CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS DIVINÓPOLIS

Yuri Dimitre Dias de Faria

PREDIÇÃO DO TEMPO DE EXECUÇÃO DE SOFTWARES DE OTIMIZAÇÃO PARA RESOLUÇÃO DE PROBLEMAS DE PROGRAMAÇÃO LINEAR INTEIRA MISTA

Divinópolis
2023

YURI DIMITRE DIAS DE FARIA

PREDIÇÃO DO TEMPO DE EXECUÇÃO DE SOFTWARES DE OTIMIZAÇÃO PARA RESOLUÇÃO DE PROBLEMAS DE PROGRAMAÇÃO LINEAR INTEIRA MISTA

Trabalho de Conclusão de Curso apresentado no curso de Graduação em Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação.

Prof. Dr. André Luiz Maravilha Silva

DIVINÓPOLIS

2023

YURI DIMITRE DIAS DE FARIA

PREDIÇÃO DO TEMPO DE EXECUÇÃO DE SOFTWARES DE OTIMIZAÇÃO PARA RESOLUÇÃO DE PROBLEMAS DE PROGRAMAÇÃO LINEAR INTEIRA MISTA

Trabalho de Conclusão de Curso apresentado no curso de Graduação em Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação.

Aprovado em 31 de janeiro de 2024.

CEFET-MG - Divinópolis

RESUMO

Neste trabalho, é estudado métodos para prever o tempo de execução e a quantidade de nós da árvore *Branch and Bound* para softwares de otimização para problemas de programação linear inteira mista. O estudo propõe verificar o uso de técnicas baseadas em Aprendizado de Máquina, incluindo *Decision Tree*, *Random Forest* e *Gradient Boosting*, para construir um modelo preditivo. Os resultados revelaram desafios na previsão do tempo de execução, indicando uma baixa representatividade dos dados e erros consideráveis, enquanto a previsão da quantidade de nós na árvore *Branch and Bound* mostrou-se satisfatória, apresentando um coeficiente de determinação elevado e erros aceitáveis

Palavras-chave: MILP; Machine Learning; Runtime prediction.

ABSTRACT

In this work, methods for predicting the runtime and the number of nodes in the Branch and Bound tree for optimization software for mixed-integer linear programming problems are studied. The study proposes to investigate the use of Machine Learning-based techniques, including Decision Tree, Random Forest, and Gradient Boosting, to build a predictive model. The results revealed challenges in predicting runtime, indicating low representativeness of the data and considerable errors, while predicting the number of nodes in the Branch and Bound tree proved to be satisfactory, showing a high coefficient of determination and acceptable errors.

Palavras-chave: MILP; Machine Learning; Runtime prediction.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANOVA Análise de Variância, do inglês Analysis of variance

B&B Branch and Bound

CPU Unidade Central de Processamento, do inglês Central Processing Unit

DL Aprendizagem Profunda, do inglês Deep Learning

DT Árvore de Decisão, do ingles Decision Tree

GB Gradient Boosting

LP Programação Linear, do inglês Linear Programming

MAE Erro Absoluto Médio, do inglês Mean Absolute Error

MILP Programação Linear Inteira Mista, do inglês Mixed Integer Linear Programming

MIP Programação de Inteira Mista, do inglês Mixed Integer Programming

ML Aprendizado de Máquina, do inglês Machine Learning

MSE Erro Quadrático Médio, do inglês Mean Squared Error

RF Árvore Aleatória, do inglês Random Forest

SUMÁRIO

1 INTRODUÇÃO

1.1 Considerações Iniciais

Atualmente, há muita discussão sobre técnicas de inteligência artificial e suas aplicações em várias áreas. Ferramentas que se baseiam principalmente em métodos heurísticos, aprendizado de máquina (ML, do inglês *Machine Learning*) e aprendizagem profunda (DL, do inglês *Deep Learning*) são amplamente utilizadas. As facilidades no desenvolvimento de soluções utilizando ML faz com que ela esteja presente em diversos setores de estudo, que não se limitam apenas à computação, mas também se estendem a áreas correlatas e diversas, como biologia, direito, artes, entre outras (Chen; Kao, 2022; Sifat, 2023). De fato, há uma crescente mudança de paradigma de como encarar a produtividade no mundo atual. Na computação, engenharia e áreas correlatas, a criação e uso de técnicas de ML já é usual. Ferramentas de ML e DL, além de serem criadas utilizando princípios estatísticos, têm sido utilizadas para resolução de problemas e auxílio à modelagem de questões. Na área de otimização, a inteligência artificial sempre foi utilizada de forma natural, com heurísticas para acelerar a resolução de instâncias. A aplicação de ML e DL em otimização também é comum, podendo ser usada para a aproximação de soluções em programação linear (LP, do inglês *Linear Programming*), problemas combinatórios e programação inteira mista (MIP, do inglês *Mixed Integer Programming*) (Wu; Lisser, 2023; Talbi, 2013; Guedes; Boas, s.d.; Hoeltzel; Chieng, 1987).

Além disso, é possível citar vários outros usos de algoritmos de ML que auxiliam na resolução de problemas de otimização, como a predição do tempo de execução. Com isso, ML pode ter um bom desempenho em correlacionar parâmetros e predizer resultados esperados, que nesse caso é o tempo de execução de um algoritmo. Muitas áreas do conhecimento já fazem uso de ML para predizer o comportamento esperado de um algoritmo em relação à sua entrada (Barry; Schumann, 2019). Embora em muitos casos possa ser usado uma abordagem analítica, abordagens utilizando ML são, em geral, mais fáceis de se conceber graças a ferramentas e *toolkits* especializados, com uma precisão bastante elevada em relação à massa de dados treinadas por meio de algoritmos interpolativos e um desempenho aceitável em algoritmos extrapolativos (Ahmed *et al.*, 2022).

Nesse contexto, embora seja muito usado em outras áreas com essa finalidade, algoritmos de aprendizagem de máquina acabaram ficando restritos ao auxílio na busca da solução de problemas, deixando outros usos como a predição do tempo de execução ou classificação de problemas ofuscada.

A programação linear inteira mista (MILP, do inglês *Mixed Integer Linear Programming*) tem aplicações variadas no cotidiano, abrangendo desde a criação de grades de horários (Fonseca, 2017) até a organização de armazenamento e guindastes dinâmicos em canteiros de obras (Riga, 2020). No entanto, devido à complexidade de problemas que envolvem múltiplas entradas e restrições, é comum recorrer a softwares comerciais de otimização para auxiliar na resolução. Em muitos casos, a execução do software pode demandar um tempo considerável, prejudicando o andamento das tarefas planejadas. Com intuito de amenizar esse impacto, esses softwares aplicam heurísticas para entregar uma solução sub-ótima, mas satisfatória. Porém, muitas vezes precisa-se realizar configurações especificas, ou escolher métodos especializados, para que a resposta seja entregue em tempo hábil.

Por essa razão, propõe-se o desenvolvimento de um modelo baseado em técnicas de ML para prever o tempo de execução de softwares de otimização na resolução de MILP. Essa previsão é essencial para um planejamento adequado, permitindo a definição de limites de tempo realistas e a adoção de estratégias adequadas. O modelo pode ser treinado com dados históricos, considerandos fatores como o tamanho do problema, a complexidade das restrições e a capacidade de processamento do hardware utilizado.

A previsão acurada do tempo de execução possibilita uma melhor gestão de projetos e programação, permitindo acompanhar e monitorar o progresso das tarefas em tempo real. Com isso, é possível reagir de forma proativa a atrasos ou problemas, ajustando prioridades ou escolhendo estrategias de soluções diferentes.

1.2 Objetivo

O objetivo geral deste trabalho é construir um modelo de predição a partir de técnicas de aprendizado de máquina para prever informações de execução de softwares de otimização para problemas de programação linear inteira mista. Para isso, são definidos os seguintes objetivos específicos:

- Selecionar e organizar instâncias de MILP para a execução do software de otimização que será utilizado para treinamento do modelo.
- Selecionar as características/atributos dos problemas.
- Explorar diferentes técnicas de predição baseados em aprendizagem de máquina.
- Efetuar comparações com modelos de predição escolhidos a fim de medir seus desempenhos.

1.3 Contribuições

Espera-se que este trabalho contribua para uma tomada de decisão mais assertiva em relação ao método utilizado para resolver problemas MILP. Essa decisão deve ser consciente, levando em consideração o esforço e o tempo requerido pelo otimizador para resolver o problema. Com base nisso, será possível determinar a necessidade de adotar outros métodos de resolução, como o desenvolvimento de uma heurística especializada para o problema que, embora não garanta a solução ótima, pode resultar em soluções de boa qualidade, mesmo diante de restrições de tempo de resposta para se obter uma solução para o problema sendo resolvido.

1.4 Organização da Monografia

A estrutura deste trabalho consiste em seis capítulos. Neste Capítulo 1, foi apresentada uma introdução ao tema tratado, fornecendo uma visão inicial do problema abordado e do contexto envolvido, seguido pela apresentação do objetivo central do trabalho, detalhando os objetivos específicos.

No Capítulo 2, é apresentado o referencial bibliográfico. Esse capítulo tem o intuito de realizar um levantamento e revisão crítica da literatura relacionada à área de Otimização e Aprendizado de Máquina. O principal objetivo é contextualizar o trabalho dentro do cenário acadêmico existente, identificando trabalhos correlatos, abordagens adotadas, e conceitos fundamentais.

No Capítulo 3, é abordado o referencial teórico, que revisa alguns conceitos e definição de otimização, assim como técnicas de Aprendizado de Máquina para a predição do tempo de execução de algoritmos. Nesse Capítulo, são explorados os

principais conceitos teóricos e trabalhos relacionados presentes na literatura.

O Capítulo 4 tem como propósito descrever a metodologia adotada para o desenvolvimento do projeto. A base teórica discutida nos capítulos anteriores é utilizada no planejamento experimental. Além disso, são consideradas métricas apropriadas para a comparação com outros métodos também abordados na literatura. Esse capítulo detalha, passo a passo, o processo metodológico empregado, fornecendo um guia claro para a replicação dos experimentos.

