UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA DE TRANSPORTES E LOGÍSTICA

JUAN CARLOS GUERRA TRINIDAD

APLICAÇÃO DE META LEARNING PARA ESCOLHA DA MELHOR META-HEURÍSTICA EM PROBLEMAS DE CAIXEIRO VIAJANTE

Joinville
2022

JUAN CARLOS GUERRA TRINIDAD

APLICAÇÃO DE META LEARNING PARA ESCOLHA DA MELHOR META-HEURÍSTICA EM PROBLEMAS DE CAIXEIRO VIAJANTE

Trabalho apresentado como requisito para obtenção do título de bacharel em Engenharia de Transportes e Logística do Centro Tecnológico de Joinville da Universidade Federal de Santa Catarina.

Orientador: Dr. Pablo Andretta Jaskowiak

Joinville
2022

JUAN CARLOS GUERRA TRINIDAD

APLICAÇÃO DE META LEARNING PARA ESCOLHA DA MELHOR META-HEURÍSTICA EM PROBLEMAS DE CAIXEIRO VIAJANTE

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de bacharel em Engenharia de Transportes e Logística, na Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Joinville (SC), 1 de agosto de 2022

Banca Examinadora:

_____

Dr. Pablo Andretta Jaskowiak
Orientador/Presidente

_____

Dr. Benjamin Grando Moreira
1° Membro
Universidade Federal de Santa Catarina

_____

Dra. Silvia Lopes de Sena Taglialenha
2° Membro
Universidade Federal de Santa Catarina

Dedico este trabalho à minha família,
em especial a mi abuelita.

AGRADECIMENTOS

Primeiramente agradeço aos meus pais, pelo apoio que me deram nesta aventura, sempre serão exemplos na minha vida. Agradeço ao professor Pablo pela paciência e dedicação ao longo desse tempo, sempre aprendi muito com você durante todo o curso de engenharia. Agradeço aos membros da banca, por terem aceitado o convite. Agradeço, em especial, às professoras Silvia Taglialenha e Ana Pinto por dar-me a oportunidade de participar em suas iniciações científicas e compartilhar os diferentes conhecimentos comigo. Agradeço à UFSC que me recebeu de braços aberto. Agradeço em especial ao programa PEC-G, por dar a oportunidade de me formar em umas das melhores faculdades federais do Brasil e da América Latina. Agradeço aos meus colegas de trabalho, que me ajudaram a evoluir profissionalmente e acreditam no meu potencial para superar diferentes desafios. Agradeço aos meus amigos, em especial as pessoas que moraram comigo, que sempre estiveram ao meu lado quando eu precisei. Agradeço a meus amigos da Kuase Rep e Neverland, por serem minha família aqui no Brasil. Agradeço a Lana, minha namorada por ter acreditado em mim e me ajudado em todo o processo. Agradeço ao Brasil em geral, por dar-me esta experiência que, sem duvida, ficou marcado na minha vida.

RESUMO

O presente trabalho tem como objetivo empregar e avaliar estratégias de aprendizado de máquina para escolher meta-heurísticas promissoras para o problema do *Traveling Salesman Problem* (TSP), o qual caracteriza-se por ser um problema de otimização combinatorial. Na grande maioria das instâncias de TSP não se sabe qual é a solução ótima. Heurísticas e meta-heurísticas são comumente usadas nos problemas de TSP para encontrar soluções de qualidade em um curto período de tempo. Uma vez que diferentes meta-heurísticas podem produzir soluções de qualidade variada, ocorre que não há uma melhor meta-heurística para todas as instâncias. Desse modo, este trabalho explora o uso de métodos de aprendizagem de máquina para criar uma meta-heurística de aprendizagem (*meta learning*), a fim de identificar quais meta-heurísticas são mais promissoras para solucionar instâncias especificas do TSP, definidas por conjuntos de características (*meta features*). Com a realização dos experimentos, observou-se que os modelos de *meta learning* podem prever com precisão quais meta-heurísticas são mais adequadas para diferentes cenários do TSP. Os resultados obtidos dos experimentos também mostram que os métodos de aprendizado utilizados no modelo tem um impacto importante na qualidade das soluções obtidas.

Palavras-chave: Meta Learning. Aprendizado de Máquina. Meta-Heurísticas. Traveling Salesman Problem.

ABSTRACT

The present work aims to employ and evaluate Machine Learning strategies to choose promising meta-heuristics for the Traveling Salesman Problem (TSP), which is characterized by being a combinatorial optimization problem. In most instances of TSP, the optimal solution is not known. Heuristics and Meta-heuristics are commonly used in TSP problems to find quality solutions in a short period of time. Since different meta-heuristics can produce solutions of varying quality, it turns out that there is no the best meta-heuristic for all instances. Thus, this work explores the use of machine learning methods to create a learning meta-heuristic (Meta-Learning), in order to identify which meta-heuristics are most promising to solve specific instances of the TSP, defined by sets of characteristics (Meta Features). By carrying out the experiments, it was observed that Meta Learning models can accurately predict which meta-heuristics are most suitable for different TSP scenarios. The experimental results also show that the learning methods used in the model have an important impact on the quality of the solutions obtained.

Keywords: Meta Learning. Machine Learning. Meta-Heuristics. Traveling Salesman Problem.

LISTA DE FIGURAS

Figura 1 – Exemplo de Grafo de TSP 12
Figura 2 – Codificação da rota (1, 2, 3, 4, 5, 6) 23
Figura 3 – Exemplo de Cruzamento do Método OX 24
Figura 4 – Exemplo de Fluxo de Trabalho de AM 27
Figura 5 – Árvore de decisão do desempenho acadêmico de estudantes de engenharia 29
Figura 6 – Estrutura da Florestas Aleatórias 30
Figura 7 – Estrutura de uma Rede Neural Artificial (RNA) com arquitetura de MLP 31
Figura 8 – Representação da modelagem de um neurônio artificial. 32
Figura 9 – Representação do classificador de k-vizinhos mais próximos (kNN). 33
Figura 10 – *Meta Learning* para selecionar MHs para o TSP 36
Figura 11 – Distribuição de Meta-Heurísticas 39
Figura 12 – Distribuição de Meta-Heurísticas após aplicação do Método *SMOTE* 40
Figura 13 – K-fold cross-validation 43
Figura 14 – Diagrama de Boxplot no Método kNN - Peso por Distância | k = 3 . . 50
Figura 15 – Diagrama de Boxplot no Método RNA com arquitetura MLP - três camadas ocultas com 25, 50 e 25 neurônios 53
Figura 16 – Diagrama de Boxplot no Método FA com 150 árvores 55

LISTA DE TABELAS

Tabela 1 – Exemplificação de Meta Dados 35
Tabela 2 – Número de instâncias com melhor solução por MH 38
Tabela 3 – Features da base de dados 41
Tabela 4 – Matrix de confusão 44
Tabela 5 – Média dos Resultados do Método kNN - Peso Uniforme | k = 3 . . . 48
Tabela 6 – Média dos Resultados do Método kNN - Peso Uniforme | k = 5 . . . 48
Tabela 7 – Média dos Resultados do Método kNN - Peso Uniforme | k = 7 . . . 48
Tabela 8 – Média dos Resultados do Método kNN - Peso por Distância | k = 3 . 49
Tabela 9 – Média dos Resultados do Método kNN - Peso por Distância | k = 5 . 49
Tabela 10 – Média dos Resultados do Método kNN - Peso por Distância | k = 7 . 50
Tabela 11 – Média dos Resultados do Método RNA com arquitetura MLP 51
Tabela 12 – Média dos Resultados do Método RNA com arquitetura MLP 51
Tabela 13 – Média dos Resultados do Método RNA com arquitetura MLP 52
Tabela 14 – Média dos Resultados do Método Floresta Aleatória com 100 árvores 54
Tabela 15 – Média dos Resultados do Método Floresta Aleatória com 50 árvores 54
Tabela 16 – Média dos Resultados do Método Floresta Aleatória com 150 árvores 54

LISTA DE SIGLAS

ACO	*Ant Colony Optimization*
AGs	Algoritmos Genéticos
AM	Aprendizado de Máquina
FA	Floresta Aleatória
GRASP	*Greedy Randomized Adaptive Search Procedure*
kNN	*k-Nearest Neighbors*
MH	Meta-Heurística
ML	*Meta Learning*
MLP	*Multilayer Perceptron*
NP-Hard	*Non-Deterministic Polynomial-Time Hardness*
RNA	*Redes Neurais Artificiais*
SA	*Simulated Annealing*
SVM	*Support Vector Machines*
TSP	*Traveling Salesman Problem*

SUMÁRIO

1 INTRODUÇÃO 11
1.1 Objetivos 15
1.1.1 Objetivo Geral 15
1.1.2 Objetivos Específicos 15
1.2 Organização do Trabalho 16

2 REFERENCIAL TEÓRICO 17
2.1 Heurísticas 17
2.2 Meta-Heurísticas 18
2.2.1 *Tabu Search* 19
2.2.2 *Simulated Annealing* 20
2.2.3 GRASP 21
2.2.4 Algoritmo Genético 22
2.2.5 Ant Colony Optimization 25
2.3 Aprendizado de Máquina 26
2.3.1 Métodos de Classificação 28
2.3.1.1 Floresta Aleatória 28
2.3.1.2 Redes Neurais Artificiais 30
2.3.1.3 k-Vizinhos Mais Próximos 32
2.4 Meta Learning 34

3 MATERIAIS E MÉTODOS 38
3.1 Dados 38
3.2 Métodos 41
3.2.1 Método de *K-fold cross-validation* 42
3.2.2 Avaliação dos Métodos 43

4 RESULTADOS E DISCUSSÕES 47
4.1 Análise dos resultados por Método 47
4.1.1 Análise das métricas no método kNN 47
4.1.2 Análise das métricas no método de Redes Neurais Artificiais com arquitetura MLP 51
4.1.3 Análise das métricas no método Floresta Aleatória 53
4.2 Análise final dos resultados 56

5 CONCLUSÕES 57

REFERÊNCIAS 59

1 INTRODUÇÃO

Na ciência da computação, matemática e engenharia os grafos são extremamente úteis para representar e analisar problemas diversos. Informalmente, um grafo é uma coleção de vértices, que é acompanhada por um conjunto de arestas que relacionam esses vértices. Quando descrevemos um grafo é comum desenhar os vértices como pontos, ou pequenos círculos no plano, e representar as arestas como linhas que unem esses pontos. As arestas podem ser rotuladas por pesos numéricos considerando-se um grafo ponderado (BONDY; MURTY, 1976).

Matematicamente, os grafos são definidos formalmente da seguinte forma, um grafo $G = (V,A)$ é composto por um conjunto não vazio V de vértices (ou nós) e um conjunto A de arestas (BONDY; MURTY, 1976). Cada aresta de G é um par ordenado de vértices (i,j), quando nos referimos de um grafo orientado, o que significa que possuem arestas com direção. As arestas indicam uma relação unidirecional, em que cada aresta só pode ser percorrida em uma única direção. Entretanto, quando cada aresta de G é um par não ordenado de vértices (i,j) nos referimos a um grafo não orientado, ou seja, possuem arestas que não têm direção. As arestas indicam uma relação de mão dupla, em que cada aresta pode ser percorrida em ambas as direções (GABOW et al., 1986).

