibilidade de se testar diferentes funcionalidades do sistema, dado a diversidade proporcionada por cada conjunto de dados e por se tratarem de *datasets* bastante utilizados em *benchmarks* de algoritmos e modelos.

Figura 10 – Exemplo de interface do *MLFlow*



Fonte: MLflow (2018).

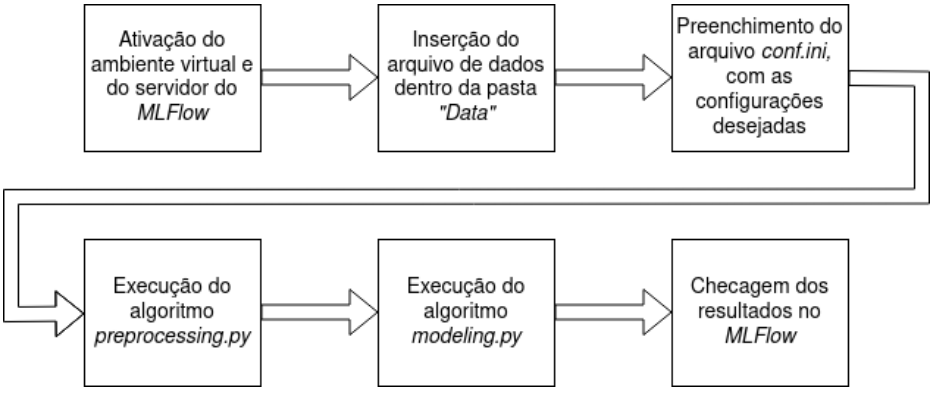
Capítulo 3

DESENVOLVIMENTO

3.1 O Projeto

Como já descrito nas seções anteriores, o objetivo deste projeto é criar um sistema que realize análises de regressão, sem a necessidade do cientista de dados trabalhar diretamente com o código. As etapas que o usuário deverá executar para realizar a análise podem ser conferidas no diagrama da Figura 11:

Figura 11 – Diagrama de execução das etapas



Fonte: Elaborada pelo autor.

No Apêndice A, é possível obter mais detalhes sobre o processo de instalação do sistema, além da estrutura de diretórios e arquivos do projeto. Neste apêndice, também é possível conferir o processo de execução das três primeiras etapas do sistema, que consiste na ativação do ambiente virtual, do servidor *MLFlow*, inserção do arquivo com os dados na pasta correta e preenchimento do arquivo *conf.ini*.

Depois de ter realizado as três primeiras etapas, basta o usuário executar os arquivos *preprocessing.py* e *modeling.py*, que estão dentro da pasta *code*. Para executar, basta digitar o seguinte comando:

```
$ python preprocessing.py && python modeling.py
```

Os resultados estarão então disponíveis no endereço configurado pelo usuário (por padrão: 0.0.0.0:5000), bastando digitar o endereço no navegador e utilizar a interface do *MLFlow*.

3.2 Atividades Realizadas

Primeiramente, traçou-se um plano sobre a estrutura geral do projeto. Neste plano, definiu-se quais seriam os módulos principais que o sistema teria, além de decidir pela utilização do *MLFlow* para registro de resultados. Com a estrutura principal do sistema definida, iniciou-se a pesquisa bibliográfica.

A pesquisa bibliográfica consistiu em pesquisar artigos, livros e revistas, procurando definir quais seriam as melhores funcionalidades para o sistema. Utilizou-se também conhecimentos obtidos no estágio obrigatório e em disciplinas da graduação, como Introdução à Ciência de Dados, Redes Neurais e Aprendizado Profundo, Estatística e Redes Complexas para auxiliar na tomada de decisões sobre as funcionalidades. Com o escopo principal e as funcionalidades definidas, iniciou-se a implementação do sistema.

Começou-se a implementação através da configuração do banco de dados e do ambiente virtual *Python*, já que eles compõem a base do sistema e todo o restante da implementação depende que eles estejam funcionando corretamente. Como comentado na Seção 3.1, o Apêndice A apresenta os detalhes de instalação e configuração. Depois de realizada a configuração, montou-se a estrutura das duas classes, definindo os métodos das classes de pré-processamento e modelagem. Os métodos implementados podem ser conferidos nas Figuras 12 e 13.

Os detalhes sobre as técnicas e tecnologias utilizadas já foram detalhados na Seção 2. Nas Subseções 3.2.1 e 3.2.2 é possível conferir uma descrição resumida de como tais técnicas foram implementadas nos métodos das classes. Já na Subseção 3.2.3, pode ser conferido o processo de testes e validações realizados na construção do regressor.

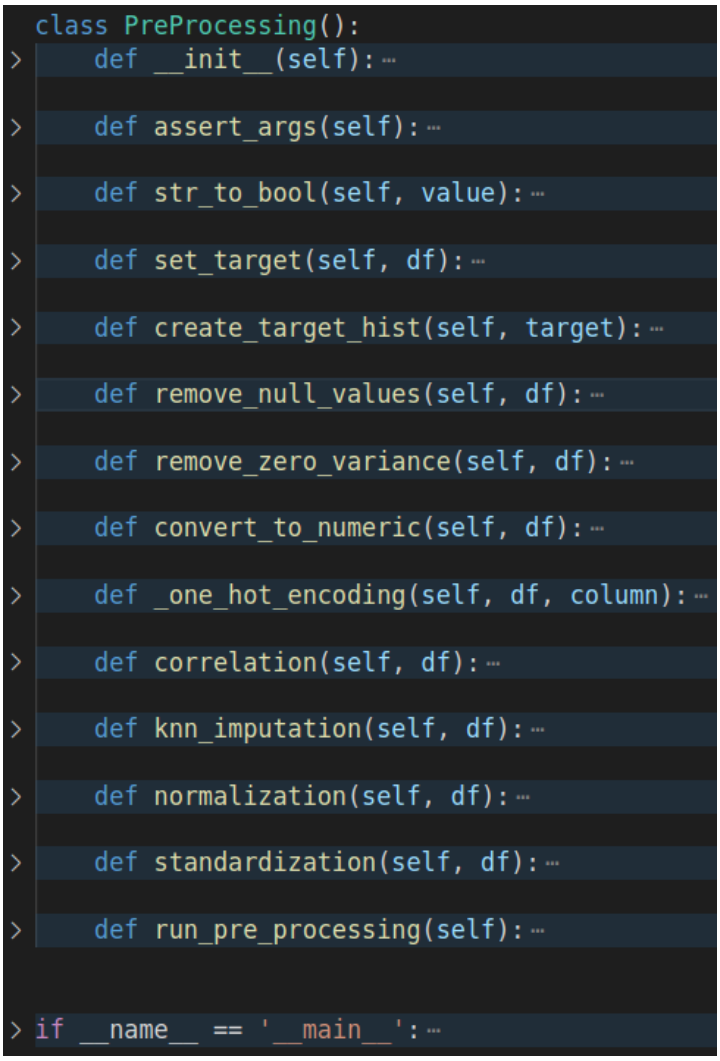
3.2.1 Implementação do módulo de Pré-Processamento