O Capítulo 5 apresenta os resultados obtidos por meio dos experimentos realizados. São analisados os dados coletados e discutidos os principais achados. Além disso, será avaliada a viabilidade e efetividade da ferramenta desenvolvida, com base nos resultados obtidos. Essa seção fornece uma avaliação crítica dos resultados, considerando suas implicações práticas e contribuições para o campo de estudo em questão.

Por fim, o Capítulo 6 são apresentadas as conclusões derivadas dos resultados obtidos ao longo do trabalho. Este capítulo destaca de maneira sistemática os principais achados, destacando se os objetivos propostos foram alcançados Além disso, esse capítulo propõe discussões sobre novas abordagens para trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

O cenário atual da pesquisa em predição de tempo de execução e otimização de algoritmos em diversos domínios, como problemas combinatórios, resolução de restrições, supercomputação, *big data* e programação linear, destaca a importância do uso de técnicas avançadas, especialmente aquelas baseadas em ML.

No contexto de problemas combinatórios, como SAT e TSP, as técnicas apresentadas no artigo de Hutter *et al.* (2014) demonstram a eficácia de abordagens baseadas em florestas aleatórias na previsão do tempo de execução. A consideração de características de instância e parâmetros dos algoritmos como entradas para o modelo de previsão contribui significativamente para uma melhor generalização em novas instâncias de problemas e algoritmos parametrizados.

A pesquisa de Truchet *et al.* (2016) se aprofunda na estimativa de tempos de execução paralelos para algoritmos aleatórios em resolução de restrições, destacando uma abordagem probabilística que se mostra eficaz. A comparação com outras abordagens sem ferramentas estatísticas e a revisão das abordagens existentes para resolução de restrições paralelas fornecem *insights* valiosos para otimizar a execução desses algoritmos.

No domínio de *runtime enforcement* (RE), Pinisetty *et al.* (2017) apresentam um framework preditivo para aplicação de RE em propriedades regulares. O artigo destaca a importância do conhecimento prévio do sistema para melhorar as políticas de aplicação, resultando em maior eficácia em termos de tempo de execução e número de estados do autômato apresentados no trabalho.

A pesquisa de Demirović *et al.* (2019) investiga a predição e otimização para o problema da mochila, explorando diferentes métodos de aprendizado. Os autores fornecem uma visão abrangente do problema, comparando métodos indiretos, diretos e semi-diretos. O estudo conclui que a melhor abordagem para problemas de predição e otimização ainda é uma questão em aberto, fornecendo visões valiosas sobre a relação com a otimização estocástica.

Barry e Schumann (2019) utilizam técnicas de aprendizado de máquina para prever o tempo de execução e ajustar os parâmetros de solucionadores matemáticos, destacando a eficácia da combinação de RF com *Multi-Objective Random Genetic*

Algorithm (MRGA). Essas descobertas contribuem para melhorar o desempenho dos solucionadores matemáticos em cerca de 40% em relação aos parâmetros padrão.

A análise de algoritmos de aprendizado para previsão de tempo de execução de *jobs* em supercomputadores, conforme realizado por Savin *et al.* (2020), destaca o método floresta randômica (RF, do inglês *Random Forest*) como o mais preciso, seguido pelo método *Gradient Boosting*. A discrepância entre o tempo requisitado pelo usuário e o tempo real de execução destaca a necessidade de métodos precisos de previsão. A Tabela 1 demonstra o comparativo da acurácia em prever a próxima tarefa dos diferentes algoritmos presentes nos experimentos dos dois supercomputadores, o MVS-10P e o MVS-100K.

Fonte: Adaptado de (Savin *et al.*, 2020)

Algoritmo	***MVS-10P***		**MVS-100K**	
	conjunto de treinamento	**amostra de teste**	**conjunto de treinamento**	**amostra de teste**
Logistic regression	0.49	0.49	0.63	0.63
Decision trees	0.79	0.66	0.81	0.73
k-Nearest neighbors	0.68	0.61	0.74	0.70
Linear discriminant analysis	0.48	0.48	0.61	0.61
Support-vector machines	0.57	0.57	-	-
Random Forest	0.68	0.68	0.73	0.73
Gradient boosting	0.59	0.59	0.58	0.58
Feedforward neural network	0.63	0.64	0.69	0.70

A pesquisa de Ardagna *et al.* (2021) aborda os desafios de prever o desempenho de aplicativos de *big data* em nuvem, apresentando três abordagens de modelagem, incluindo modelos analíticos baseados em filas e um simulador *ad-hoc* chamado dagSim. Os resultados indicam que o modelo de rede de filas hierárquicas e o dagSim são eficazes na previsão do desempenho dos aplicativos de *big data*, destacando a importância da escolha do modelo dependendo do contexto.

Ahmed *et al.* (2022) realizam uma comparação de desempenho entre métodos analíticos e algoritmos regressivos de ML para a predição de tempo de execução em processamento paralelo de dados. O estudo destaca que o *Gradient Boosting*, um regressor de ML, é mais preciso que modelos analíticos existentes em situações dentro do intervalo de treinamento.

O artigo de Wang *et al.* (2022) introduz o modelo *RunningNet*, que utiliza uma arquitetura de rede neural recorrente (RNN, do ingles *Recurrent neural network*) com atenção para prever o tempo de execução de trabalhos em sistemas de

supercomputação. Os resultados indicam que o *RunningNet* supera outros métodos existentes, fornecendo estimativas precisas e destacando a importância do comportamento do usuário para melhorar a alocação de recursos.

Zhang *et al.* (2023) exploram o uso de ML na resolução de problemas de MIP em otimização combinatória. A pesquisa destaca a viabilidade de abordagens heurísticas baseadas em ML para problemas NP-difícil, oferecendo soluções com base em padrões de treino.

Finalmente, Wu e Lisser (2023) investigam o uso de redes DL para a resolução de problemas de programação linear. O modelo de rede neural é construído como uma previsão do problema LP, fornecendo uma solução aproximada. A utilização de variáveis de entrada dos problemas LP como entradas para a rede destaca a aplicabilidade dessas técnicas na otimização matemática.

3 REFERENCIAL TEÓRICO

Neste capítulo serão apresentados tópicos que dão suporte para o desenvolvimento do trabalho. Na Seção 3.1 é feita uma breve apresentação de problemas de otimização e suas especializações. Em seguida, na Seção 3.2, é apresentado estratégias para resolução para problemas de otimização, além de apresentar os softwares de otimização e suas estratégias. Por fim, a Seção 3.3 é destinada a apresentar modelos e técnicas para predição, além de detalhar as estratégias utilizadas neste do trabalho.

3.1 Otimização Linear

Os problemas de otimização podem ser descritos de uma forma geral como apresentado abaixo:

$$\begin{aligned} &\text{Otimizar } f(x) \\ &\text{sujeito a: } g(x) \leq b \\ &x \in \mathbb{R}^n \end{aligned} \tag{3.1}$$

em que se deve determinar o vetor $x \in \mathbb{R}^n$, denominado variáveis de decisão, de forma que a função $f : \mathbb{R}^n \to \mathbb{R}$ seja otimizada. A função f é denominada função objetivo e o termo “otimizar” se refere à minimização ou maximização desta função. No entanto, os valores possíveis para o vetor x está sujeito a um conjunto de restrições $g(x) \leq b$, sendo $g : \mathbb{R}^n \to \mathbb{R}^m$ a função de restrições e $b \in \mathbb{R}^m$ é um vetor coluna de termos independentes das restrições. Valores de x que satisfazem as restrições são denominados de soluções viáveis. O conjunto de todas as soluções viáveis formam o espaço de soluções (Bazaraa; Jarvis; Sherali, 2004).

Quando as funções f e g são lineares em x, tem-se uma categoria especial de problemas de otimização, denominados problemas de programação linear (LP, do inglês *Linear Programming*) (Bazaraa; Jarvis; Sherali, 2004; Goldbarg, 2005). Um problema de

LP pode, então, ser escrito como:

$$\begin{aligned} &\text{Otimizar } c^T x \\ &\text{sujeito a: } Ax \leq b \\ &x \geq 0 \end{aligned} \tag{3.2}$$

onde, $A \in \mathbb{R}^{m \times n}$ é a matriz de coeficientes das restrições e $c \in \mathbb{R}^n$ é um vetor coluna de coeficientes de custo.

Uma vantagem de problemas modelados como problemas de LP está na eficiência dos algoritmos existentes para sua solução, como o algoritmo Simplex (Dantzig, 1963). O Simplex, apesar de ter complexidade exponencial no pior caso, tende a ser rápido na prática (Cormen *et al.*, 2022).

Quando é tido um problema LP que apresenta uma ou mais variáveis que assumem, necessariamente, valores inteiros, tem-se um MILP. Dessa forma, podemos dividir as variáveis de decisão em dois grupos: o grupo de variáveis contínuas $x \in \mathbb{R}^n_+$ e o conjunto de variáveis inteiras $y \in \mathbb{Z}^p_+$. Um problema de MILP pode ser escrito como:

$$\begin{aligned} &\text{Otimizar } c^T x + h^T y \\ &\text{sujeito a: } Ax + Gy \leq b \\ &\qquad x \geq 0, \text{ e } x \in \mathbb{R}^n \\ &\qquad y \geq 0, \text{ e } y \in \mathbb{Z}^p \end{aligned} \tag{3.3}$$

onde, $G \in \mathbb{R}^{m \times p}$ é a matriz dos coeficientes das restrições associada às variáveis inteiras e $h \in \mathbb{R}^p$ é um vetor coluna de coeficientes de custo.

Esse trabalho tem como foco os problemas de programação linear inteira mista. Assim, as subseções a seguir apresentam as principais estratégias adotadas pelos softwares comerciais de otimização para resolução dessa categoria de problemas.

3.2 Estratégia para Resolução de Problemas

O espaço de soluções em problemas de MILP é um conjunto enumerável. Portanto, a primeira estratégia que se pode imaginar para sua resolução é enumerar e avaliar todas as possíveis alternativas e assim, determinar a solução ótima. No entanto,

esses problemas tendem a apresentar um rápido crescimento do número de possíveis respostas, chamado de “explosão combinatória” (Wolsey, 1998).

Dessa forma, enumerar de forma explícita todas as alternativas para se resolver problemas MILP torna-se impraticável. Com isso, são necessárias estratégias eficientes para lidar com a explosão combinatória, evitando a avaliação de todas as soluções possíveis e, ainda assim, garantir a otimalidade da resposta retornada.