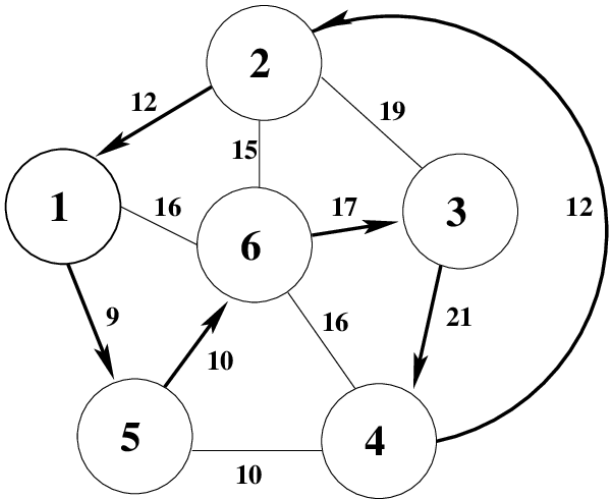
Na teoria dos grafos, um percurso é uma sequência de vértices e arestas que compreendem um grafo. Dois vértices são conectados ou acessíveis se houver um percurso que permita ir de um ao outro; caso contrário, os vértices são ditos desconectados ou inacessíveis (SUTARIA, 2016). Quando um dos conjuntos do grafo é explorado em sua totalidade, os percursos são categorizado como Hamiltonianos, quando ocorre a visita de cada vértice do grafo; Euleriano, quando é visitada cada aresta do grafo. (HILLIER; LEIBERMAN, 2010).

O foco deste trabalho será o problema *Traveling Salesman Problem* (TSP), em português, Problema do Caixeiro Viajante (PCV). O TSP se baseia no percurso hamiltoniano, com a restrição adicional de encontrar o circuito hamiltoniano de menor custo (COOPER; NICOLESCU, 2019). De acordo com Schrijver (2000), o TSP é tradicionalmente descrito como um problema de otimização cujo objetivo é minimizar o custo total observado ao visitar todos os nós de um grafo e voltar ao nó inicial, considerando a restrição de unicidade nas visitações. Sendo custo definido como o peso de uma aresta que conecta dois nós, como pode ser observado na Figura 1.

Todas as arestas que conectam os nós tem um custo. Esse custo associado a uma aresta pode representar no mundo real o tempo de deslocamento ao longo de duas cidades; em outras situações, pode ser o comprimento ou o custo de uma passagem

entre os extremos da aresta, por exemplo. O caminho concluído, que compreende a passagem por todos os nós e retorno ao nó inicial, é conhecido como um *tour* (circuito hamiltoniano fechado). O TSP é um problema de otimização combinatória que devido a sua elevada complexidade, é classificado como *NonDeterministic Polynomial-Time Hardness* (NP-Hard). Devido a esta classificação o uso de técnicas exatas têm se restringido apenas a pequenas instâncias do problema (HILLIER; LEIBERMAN, 2010).

Figura 1 – Exemplo de Grafo de TSP



Fonte: (SARUBBI; LUNA, 2007)

Importante relembrar que o conjunto V consiste em n nós e A representa o conjunto de arestas entre os nós. A distância entre os nós i e j é dada por $c_{i,j}$. Se as variáveis de decisão $x_{i,j}$ são definidas para todo $(i,j) \in A$, de modo que elas assumem o valor 1 se a aresta (i,j) for parte da solução e 0 caso contrário, como é observado na Equação (1) (MEDOZA, 2017) .

$$x_{i,j} = \begin{cases} 1, & \text{se a aresta i à j for parte da solução} \\ 0, & \text{caso contrario,} \end{cases} \tag{1}$$

Assim, temos que o problema de programação linear associado ao TSP pode ser representado pelas Equações (2)-(8), formuladas por Nemhauser e Wolsey (1988).

Minimizar:

$$z = \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{2}$$

Sujeito à:

$$\sum_{(i,j)\in A} x_j = 1 \qquad \forall i \in V \tag{3}$$

$$\sum_{(i,j)\in A} x_i = 1 \qquad \forall j \in V \tag{4}$$

$$(n-1) \geq n x_{ij} + u_i - u_j \qquad \forall (i,j) \in A \tag{5}$$

$$\sum \sum x_{ij} = n \tag{6}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in V \tag{7}$$

$$1 \leq u_i \leq n \qquad \forall (i) \in V \tag{8}$$

A função objetivo é representada na Equação (2) que minimiza a distância total do tour. A Equação (3) é uma restrição indicando que apenas uma aresta pode chegar em cada nó. Enquanto a Equação (4) indica que somente uma aresta pode sair de cada nó. Para garantir que não exista sub-rotas são usadas as Equações (5) e (6), nas quais as variáveis u_i são auxiliares para eliminar as subrotas e representa a ordem em que o nó i e j são visitados na solução. Já as Equações (7) e (8) são utilizadas para definir o tipo das variáveis.

A importância do TSP pode ser verificada em aplicações de áreas como manufatura flexível, sistemas de transportes, problemas de roteamento e comunicação de dados, e nesses casos é amplamente utilizado como *benchmark* (RAMOS, 2001). Segundo Ballou (2006), as melhores rotas podem ser determinadas pela utilização do racional humano, mas em casos nos quais o número de pontos atendidos e restrições são menores. Entretanto, quando temos problemas com dimensões maiores, isto é, muitos vértices e arestas, uma visualização lógica de boas soluções é impraticável.

Assim, se faz necessário o uso de ferramentas computacionais para dar suporte à tomada de decisões. No entanto, há situações nas quais as ferramentas computacionais não podem resolver as instâncias de forma exata, independentemente de sua capacidade. Importante destacar que o uso de ferramentas computacionais que utilizam métodos heurísticos e meta-heurísticos é comum, visto que esses métodos geralmente fornecem soluções de qualidade apropriada com um tempo de processamento aceitável, atendendo necessidades comerciais (BALLOU, 2006).

Os procedimentos heurísticos encontram uma solução viável, mas não necessariamente uma solução ótima, para um problema específico (HILLIER;

LIEBERMAN, 2010). Em alguns casos podemos estabelecer limites para identificar a qualidade da solução, porém, nem sempre é possível ter uma garantia da qualidade (HILLIER; LIEBERMAN, 2010).

Sabe-se que os métodos heurísticos tendem a ser específicos por natureza, isto é, cada método heurístico é geralmente desenvolvido para atender a um tipo de problema específico em vez de uma variedade de aplicações (HILLIER; LIEBERMAN, 2010). Muitas heurísticas têm sido desenvolvidas como a heurística *multi-start* com busca local (SOUSA; OCHI, 2016) ou busca em vizinhança variável (MLADENOVIĆ; HANSEN, 1997).

Considerando a existência de outras técnicas, destacam-se as meta-heurísticas (MHs), que diferem das heurísticas por apresentarem uma estrutura algorítmica aplicável a diferentes problemas de otimização e com um número reduzido de modificações para o tratamento de cada problema específico (ZAPELINI, 2009). Dentre as meta-heurísticas mais comumente empregadas em problemas de TSP destacam-se: Tabu search, Simulated Annealing e Algoritmos Genéticos (KRAMER, 2014).

Mesmo que as MHs não cheguem a um resultado ideal, elas trazem soluções consideradas razoáveis sobre a instância do problema em um período de tempo aceitável. As MHs permitem que a resolução dos problemas seja obtida de forma mais fácil devido à economia de tempo, e assim, auxiliam na geração de soluções melhores (HILLIER; LIEBERMAN, 2010). As pesquisas em MHs geralmente possuem dependência significativa do envolvimento humano, principalmente no seu próprio desenvolvimento (GASTINEAU, 2018). Nos últimos anos vários estudos têm aplicado técnicas que abordam os conceitos de aprendizado de máquina (AM) como uma MH para problemas combinatórios (FONSECA, 2018).

O AM é definido como o campo científico que dá às máquinas a capacidade de aprender sem serem estritamente programadas (GERON, 2019). Os algoritmos de AM são comumente classificados em várias categorias e dependem do tipo de aprendizagem e técnica, assim, por meio de modelos de AM e da base de dados utilizada. (LIAKOS et al., 2018).

Um dos grandes desafios no AM é identificar quando um algoritmo é mais adequado que outro para resolver problemas particulares. As abordagens tradicionais de seleção de algoritmos envolvem, em geral, procedimentos dispendiosos de tentativa e erro, ou requerem conhecimento especializado, que nem sempre é fácil de adquirir (PRUDENCIO; SOUTO; LUDERMIR, 2011). Nesse contexto, o *meta learning* (ML) surge como solução eficaz que busca prever automaticamente o desempenho do algoritmo, auxiliando aos usuários na escolha dos algoritmos mais adequados para lidar com os problemas (BRAZDIL; SOARES; VILALTA, 2009).

Utilizando como base um conjunto de instâncias de um problema, que possuem um conjunto de características que os definem, e uma base de soluções obtidas dos

problemas a partir de diferentes algoritmos, o *meta learning* tem como objetivo mapear o conjunto de características do problema, e com ajuda de algoritmos de AM, faz uma relação com o desempenho dos algoritmos aplicados, criando-se assim um modelo que aprende a identificar o algoritmo mais promissor para resolução do problema (SOUZA, 2010).

Os componentes principais do *meta learning* são: o conjunto de instâncias do problema, o conjunto de atributos que descrevem as características das instâncias do problema, que são definimos como *meta features* ou em português como meta-atributos, e o conjuntos de soluções obtidas a partir de diferentes algorítimos. Esse conjunto de informações é chamado de *meta data* ou em português como meta dados (KANDA; SOARES; BRAZDIL, 2016).

Na literatura encontramos exemplos de aplicação do *meta learning*, como o trabalho realizado por Kanda, Soares e Brazdil (2016), que aplica até 41 *meta features* diferentes, para utilizar a técnica de *meta learning* e produzir uma classificação de várias meta-heurísticas, para assim obter a escolha da meta-heurística com o melhor desempenho em uma instância do TSP. A aplicação de *meta learning* no contexto de instâncias de TSP é definida por Kanda, Soares e Brazdil (2016) de duas maneiras: primeiro, a aprender qual MH escolher e, em segundo lugar, como aprender sobre o TSP por meio de *meta features*.

Neste sentido, este trabalho contempla a aplicação e avaliação de um modelo de *meta learning* em problemas de TSP, incluindo a revisão desse, descrições de MHs usadas e metodologia de AM, bem como apresentação de conjuntos de *meta features* usados para caracterizar as instâncias do TSP, para aplicar a metodologia de *meta learning*. Ao final, é apresentado um cenário experimental com os detalhes dos resultados obtidos e analisados.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Aplicar o *meta learning* para escolha da meta-heurística mais apropriada para resolução de cada TSP.

1.1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

a) Estudar conceitos de aprendizado de maquina e *meta learning* considerando sua utilização para escolha de meta-heurísticas apropriadas à instâncias específicas de TSP;
b) Implementar a metodologia de *meta learning*;

c) Avaliar os resultados sob a perspectiva de diferentes métricas de avaliação.