Ambos os módulos apresentam um *runner*, que nada mais é que um método responsável por coordenar a execução dos outros métodos. O *runner* do módulo de pré-processamento pode ser conferido no código-fonte 2.

Código-fonte 2: Método *runner* do módulo de pré-processamento

```
1  def run_pre_processing(self):
2      df = self.assert_args()
3      df = self.set_target(df)
4      df = self.convert_to_numeric(df)
5      df = self.remove_zero_variance(df)
6      df = self.remove_null_values(df)
7      df = self.correlation(df)
8      df = self.knn_imputation(df)
9      df = self.normalization(df)
10     df = self.standardization(df)
11     df.to_pickle(f'../data/{self.run_id}/data.pkl')
```

Figura 12 – Métodos criados na classe responsável pelo pré-processamento



Fonte: Elaborada pelo autor.

O método construtor *__init__* e o método *assert_args* são responsáveis por coletar as informações do arquivo de configuração, validando-as. Esses métodos verificam por exemplo, se o arquivo com os dados realmente existe e se os limiares definidos estão com valores válidos. Um trecho exemplo do que foi implementado pode ser conferido no Código-fonte 3:

Código-fonte 3: Exemplos de *assert* utilizados na implementação

```
1 assert file_exists is True, (
2     'File_name file does not exist. Check the conf.ini file'
3 )
4 assert self.null_threshold >= 0 and self.null_threshold <= 1, (
5     'Null_threshold value is not valid, use a value between 0 and 1'
6     ' in the conf.ini file'
7 )
```

```
 8 assert self.target_name in df.columns, 'Target_name does not exist'
 9 assert isinstance(self.imputation, bool), (
10     'Imputation variable is not valid, use True or False'
11 )
```

O método *str_to_bool* é um método auxiliar, que ajuda na validação de variáveis booleanas escritas fora do padrão pelo usuário no arquivo de configuração. Ao invés de aceitar apenas *False* por exemplo, são aceitas as seguintes *strings*: '*false*', 'falso', 'f', '0', 'no', 'n'.

Os métodos *set_target* e *create_target_hist* são responsáveis por manipular a variável alvo do sistema que será predita no módulo de modelagem. A variável alvo é separada das outras variáveis e um histograma é criado com os seus valores. A biblioteca *pickle* é utilizada para salvar a variável alvo em um arquivo e a biblioteca *pandas* é utilizada para realizar a traçagem do histograma (juntamente com o auxílio da biblioteca *scipy*, utilizada para definir a largura ideal das barras, e da biblioteca *matplotlib* para salvar o histograma em um arquivo).

Os métodos *convert_to_numeric* e *_one_hot_encoding* são responsáveis por converter variáveis não numéricas para numéricas, aplicando quando necessário a técnica de *One-Hot Encoding*. Já o método *knn_imputation* é responsável por realizar a imputação de valores faltantes nas variáveis. Para realizar a técnica *One-Hot Encoding*, utilizou-se a ferramenta *get_dummies* da biblioteca *pandas*. Para realizar a imputação de valores faltantes, utilizou-se a ferramenta *KNNImputer* da biblioteca *sklearn*.

Os métodos *remove_zero_variance*, *remove_null_values* e *correlation* são responsáveis por remover as variáveis com variância nula, variáveis contendo mais valores nulos do que o limiar definido pelo usuário e variáveis altamente correlacionadas entre si. Para implementar esses métodos, utilizou-se as ferramentas *var()*, *dropna()* e *corr()* da biblioteca *pandas*. Com o auxílio da ferramenta *corr()* também salvou-se um arquivo contendo a correlação entre todas as variáveis.