3.2.1 Algoritmos Exatos

Os algoritmos exatos são aqueles que garantem a solução ótima para um problema de otimização. Como o número de soluções de problemas de MILP cresce exponencialmente em função do número de variáveis de decisão e de restrições, a completa enumeração das soluções é impraticável (Wolsey, 1998). Com isso, surge a seguinte questão: “Como evitar a enumeração completa das soluções sem abdicar da garantia de se encontrar a solução ótima para o problema?”.

A seção seguinte apresenta o algoritmo exato implementado pelos softwares de otimização para a resolução de problemas de MILP, chamado *Branch and Bound* (B&B). Vale ressaltar que o B&B é o algoritmo base e implementado nos softwares de otimização, e guiam o processo de busca, contando, ainda, com vários outros mecanismos como pré-processamentos, buscas heurísticas e reformulações do problema para o cálculo da solução.

3.2.1.1 Branch and Bound

O algoritmo *Branch and Bound* B&B (Land; Doig, 2010) realiza uma enumeração implícita do conjunto de soluções mediante uma estratégia baseada no princípio de dividir para conquistar. Com isso, grande parte das soluções deixam de ser explicitamente avaliadas sem que se perca a garantia de se encontrar a solução ótima (Bertsimas *et al.*, 1997).

Seja F o conjunto de soluções viáveis para um problema de MILP em que se deseja,

sem perda de generalidade, minimizar a função objetivo:

$$\begin{aligned} &\text{Minimizar } c^T x \\ &\text{sujeito a: } x \in F \end{aligned} \tag{3.4}$$

em que o conjunto F pode ser particionado em subconjuntos finitos $F_1, \ldots, F_k$, sendo $F = F_1 \cup \ldots \cup F_k$ e cada subproblema é resolvido separadamente. Dessa forma, o problema completo descrito acima pode ser representado como k subproblemas da seguinte forma:

$$\begin{aligned} &\text{Minimizar } c^T x \\ &\text{sujeito a: } x \in F_i, \qquad i = 1, \ldots, k \end{aligned} \tag{3.5}$$

As soluções dos subproblemas são comparadas e, então, é escolhida a melhor. Cada subproblema tem complexidade quase tão difícil quanto o problema original F, o que sugere a resolução pelo mesmo método. Com isso, é criado uma árvore de subproblemas (Bertsimas *et al.*, 1997).

É possível assumir a existência de um algoritmo bastante eficiente que, para cada F_i de interesse, calcula-se um limite inferior $b(F_i)$ para o custo ótimo do subproblema correspondente, isto é,

$$b(F_i) \leq \min_{x \in F_i} c^T x. \tag{3.6}$$

Considere U, denominado limite superior, como o valor da função objetivo para a melhor solução viável encontrada até o momento. Se o limite inferior $b(F_i)$ correspondente a um subproblema particular satisfaz $b(F_i) \geq U$, então o subconjunto pode ser descartado já que a solução ótima para esse subproblema não é melhor que a melhor solução viável conhecida até o momento (Bertsimas *et al.*, 1997). Dessa forma, o B&B evita a enumeração completa de soluções, sem perda da garantia de otimalidade.

3.3 Predição

A predição é um processo de estimativa dado situações e conhecimento do sistema. Dado observações feitas em um sistema, algoritmos e métodos estatísticos podem ser aplicados a fim de achar correlações entre as características dos eventos, podendo assim ser dito a resposta dado uma extrapolação das amostras presentes no

estudo. A predição pode ser feita para aproximar a amostra de um conjunto de outras amostras, uma classificação, ou do seu comportamento singular, uma regressão (Norvig; Russell, 2021).

3.3.1 Preparação da Base de Dados

Antes de aplicar técnicas de ML, é necessário preparar a base de dados. Existem muitos processos necessários e boas práticas na preparação da base, como tratamento de dados faltantes por meio da introdução de valores ou eliminação da amostra, codificação de atributos qualitativos para quantitativos, tratamento de *outliers* e normalização dos dados.

3.3.1.1 Eliminação de *Outliers* pelo Método Z-score

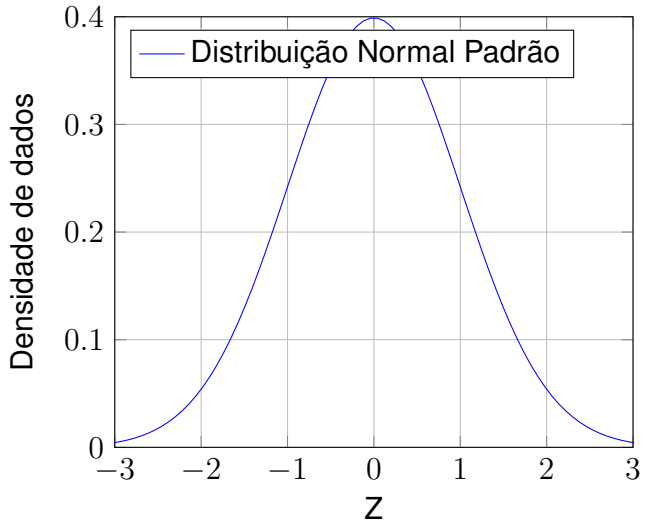
Ao trabalhar com algoritmos de ML, a similaridade dos dados é um fator de importância para gerar um sistema que consiga prever de forma aproximada. Dessa forma, é necessário observar e tratar os chamados *outliers*, que são dados que se dispersam do padrão da base.

O modelo Z-score é concebido para a previsão de falência com base em análise tradicional de índices financeiros e econômicos, aliado ao método de discriminação múltipla (MDA, do inglês *Multiple Discriminant Analysis*). Um Z-score é denominado pontuação padrão e pode ser representado em uma curva de dispersão com valores variando de -3 a 3. A definição do método é expressa pela fórmula:

$$Z = \frac{x - \mu}{\sigma} \tag{3.7}$$

onde x é o valor a ser verificado para o *score*, μ representa a média da população para a métrica, e σ é o desvio padrão para aquela métrica (Altman, 2000). O gráfico da Figura 1 mostra uma distribuição normal, com valores de Z, de -3 a 3, pontuados ao longo do gráfico. Utilizando um valor de Z, é possível eliminar os valores que se distanciam significativamente da média da distribuição normal a partir do ponto escolhido, eliminando assim os *outliers*.

Figura 1 – Gráfico de distribuição normal pontuado por Z-score



3.3.1.2 Normalização pelo Método de Reescala

Quando se trabalha com bases de dados numéricas, é essencial realizar uma análise do comportamento dos dados. Muitas vezes, valores podem exercer domínio sobre outros quando ocorre uma variação muito grande, resultando em uma sobreposição do valor dominante em relação ao outro e tornando o sistema frágil.

A normalização dos dados é frequentemente empregada para lidar com situações de grande variação de valores. Neste trabalho, é abordada a normalização de amplitude por reescala.

A normalização por reescala (*min-max*) define uma nova escala de valores, utilizando limites mínimo e máximo. Inicialmente, são definidos os valores mínimos (min) e máximos (max) para a nova escala. Assim, a reescala pode ser calculada pela seguinte equação:

$$v_{\text{Novo}} = min + \frac{v_{\text{atual}} - min}{\text{maior} - \text{menor}} \cdot (max - min) \tag{3.8}$$

Onde min e max são os limites da nova escala, maior e menor são os limites da escala anterior, e v_{atual} é o valor que será reescalado.

3.3.1.3 Seleção de Atributos Univariada com SelectKBest e ANOVA

O método de seleção univariada de características opera analisando a intensidade da relação entre cada conjunto de dados de entrada e o objetivo, utilizando testes estatísticos univariados como base. O *SelectKBest* emprega a combinação de p-valores e pontuações do teste F para avaliar o desempenho dos parâmetros, identificando e escolhendo as características relevantes. Ao adicionar uma estimativa da quantidade de dependência linear entre duas variáveis aleatórias, o teste F confere maior robustez estatística a esse método em comparação com a abordagem de invólucro, que seleciona características exclusivamente com base nos p-valores (Otchere *et al.*, 2022).

Para este trabalho, é utilizado o teste F baseado na técnica estatística de análise de variância (ANOVA). O teste ANOVA consiste em comparar a média da população amostral e identificar se essas médias diferem significativamente entre elas, utilizando mais de duas amostras por vez (Stahle; Wold, 1989).

Podemos definir o teste ANOVA da seguinte forma:

$$QM_{\text{trat}} = \frac{n \sum_{j=1}^{g} (m_j - M)^2}{g - 1} \tag{3.9}$$

$$QM_{\text{erro}} = \frac{\sum_{j=1}^{g} (n_j - 1) \cdot V_j}{N - g} \tag{3.10}$$

$$F = \frac{QM_{\text{trat}}}{QM_{\text{erro}}} \tag{3.11}$$

onde m_j é a média amostral do índice j, M é a média global, g é o grau de liberdade, n é a quantidade de amostras, N é a quantidade total de dados, V_j é a variância do índice j, e F é o valor de comparação com os dados.

Escolhendo uma quantidade k de atributos das amostras, é possível calcular o valor F_n e, utilizando um valor critico $\frac{g-1}{N-g}$ delimitando uma área de aceitação, determinar se a hipótese de que os atributos são correlatos. Assim, o SelectKBest itera pelos K total de atributos, utilizando um subconjunto k para verificar uma correlação entre eles.