1.2 ORGANIZAÇÃO DO TRABALHO

Visando alcançar os objetivos propostos, este trabalho é organizado da seguinte maneira. O Capítulo 2 aborda o referencial teórico acerca dos conceitos utilizados no trabalho, o qual consiste em uma breve revisão dos conceitos abordados no trabalho, ou seja, são apresentados os conceitos de Meta-heurísticas e aprendizado de máquina. No Capítulo 3 é apresentada a metodologia adotada para realização das análises e descrição dos dados obtidos de forma detalhada. No Capítulo 4 são expostos os resultados do experimentos utilizando *meta learning*, e por fim, no Capítulo 5 são apresentadas as considerações finais do trabalho, as quais abrangem as conclusões obtidas a partir dos resultados experimentais.

2 REFERENCIAL TEÓRICO

As técnicas utilizadas para resolver o TSP podem ser categorizadas em algoritmos heurísticos, meta-heurísticos e exatos e, destes, as abordagens exatas são usadas apenas para problemas relativamente pequenos (SINGH, 2016).

Em conformidade com os estudos de Bektas (2006), embora o TSP seja conceitualmente simples é difícil obter uma solução boa ou próxima da ótima para instâncias maiores. Ou seja, quando o tamanho do problema é aumentado pelo números de nós, os métodos exatos não são capazes de resolvê-lo. Portanto, requer-se a utilização de métodos heurísticos ou meta-heurísticos para solucioná-lo em um tempo razoável. Isso posto, este capítulo aborda os fundamentos teóricos das heurísticas, meta-heurísticas e os algoritmos de aprendizado de máquina utilizados para a modelagem do *Meta Learning*.

2.1 HEURÍSTICAS

Este termo deriva da palavra grega *heuriskein*, que significa encontrar ou descobrir, e é usado na otimização para descrever uma classe de algoritmos de resolução de problemas (MEDOZA, 2017). Segundo Hillier e Lieberman (2010), as heurísticas são procedimentos desenvolvidos para encontrar as soluções de problemas, usando métodos intuitivos que produzem soluções que podem não ser ótimas, mas viáveis, dado um prazo limitado.

No contexto científico, a otimização é o processo de tentar encontrar a melhor solução possível para um determinado problema. Em um problema de otimização existem diferentes soluções e um critério para discriminá-las. Precisamente, esses problemas podem ser definidos por encontrar o valor de algumas variáveis de decisão para as quais uma determinada função objetivo (medida de desempenho) atinge seu valor máximo ou mínimo (MEDOZA, 2017).

Segundo Bodin (1983), as heurísticas podem ser classificadas em três categorias: as de construção, as de melhoria e as compostas (de construção mais melhoramento). As heurísticas de construção são heurísticas que constroem uma solução completa do zero e pela adição sequencial de componentes a uma solução parcial, e a heurística é finalizada quando a solução esta completa. As heurísticas de melhoria começam com uma solução completa e depois tentam melhorar ainda mais a solução atual por meio de buscas locais no entorno da vizinhança.

No campo de TSP podemos definir a heurística de construção como heurística que utiliza de regras determinadas para inserção de nós na rota que está em construção. Nas heurísticas de melhoria utiliza-se uma rota inicial já construída, com o objetivo de

melhorar a solução da rota trazendo mudanças na rota para melhorar o custo da rota total (ISMAIL, 2019).

No âmbito geral do TSP existem várias heurísticas implementadas para encontrar uma solução viável do problema, um método frequentemente utilizado é a heurística do vizinho mais próximo, que é uma técnica amplamente utilizada em instâncias do TSP. No entanto, isso não significa que não possa ser útil em outras disciplinas (MICO, 1996). O vizinho mais próximo é uma heurística construtiva de fácil implementação. O principal objetivo é determinar uma rota como solução do problema de TSP. A solução começa em um nó aleatório e depois visita o nó mais próximo do nó inicial. Após, ele visita o nó não visitado mais próximo e repete esse processo até ter visitado todos os nós, no final, ele retorna ao nó inicial. Os passos do algoritmo são os seguintes (ROSENKRANTZ; STEARNS; LEWIS, 1977):

1. Selecionar um nó aleatório.
2. Encontrar o nó não visitado mais próximo e selecionar.
3. Resta algum nó não visitado? Se sim, retornar para o passo 2.
4. Retornar para o nó inicial.

Essa metodologia é usada inicialmente para encontrar uma rota completa entre os nós de origem e destino, para servir de referência e posteriormente comparar com outros métodos (OCAMPO; SANTA; GRANADA, 2013).

2.2 META-HEURÍSTICAS

Osman e Kelly (1996) definem os procedimentos meta-heurísticos como uma classe de métodos aproximados que são projetados para resolver problemas difíceis de otimização combinatória, nos quais as heurísticas clássicas não são eficazes. As meta-heurísticas fornecem uma estrutura geral para a criação de novos algoritmos híbridos, combinando diferentes conceitos derivados da evolução biológica e mecanismos estatísticos (HILLIER; LIEBERMAN, 2010). Fazendo uma comparação, as heurísticas são utilizadas na busca de soluções para um problema determinado, enquanto as MHs possuem uma abrangência maior, ou seja, são capazes de encontrar soluções em problemas mais variados (BARBOZA, 2005).

Ao adentrar à definição de MHs no âmbito de TSP, Santos (2016) as descreve como métodos aplicados para encontrar resoluções de problemas de otimização combinatória/discreta, podendo dizer ainda que as MHs apresentam soluções de qualidade variada para vários cenários de TSP. De acordo com Gastineau (2018), as MHs são descritas ainda como solucionadores de aproximação simples de um subconjunto específico de instâncias de TSP. No entendimento desse autor, mesmo que a maioria das MHs encontre um resultado razoavelmente preciso em um curto espaço de tempo, seu desempenho é ditado pelo tipo de problema ao qual se aplica.

Considerando as definições acima, as meta-heurísticas foram escolhidas para minimizar o custo em relação as instâncias do TSP. Assim, neste trabalho foram escolhidas as seguintes MHs, as quais são detalhadas logo abaixo: Tabu Search (GLOVER, 1986), Simulated Annealing (KIRKPATRICK; VECCHI, 1983), Greedy Randomized Adaptive Search Procedure (GRASP) (FEO; RESENDE, 1995), Algoritmos Genéticos (XING; CAI, 2008) e Ant Colony Optimization (DORIGO; GAMBARDELLA, 1997).

2.2.1 *Tabu Search*

A *Tabu Search*, ou Busca Tabu em português, é um método de otimização iterativo, proposto por Glover (1986), relativamente simples, que se move da solução atual para outra, na "vizinhança"da solução atual (BIANCHI; GUTJAHR, 2009). Essa meta-heurística se assemelha aos métodos gulosos comuns que segundo Bang-Jensen, Gutin e Yeo (2004) são algoritmos que abordam a resolução de um problema, selecionando a melhor opção disponível em cada estágio, e não se preocupa se o melhor resultado atual trará o resultado ideal geral.

No entanto, o *Tabu Search* em vez de sempre optar pela melhor solução próxima, possivelmente ficando preso em um mínimo local, usa a memória para escolher estrategicamente uma nova solução ideal contida na vizinhança da solução atual (GLOVER; LAGUNA, 1997). Essa memória registra uma quantidade fixa de movimentos feitos nas iterações recentes, impedindo a revisitação desse movimentos. Essa memoria é conhecida como lista “tabu” (GLOVER; WERRA, 1993). Ou seja, a ideia básica da *Tabu Search* é penalizar movimentos que levam a solução para espaços de busca previamente visitados, também conhecidos como tabu. Nesse sentido, pode-se dizer que há um certo aprendizado e que a busca é inteligente (MEDOZA, 2017).

Em resumo, o *Tabu Search* é uma combinação de busca local com um mecanismo de memória de curto prazo. Seus elementos-chave são a lista tabu que restringe a busca classificando determinados movimentos como proibidos, para evitar cair em soluções geradas recentemente e as regras de parada, usando um critério de parada, como um número fixo de iterações, ou um número fixo de iterações consecutivas sem uma melhoria no valor da função objetivo (HILLIER; LIEBERMAN, 2010).

Para implementar um algoritmo de *Tabu Search* para resolver um problema do TSP, basicamente, precisamos ter uma solução inicial e ela pode ser gerada aleatoriamente. Ao longo do uso do algoritmo *Tabu Search* para o TSP, a seleção da solução viável inicial é um dos passos significativos para a obtenção de uma solução aceitável (BASU, 2012).

O *Tabu Search* no TSP se move de uma solução para próxima em busca de uma solução viável. O método de passar de uma solução para outra é descrito por

um conjunto de regras e é chamado de movimento. O conjunto de todas as soluções que podem ser alcançadas a partir de uma dada solução usando um movimento pré-especificado é chamado de vizinhança da solução (BASU, 2012). O movimento utilizado pode ser o de 2-opt que envolve a remoção de duas arestas não adjacentes de um rota existente e duas novas arestas são inseridas conectando os nós de início e fim das arestas excluídas para criar uma nova rota sem criar sub-rotas (BASU, 2012). Esse tipo de movimento tem sido usado no contexto de TSP e problemas relacionados (BASU, 2012).

2.2.2 *Simulated Annealing*

Simulated Annealing (SA) é uma meta-heurística inspirada no processo de aquecimento de metais desenvolvido por Metropolis, em 1953 (YOUSSEF; SAINT, 2001), ou seja, quando o metal é levado a temperaturas altas ocorre a fusão, procedimento esse que é definido como *annealing* e ocorre a circulação livre de átomos; com o controle do resfriamento, os átomos acabam se organizando de um modo ordenado e estável, resultando em uma estrutura uniforme, diminuindo eventuais defeitos do material. Se ocorrer o resfriamento inesperadamente, a estrutura pode apresentar instabilidade, assim, Metropolis (1953) para solucionar possíveis problemas de recozimento apresentou um algoritmo desse processo, pelo qual simulou alterações de energia com a diminuição de temperatura até entrar em um estágio estável (SALAMON; SIBANI; FROST, 2002).

Segundo o Silva (2012), o SA é um método baseado em uma analogia com a termodinâmica de como um material é aquecido a uma alta temperatura e depois resfriado lentamente até atingir sua configuração de rede cristalina mais regular possíve, ou seja, seu estado mínimo de energia de rede. Portanto, estará livre de defeitos em seu material se o cronograma de resfriamento for suficientemente lento, e a configuração final resulta em um sólido com integridade estrutural superior (IGNACIO; GALVãO, 2000) (SILVA, 2012). De um modo geral, o SA usa exatamente essa abordagem para encontrar o menor valor de uma função de erro, a qual é análoga à energia no caso termodinâmico (BAILER-JONES; BAILER-JONES, 2002). A solução de vários problemas de otimização combinatória pode ser encontrada pela utilização da meta-heurística SA (RODRIGUES et al., 2004).