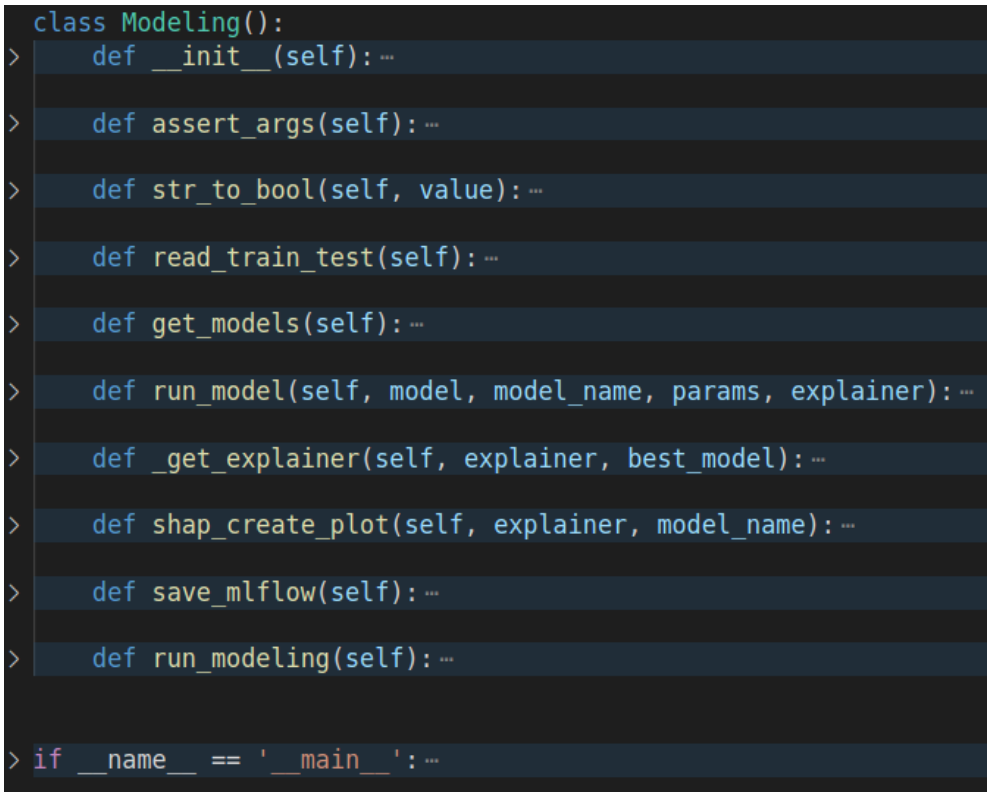
Por fim, os métodos *normalization* e *standardization* são responsáveis por aplicar a normalização ou a estandardização (utilizando a ferramenta *StandardScaler()* da biblioteca *sklearn*) dos dados. O conjunto de dados manipulado e pré-processado é então salvo em um arquivo *.pkl*, através da biblioteca *pickle*.

3.2.2 Implementação do módulo de modelagem

O método construtor *__init__* e os métodos *assert_args*, *str_to_bool* e o *run_modeling* apresentam funcionamento análogo ao módulo de pré-processamento, sendo métodos que coletam informações do arquivo de configuração, validam tais informações e controlam a execução dos outros métodos. O *runner* deste módulo pode ser conferido no Código-fonte 4.

Código-fonte 4: Método *runner* do módulo de modelagem

Figura 13 – Métodos criados na classe responsável pela modelagem



Fonte: Elaborada pelo autor.

```
1 def run_modeling(self):
2     self.assert_args()
3     self.read_train_test()
4     self.get_models()
5     self.save_mlflow()
```

O método *read_train_test* é responsável por ler os arquivos com os dados processados pelo módulo anterior, dividindo-os em conjuntos de treino e teste através da ferramenta *train_test_split* da biblioteca *sklearn*.

O método *get_models* é responsável por selecionar os modelos definidos pelo usuário. Um trecho desse método pode ser conferido no Código-fonte 5. Como pode ser conferido, esse método possui diversos hiperparâmetros para cada modelo que serão utilizados no método *run_model* para realizar o *Grid-Search*, definindo o melhor modelo para o problema.

Código-fonte 5: Método *get_models* da classe de modelagem

```
1 def get_models(self):
2     params = {
3         'XGB': {
4             'learning_rate': [0.01, 0.05],
```

```
            'n_estimators': [100, 250, 500],
            'max_depth': [5, 7, 9],
            'random_state': [42]
        },
        'MLP': {
            'hidden_layer_sizes': [
                (3,), (5,), (7,),
                (3, 7, 3),
                (25, 50, 25,),
                (10, 25, 10),
                (5, 10, 15, 10, 5),
                (100,)
            ],
            'activation': ['relu'],
            'solver': ['adam'],
            'alpha': [0.00001, 0.0001, 0.001],
            'random_state': [42]

        }
    }

    if 'XGB' in self.models_list:
        self.run_model(
            XGBRegressor(), 'XGBoost', params['XGB'], 'tree'
        )
    if 'MLP' in self.models_list:
        self.run_model(
            MLPRegressor(), 'MLP', params['MLP'], 'explainer'
        )
```

O método *run_model* receberá os parâmetros definidos passados pelo método *get_models*, realizando o treinamento e avaliação dos modelos. Todo o processo de definição dos modelos, treinamento e avaliação são feitos utilizando ferramentas da biblioteca *sklearn*. Além disso, dois métodos auxiliares são utilizados: *get_explainer* e *shap_create_plot*. Esses métodos auxiliares serão responsáveis por criar os gráficos de importância de variáveis utilizando a técnica *SHAP Values*.

Por fim, o método *save_mlflow* é responsável por salvar os resultados do processo de modelagem no servidor *MLFlow*, estando então disponíveis para consulta.

3.2.3 Teste e validação do sistema

Para desenvolver e ir aperfeiçoando o sistema, utilizou-se o conjunto de dados *Student Performance*. Assim que cada funcionalidade era implementada, o sistema era executado para garantir que tudo estivesse funcionando corretamente. Ao fim da implementação do sistema,

realizou-se um último teste para verificar a execução das funcionalidades e iniciou-se o processo de validação.

O processo de validação ocorreu através da execução do sistema utilizando como entrada os outros dois conjuntos de dados existentes: *The Boston Housing* e *Wine Quality*. O conjunto de dados *Student Performance* apresenta uma série de variáveis que descrevem estudantes de uma escola e o objetivo do problema de regressão é prever qual será a nota da terceira avaliação. O conjunto de dados *The Boston Housing* contém inúmeras variáveis quantitativas e qualitativas que descrevem casas em Boston e o objetivo do problema de regressão é prever qual o valor estimado da casa. Por fim, o conjunto de dados *Wine Quality* possui uma série de variáveis sobre vinhos, tendo como objetivo a atribuição de uma nota que defina a qualidade do vinho de acordo com as características presentes nas variáveis.

Para garantir que todas as funcionalidades fossem validadas, criou-se novas variáveis artificiais nos conjuntos de dados. Em seguida, executou-se o sistema para os dois *datasets*, validando assim o sistema.