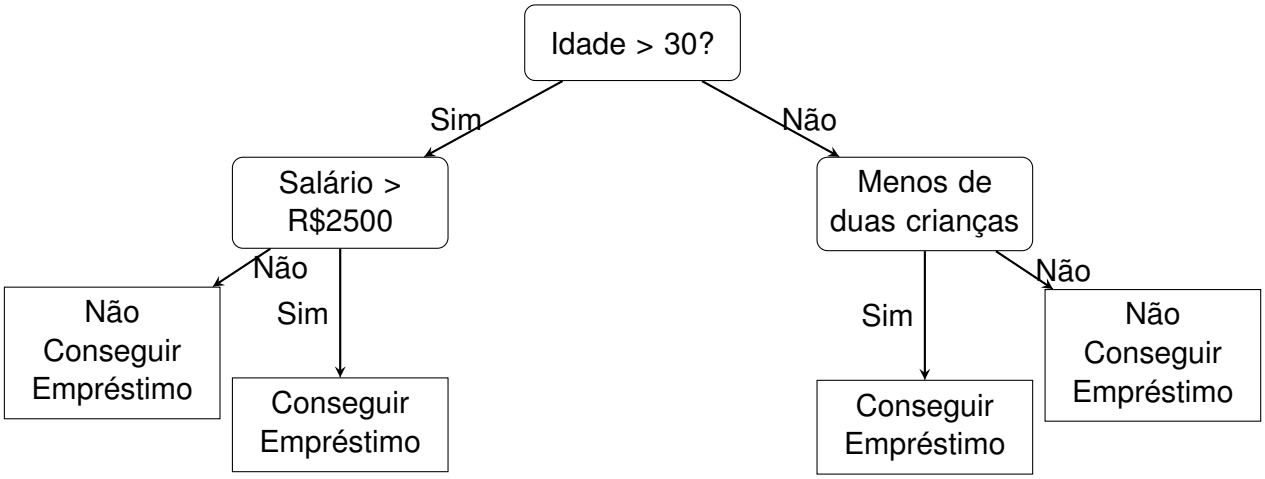
3.3.2 Algorítimos Preditivos

Nesse trabalho, são utilizados algoritmos de ML preditivos regressivos baseados em *Decision Tree* (DT), o método *Random Forest* (RF) e o *Gradient Boosting* (GB), além do próprio DT. Métodos de DT utilizam estratégia de divisão para a resolução de problema de decisão. As soluções de subproblemas podem ser combinadas, para produzir uma solução do problema complexo (Faceli *et al.*, 2021). Uma árvore é construída por particionamento recursivos no espaço das covariáveis, com cada particionamento recebendo o nome de nó e cada resultado recebe o nome de folha (Izbicki; Santos, 2020).

3.3.2.1 Decision Tree

Assim como no método B&B, apresentado na Subseção 3.2.1.1, uma árvore de decisão (DT) utiliza a estratégia de divisão e conquista para a resolução do problema. Dado um problema complexo, este é dividido em subproblemas menores de forma recursiva, gerando uma árvore combinatorial. Essa subdivisão do problema é aplicada para tornar a busca da solução menos custosa, sendo uma combinação das subsoluções dos subproblemas. A Figura 2 é uma representação visual e uma DT para um problema de empréstimo.

Figura 2 – Árvore de Decisão para empréstimo



Formalmente, uma DT é um grafo acíclico direcionado no qual cada nó é um nó de divisão ou um nó folha. Um nó folha é rotulado como uma função que considera apenas os

valores das variáveis alvo. Em geral, nós folha em problemas de regressão possuem uma função que minimiza a função de custo do erro. Por outro lado, um nó de divisão contém um teste condicional baseado nos valores de cada atributo. De forma padrão, esses nós possuem testes univariados, nos quais as condições envolvem apenas um único atributo e valores no domínio desse atributo (Faceli *et al.*, 2021).

Algorithm 1 Algorítimo Decision Tree

Entrada: Um conjunto de treinamento $D = \{(x_i, y_i), i = 1, \ldots, n\}$
Saída: Árvore de Decisão

1: **Função** GERA ÁRVORE(D)
2: **Se** critério de parada(D) = Verdadeiro **então**
3: **Retorna** um no folha rotulado com a constante que minimiza a função perda.
4: Escolha o atributo que maximiza o critério de divisão em D
5: **Para cada** partição dos exemplos D_i baseado nos valores do atributo escolhido **faça**
6: Induz uma subárvore $arvore_i = Gerarvore(D_i)$
7: **Retorna** Árvore contendo um nó de decisão baseado no atributo escolhido, e descendentes $arvore_i$

3.3.2.2 Random Forest

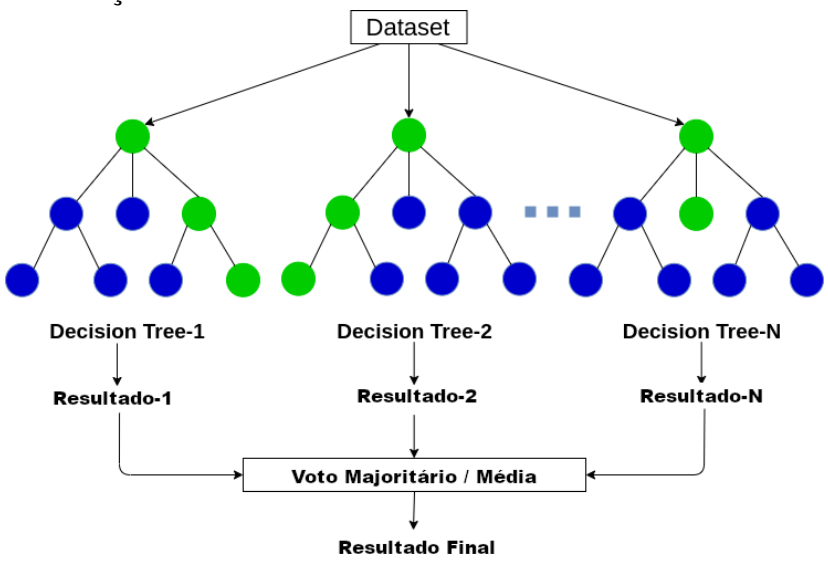
Algoritmos de regressão simples utilizando árvores, como apresentado anteriormente, não possuem um poder preditivo alto comparado a demais estimadores. Um dos métodos que visa contornar essa limitação é chamado floresta aleatória (RF, do inglês *Random Forest*). A estratégia aplicada pelo algoritmo RF, chamada de *bagging*, consiste em uma combinação de diversas árvores para realizar a predição de um problema. Essas separações passam pela técnica de *bootstrap*, que é uma abordagem estatística para estimar a variabilidade de um modelo. Esse modelo estatístico consiste na separação de amostras de *bootstrap* de um conjunto com n observações em k observações do conjunto original, com $k \leq n$. A ideia da técnica *bootstrap* é não necessitar de novas coletas de dados, realizando simulações repetidas a partir dos dados disponíveis. Empregando essa técnica, a estratégia *bagging*, da qual o algoritmo RF se baseia, constrói múltiplos modelos de forma independente e, em seguida, combinar suas previsões para produzir um resultado robusto e geralmente mais preciso que um modelo

individual (James *et al.*, 2013).A proposição matemática do modelo *bagging* pode ser escrita da seguinte maneira:

$$g(x) = \frac{1}{B}\sum_{b=1}^{B} g_b(x) \tag{3.12}$$

Sendo $g(x)$ a função de predição dada pelo *bagging*, $g_b(x)$ a função de predição da b-ésima árvore e B é o número total de árvores (Izbicki; Santos, 2020). A Figura 3 é uma representação visual de uma RF.

Figura 3 – Representação visual de Random Forest



A técnica de RF se difere de outros modelos de combinação de árvores utilizando a técnica *bagging* pela introdução de aleatoriedade na seleção de características em cada divisão de nó. Essa seleção aleatória introduz uma maior diversidade nas árvores individuais e ajuda a reduzir a correlação entre elas (Izbicki; Santos, 2020)(Kuhn; Johnson, 2013). Com isso, é possível abstrair o algoritmo RF da seguinte maneira:

Algorithm 2 Algoritmo Random Forest

Entrada: Conjunto de dados original D e número de modelos a construir m

Saída: Floresta Aleatória de Árvores de Decisão

1: **Função** FLORESTAALEATORIA(D, m)
2: **Para cada** $i = 1$ até m **faça**
3: Gerar uma amostra bootstrap dos dados originais
4: Treinar um modelo de árvore com base nesta amostra
5: **Para cada** cada divisão **faça**
6: Selecionar aleatoriamente k ($< P$) dos preditores originais
7: Selecionar o melhor preditor entre os k preditores e particionar os dados
8: Utilizar critérios típicos de parada de árvore para determinar quando uma árvore está completa (mas não podar)

3.3.2.3 Boosting

As técnicas *Boosting* são formas de *ensemble* de preditores mais fracos, assim como o *Bagging* citado na Subseção 3.3.2.2. Contudo, a montagem do preditor é feita de forma diferente. A forma mais usual, o estimador $g(x)$ é construído de forma incremental a cada iteração do modelo. O estimador começa com um alto viés, com $g(x) = 0$, e a cada passo é atualizado o valor g, adicionando uma função resíduo $r_i = Y_i - g(x_i)$, de modo a diminuir esse viés e aumentar a variância. Formalmente, a função *boosting* consiste no seguinte algorítimo:

Algorithm 3 Boosting

1: Definimos $g(x) = 0$ e $r_i = y_i, \forall i = 1, \ldots, n$.
2: **Para cada** $b = 1, \ldots, B$ **faça**
3: É ajustado uma árvore com p folhas para $(x_1, r_1), \ldots, (x_n, r_n)$. Seja $g^b(x)$ sua respectiva função de predição.
4: Atualiza g e os resíduos: $g(x) \leftarrow g(x) + \lambda g^b(x)$ e $r_i \leftarrow Y_i - g(x)$
5: **Retorna** Modelo final $g(x)$

Sendo r_i a função residual para a iteração i, como uma *DT*, e λ a taxa de aprendizagem do modelo, variando de 0 a 1, para evitar o super-ajuste. Tipicamente, λ tem um valor bem pequeno ($0,01$ por exemplo), $B \approx 1000$ e p é da ordem de 2 ou 4.

O *gradient-boosted trees*, ou *Gradient boosting*, é a técnica *boosting* aplicada diretamente para a montagem de DT. Geralmente tem um melhor desempenho que outros métodos utilizando árvores de decisão, como o RF apresentado na Subseção 3.3.2.2 (Kuhn; Johnson, 2013). É importante que o emprego da técnica *boosting* utilize árvores com profundidade pequena de modo a evitar o super-ajuste.