Ao implementar o algoritmo de SA para um problema de otimização é necessário selecionar um cronograma de temperatura apropriada. Em razão da analogia com o processo de termodinâmica que ocorre na vida real, T representa uma temperatura no algoritmo de SA. Esse cronograma precisa especificar o valor inicial, sendo um valor alto de T, bem como os valores seguintes progressivamente menores (HILLIER; LIEBERMAN, 2010). Além disso, é necessário definir o número de iterações $MaxIter()$ que serão realizadas a cada valor de T. Outro parâmetro que é

utilizado em cada iteração é o Z_c que é a solução experimental atual, e o parâmetro Z_n é definido como o candidato de ser a próxima solução experimental (HILLIER; LIEBERMAN, 2010), que é proveniente da aplicação do procedimento 2-opt como vizinhança. Os parâmetros aqui definidos são fatores de extrema importância para que se obtenha maior eficiência no algoritmo de SA (HILLIER; LIEBERMAN, 2010).

O algoritmo a seguir mostra a fundamentação do Método SA aplicado para um problema de minimização como o do TSP (OLIVEIRA; PEREIRA; SOUZA, 2018).

Algoritmo 1 Algoritmo do método *Simulated Annealing*

$Z_c \leftarrow$ Solução Inicial
$Custo_{Atual} \leftarrow Custo(Z_c)$
$T \leftarrow T_{Inicial}$
enquanto $T > T_{Final}$ **faça**
  para $i = 1 \rightarrow MaxIter(T)$ **faça**
    $Z_n \leftarrow Vizinho(Z_c)$
    $Custo_{Novo} \leftarrow Custo(Z_n)$
    $\Delta Custo \leftarrow Custo_{Atual} - Custo_{Novo}$
    $x \in |0,1|$
    se $\Delta Custo \leq 0$ ou $e^{\frac{\Delta Custo}{T}} > x$ **então**
      $Z_c \leftarrow Z_n$
      $Custo_{Atual} \leftarrow Custo_{Novo}$
    fim se
  fim para
  $T \leftarrow T_{Inicial+1}$
fim enquanto
Retornar Z_c

Cada interação do SA tem-se como objetivo buscar movimentos da Z_c para encontrar uma vizinhança da solução com melhor resultado (HILLIER; LIEBERMAN, 2010). Segundo Hillier e Lieberman (2010), o método usual de implementação para determinar se a vizinhança da solução será aceito é comparar com um numero aleatório entre 0 e 1, se o número aleatório for menor que $e^{\frac{\Delta Custo}{T}}$ será aceito, caso contrário, será rejeitado.

2.2.3 GRASP

O *Greedy Randomized Adaptive Search Procedures* (GRASP) proposto por Feo e Resende (1995), corresponde a um processo iterativo que contempla duas fases, a de construção, na qual uma solução viável é produzida de forma iterativa, considerando adição um nó em cada etapa. Em cada iteração a escolha do próximo nó a ser adicionado à solução parcial é determinada por uma função gulosa. Esta função mede o benefício de adicionar cada um dos nós à solução construída e escolher o melhor. Observe que essa medida é míope no sentido de que não leva em consideração

o que acontecerá em iterações sucessivas ao fazer uma escolha (MEDOZA, 2017).

Na sequência, aplica-se uma fase de busca local, na qual se busca um ótimo local na vizinhança da solução construída. Nessa fase geralmente é aplicado um procedimento de busca local simples. O GRASP é baseado na execução de várias iterações da fase 1 e fase 2 , portanto, segundo o Festa e Resende (2002), não é benéfico para o método se debruçar muito na melhoria de uma determinada solução para ao final manter a melhor solução como resultado (FESTA; RESENDE, 2002). Cumpre destacar que o GRASP não utiliza históricos para realizar os processos de busca, sendo capaz de manter armazenada uma ou até mesmo diversas soluções considerando sempre as melhores. De um modo geral, o GRASP não é complexo, já que pode ser incorporado com outras formas de busca (FREDDO; BRITO, 2007).

A implementação do algoritmo utilizado para resolução dos problemas de TSP no trabalho é descrito abaixo.

Algoritmo 2 Algoritmo do método GRASP

```
Melhor Solução ← ∅
para i = 1 → Numero de iterações faça
    Solução Atual ← Construcao()
    Solução Atual ← LocalSearch(Solução Atual)
    se Solução Atual < Melhor Solução então
        Melhor Solução ← Solução Atual
    fim se
fim para
Retornar Melhor Solução
```

O primeiro procedimento, $Construcao()$, leva em consideração todos os nós disponíveis para inserí-los nas rotas. O procedimento $LocalSearch()$ é uma variante do procedimento 2-opt, onde uma rota e dois nós da rota são selecionados para troca. A alteração é aceita desde que a distância total da rota seja diminuída.

2.2.4 Algoritmo Genético

O algoritmo genético (AG) proposto por Holland (1973), reconhecido como um método de otimização poderoso e amplamente aplicável, especialmente para problemas de otimização global e problemas *NP Hard*, e foi inspirado nos estudos de Darwin sobre seleção natural e genética (GOLDBERG, 1988). Em suma, o algoritmo genético é um método de busca heurística adaptativa baseado em genética de populações, que não foi criado para solucionar um problema específico, e sim padronizar a adaptação da mesma forma que ocorre na seleção natural. Em AG é possível ter uma diversidade de soluções para um problema que pode evoluir conforme operadores genéticos de mutação, cruzamento e seleção, guiados por probabilidades com base em teorias biológicas, apresentando soluções melhores conforme o andamento de um processo

evolutivo (BARBOZA, 2005).

O AG compreende um conjunto de soluções viáveis ($populacao$) com cada solução correspondendo a um cromossomo. Essas soluções ($pais$) são selecionadas da população de acordo com sua aptidão (valor da função objetivo), e são combinadas para formar novas soluções ($filhos$) (BETTINGER et al., 2002). Este processo é repetido até que um critério de parada (por exemplo, número de gerações, melhoria da melhor solução ou homogeneidade de soluções) seja concluído. O AG tem dois operadores básicos de geração de soluções/população, o crossover (($Cruzamento()$) e a $Mutacao()$. Uma rotina de crossover denota o lugar onde dois pais são divididos, e então recombinados para formar descendentes, permitindo que genes benéficos entre dois pais diferentes sejam combinados em seus descendentes e, esperançosamente, produzam melhores soluções (HOLLAND, 1973), (WOZNIAK; POLAP, 2015). A mutação, geralmente aplicada de maneira aleatória, fornece variação e ocasionalmente introduz material benéfico nos cromossomos filhos (DAVIS, 1987), (MITCHELL, 1998).

Podemos ressaltar que os AGs para qualquer problema frequentemente usam métodos de codificação, que são métodos mais apropriados de representação de soluções, um exemplo dos métodos são as cadeias de dígitos binários, que podem facilitar a geração de filhos e criação de mutações. O desenvolvimento de uma estrutura de codificação apropriada para o problema de TSP é fundamental para o desenvolvimento de um AG eficaz para qualquer aplicação (HILLIER; LIEBERMAN, 2010).

No algoritmo proposto no trabalho, é aplicada a geração aleatória $GerarPopulacaoIncial()$ para gerar a população inicial de rotas. As rotas criadas são codificadas por uma matriz de números inteiros representando o sucessor e o predecessor de cada nó, como é ilustrado na Figura 2.

Figura 2 – Codificação da rota (1, 2, 3, 4, 5, 6)

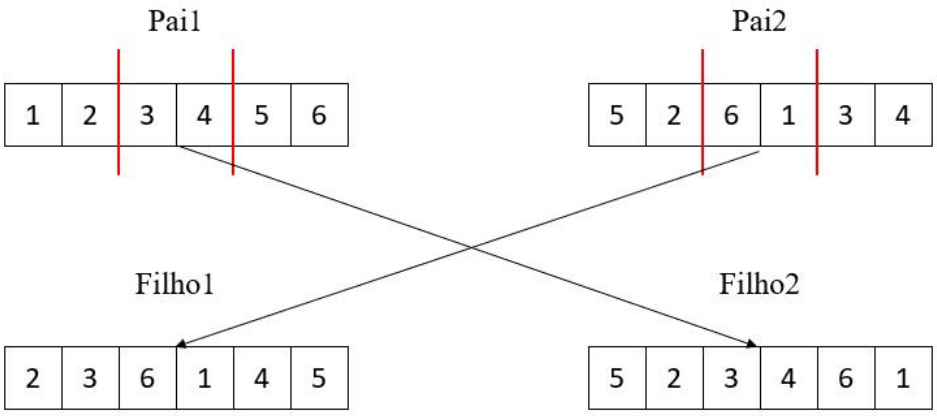
1	2	3	4	5	6

Fonte: Autor (2022)

Após a seleção de dois pais pelo método de $Selecao()$, ocorre o $Cruzamento()$. O $Cruzamento()$ é um operador que cruza os dois pais (cromossomos) chamados Pai1 e Pai2 para produzir dois descendentes (soluções) chamados Filho1 e Filho2. Neste trabalho, é escolhido como operador de $Cruzamento()$ o método *Ordered Crossover* (OX).

O método OX seleciona dois pontos de corte aleatórios dividindo o Pai1 e Pai2 em três seções: esquerda, médio e direita. Após isso, o Filho1 herda a seção do médio do Pai2, e sua seção da esquerda e direita é preenchido, a partir do Pai1, é copiado os números inteiros não utilizados restantes do Pai 1 para o Filho1. Depois é repetido os mesmo passos para o Filho2 com o papel dos pais invertido. Um exemplo do método OX é ilustrado na Figura 3.

Figura 3 – Exemplo de Cruzamento do Método OX



Fonte: Autor (2022)

Para o operador de $Mutacao()$ é aplicado a heurística 2-opt para TSP, explicado anteriormente neste trabalho. O pseudocódigo do AG é descrito abaixo.

Algoritmo 3 Algoritmo Genético

```
População ← GerarPopulacaoInicial()
Melhor Solução ← MelhorSolucao(População)
enquanto i < Máximo Numero de Gerações faça
    Pais ← Selecao(População)
    Filhos ← Cruzamento(Pais)
    Filhos ← Mutacao(Filhos)
    se MelhorSolucao(Filhos) < Melhor Solução então
        Melhor Solução ← MelhorSolucao(Filhos)
    fim se
    i ← i + 1
    Atualizar(População)
fim enquanto
Retornar Melhor Solução
```

2.2.5 Ant Colony Optimization

O *Ant Colony Optimization* (ACO), ou otimização por colônia de formigas em português, proposta por Dorigo e Gambardella (1997), originou-se pela observação de colônias de formigas reais, que posteriormente foram incorporadas as técnicas de busca local (DORIGO; GAMBARDELLA, 1997). A observação das formigas teve como objetivo descobrir como esses insetos conseguiam encontrar o caminho mais curto até o local da fonte de alimento, ocasião em que se descobriu uma substância química, o feromônio, a qual permite que as formigas se comuniquem entre elas; ao percorrer a rota, elas liberam essa substância a fim de comunicar para as outras o caminho que deve ser seguido (DORIGO; GAMBARDELLA, 1997). Um importante e interessante comportamento das formigas é a capacidade delas encontrarem o menor caminho entre uma fonte de comida e o seu ninho pelo feromônio deixado na trilha (DORIGO; STUTZLE; BIRATTARI, 2006). Assim, o ACO é uma meta-heurística inspirada no comportamento e seguimento de trilhas de feromônios das formigas. Formigas artificiais em ACO são procedimentos de construção de soluções estocásticas que constroem soluções possíveis para a instância de um problema, explorando informações de um feromônio artificial (DORIGO; STUTZLE, 2019).