3.3 Resultados

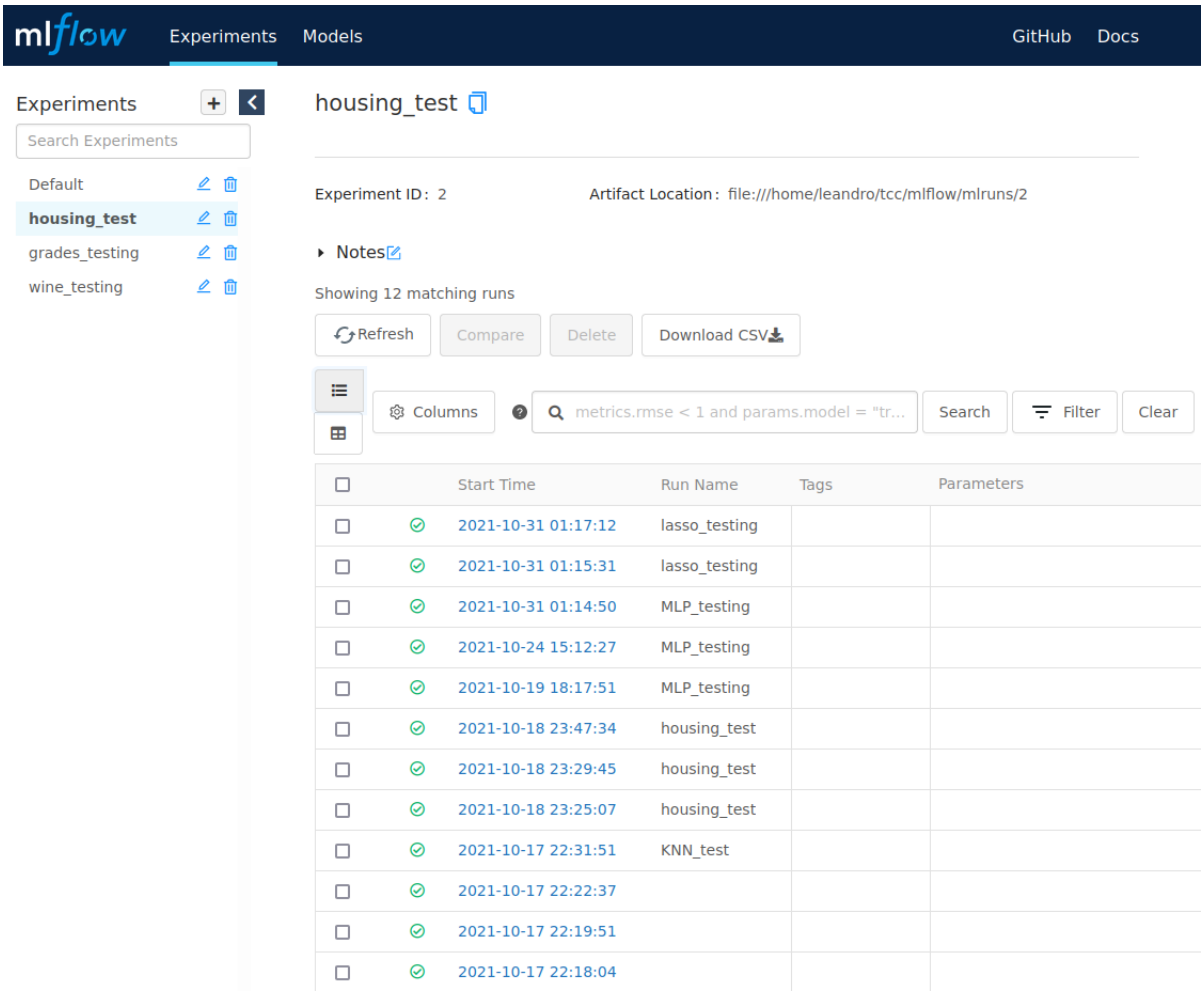
Testou-se o sistema nos três conjuntos de dados propostos e o sistema funcionou como esperado para todas as entradas. Desta forma, espera-se ter atingido um nível de generalização e tolerância à falhas bons o suficiente para suportar uma série de problemas de regressão diferentes. Na Figura 14 é possível conferir a interface do *MLFLow*, listando todos os experimentos. Na Figura 15 é possível conferir o histograma com a variável alvo. Nas Figuras 16 e 17 é possível conferir as saídas das etapas de pré-processamento e modelagem, respectivamente. Por fim, nas Figuras 18 e 19 é possível conferir os gráficos sumário e enxame de importância de variáveis. As saídas foram geradas para um teste utilizando o *dataset The Boston Housing Prices*.

3.4 Dificuldades e Limitações

As principais dificuldades enfrentadas no desenvolvimento do projeto estão relacionadas à escolha das técnicas e tecnologias utilizadas. Como os dados possuem variações gigantescas em diversos aspectos, como distribuição, dimensionalidade, quantidade e magnitude por exemplo, escolher técnicas que funcionassem em diferentes cenários e trouxessem resultados satisfatórios tornou-se de um grande desafio.