3.3.3 Validação Cruzada com Método K-fold e Coeficiente R^2

Para garantir que a rede de aprendizado obteve um desempenho satisfatório de ajuste aos dados, é executada uma técnica de validação cruzada, que consiste na aplicação de um algoritmo, utilizando geralmente a divisão da base de dados para treino, e a verificação de uma métrica de desempenho. A métrica de desempenho utilizada neste trabalho é o coeficiente de determinação R^2, que pode ser expresso da seguinte maneira:

$$R^2 = 1 - \frac{RSS}{TSS} \tag{3.13}$$

Onde RSS é a soma dos quadrados dos resíduos, representando a quantidade de variabilidade não explicada pelo modelo, TSS é a soma total dos quadrados, representando a variabilidade total na variável dependente, e R^2 é o coeficiente de determinação, variando de 0 (nenhuma variabilidade na variável dependente) a 1 (o modelo explica toda a variabilidade na variável dependente) (James *et al.*, 2013).

O método k-fold é uma técnica de validação cruzada que utiliza a divisão do subconjunto de dados em k outro subconjunto (ou *folds*). O treinamento e validação desse subconjunto é realizado k vezes, com cada uma das repetições utilizando um *fold* como conjunto de teste e os $k-1$ restantes como treinamento. O desempenho do modelo é calculado pela média de desempenhos nas k iterações:

$$\text{Média} = \frac{1}{k}\sum_{i=1}^{k} D_i \tag{3.14}$$

onde k é o número de *folds* e D é a métrica avaliada, neste caso, o coeficiente de

determinação, na iteração i. Esse método de avaliação garante robustez ao modelo avaliado, além de oferecer uma ferramenta de desempenho para efetuar o ajuste fino (*tuning*) do modelo de predição (Fushiki, 2011).

3.3.4 Métricas de qualidade

Para atestar a qualidade do modelo de aprendizado utilizado, é empregado métricas para averiguar a acurácia do modelo. Para isso, foram selecionadas três métricas usadas na literatura para problemas de regressão: R^2, que foi introduzido na Subseção 3.3.3, o Erro Quadrático Médio e o Erro Absoluto Médio. Essas métricas são utilizadas para auxiliar nas análises dos resultados, fornecendo informações quanto ao erro e o acomodamento do modelo aos dados empregados.

3.3.4.1 Erro Quadrático Médio (MSE)

O MSE mede a qualidade do estimador e é derivado do quadrado da distância euclidiana. Sempre positivo, diminui à medida que o erro se aproxima de zero, indicando um melhor modelo em comparação com outro de valor mais alto. Sua formulação matemática é dada por:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{3.15}$$

onde n é o número de amostras, Y é o vetor de valores observados da variável a ser prevista, e $\hat{Y}$ é o vetor de variáveis previstas (Kuhn; Johnson, 2013).

O MSE é uma métrica sensível, sendo eficaz para penalizar erros grandes. Sua sensibilidade o torna uma métrica útil e de fácil interpretação para comparação de modelos, pois um modelo com MSE menor geralmente indica melhor capacidade de previsão. No entanto, essa mesma sensibilidade o torna suscetível a ruídos de *outliers*, podendo dificultar a análise.

3.3.4.2 Erro Absoluto Médio (MAE)

O MAE, assim como o MSE, é uma métrica usada para avaliar a precisão de modelos de previsão. Ao contrário do MSE, o MAE calcula a média das diferenças

absolutas entre os valores previstos pelo modelo e os valores reais observados nos dados. Quanto mais próximo de zero, melhor é o desempenho do estimador. Sua formulação matemática é dada por:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i| \tag{3.16}$$

onde n é o número total de observações nos dados, Y_i são os valores reais observados, e $\hat{Y}_i$ são os valores previstos pelo modelo.

Diferentemente do MSE, o MAE é robusto a *outliers*, pois não eleva o erro i ao quadrado. Essa característica, aliada a uma interpretação intuitiva e uma ênfase equitativa a todos os erros, faz do MAE uma boa métrica para avaliar modelos (Hyndman; Athanasopoulos, 2018).

4 METODOLOGIA

4.1 Considerações Iniciais

A previsão de características de execução, como o tempo, torna-se desafiadora de modelar analiticamente quando a quantidade de parâmetros a serem considerados se torna extensa e a natureza dessas características se torna complexa. Embora as soluções analíticas sejam menos dispendiosas em termos de processamento, as técnicas de ML, em conjunto com bibliotecas auxiliares para modelagem, oferecem maior facilidade na construção do modelo e na avaliação da precisão da previsão. Nesse contexto, recorremos a algoritmos de ML para desenvolver a ferramenta proposta neste trabalho.

4.2 Planejamento Experimental

A modelagem do sistema de previsão segue uma sequência de passos bem definidos, que incluem pesquisa literária, organização dos dados, medição da execução, treinamento do modelo e validação cruzada.

Inicialmente, é necessário determinar quais características de desempenho desejamos observar e prever. Embora o tempo de execução da máquina não seja uma métrica precisa para avaliar o desempenho de algoritmos isolados, sendo diversa entre sistemas, será utilizado como métrica para avaliar a execução do software de otimização, comumente chamado de *solver*. Além disso, outras características previstas incluem o número de nós da árvore *Branch-and-Bound*, relacionados ao método de otimização em si.

Com essas características em mente, conduzimos uma análise na literatura para identificar os métodos mais indicados para prever características de desempenho. Selecionamos algoritmos baseados na árvore de decisão, como *Decision Tree* (Subseção 3.3.2.1), *Random Forest* (Subseção 3.3.2.2) e *Gradient Boosting* (Subseção 3.3.2.3). Essa escolha baseou-se em resultados eficientes encontrados na literatura para a predição de características de execução de algoritmos (Ahmed *et al.*, 2022) (Savin *et al.*, 2020).

A próxima etapa envolve a organização dos dados para a execução do *solver*. É

utilizada a base MIPLIB (Gleixner *et al.*, 2021), empregada por *solver*s comerciais para ajuste de parâmetros internos. A organização e separação dos dados são explicadas na seção Seção 4.3. Esses dados são utilizados para medir os parâmetros de desempenho do *solver*, treinando a rede. O *solver* escolhido para resolver os problemas selecionados é o Gurobi, amplamente utilizado na indústria e reconhecido como uma das principais ferramentas para resolver MILP (Gurobi Optimization, LLC, 2023). A execução do *solver* é realizada em uma máquina com dois processadores Intel(R) Xeon(R) Silver 4116 de 2.10 GHz, 156 GiB de memória principal e sistema operacional Ubuntu 20.04 (64 bits).

É utilizada a configuração padrão do solver seus atributos em todas as execuções. Tanto o tempo de execução quanto o número de nós da árvore B&B estão diretamente relacionados à capacidade de processamento do sistema. Além disso, conforme descrito na Seção 4.3, alguns problemas da base MIPLIB (Gleixner *et al.*, 2021) não são considerados devido a suas características específicas.

Após a coleta das informações de execução das amostras, é iniciada a etapa de pré-processamento e treinamento do modelo. Para isso, é utilizada a linguagem *Python* com a biblioteca *Scikit-Learn*[1] e outras auxiliares como *Pandas*, na versão 2.1.4, e *Numpy*, na versão 1.26.3, para a construção dos modelos de aprendizagem.

O desempenho do sistema é medido por diversos fatores, abordados na Seção 4.4.

4.3 Base de Problemas de Referência

Para desenvolver o sistema de predição por meio de métodos de ML, é essencial dispor de um conjunto de dados para treinamento, teste e validação do modelo. Nesse sentido, é utilizada a biblioteca MIPLIB (Gleixner *et al.*, 2021). A MIPLIB é uma compilação de problemas de Programação Linear Inteira Mista (MILP) com diversas características, sendo empregada para avaliar o desempenho de *solver*s comerciais em diversos aspectos, como otimização, capacidade de provar inviabilidade de modelos e manipulação de erros numéricos. Optaremos pela versão mais recente, datada de 2017, que contém uma seleção de 1065 instâncias.

As instâncias disponíveis na MIPLIB possuem características que são exploradas neste trabalho para a construção do modelo de predição. Essas características são

[1] https://scikit-learn.org

agrupadas em dois conjuntos principais: “Tamanho”, que engloba informações sobre as dimensões do problema, e “Restrições”, que compreende a quantidade de cada tipo de restrição no problema. Adicionalmente, cada tipo de característica possui duas categorias de valores: “Original” e “*Presolved*”, sendo esta última uma versão que passou por pré-processamento. Neste trabalho, consideraremos apenas as características originais do problema. A Tabela 2 apresenta características das instâncias.

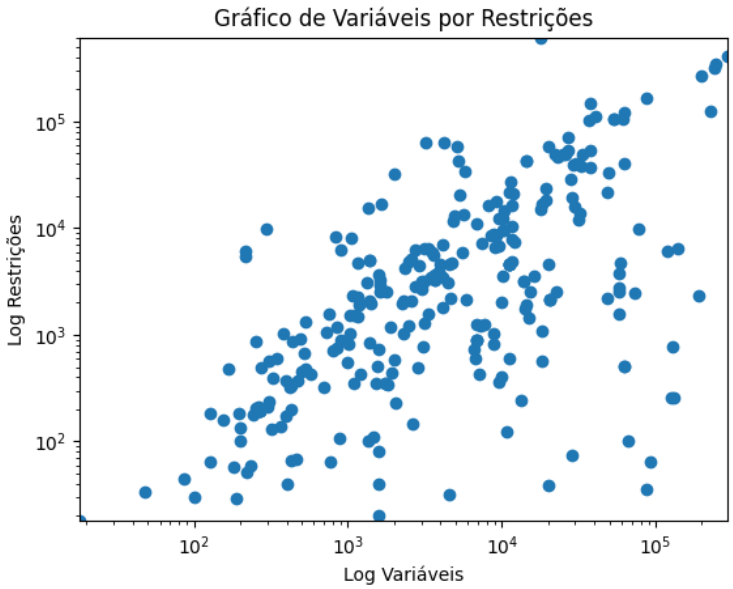
Fonte: Adaptado de (Gleixner *et al.*, 2021)

Tamanho	**Restrições**
Variáveis	Total
Restrições	Vazio
Binarias	Solta
Inteiras	Singular
Continuas	Agregações
Inteiros Implícitos	Precedência
Variáveis Fixas	Variáveis de limite
Densidade de não zeros	Conjunto de particionamento
Não zeros	Conjunto pacote
	Conjunto cobertura
	Cardinalidade
	Mochila invariante
	Equações da mochila
	Empacotamento
	Mochila
	Inteiro de mochila
	Binários mistos
	Linear geral
	Indicador

A MIPLIB é uma base bastante complexa, desenvolvida com o propósito de testar *solvers* matemáticos e verificar sua integridade. Essa base abrange uma ampla gama de problemas, apresentando diversas características para avaliação. Em termos gerais, os problemas tendem a ter valores paramétricos próximos da média. No entanto, é possível identificar *outliers* em todas as classes de características da base, o que pode representar um desafio para o aprendizado do sistema.