A base do ACO para resolver o TSP consiste expressar o caminho percorrido por cada formiga na colônia de formigas como solução do problema, e todos os caminhos de toda a colônia de formigas são um espaço de solução do problema (XUAN-SHI; YUN, 2021). As formigas liberam mais feromônio em um caminho curto e evaporam lentamente, portanto, a concentração de feromônio é alta. À medida que o tempo avança, mais formigas escolhem o caminho e a concentração de feromônio aumenta gradualmente (XUAN-SHI; YUN, 2021). Finalmente, toda a colônia de formigas será concentrada no caminho ótimo sob a ação do mecanismo de *feedback* positivo, e esse caminho é a solução ótima para o problema (XUAN-SHI; YUN, 2021).

As formigas artificiais se movem aplicando uma política de decisão local estocástica baseada em dois parâmetros, chamados trilhas de feromônio τ e visibilidade (ou atratividade) η (PACIELLO et al., 2022). As trilhas de feromônios refletem que os estados mais visitados são altamente desejáveis, implementando assim um processo autocatalítico ou de *feedback*, enquanto a visibilidade reflete que estados menos custosos, avaliados de acordo com os objetivos a serem otimizados, também são desejáveis (PACIELLO et al., 2022). A política de transição baseia-se em atribuir a probabilidade de passar do estado i para o estado j de acordo com a Equação 9:

$$p_{i,j} = \begin{cases} \frac{\tau_{i,j}^{\alpha}\eta_{i,j}^{\beta}}{\sum_{x \in J}\tau_{i,x}^{\alpha}\eta_{i,x}^{\beta}} & \text{se} j \in J_i \\ 0 & \text{caso contrário,} \end{cases} \tag{9}$$

onde J_i representa a vizinhança de estados alcançáveis a partir do estado i,

α e β são parâmetros definidos que refletem a importância relativa dos feromônios e visibilidade, respectivamente.

Ao chegar em uma solução ou durante sua construção, a formiga avalia a solução e modifica as trilhas de feromônios nos componentes de uma matriz de feromônios τ armazenando assim o conhecimento de áreas já exploradas (PACIELLO et al., 2022). Esta informação de feromônio irá guiar a busca por futuras formigas. O algoritmo também pode incluir um processo de evaporação da trilha de feromônio. A atualização e evaporação dos feromônios é feita de acordo com a seguinte equação:

$$\tau_{i,j} = (1 - \rho)\tau_{i,j} + \rho\Delta\tau \tag{10}$$

onde 0 < ρ < 1 é uma constante que representa o coeficiente de evaporação e $\Delta\tau$ é a quantidade de feromônio depositado pelas formigas que percorreram o caminho (i,j) (SMARANDACHE, 2017). O pseudocódigo do ACO é descrito abaixo.

Algoritmo 4 Algoritmo do método ACO

enquanto $CondicaoParada() != True$ **faça**
  para $Formiga = 1 \rightarrow$ Numero de Formigas **faça**
    $x \leftarrow ConstruirSolucao()$
    $\eta_{i,j} \leftarrow AvaliarSolucao(x)$
    $\tau_{i,j} \leftarrow AtualizarFeromonios(x)$ ▷ *de acordo com a equação (10)*
  fim para
fim enquanto
Melhor Solução $\leftarrow Melhor(\eta_{i,j})$
Retornar Melhor Solução

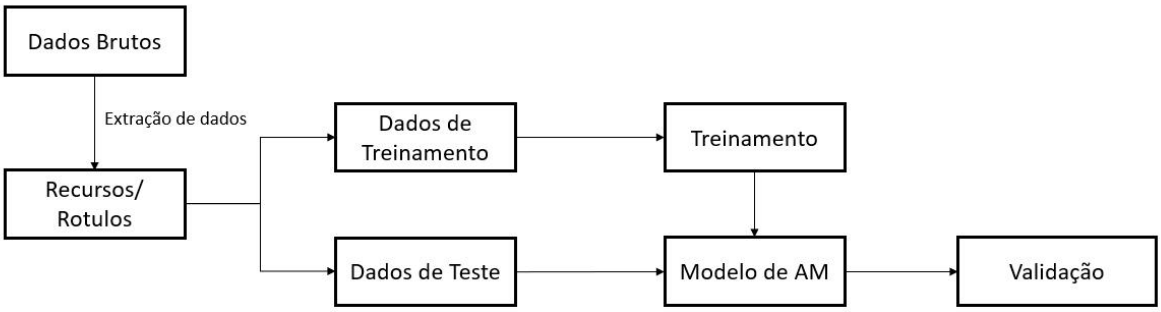
2.3 APRENDIZADO DE MÁQUINA

O aprendizado de máquina (AM) pode ser definido como a ciência da programação de computadores, para que esses aprendam com os dados (GERON, 2019). O AM é um ramo da ciência da computação que visa permitir que os computadores “aprenderam” sem serem programados diretamente (SAMUEL, 1959).

No entendimento de Mitchell (1997), O AM é uma área que estuda como construir programas de computador que melhoram seu desempenho em algumas tarefa graças à experiência. Nesse sentido, o AM é baseado em ideias de várias disciplinas, como inteligência artificial, estatística e probabilidade, teoria da informação, psicologia e neurobiologia, teoria de controle e complexidade computacional.

A Figura 4 mostra o fluxo de trabalho geral para as abordagens de aprendizado de máquina. O fluxo começa com um conjunto de dados brutos no qual é realizado processo de extração de dados que refere-se ao processo de transformar dados brutos em recursos numéricos, que podem ser processados, preservando as informações no

Figura 4 – Exemplo de Fluxo de Trabalho de AM



Fonte: Autor (2022)

conjunto de dados original (BASAVARAJU et al., 2019). O conjunto de dados extraído em alguns casos pode conter rótulos de classe que são os atributos que desejamos prever, e o conjunto de dados de entrada que são os atributos dos dados chamados de recursos, são os dados utilizados para gerar a saída. Depois que os recursos e os rótulos de classe são extraídos, os dados são divididos em dois conjuntos: o conjunto de dados de treinamento e o conjunto de dados de teste. O conjunto de treinamento é usado para treinar o algoritmo e desenvolver o modelo de AM a partir de dados. O conjunto de dados de teste é então usado para validar o desempenho do treinamento (BASAVARAJU et al., 2019).

Segundo o Mitchell (1997), para utilizar a abordagem de aprendizagem, deve-se considerar uma série de decisões que incluem a seleção do tipo de treinamento, a função objetivo a ser aprendida e o algoritmo para aprender a partir de exemplos de treinamentos. Um AM pode ser utilizado para diversos fins e áreas. Segundo Geron (2019) o AM por exemplo é usado para filtro de spam de e-mails, mas não se limita a esse, tem-se também os sistemas que recomendam anúncios, detectam fraude, etc.

Conforme descreve Geron (2019), existem diversos tipos de sistemas de AM que podem ser divididos nas duas vertentes principais, conforme a quantidade e a forma de supervisão que recebem: não supervisionado e supervisionado.

No primeiro, chamado de aprendizado não supervisionado, não há rótulos, ou seja, o sistema precisa identificar as soluções sem a ajuda de um rótulo. O modelo agrupará os dados de acordo com semelhanças, encontrando estruturas e padrões ocultos em dados não rotulados (GAMARRA; VERA, 2018). O aprendizado não supervisionado usa uma variedade de algoritmos para ajustar os dados em grandes grupos, *clusters* e associações, uma aplicação central da aprendizagem não supervisionada é no campo da estimativa de densidade em estatística, como encontrar a função de densidade de probabilidade (JORDAN; BISHOP, 2004). Outro exemplo em que é usado é no agrupamento de informações coletadas sobre os usuários em um

site ou em um aplicativo e para que o modelo detecte diversas características que eles têm em comum (GAMARRA; VERA, 2018).

Já no aprendizado supervisionado, o principal objetivo é treinar um modelo preditivo a partir de dados previamente rotulados. O termo supervisionado refere-se ao fato de utilizarmos um grupo de amostras cujo dado de saída já são conhecidos (GERON, 2019). Dependendo da natureza da variável que queremos prever (variável dependente) teremos um problema de classificação ou regressão. Quando a variável dependente ou a ser predita for categórica, trata-se de um problema de classificação, por exemplo, podemos usar o AM para classificar e-mails como *spam* ou não *spam*, ou classificar imagens em diferentes categorias, como animais, plantas ou objetos.

Quando a variável for contínua estaremos diante de um problema de regressão (GERON, 2019). Um dos usos mais comuns da regressão é prever as vendas futuras de um produto. Isso pode ser feito analisando dados de vendas anteriores para identificar padrões. Por exemplo, se os dados de vendas anteriores mostrarem que as vendas de um produto aumentam quando a temperatura está acima de um determinado limite, o algoritmo de regressão pode ser usado para prever vendas futuras do produto com base na temperatura atual (SALTOS; VILLACIS, 2022).

2.3.1 Métodos de Classificação

Dentre os modelos de AMs existentes, este trabalho foca no aprendizado supervisionado, que é aquele em que existem os chamados rótulos e podem generalizar o conhecimento assimilado para um conjunto de dados desconhecidos. Partindo do aprendizado supervisionado, encontramos dois tipos de algoritmos, quais sejam: classificação e regressão (ALPAYDIN, 2010). Os métodos de classificação abordados incluem Floresta Aleatória, Redes Neurais Artificiais e k-Vizinhos Mais Próximos, os quais serão discutidos a seguir.

2.3.1.1 Floresta Aleatória

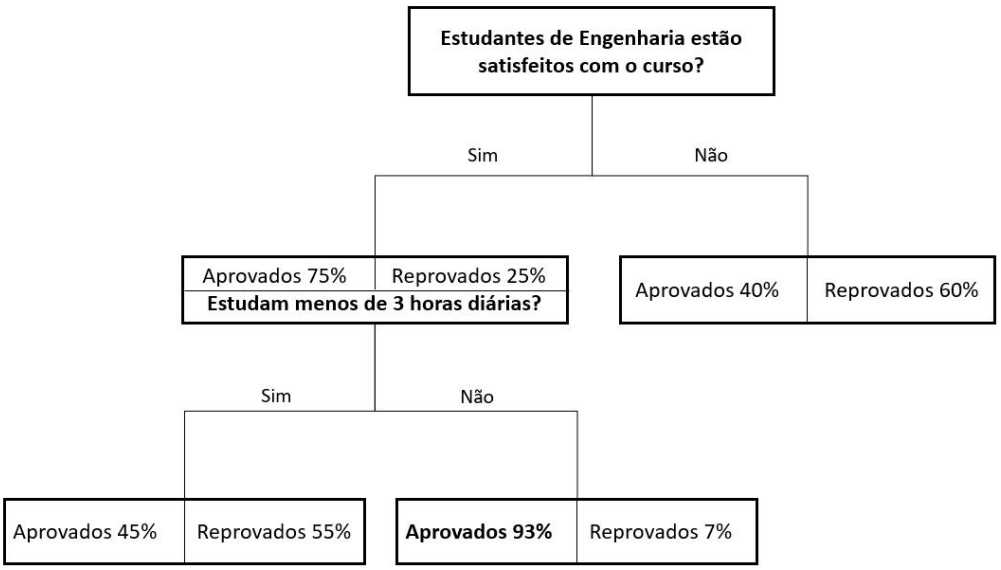
O classificador de floresta aleatória (FA), ou *Random Forest* em inglês, é um método AM largamente utilizado em problemas de classificação. É um algoritmo de aprendizagem supervisionada que utiliza o *ensemble learning*, uma técnica que combina um conjunto de modelos elaborados para obtenção das melhores combinações, para uma tarefa de classificação (SANTOS, 2020b). Assim, o FA usa o *ensemble* de árvores de decisão, que em conjunto encontram melhores predições do que quando o trabalho é feito por apenas uma árvore (DIETTERICH, 2002).