As limitações encontradas no sistema estão principalmente relacionadas ao alto consumo de poder computacional (processamento e memória). Para realizar a modelagem em um conjunto de dados com milhões de linhas ou até realizar manipulações utilizando a biblioteca *Pandas*, uma grande quantidade de memória é consumida. Logo, caso o sistema venha a ser utilizado em computadores pessoais, as limitações do sistema estarão ligadas diretamente ao poder

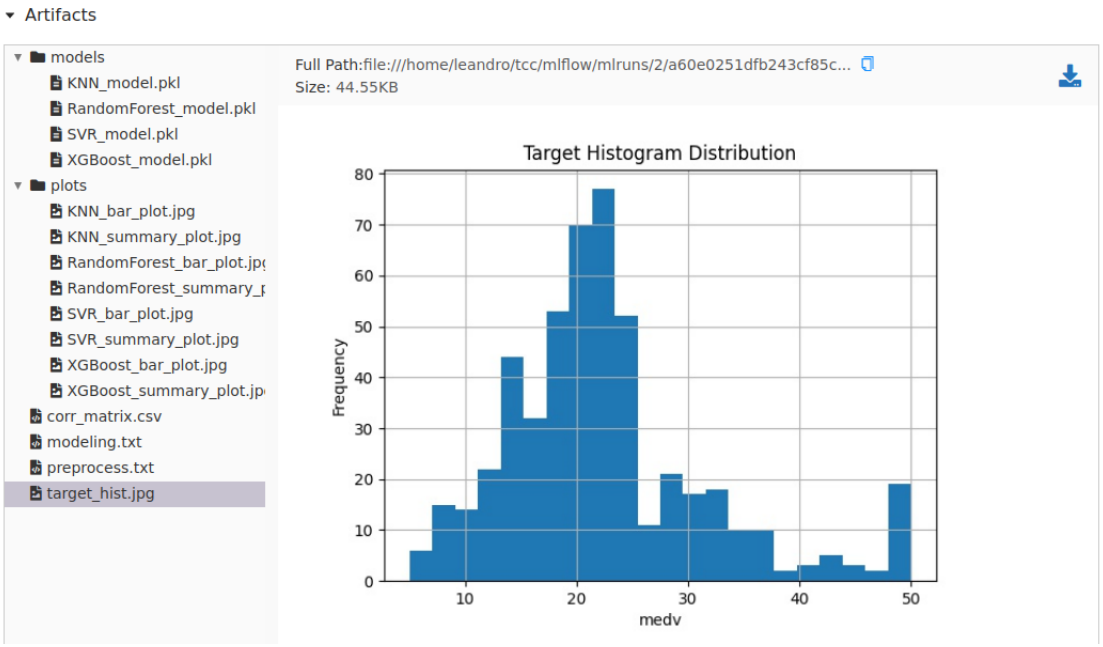
Figura 14 – Lista com os experimentos realizados com os *datasets*



Fonte: Elaborada pelo autor.

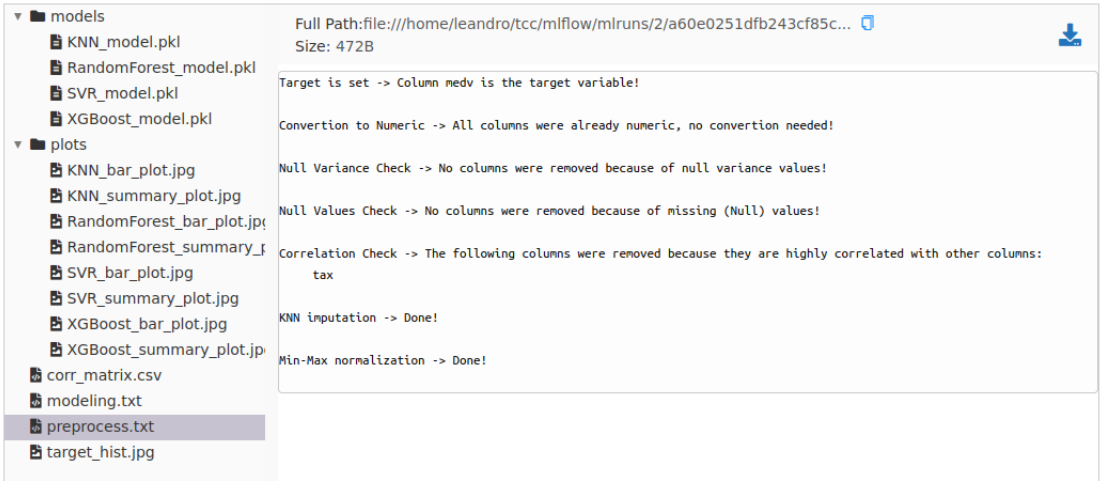
computacional da máquina utilizada. Nestes casos, recomenda-se a utilização de *clusters* ou soluções em nuvem fornecidas por empresas como a Amazon por exemplo, que oferece através do serviço Web AWS-EC2 (*Amazon Web Service - Elastic Compute Cloud*) instâncias com capacidade computacional redimensionável.

Figura 15 – Histograma com a variável alvo



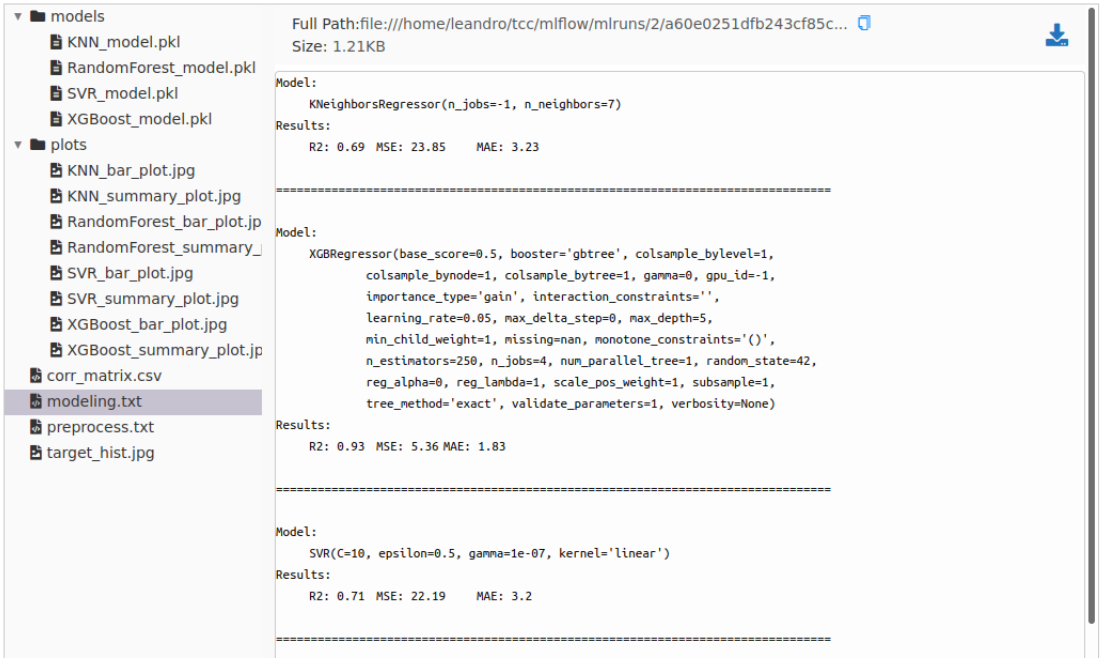
Fonte: Elaborada pelo autor.