A Figura 4 ilustra a quantidade de variáveis em relação ao número de restrições para cada instância. Para essa análise, é aplicado um filtro Z-score (Subseção 3.3.1.1) com um limite de valor de 3 e -3, removendo assim os valores considerados *outliers*.

Figura 4 – Gráfico do número de variáveis por restrições



Mesmo após a aplicação da eliminação Z-score, ainda é possível observar a presença de *outliers* nas características das instâncias, como evidenciado nos gráficos de caixa das Figura 5 e Figura 6. Essa persistência de *outliers* pode representar um desafio no treinamento do modelo e resultar em previsões imprecisas. No entanto, é notável que a maioria das instâncias fornecidas pela base apresenta uma quantidade homogênea de características, o que é vantajoso na construção do sistema de predição.

Figura 5 – Gráfico de caixa da quantidade de variáveis

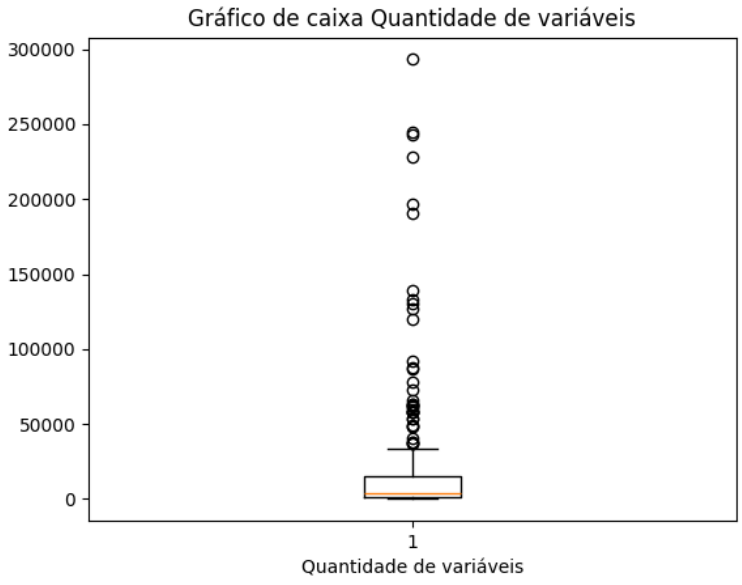


Figura 6 – Gráfico de caixa da quantidade de restrições

Na solução proposta neste trabalho, é feito um tratamento nas amostras da MIPLIB para o treinamento e validação do método de ML. São selecionadas apenas as instâncias para as quais a solução ótima é conhecida e são classificadas como fáceis e/ou médias. As instâncias classificadas como "*open*" ainda não foram resolvidas até a otimalidade e, portanto, estão fora do escopo deste trabalho. Além disso, problemas numéricos não são incorporados ao conjunto de treinamento, pois também estão fora do escopo. Com essa abordagem, foram escolhidas 332 instâncias com as características desejadas.

4.4 Indicadores de Qualidade dos Modelos

Ao lidar com métodos de ML, é crucial realizar uma análise crítica da acurácia do sistema, considerando vários indicadores fornecidos pelo modelo. São utilizados o coeficiente de determinação (R^2), mencionado anteriormente em Subseção 3.3.3, para avaliar o desempenho. Além disso, são analisados o Erro Quadrático Médio e o Erro Absoluto Médio, pois essas métricas têm se mostrado relevantes na literatura (Khoobjou, 2017)(Izbicki; Santos, 2020) (F.y, 2017).

4.5 Experimentos

A parte experimental consiste em três etapas: coleta e tratamento dos dados, treinamento dos modelos e coleta dos resultados. A seguir, é mostrado como cada etapa é construída e como cada técnica e algoritmo é aplicada.

4.5.1 Coleta e Tratamento dos Dados

Assim como citado anteriormente, esse trabalho utiliza a base MIPLIB, que pode ser adquirida no site oficial do repositório. Embora seja fornecido um arquivo comprimido com todas as instâncias dos problemas, não há uma forma prática ofertada pelo site para se adquirir instâncias específicas em massa. Para isso, foi utilizada uma ferramenta *scraping* para poder selecionar de forma eficiente as necessárias, além de extrair as informações características de cada instância, já que não é disponibilizado pelo site em arquivo. Após isso, é executado o *solver* Gurobi a partir de um *script Python*, na versão 3.10.11, utilizando a biblioteca *gurobipy*, na versão 11.0.0, de forma paralela utilizando 10 subprocessos, cada processo limitado a uma única *thread*. O *solver* se manteve nas configurações originais e todas as instâncias alcançaram *status* ótimo. As informações sobre o resultado da otimização, como a quantidade de nós da árvore B&B e o tempo de execução, foram salvos em um arquivo *csv* para melhor manipulação.

Com a coleta de informações sobre a otimização realizada, foi gerado dois arquivos do tipo *csv*: um contendo as informações das instâncias e outro contendo informações da otimização dessa instância. Foi executado uma divisão, em cada instância, da quantidade de tipos de variáveis e restrições pela quantidade total de cada uma delas. Essa etapa foi realizada na tentativa de amortecer o impacto das diferenças de magnitudes destas informações no treinamento do modelo.

4.5.2 Treinamento dos modelos

Para o treinamento dos modelos, foi pensado uma forma iterativa que proporciona uma alta possibilidade de ajustes. Foi realizado uma agregação de dados de interesse ao ajuste e realizada uma permutação com cada elemento, criando um conjunto com todas as

combinações possíveis. Assim, são treinados p_n modelos utilizando os parâmetro p, sendo $p \in P, P =$ *Conjunto permutacao(z,k,m,n)* e p_n a quantidade de elementos do conjunto P. Os dados permutados são:

- z: Uma lista de dez valores, de um a três, utilizados como limiar do Z-Score.
- n: Uma lista de valores, de cinco a vinte e três, utilizados pelo SelectKBest para a quantidade de atributos selecionados.
- k: Uma lista de números, de dois a cinco, utilizadas para a quantidade de cortes na validação cruzada KFold.
- m: Uma lista contendo os modelos de predição: DT, RF e GB

Para o *tunning* do modelo, foi utilizada a função *GridSearchCV* da biblioteca *Scikit-learning*, na versão 1.3.2. Essa função itera entre os parâmetros fornecidos do modelo e valida a eficiência pelo método de validação cruzada, com o método R^2, escolhido como métrica de qualidade. Para o *tunning* dos algoritmos de predição são considerados a profundidade máxima da árvore, o número mínimo de amostras por separação e o número mínimo de amostras por folha. Além disso, é executada uma separação de dados entre treinamento e teste do modelo, utilizando a proporção 90/10 respectivamente, e uma normalização dos dados de treinamento utilizando o método da reescala. Os atributos de cada instância utilizados para o treinamento do modelo são do tipo numérico, sendo eles: quantidade total de variáveis, quantidade dos tipos de variáveis, quantidade de restrições, quantidade do tipo de restrições e a densidade de valores não zeros. Dos dados resultantes da otimização, foram utilizados o tempo de execução e a quantidade de nós da árvore B&B, utilizando apenas uma característica para a montagem do modelo por vez. A estrutura a seguir mostra o fluxo de execução do algoritmo:

4.5.3 Coleta dos Resultados

Após o treinamento ser executado para cada instância, utilizando as configurações p, são calculadas métricas de desempenho para analisar o comportamento do modelo. Assim como citado na Seção 4.4, as métricas aplicadas foram R^2, MSE e MAE. Com isso, foi construída uma base de dados com 4.562 resultados, sendo o tamanho de p_n. Foram coletados os parâmetros ideais de cada modelo para cara iteração p fornecidos

Algorithm 4 Algorítimo Treinamento Modelos

X ← dados características das instâncias
y ← dados resultados otimização
z ← lista de limiares Z-score
k ← lista de quantidade de atributos de treinamento
n ← lista de quantidade separação da validação cruzada
m ← lista modelos de predição
P ← GeraConjuntoPermutação(z,k,n,m);
Para cada elemento p do conjunto P **faça**
    X_{treino},y_{treino},X_{teste},y_{teste} = SeparTreinoTeste(X,y);
    params ← Parâmetros do modelo p_m
    idx ← IndicesOutliers(y_{treino},p_z)
    y_{treino}.drop(idx) , X_{treino}.drop(idx)
    selector ← SelectKBest(ANOVA,p_n)
    pipeline ← Pipeline(MinMaxScaler,selector,p_m)
    vc ← KFold(p_n)
    grid ← GridSearchCV(pipeline,params,vc,R^2)
    grid.treinar(X_{treino},y_{treino})
    $y_{previsto}$ ← grid.prever(X_{teste})

pela *GridSearchCV* para cada uma das p_n iterações, o que permitiu replicar os resultados adquiridos. Os resultados obtidos serão explorados no Capítulo 5

5 RESULTADOS COMPUTACIONAIS

Neste capítulo, procedemos à exposição dos resultados obtidos a partir do treinamento do modelo proposto no capítulo anterior, proporcionando uma visão mais aprofundada sobre o desempenho e a eficácia das abordagens apresentadas.

5.1 Considerações Iniciais

A etapa subsequente à execução dos procedimentos experimentais, conforme delineados no Capítulo 4, envolveu a meticulosa coleta de dados. Este processo estabelece as bases para uma análise criteriosa, fundamentada nas métricas propostas neste trabalho, visando avaliar a viabilidade e o desempenho dos modelos que foram apresentados. É relevante destacar que, em consonância com as diretrizes discutidas na Subseção 4.5.2, além das métricas sugeridas, procede-se à compilação dos parâmetros do conjunto de permutação P. Adicionalmente, foram registrados os parâmetros resultantes da aplicação da função *GridSearchCV*, após a conclusão da validação cruzada e o refinamento do modelo.