A árvore de decisão é uma técnica que prepara, investiga e explora dados para descobrir novas informações. É solução para problemas de previsão, classificação e segmentação (BERLANGA; RUBIO; VILÀ, 2013). As árvores de decisão criam um

modelo de classificação baseado em fluxogramas. Eles classificam os casos em grupos ou preveem valores de uma variável dependente com base em valores de variáveis independentes (BERLANGA; RUBIO; VILÀ, 2013).

Para exemplificar graficamente o que é uma árvore de decisão, os autores Berlanga, Rubio e Vilà (2013) propõem o exemplo ilustrado na Figura 5. O objetivo é saber quais alunos matriculados no primeiro ano de engenharia têm maior probabilidade de aprovar em todas as disciplinas e quais características estão associadas a esse sucesso acadêmico. Nesse caso, a variável dependente é o desempenho acadêmico no primeiro ano. Depois de inserir os dados necessários no programa, ele nos retorna um diagrama que nos permite verificar que a satisfação com o curso e as horas de estudo por dia são as variáveis independentes que determinam principalmente o sucesso acadêmico de um aluno.

Figura 5 – Árvore de decisão do desempenho acadêmico de estudantes de engenharia

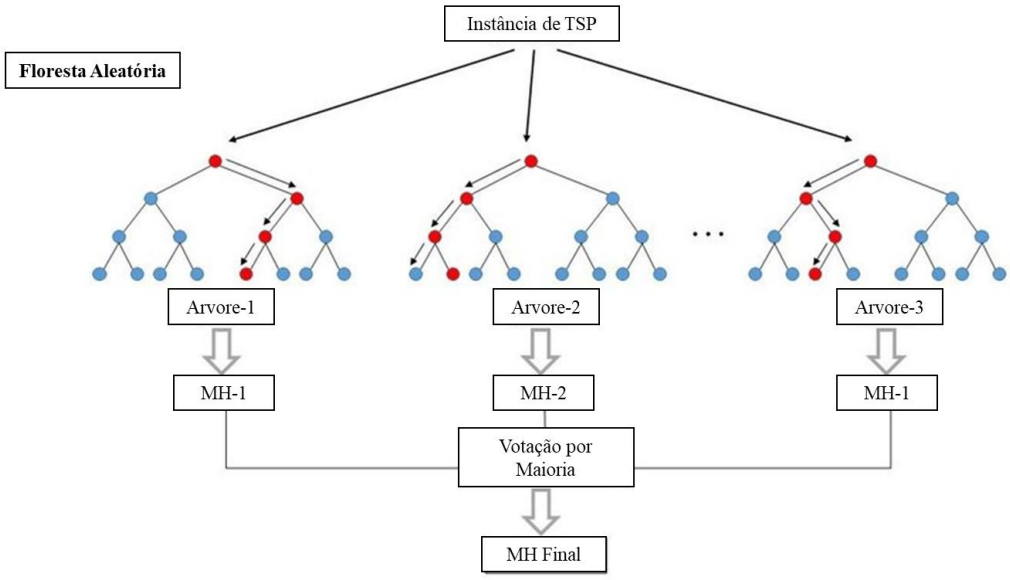


Fonte: Adaptado de (BERLANGA; RUBIO; VILÀ, 2013)

Como podemos observar na estrutura da FA ilustrada na Figura 6, o algoritmo tem uma estrutura similar a um fluxograma, com várias arvores que são constituídas com vários “nós” onde uma condição é verificada.

As arvores são obtidas a partir do mesmo conjunto de dados, aplicando o método de *ensemble learning* chamado de *Bootstrap Aggregation* (ou *Bagging* para abreviar), no qual consiste em treinar cada árvore da floresta com um subconjunto dos dados de treinamento (GERON, 2019). A partir desse ponto descrevemos uma característica muito importante do método, esse subconjunto de treinamento é selecionado aleatoriamente, por essa razão o nome é “Florestas Aleatórias” (GERON,

Figura 6 – Estrutura da Florestas Aleatórias



Fonte: Adaptado de (DIMITRIADIS; LIPARAS, 2018)

2019).

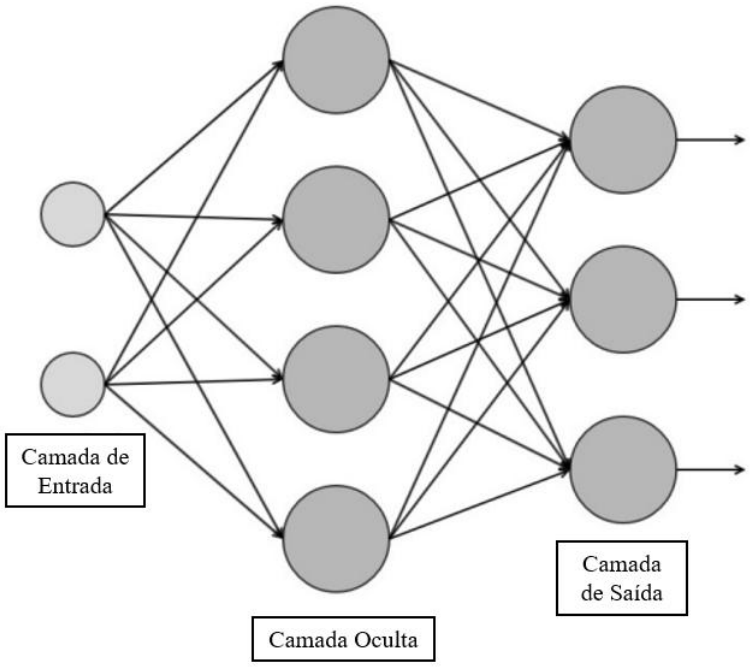
Conforme estudos de Silva (2020) em tarefas de classificação, a saída da FA é a classe predita pela maioria das árvores. As FAs corrigem o hábito das árvores de decisão de ter *overfitting* (sobreajuste) em seus treinamentos (GERON, 2019). O algoritmo introduz aleatoriedade extra ao ter uma maior diversidade de árvores, geralmente produzindo um modelo melhor (GERON, 2019).

2.3.1.2 Redes Neurais Artificiais

As redes neurais artificiais (RNA) foram introduzidas em 1943 por Warren McCulloch e Walter Pitts (HAYKIN, 1998), que criaram um modelo computacional simples com base em uma lógica proposicional de como os neurônios biológicos podem trabalhar juntos no cérebro para realização de cálculos complexos (GERON, 2019). Hoje, existe uma grande quantidade de dados utilizados para treinamentos de redes neurais, fazendo com que as RNAs avancem frequentemente a outras formas de AM com o *Deep Learning* que é utilizado para problemas de um grau maior de complexidade (SPOLTI, 2018).

Nas estruturas de uma RNA convencional existem diferentes topologias para organizar os neurônios, mas para modelos de aprendizado supervisionado, como os de classificação, a MLP (*Multilayer Perceptron*) é uma das mais usadas (GERON, 2019). A MLP é composta por pelo menos três camadas diferentes: a camada de entrada, uma ou mais camadas ocultas e a camada de saída (RABELO, 2014).

Figura 7 – Estrutura de uma Rede Neural Artificial (RNA) com arquitetura de MLP



Fonte: Adaptado de (RABELO, 2014)

Como podemos observar na Figura 7 cada camada consiste em um certo número de neurônios, cada neurônio se interconecta com todos os neurônios na camada seguinte e cada conexão representa um peso que contribui para o cálculo. Com uma função de ativação adequada e uma combinação de pesos em cada neurônio é possível gerar a predição de uma variável dependente categórica (GERON, 2019).

Em termos matemáticos, é possível descrever um neurônio k escrevendo as seguintes equações:

$$u_k = \sum_{j=1}^{m} w_{k,j} x_j \tag{11}$$

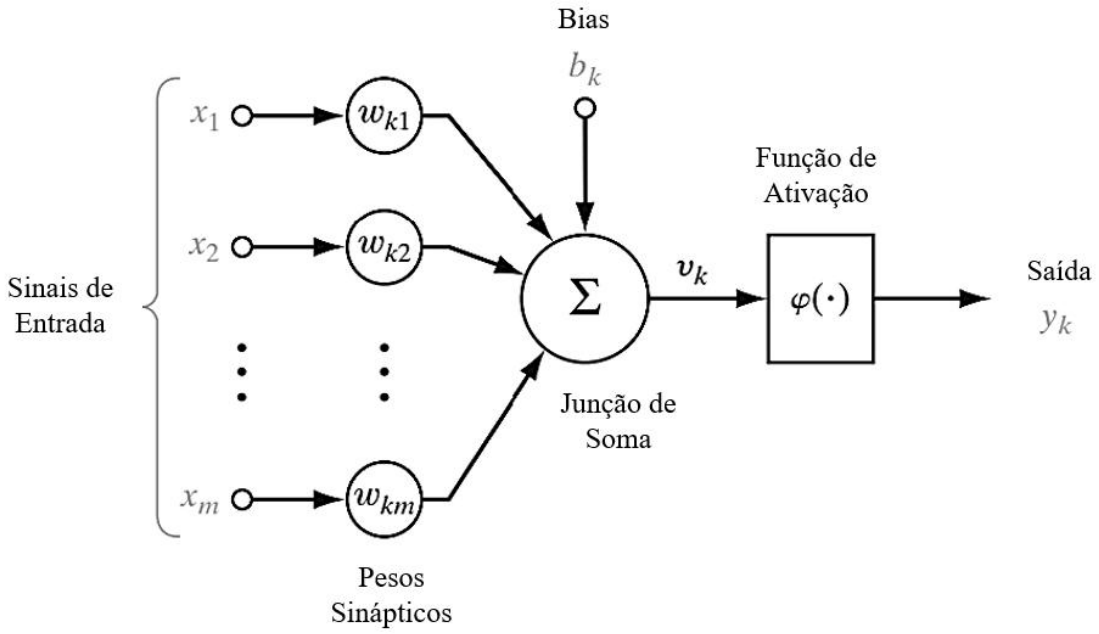
$$v_k = (u_k + b_k) \tag{12}$$

$$y_k = \varphi(v_k) \tag{13}$$

onde x_1, x_2,...,x_m são os sinais de entrada; $w_{k,1}$, $w_{k,2}$,...,$w_{k,m}$ são os pesos sinápticos do neurônio k; u_k é a saída do combinador linear devido aos sinais de entrada; b_k é o bias, que tem o efeito de aumentar ou reduzir a polarização na função de ativação; $\varphi(.)$ é a função de ativação do neurônio; e y_k é o sinal de saída do neurônio

(HAYKIN, 1998). O uso do bias b_k tem a função de aplicar alguma transformação na saída u_k do combinador linear (HAYKIN, 1998). O modelo matemático de um neurônio artificial é mostrado na Figura 8.