Figura 16 – Resultados da etapa de pré-processamento



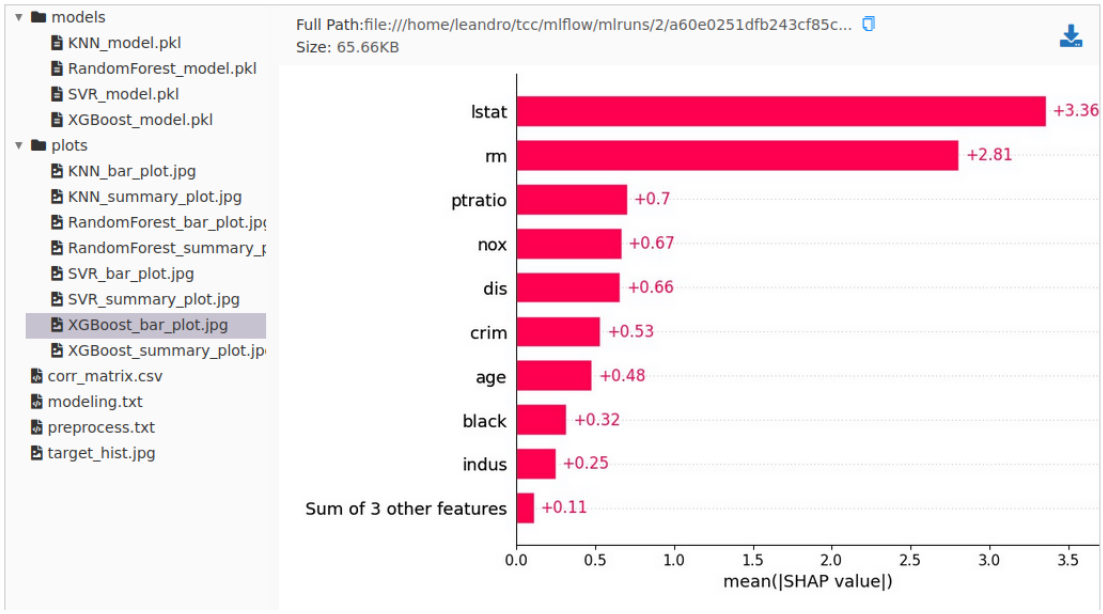
Fonte: Elaborada pelo autor.

Figura 17 – Resultados da etapa de modelagem, contendo os modelos e métricas de desempenho



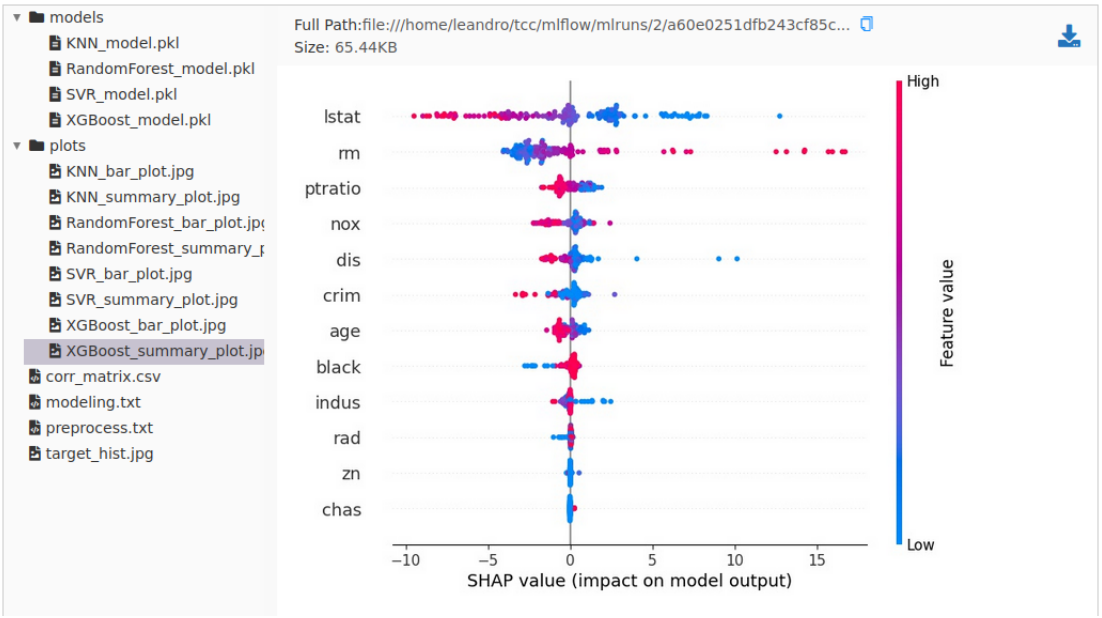
Fonte: Elaborada pelo autor.

Figura 18 – Gráfico *SHAP* com o sumário dos dados



Fonte: Elaborada pelo autor.

Figura 19 – Gráfico *SHAP* enxame com informações de importância de variáveis



Fonte: Elaborada pelo autor.

Capítulo 4

CONCLUSÃO

4.1 Contribuições

Realizar um trabalho que necessitou da integração de diferentes ferramentas, técnicas e tecnologias tratou-se de uma experiência valiosa e que com certeza proporcionou inúmeros ensinamentos.