5.2 Tabelas e Resultados

Para cada característica prevista, foram geradas duas categorias de tabelas: uma contendo os piores casos e outra apresentando os melhores casos, utilizando cada métrica como critério comparativo. Em ambas as tabelas, são detalhadas as características que influenciaram essa medida, além das demais métricas associadas. No contexto das métricas MAE e MSE, a superioridade do desempenho do experimento é indicada por valores menores, enquanto no caso de R^2, um valor mais próximo de 1 reflete um melhor resultado.

Tabela 3 Melhores valores das métricas para previsão de tempo de execução

Métrica	Z-score	n	Modelo	k	Profundidade máxima	Minímas por folha	Minímas por separação	MAE	MSE	R^2
R^2	2,778	19,0	DecisionTree	4,0	3,0	2,0	1,0	900,418	3347010,168	0,402
R^2	2,778	19,0	DecisionTree	5,0	3,0	2,0	1,0	900,418	3347010,168	0,402
R^2	2,778	20,0	DecisionTree	3,0	3,0	2,0	1,0	900,418	3347010,168	0,402
R^2	2,778	20,0	DecisionTree	4,0	3,0	2,0	1,0	900,418	3347010,168	0,402
R^2	2,778	22,0	DecisionTree	2,0	3,0	2,0	1,0	900,418	3347010,168	0,402
R^2	2,778	22,0	DecisionTree	3,0	3,0	2,0	1,0	900,418	3347010,168	0,402
R^2	2,778	22,0	DecisionTree	4,0	3,0	2,0	1,0	900,418	3347010,168	0,402
R^2	2,778	23,0	DecisionTree	2,0	3,0	2,0	1,0	900,418	3347010,168	0,402
R^2	3,0	19,0	DecisionTree	4,0	3,0	2,0	1,0	900,418	3347010,168	0,402
R^2	3,0	19,0	DecisionTree	5,0	3,0	2,0	1,0	900,418	3347010,168	0,402
R^2	3,0	20,0	DecisionTree	3,0	3,0	2,0	1,0	900,418	3347010,168	0,402
R^2	3,0	20,0	DecisionTree	4,0	3,0	2,0	1,0	900,418	3347010,168	0,402
R^2	3,0	22,0	DecisionTree	2,0	3,0	2,0	1,0	900,418	3347010,168	0,402
R^2	3,0	22,0	DecisionTree	3,0	3,0	2,0	1,0	900,418	3347010,168	0,402
R^2	3,0	22,0	DecisionTree	4,0	3,0	2,0	1,0	900,418	3347010,168	0,402
R^2	3,0	23,0	DecisionTree	2,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MAE	2,778	20,0	DecisionTree	5,0	7,0	2,0	1,0	791,077	3505782,096	0,373
MAE	3,0	20,0	DecisionTree	5,0	7,0	2,0	1,0	791,077	3505782,096	0,373
MSE	2,778	19,0	DecisionTree	4,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MSE	2,778	19,0	DecisionTree	5,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MSE	2,778	20,0	DecisionTree	3,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MSE	2,778	20,0	DecisionTree	4,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MSE	2,778	22,0	DecisionTree	2,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MSE	2,778	22,0	DecisionTree	3,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MSE	2,778	22,0	DecisionTree	4,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MSE	2,778	23,0	DecisionTree	2,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MSE	3,0	19,0	DecisionTree	4,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MSE	3,0	19,0	DecisionTree	5,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MSE	3,0	20,0	DecisionTree	3,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MSE	3,0	20,0	DecisionTree	4,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MSE	3,0	22,0	DecisionTree	2,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MSE	3,0	22,0	DecisionTree	3,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MSE	3,0	22,0	DecisionTree	4,0	3,0	2,0	1,0	900,418	3347010,168	0,402
MSE	3,0	23,0	DecisionTree	2,0	3,0	2,0	1,0	900,418	3347010,168	0,402

Tabela 4 Piores valores das métricas para previsão de tempo de execução

Métrica	Z-score	n-parâmetros	Modelo	k-folds	Profundidade máxima	Minímas por folha	Minímas por separação	MAE	MSE	R^2
R^2	1,0	16,0	DecisionTree	4,0	5,0	5,0	2,0	1008,403	6397283,669	-0,143
R^2	1,222	16,0	DecisionTree	4,0	5,0	5,0	2,0	1008,403	6397283,669	-0,143
MAE	2,333	5,0	GradientBoosting	2,0	3,0	10,0	4,0	1240,092	4854485,919	0,132
MAE	2,556	5,0	GradientBoosting	2,0	3,0	10,0	4,0	1240,092	4854485,919	0,132
MSE	1,0	16,0	DecisionTree	4,0	5,0	5,0	2,0	1008,403	6397283,669	-0,143
MSE	1,222	16,0	DecisionTree	4,0	5,0	5,0	2,0	1008,403	6397283,669	-0,143

Tabela 5 Melhores valores das métricas para previsão de quantidade de nós B&B

Métrica	Z-score	n-parâmetros	Modelo	k-folds	Profundidade máxima	Minímas por folha	Minímas por separação	MAE	MSE	R^2
R^2	3,0	23,0	GradientBoosting	3,0	3,0	5,0	1,0	58067,255	9792419716,155	**0,924**
MAE	3,0	22,0	GradientBoosting	5,0	3,0	4,0	1,0	53044,786	11750414570,11	0,909
MSE	3,0	23,0	GradientBoosting	3,0	3,0	5,0	1,0	58067,255	9792419716,155	**0,924**

Tabela 6 Piores valores das métricas para previsão de quantidade de nós B&B

Métrica	Z-score	n-parâmetros	Modelo	k-folds	Profundidade máxima	Minímas por folha	Minímas por separação	MAE	MSE	R^2
R^2	2,778	15,0	DecisionTree	2,0	3,0	10,0	1,0	168305,981	314475480863,671	-1,435
R^2	2,778	15,0	DecisionTree	3,0	3,0	10,0	1,0	168305,981	314475480863,671	-1,435
R^2	2,778	22,0	DecisionTree	2,0	3,0	10,0	1,0	168305,981	314475480863,671	-1,435
R^2	2,778	23,0	DecisionTree	2,0	3,0	10,0	1,0	168305,981	314475480863,671	-1,435
MAE	2,778	16,0	DecisionTree	2,0	3,0	10,0	2,0	219532,951	279447838838,971	-1,164
MSE	2,778	15,0	DecisionTree	2,0	3,0	10,0	1,0	168305,981	314475480863,671	-1,435
MSE	2,778	15,0	DecisionTree	3,0	3,0	10,0	1,0	168305,981	314475480863,671	-1,435
MSE	2,778	22,0	DecisionTree	2,0	3,0	10,0	1,0	168305,981	314475480863,671	-1,435
MSE	2,778	23,0	DecisionTree	2,0	3,0	10,0	1,0	168305,981	314475480863,671	-1,435

Observa-se que, em muitos casos, há uma correlação entre diferentes métricas, como evidenciado na Tabela 6, onde os piores cenários de R^2 também coincidem com os piores cenários de MSE. Essa associação decorre do fato de que as métricas refletem o comportamento médio do modelo, sendo o R^2 uma medida da quantidade de variância capturada pelo modelo. Tanto MSE quanto MAE elevados revelam a presença de muitos *outliers*, mesmo após a exclusão por Z-score. Tal fenômeno é atribuído à natureza generalista da base de treinamento, que não se concentra em problemas específicos, mas sim problemas MIP de maneira abrangente, resultando em características heterogêneas. De modo geral, o modelo de DT destaca-se, sendo responsável pela maioria das melhores e piores métricas.

Observa-se também que os valores da métrica MAE na Tabela 3 e Tabela 4, de modo geral, são aproximados, o que pode viabilizar a previsão do tempo de execução, dado o conhecimento prévio da média do erro. No entanto, ao examinar a Tabela 3, nota-se que o valor de R^2 é inferior a 0,5, indicando que o modelo não se ajustou adequadamente à base de dados. A previsão do tempo de execução é um desafio, dada a multiplicidade de características internas e externas ao problema que podem influenciar a execução do solver. Portanto, é crucial exercer cautela ao utilizar o modelo para essa finalidade, considerando sempre uma margem de erro na predição dessa característica.

A predição dos nós na árvore B&B apresentou resultados satisfatórios. Conforme evidenciado na Tabela 5, alcançou-se um R^2 superior a 0,9 em todas as métricas. Além disso, observa-se que o conjunto de permutação é o mesmo nos melhores casos para R^2 e MSE, indicando um modelo consistente. O método GB se destacou como o melhor modelo para prever essa característica em todas as métricas, demonstrando robustez para esse problema.

Embora seja comum na literatura o algoritmo de DT ser considerado um regressor de melhor qualidade em comparação com aqueles que utilizam métodos de *ensemble*, é evidente uma tendência de dominação nos resultados, tanto para melhores e piores

valores, para a previsão de tempo de execução. Essa tendência pode ser atribuída a uma série de fatores. O tamanho do conjunto de dados pode ter exercido uma influência significativa na preferência por outros modelos, uma vez que o desempenho do DT tende a ser mais eficaz em conjuntos de dados menores. Além disso, as Árvores de Decisão podem apresentar um desempenho superior quando o conjunto de dados possui características proeminentes, como observado nas características “Quantidade de Variáveis” e “Quantidade de Restrições”.

A capacidade de prever nós revelou-se bastante satisfatória, uma vez que, de modo geral, o número de nós B&B está diretamente relacionado às características da instância e ao método de resolução do solver. Dado que todas as instâncias foram executadas de maneira uniforme, utilizando os parâmetros padrões do solver, as características inerentes aos problemas impactaram diretamente na resolução dos mesmos. Desta forma, os padrões do conjunto de dados passaram a ter uma interpretação distinta, com instâncias correlacionadas umas com as outras por razões além do tempo de execução. Nesse contexto, o modelo de GB destacou-se, possivelmente devido à identificação de novos padrões nas características do problema e à introdução de uma complexidade diferenciada ao conjunto de dados.

6 CONCLUSÃO

Este trabalho dedicou-se à previsão de características de execução de softwares de otimização para a resolução de problemas de programação linear inteira mista. Esses problemas apresentam características lineares e variáveis inteiras, visando maximizar ou minimizar uma função objetivo.