Figura 8 – Representação da modelagem de um neurônio artificial.



Fonte: Adaptado de (HAYKIN, 1998)

A função de ativação $\varphi(.)$, define a saída de um neurônio em função do campo local induzido v_k (HAYKIN, 1998). Existem três tipos básicos de funções de ativação: função limiar, função de unidade linear retificada e função sigmoide (HAYKIN, 1998).

O treinamento de uma RNA é essencial para otimização de cada peso com base no dados no conjunto de treinamento. Em MLPs o método de otimização de peso mais comumente usado é o algoritmo de *Back-Propagation*, que analisa iterativamente os erros e otimiza cada valor de peso com base nos erros gerados pela próxima camada (GERON, 2019).

2.3.1.3 k-Vizinhos Mais Próximos

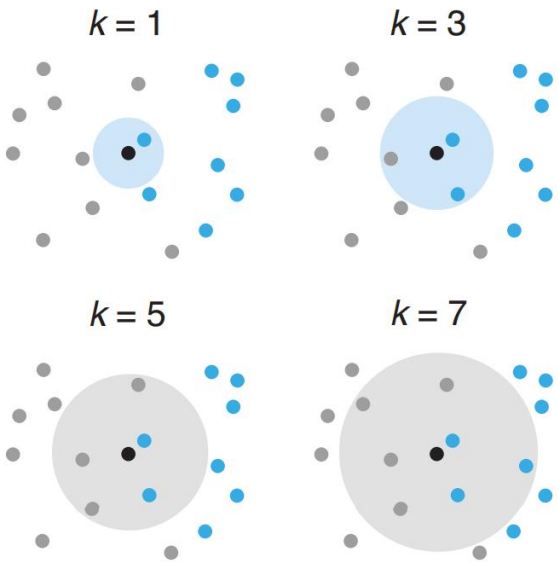
O algoritmo do k-Vizinhos mais Próximos (*k-Nearest Neighbors* em ingles-kNN) é um algoritmo simples capaz de classificar um conjunto de dados de forma muito rápida e eficiente, o k-NN usa uma medida de similaridade para comparar os dados de teste com os dados de treinamento (GERON, 2019).

Seu funcionamento é muito simples: são armazenados exemplos de treinamento de dados históricos e quando é necessário classificar um novo objeto, os

objetos mais semelhantes são extraídos e sua classificação é utilizada para classificar o novo objeto (OREA; VARGAS; ALONSO, 2005). O kNN usa a proximidade para fazer classificações ou previsões sobre a classe de um ponto de dados individual (OREA; VARGAS; ALONSO, 2005).

Para problemas de classificação, um rótulo de classe é atribuído com base na "votação majoritária", ou seja, é usado o rótulo que é mais frequentemente representado em torno de um determinado ponto de dados (CHENG et al., 2014). A "votação majoritária"tecnicamente exige uma maioria superior a 50%, o que funciona principalmente quando há apenas duas categorias. Quando se tem várias classes, por exemplo quatro categorias, não precisa necessariamente de 50% dos votos para chegar a uma conclusão sobre uma classe, é possível atribuir um rotulo de classe com uma votação superior a 25% (BZDOK; KRZYWINSKI; ALTMAN, 2018).

Figura 9 – Representação do classificador de k-vizinhos mais próximos (kNN).



Fonte: Adaptado de (BZDOK; KRZYWINSKI; ALTMAN, 2018)

Na Figura 9 é observado uma ilustração do classificador de k-vizinhos mais próximos, no qual o algoritmo de kNN atribui uma rótulo de classe a um ponto não classificado (preto) com base na votação majoritária dos k vizinhos mais próximos no conjunto de treinamento (pontos cinza e azul). São mostrados os casos para k = 1, 3, 5 e 7; os k vizinhos estão circunscritos no círculo, que é colorido pelo voto da classe majoritária.

O kNN é um método de aprendizado que não constrói um modelo ou uma função, mas produz os k registros mais próximos do conjunto de dados de treinamento que são os mais semelhantes ao ponto que devem ser categorizados (GERON, 2019).

2.4 META LEARNING

Antes de adentrar ao que este trabalho propõe precisamos explanar acerca do *Meta Learning*. Segundo Jankowski, Grabczewski e Włodzisław (2011), o termo “meta-aprendizagem” em geral abrange uma ampla gama de abordagens direcionadas a "aprender a aprender". O *Meta Learning*, operando no espaço de transformações de dados disponíveis e técnicas de otimização, deve aprender com a experiência de resolver diferentes problemas, fazer inferências sobre transformações úteis em diferentes contextos e, finalmente, ajudar a construir algoritmos de aprendizagem interessantes que possam revelar vários aspectos do conhecimento escondidos em dados (JANKOWSKI; GRABCZEWSKI; WłODZISłAW, 2011).

Portanto o objetivo geral do *Meta Learning* é maximizar a probabilidade de encontrar possivelmente a melhor solução de um dado problema P dentro de um espaço de busca no menor tempo possível (JANKOWSKI; GRABCZEWSKI; WłODZISłAW, 2011). O *Meta Learning* se destaca por apresentar diferenças do AM, já que esse último trabalha sobre um conjunto de dados e o *Meta Learning* utiliza-se do acúmulo de experiência da performance de utilização de algoritmos de AM. Em suma, o *Meta Learning* se distingue por enfatizar a relação entre métodos de aprendizagem e seus problemas (SOUZA, 2010). Isto é, o *Meta Learning* é capaz de estabelecer sob quais propriedades do problema cada algoritmo é mais aplicável, sendo capaz ainda de levar a opções de aplicação mais apropriada.

Considerando o estudo por Souza (2010) o processo de recomendação de algoritmos aplicando o *Meta Learning* inicia-se com a obtenção de um conjunto adequado de problemas. Nesse processo duas fases são utilizadas para cada instância de problema. A primeira fase define-se pela avaliação dos algoritmos de AM, que no caso deste trabalho são as MHs. Na segunda fase se atribui a extração dos atributos dos problemas. Ao vincular as informações resultantes das duas fases se obtem um Meta Exemplo, esse é composto por Meta-Atributos, os quais podemos dividir em dois, que são os Meta-Atributos de entrada que se classificam pelas características dos problemas; e os Meta-Atributos de saída que são os desempenhos dos algoritmos de AM, no caso deste trabalho as MHs. Esse conjunto de Meta Exemplos chama-se Meta Dados. Posteriormente, a esses Meta Dados é aplicado a um algoritmo de AM, pelo qual será capaz de realizar recomendações de algoritmos no ambiente de *Meta Learning*.

Segundo Souza (2010), os *Meta Features*, ou também chamados de Meta Atributos, que tem como objetivo de descrever as propriedades das instâncias dos problemas e atribuir informações que possam ajudar a observar o desempenho dos algoritmos utilizados. O autor cita como exemplo que alguns algoritmos não atuam de modo satisfatório na companhia de atributos desnecessários, como o kNN, no entanto,

outros algoritmos como as Redes Neurais tem melhor desempenho pois possuem métodos próprios para considerar quais atributos são mais relevantes.

Para tratar sobre recomendação de algoritmos com melhor desempenho na instância dos problemas, Kalousis (2002) explica que existem três abordagens. A primeira constitui-se em oferecer o melhor algoritmo, a segunda destaca-se por oferce um conjunto de algoritmos com o melhor desempenho e a terceira caracteriza-se por fornecer um ranking dos algoritmos com melhor desempenho relacionados à base de dados.

De acordo com Kanda, Soares e Brazdil (2016) no contexto de instâncias de TSP, o *Meta Learning* consiste em gerar um metamodelo que mapeia características das instâncias de TSP, ou seja, "*Meta Features*", para o desempenho de algoritmos, especificamente as meta-heurísticas utilizadas para o TSP. O metamodelo é normalmente alguma forma de algoritmo de aprendizado de classificação/ranking, como aqueles criados pela comunidade de aprendizado de máquina. Esse algoritmo de aprendizado então produz uma correspondência entre certas *Meta Features* do TSP e as meta-heurísticas que tendem a produzir os melhores resultados (GASTINEAU, 2018). O *Meta Learning* engloba tanto uma MH de aprendizagem "uma heurística que aprende a escolher outras heurísticas", e aprendizagem sobre o domínio do problema indiretamente por meio de Meta Dados.

Tabela 1 – Exemplificação de Meta Dados

Meta Exemplos	***Meta Features*** $_1$	**...**	***Meta Features*** $_m$	**Solução MH** $_1$	**...**	**Solução MH** $_h$
TSP $_1$						
			.			
			.			
			.			
TSP $_n$						

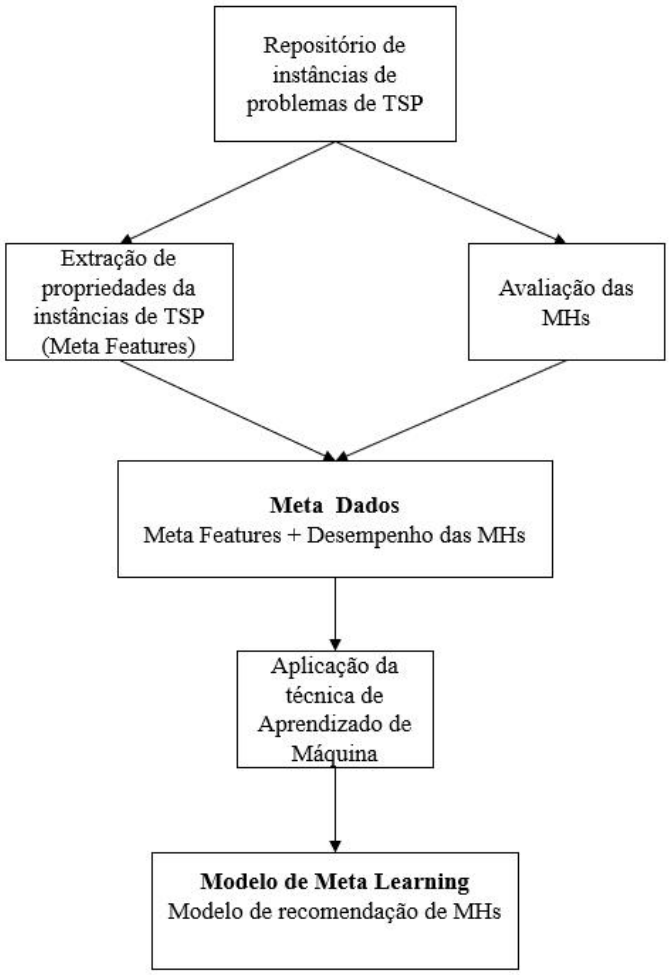
Fonte: Autor (2022)

Basicamente, o *Meta Learning* aplicado em instâncias de TSP ocorre nas duas fases explicadas anteriormente: 1) geração de *Meta Features* e 2) criação do metamodelo (KANDA; SOARES; BRAZDIL, 2016).