No nível pessoal, o trabalho realizado trouxe a experiência da execução de um projeto de longo prazo, sendo necessário ter organização e disciplina para realizar um planejamento detalhado das atividades que seriam desenvolvidas, fazendo com que o projeto pudesse ser finalizado à tempo, levando em conta inclusive eventuais contratempos que poderiam ocorrer.

No nível profissional, obteve-se um conhecimento valioso em relação à pesquisas bibliográficas, que permitem que um profissional esteja sempre atualizado com o estado da arte de diferentes áreas do conhecimento. Além disso, os conceitos aprendidos certamente agregam aos conhecimentos já existentes sobre os temas estudados, permitindo a formação de um profissional mais completo e preparado para o mercado de trabalho.

REFERÊNCIAS

CHEN T.; GUESTRIN, C. XGBoost: A scalable tree boosting system. In: PROCEEDINGS OF THE 22ND ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING. São Francisco, Califórnia, EUA, 2016. Disponível em: <http://doi.acm.org/10.1145/2939672.2939785>. Citado na página 19.

CORTEZ, P.; CERDEIRA, A.; ALMEIDA, F.; MATOS, T.; REIS, J. Modeling wine preferences by data mining from physicochemical properties. **Decis. Support Syst.**, v. 47, n. 4, p. 547–553, 2009. Disponível em: <http://dblp.uni-trier.de/db/journals/dss/dss47.html#CortezCAMR09>. Citado na página 33.

CORTEZ, P.; SILVA, A. M. G. Using data mining to predict secondary school student performance. In: A. BRITO AND J. TEIXEIRA EDS., PROCEEDINGS OF 5TH FUTURE BUSINESS TECHNOLOGY CONFERENCE (FUBUTEC 2008). Porto, Porgugal, 2008. ISBN 978-9077381-39-7. Citado na página 33.

DASH, M.; LIU, H. Feature selection for classification. **Intelligent Data Analysis**, v. 1, p. 131–156, 1997. Citado na página 22.

FREEDMAN, D.; DIACONIS, P. **On the Histogram as a Density Estimator: L 2 Theory**. 1981. Citado na página 21.

HARRISON, D.; RUBINFELD, D. Hedonic housing prices and the demand for clean air. **Journal of Environmental Economics and Management**, v. 5, p. 81–102, 03 1978. Citado na página 33.

HE, X.; ZHAO, K.; CHU, X. Automl: A survey of the state-of-the-art. **Knowledge-Based Systems**, v. 212, p. 106622, 2021. ISSN 0950-7051. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950705120307516>. Citado na página 17.

HUNTER, J. D. Matplotlib: A 2d graphics environment. **Computing in science & engineering**, IEEE, v. 9, n. 3, p. 90–95, 2007. Citado na página 19.

LUNDBERG, S. M.; LEE, S.-I. ***SHAP Values Github Page***. 2017. <https://github.com/slundberg/shap>. [Online: Accessed 1-Nov-2021]. Citado 2 vezes nas páginas 31 e 32.

____. A unified approach to interpreting model predictions. In: GUYON, I.; LUXBURG, U. V. *et al.* (Ed.). **Advances in Neural Information Processing Systems 30**. Curran Associates, Inc., 2017. p. 4765–4774. Disponível em: <http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf>. Citado na página 19.

MAKRIDAKIS, S. The forthcoming artificial intelligence (ai) revolution: Its impact on society and firms. **Futures**, v. 90, p. 46–60, 2017. ISSN 0016-3287. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0016328717300046>. Citado na página 17.

MCKINNEY, W. *et al.* Data structures for statistical computing in python. In: **Proceedings of the 9th Python in Science Conference**. Austin, TX: [s.n.], 2010. v. 445, p. 51–56. Citado na página 19.

MLFLOW. ***MLFlow Documentation Page***. 2018. <https://mlflow.org/docs/latest/tutorials-and-examples/tutorial.html>. [Online: Accessed 1-Nov-2021]. Citado na página 35.

PEDREGOSA, F.; VAROQUAUX, G. *et al.* Scikit-learn: Machine learning in python. **Journal of machine learning research**, v. 12, n. Oct, p. 2825–2830, 2011. Citado na página 19.

REFAEILZADEH, P.; TANG, L.; LIU, H. Cross-validation. In: ____. **Encyclopedia of Database Systems**. New York, NY: Springer New York, 2016. p. 1–7. ISBN 978-1-4899-7993-3. Disponível em: <https://doi.org/10.1007/978-1-4899-7993-3_565-2>. Citado na página 25.

ROSSUM, G. V.; DRAKE, F. L. **Python 3 Reference Manual**. Scotts Valley, CA: CreateSpace, 2009. ISBN 1441412697. Citado na página 19.

SOLA, J.; SEVILLA, J. Importance of input data normalization for the application of neural networks to complex industrial problems. **IEEE Transactions on Nuclear Science**, v. 44, n. 3, p. 1464–1468, 1997. Citado na página 24.

TROYANSKAYA, O.; CANTOR, M.; SHERLOCK, G.; BROWN, P.; HASTIE, T.; TIBSHIRANI, R.; BOTSTEIN, D.; ALTMAN, R. B. Missing value estimation methods for DNA microarrays . **Bioinformatics**, v. 17, n. 6, p. 520–525, 06 2001. ISSN 1367-4803. Disponível em: <https://doi.org/10.1093/bioinformatics/17.6.520>. Citado na página 23.

WANG, W.; CHAKRABORTY, G.; CHAKRABORTY, B. Predicting the risk of chronic kidney disease (ckd) using machine learning algorithm. **Applied Sciences**, v. 11, n. 1, 2021. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/11/1/202>. Citado 2 vezes nas páginas 28 e 30.