No âmbito deste estudo, foi proposto um modelo de ML para prever o tempo de execução e a quantidade de nós na árvore B&B. Técnicas baseadas em árvore de decisão, como DT, RF e GB, foram utilizadas para construir o modelo preditivo. A técnica K-fold foi adotada para realizar a validação cruzada e o ajuste dos algoritmos preditivos.

Os resultados obtidos revelaram desafios na previsão do tempo de execução, evidenciando um baixo índice de representatividade dos dados, com variação moderada e um erro considerável para esse tipo de previsão. Por outro lado, a previsão da quantidade de nós na árvore B&B mostrou-se satisfatória, apresentando um coeficiente de determinação elevado e erros aceitáveis para esse parâmetro.

Como continuação deste estudo, pode-se sugerir uma análise mais aprofundada de outros modelos preditivos, como redes neurais e sistemas de votação. Esses modelos podem suprir as deficiências observadas nos modelos de árvore para prever o tempo de execução.

Outra proposta de continuidade inclui a utilização de uma base de dados mais extensa para enriquecer o modelo. Uma base com um maior número de amostras evita o subtreinamento dos modelos preditivos e contribui para a melhoria da acurácia do sistema. Sugere-se explorar não apenas outras instâncias além das utilizadas neste trabalho, mas também considerar outras referências de repositórios.

Por fim, considera-se a possibilidade de uma abordagem analítica. Embora mais desafiadoras em termos de concepção, abordagens analíticas proporcionam um maior controle e demandam menos recursos computacionais em comparação com modelos de aprendizado de máquina.

REFERÊNCIAS

AHMED, Nasim *et al.* Runtime prediction of big data jobs: performance comparison of machine learning algorithms and analytical models. **Journal of Big Data**, Springer Science e Business Media Deutschland GmbH, v. 9, 1 dez. 2022. ISSN 21961115. DOI: 10.1186/s40537-022-00623-1.

ALTMAN, Edward I. Predicting FinancialDistress of Companies : Revisiting the Z-Score and ZETA ® Models *. *In:* disponível em: https://api.semanticscholar.org/CorpusID:7423877.

ARDAGNA, D. *et al.* Predicting the performance of big data applications on the cloud. **Journal of Supercomputing**, Springer, v. 77, p. 1321–1353, 2 fev. 2021. ISSN 15730484. DOI: 10.1007/s11227-020-03307-w.

BARRY, Michael; SCHUMANN, René. Strategies for runtime prediction and mathematical solvers tuning. *In:* v. 2, p. 669–676. ISBN 9789897583506. DOI: 10.5220/0007387606690676.

BAZARAA, Mokhtar S.; JARVIS, John J.; SHERALI, Hanif D. **Linear Programming and Network Flows**. Hardcover. [*S. l.*]: Wiley-Interscience, 17 dez. 2004. p. 744. ISBN 978-0471485995.

BERTSIMAS, Dimitris *et al.* **Introduction to Linear Optimization (Athena Scientific Series in Optimization and Neural Computation, 6)**. Hardcover. [*S. l.*]: Athena Scientific, 1 fev. 1997. p. 608. ISBN 978-1886529199.

CHEN, Hung-Cheng; KAO, Yi-Fang. Mirroring Reality: Surreal Tourist Gaze of Chinese Landscape Painting by Artificial Neural Networks. *In:* p. 155–159. DOI: 10.1109/ICKII55100.2022.9983554.

CORMEN, Thomas H. *et al.* **Introduction to Algorithms, fourth edition**. Hardcover. [*S. l.*]: The MIT Press, 5 abr. 2022. p. 1312. ISBN 978-0262046305.

DANTZIG, George B. **Linear Programming and Extensions**. Hardcover. [*S. l.*]: Princeton University Press, 1963. p. 632.

DEMIROVIĆ, Emir *et al.* An Investigation into Prediction + Optimisation for the Knapsack Problem. *In:* 11494 LNCS, p. 241–257. ISBN 9783030192112. DOI: 10.1007/978-3-030-19212-9_16.

F.Y, Osisanwo. Supervised Machine Learning Algorithms: Classification and Comparison. **International Journal of Computer Trends and Technology**, 2017. DOI: 10.14445/22312803/IJCTT-V48P126.

FACELI, Katti *et al.* **Inteligência artificial: uma abordagem de aprendizado de máquina**. [*S. l.*]: LTC, 2021.

FONSECA, George H. G. Integer programming techniques for educational timetabling. **Eur. J. Oper. Res.**, 2017.

FUSHIKI, Tadayoshi. Estimation of prediction error by using K-fold cross-validation. **Statistics and Computing**, Springer, v. 21, p. 137–146, 2011.

GLEIXNER, Ambros *et al.* MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. **Mathematical Programming Computation**, 2021. DOI: 10.1007/s12532-020-00194-3.

GOLDBARG, Marco. **Otimização Combinatória e Programação Linear**. [*S. l.: s. n.*], jan. 2005. ISBN 85-352-1520-4.

GUEDES, Matheus; BOAS, Vilas. **Decision Trees for the Algorithm Selection Problem: Integer Programming Based Approaches**. [*S. l.*].

GUROBI OPTIMIZATION, LLC. **Gurobi Optimizer Reference Manual**. [*S. l.: s. n.*], 2023. Disponível em: https://www.gurobi.com.

HOELTZEL, D A; CHIENG, W H. **Statistical Machine Learning for the Cognitive Selection of Nonlinear Programming Algorithms in Engineering Design Optimization**. [*S. l.*]: American Society of Mechanical Engineers, abr. 1987. DOI: 10.1115/detc1987-0009.

HUTTER, Frank *et al.* Algorithm runtime prediction: Methods evaluation. **Artificial Intelligence**, Elsevier B.V., v. 206, p. 79–111, 1 2014. ISSN 00043702. DOI: 10.1016/j.artint.2013.10.003.

HYNDMAN, Robin John; ATHANASOPOULOS, George. **Forecasting: Principles and Practice**. 2nd. Australia: OTexts, 2018.

IZBICKI, Rafael; SANTOS, Tiago Mendonça dos. **Aprendizado de máquina: uma abordagem estatística**. [*S. l.: s. n.*], 2020. 268 p.

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

KHOOBJOU, E. On hybrid intelligence-based control approach with its application to flexible robot system. **Human-centric Computing and Information Sciences**, 2017. DOI: 10.1186/s13673-017-0086-5.

KUHN, Max; JOHNSON, Kjell. **Applied Predictive Modeling**. Hardcover. [*S. l.*]: Springer, 17 maio 2013. p. 613. ISBN 978-1461468486.

LAND, Ailsa H.; DOIG, Alison G. An Automatic Method for Solving Discrete Programming Problems. *In:* **50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art**. Edição: Michael Jünger. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 105–132. ISBN 978-3-540-68279-0. DOI: 10.1007/978-3-540-68279-0_5.

NORVIG, Peter; RUSSELL, Stuart. **Artificial Intelligence: A Modern Approach, Global Edition**. Paperback. [*S. l.*]: Pearson, 13 maio 2021. ISBN 978-1292401133.

OTCHERE, Daniel Asante *et al.* Application of gradient boosting regression model for the evaluation of feature selection techniques in improving reservoir characterisation predictions. **Journal of Petroleum Science and Engineering**, v. 208, p. 109244, 2022. ISSN 0920-4105. DOI: https://doi.org/10.1016/j.petrol.2021.109244.

PINISETTY, Srinivas *et al.* Predictive runtime enforcement. **Formal Methods in System Design**, Springer New York LLC, v. 51, p. 154–199, 1 ago. 2017. ISSN 15728102. DOI: 10.1007/s10703-017-0271-1.

RIGA, Katrin. Mixed integer programming for dynamic tower crane and storage area optimization on construction sites. **Automation in Construction**, 2020.

SAVIN, G. I. *et al.* Jobs Runtime Forecast for JSCC RAS Supercomputers Using Machine Learning Methods. **Lobachevskii Journal of Mathematics**, Pleiades journals, v. 41, p. 2593–2602, 12 dez. 2020. ISSN 18189962. DOI: 10.1134/S1995080220120343.

SIFAT, Ridwan Islam. ChatGPT and the Future of Health Policy Analysis: Potential and Pitfalls of Using ChatGPT in Policymaking. **Annals of Biomedical Engineering**, 2023. ISSN 1573-9686. DOI: 10.1007/s10439-023-03204-2.

STAHLE, Lars; WOLD, Svante. Analysis of variance (ANOVA). **Chemometrics and Intelligent Laboratory Systems**, v. 6, n. 4, p. 259–272, 1989. ISSN 0169-7439. DOI: https://doi.org/10.1016/0169-7439(89)80095-4.

TALBI, El-Ghazali. Combining metaheuristics with mathematical programming, constraint programming and machine learning. **4OR**, v. 11, p. 101–150, 2 2013. ISSN 1614-2411. DOI: 10.1007/s10288-013-0242-3.

TRUCHET, Charlotte *et al.* Estimating parallel runtimes for randomized algorithms in constraint solving. **Journal of Heuristics**, Springer New York LLC, v. 22, p. 613–648, 4 ago. 2016. ISSN 15729397. DOI: 10.1007/s10732-015-9292-3.

WANG, Qiqi *et al.* Predicting job finish time based on parameter features and running logs in supercomputing system. **Journal of Supercomputing**, Springer, v. 78, p. 18551–18577, 17 nov. 2022. ISSN 15730484. DOI: 10.1007/s11227-022-04582-5.

WOLSEY, L.A. **Integer Programming**. [*S. l.*]: Wiley, 1998. (Wiley Series in Discrete Mathematics and Optimization). ISBN 9780471283669.

WU, Dawen; LISSER, Abdel. A deep learning approach for solving linear programming problems. **Neurocomputing**, Elsevier B.V., v. 520, p. 15–24, fev. 2023. ISSN 18728286. DOI: 10.1016/j.neucom.2022.11.053.

ZHANG, Jiayi *et al.* **A survey for solving mixed integer programming via machine learning**. v. 519. [*S. l.*]: Elsevier B.V., jan. 2023. p. 205–217. DOI: 10.1016/j.neucom.2022.11.024.