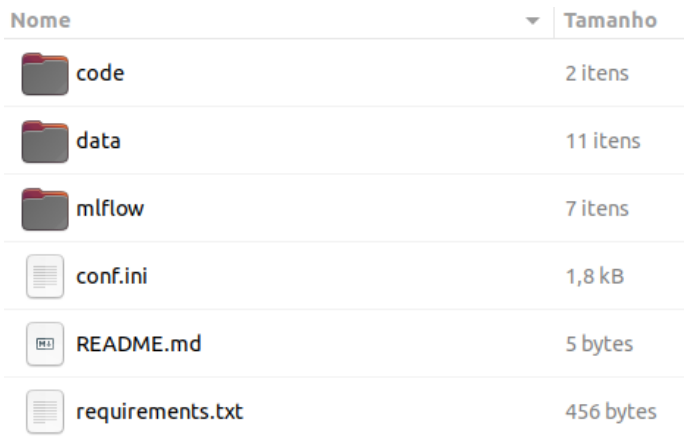
APÊNDICE A

ESTRUTURA DO PROJETO E INSTALAÇÃO DO SISTEMA

A.1 Estrutura do Projeto

Todo o projeto foi desenvolvido e testado utilizando o sistema operacional Ubuntu 20.04. A estrutura de arquivos no diretório do projeto pode ser conferida na Figura 20.

Figura 20 – Diretório do projeto



Fonte: Elaborada pelo autor.

Dentro da pasta *code* existem os arquivos *python* que realizam as tarefas de pré-processamento e modelagem. Já a pasta *data* deverá possuir os arquivos *.csv* que originarão as análises. A pasta *mlflow* é utilizada para armazenamento de resultados e execuções, não sendo necessária sua manipulação.

O arquivo *conf.ini* apresenta as configurações que o usuário deve preencher, já o arquivo *requirements.txt* possui os pacotes e bibliotecas necessários para a execução do projeto. Por fim, o arquivo *README.md* é utilizado apenas para trazer informações sobre o projeto em repositórios *git*.

O arquivo de configuração *conf.ini* pode ser conferido no Código-fonte 6. As linhas que começam com # são comentários e explicam o que deve ser preenchido em cada variável.

Código-fonte 6: Arquivo conf.ini

```
# Note: fill all the variables (including the strings) without any
# quotation marks ("" or '')

[PREPROCESS]
# Name of the csv file with the data. The csv file must be inside
# the data folder.
file_name = winequality-red.csv
# Null values acceptable threshold. All the columns that have more than
# (null_threshold*100)\% null values will be removed. If you don't want
# to remove any columns, use null_threshold = 0.
null_threshold = 0.1
# Correlation threshold to drop highly correlated variables. If you don't
# want to remove any columns, use corr_threshold = 1.
corr_threshold = 0.8
# Name of the target that will be used to train the model.
target_name = quality
# Check if you want to replace missing data using the KNN imputation method.
imputation = False
# Check if you want to normalize the data using min-max scaler.
# If you use normalization, you can't use standardization.
norm = False
# Check if you want to rescale the data using a standardization method.
# If you use standardization, you can't use normalization.
standard = True
# Run_id that identify the execution.
run_id = 5

[MODELING]
# Select what models you want to test. To select a specific model,
# copy the word inside the parentheses and write it in the models
# parameter, separating each model using a comma (,).
# The available models are:
# 1) KNeighborsRegressor (KNN)
# 2) XGBRegressor (XGB)
# 3) LassoRegressor (LR)
# 4) RandomForestRegressor (RFR)
# 5) MLPRegressor (MLP)
models = RFR, KNN
# Number of folds for K-Fold Cross-Validation
folds = 5
# Test size for train-test split. A good value is between
```

```
42 test_size = 0.2
43 # Name of the experiment that will be saved in MLFlow
44 experiment_name = wine_testing
45 # Name of the run that will be saved in MLFlow
46 run_name = rfr_knn_testing
```

A.2 Instalação do sistema e configurações adicionais

Antes da primeira execução do sistema, é necessário instalar o ambiente virtual. Estando dentro da pasta do projeto, a instalação é feita através do seguinte comando:

```
$ python3 -m venv .venv
```

Com o ambiente virtual instalado, é necessário ativá-lo utilizando o seguinte comando:

```
$ source .venv/bin/activate
```

Em seguida, deve ocorrer a instalação dos pacotes e bibliotecas necessários, através da utilização do arquivo de texto *requirements.txt* e podem ser instalados através do seguinte comando:

```
$ pip install -r requirements.txt
```

Para configurar a base *PostgreSQL*, responsável por armazenar as informações do *MLFlow*, é necessário utilizar o seguinte comando:

```
$ sudo -u postgres psql
```

E então digitar o seguinte comando:

```
CREATE DATABASE mlflow_db;
CREATE USER mlflow_user WITH ENCRYPTED PASSWORD 'password';
GRANT ALL PRIVILEGES ON DATABASE mlflow_db TO mlflow_user;
```

Obs: a senha 'password' pode ser mudada para uma senha de sua preferência.

Por fim, para ativar o servidor *MLFlow* basta utilizar o seguinte comando com o ambiente virtual ativado:

```
mlflow server --backend-store-uri
postgresql://mlflow_user:password@0.0.0.0/mlflow_db
-h 0.0.0.0 --default-artifact-root
file:/home/<path_to_project>/mlflow/mlruns/
--port 5000
```

Obs: é possível colocar o comando de ativação do *MLFLow* no arquivo *.bashrc*, associando um *alias* ao comando para facilitar a execução. No arquivo, basta colocar a seguinte expressão:

```
alias mlflow_server='source .venv/bin/activate &&
mlflow server --backend-store-uri
postgresql://mlflow_user:password@0.0.0.0/mlflow_db
-h 0.0.0.0
--default-artifact-root file:/home/<path_to_project>/mlflow/mlruns/
--port 5000'
```

Desta forma, basta utilizar o comando *mlflow_server* que o ambiente virtual e o servidor serão ativados automaticamente.

O servidor ficará ativo e disponível para acesso no endereço *0.0.0.0:5000*, bastando utilizar um navegador de internet para acessá-lo.