Na primeira fase, um conjunto de instâncias de TSP é usado para gerar os Meta Dados. Os Meta Dados são armazenados em uma matriz cujas linhas são Meta Exemplos e colunas são *Meta Features*, conforme representado na Tabela 1. Um Meta Exemplo é uma instância particular do TSP e os *Meta Features* são atributos que representam propriedades relevantes que descrevem essas instâncias (KANDA; CARVALHO; SOARES, 2011). Cada Meta Exemplo é rotulado com a qualidade da solução obtida por diferentes MHs quando aplicados à instância TSP correspondente. No caso do TSP é o custo da melhor solução obtido pela correspondente MH. A

estrutura geral de uma *Meta Learning* é ilustrada na Figura 10.

Figura 10 – *Meta Learning* para selecionar MHs para o TSP



Fonte: Adaptado de (KANDA; SOARES; BRAZDIL, 2016)

De acordo com Kanda, Carvalho e Soares (2011) estudos de *Meta Learning* tradicionais em TSP constroem features idealizados para o problema dado (como o número de vértices em um grafo, custo médio dos vértices, coeficientes de agrupamento calculados de várias maneiras, etc.), o que não muito diferente do que é feito para outros problemas de aprendizado de máquina. No entanto, na metodologia apresentada por Kanda, Soares e Brazdil (2016) os melhores resultados são encontrados usando os chamados “features complexos”. Features complexos são aqueles que são computacionalmente difíceis de calcular e podem aumentar drasticamente o tempo de execução de um algoritmo de *Meta Learning* (GASTINEAU, 2018). Afinal, podemos concluir que *Meta Learning* em AM se refere a algoritmos de aprendizado de máquina que aprendem com a saída de outros algoritmos.

Neste trabalho o objetivo é identificar por meio do *Meta Learning* qual MH tem melhor desempenho nas instâncias novas, ou seja, todavia desconhecidas pelo

algoritmo de *Meta Learning*, consideradas neste trabalho. Algoritmos de AM aprendem com dados históricos, por exemplo, algoritmos de aprendizado supervisionado aprendem como mapear exemplos de padrões de entrada para exemplos de padrões de saída para resolver problemas de modelagem preditiva de classificação e regressão (GERON, 2019). Os algoritmos de *Meta Learning* aprendem com a saída de outros algoritmos que aprendem com os dados. Isso significa que o *Meta Learning* requer a presença de outros algoritmos que já foram treinados com os dados.

3 MATERIAIS E MÉTODOS

3.1 DADOS

Tendo em vista que o *Meta Learning* requer um grande número de exemplos para otimizar adequadamente o objetivo proposto, neste trabalho utilizou-se as 2760 instancias de TSP propostas por Gastineau (2018), no qual realizou a geração de grafos aleatórios. Destaca- se que todas as instâncias geradas no conjunto de dados final são totalmente conectadas, ou seja, cada nó está conectado a todos os outros nós.

Para criação do Meta Dados, utilizou-se as meta-heurísticas de Simulated Annealing, Tabu Search, GRASP, Algoritmos genéticos e Ant Colony Optimization as quais foram implementadas em Python (PYTHON, 2021) por Gastineau (2018), e executadas em cada instância de TSP. A Tabela 2 e a Figura 11 descrevem o número de instâncias de TSP que cada meta-heurísticas tem o melhor resultado. Observou-se que com as cinco meta-heurísticas temos um conjunto de dados desbalanceado.

Tabela 2 – Número de instâncias com melhor solução por MH

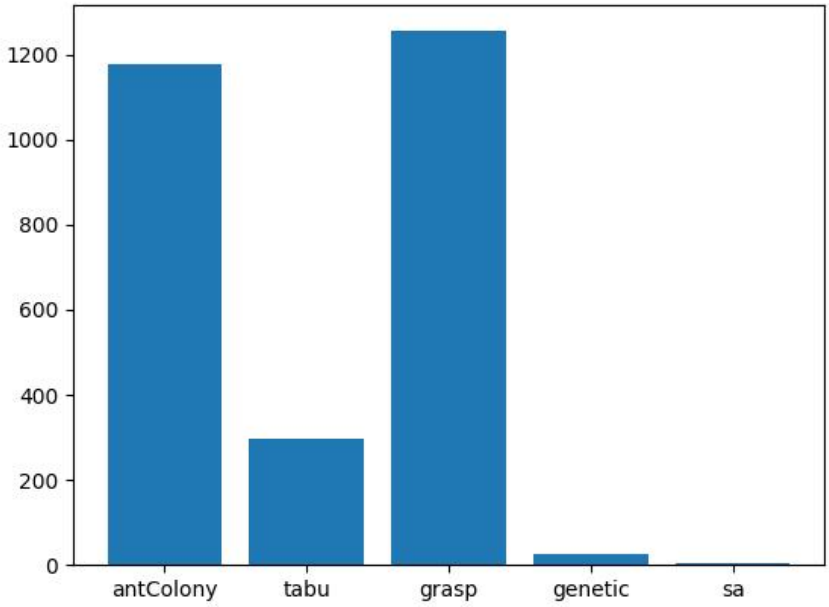
Meta-heurísticas	**Numero de instâncias**	**Proporção (%)**
Ant Colony Optimization	1178	(42,7%)
Algoritmos Genéticos	24	(0,9%)
GRASP	1255	(45,5%)
Simulated Annealing	6	(0,2%)
Tabu Search	297	(10,7%)
Total	**2760**	**(100%)**

Fonte: Autor (2022)

Deve-se notar que tanto a meta-heurística de Ant Colony Optimization e GRASP são as meta-heurísticas mais frequentes como a melhor solução das instâncias. Da mesma forma, observamos que uma das duas é a melhor solução em mais de 88% do conjunto de instâncias. Isto é, um sinal de que o conjunto de dados está desbalanceado o que resultará em um viés significativo para aquelas meta-heurísticas.

O problema com um conjunto de dados desbalanceado é que há poucos exemplos da classe minoritária para que um modelo aprenda efetivamente o limite de decisão (CHAWLA et al., 2002). Uma maneira de resolver esse problema é superamostrar os exemplos nas classes minoritárias. Isso pode ser alcançado simplesmente duplicando exemplos da classe minoritária no conjunto de dados de treinamento antes de ajustar um modelo. Isso pode equilibrar a distribuição de classes, mas não fornece nenhuma informação adicional ao modelo (CHAWLA et al., 2002).

Figura 11 – Distribuição de Meta-Heurísticas



Fonte: Autor (2022)

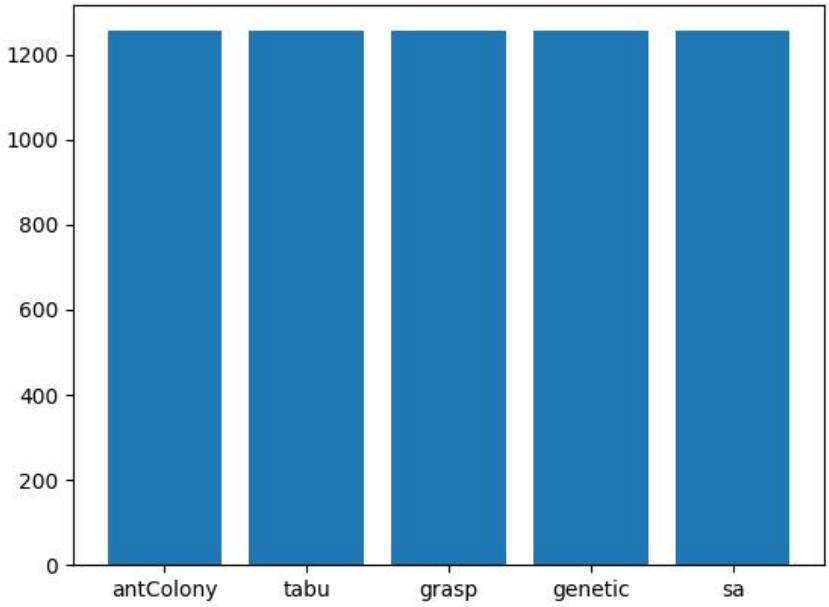
Uma melhoria na duplicação de exemplos das classes minoritárias é sintetizar novos exemplos das classes minoritárias. Este é um tipo de aumento de dados para dados tabulares e pode ser muito eficaz. Talvez a abordagem mais amplamente usada para sintetizar novos exemplos é chamada de Técnica de Sobreamostragem de Minorias Sintéticas, ou *SMOTE* para abreviar (CHAWLA et al., 2002). Esta técnica foi descrita por Chawla et al. (2002) em seu artigo nomeado *"SMOTE: Synthetic Minority Over-sampling Technique"*.

O *SMOTE* funciona selecionando exemplos próximos no espaço de características, criando uma linha entre os exemplos no espaço de recursos e gerando uma nova amostra em um ponto ao longo dessa linha (CHAWLA et al., 2002). Especificamente, um exemplo aleatório da classe minoritária é escolhido primeiro. Então k dos vizinhos mais próximos para esse exemplo são encontrados. Um vizinho selecionado aleatoriamente é escolhido e um exemplo sintético é criado em um ponto selecionado aleatoriamente entre os dois exemplos (CHAWLA et al., 2002).

Portanto, foi utilizado o método *SMOTE* para balancear nosso conjunto de dados. A Figura 12 representa nosso conjunto de dados depois de aplicar o método *SMOTE*.

A base de dados de *features* das instâncias de TSP gerada por Gastineau (2018) possui 39 features, que foram baseadas no trabalho de Kanda, Soares e Brazdil (2016), porém nem todos foram utilizados para este trabalho. As *features* utilizadas

Figura 12 – Distribuição de Meta-Heurísticas após aplicação do Método *SMOTE*



Fonte: Autor (2022)

neste trabalho implementadas por Gastineau (2018), estão listadas na Tabela 3.

A maioria das *Meta Features* descrevem propriedades das arestas e vértices de um grafo como exemplo o número de vértices. Também podemos ver *Meta Features* associadas ao custo de aresta que podem fornecer informações importantes sobre a estratégia de busca mais adequada para um determinado grafo. Outra informação relevante que podemos extrair de um grafo é o custo do vértice, que é medido como o custo médio das arestas conectadas ao vértice. Outras *Meta Features* mais complexas foram utilizadas como a distância geodésica média, que mede a distância média mais curta entre cada par de nós. Para capturar informações entre os vértices, foi calculado pela medida de eficiência global, que assume que a eficiência de envio de informações entre dois vértices é inversamente proporcional à sua distância.

Portanto, para ter maior diversificação e conseguir fazer a comparação em nosso conjuntos de dados, no presente trabalho foram criado 5 diferentes tipos de subconjuntos de dados, descrito na sequéncia:

- Subconjunto 1: Dados somente com *Features* de Vértices;
- Subconjunto 2: Dados somente com *Features* de Arestas;
- Subconjunto 3: Dados com *Features* Vértices e Arestas;
- Subconjunto 4: Dados com *Features* Complexas;
- Subconjunto 5: Dados com todas as *Features*.

Tabela 3 – Features da base de dados

Tipo de Features	**Features**	**Descrição**
Vértices	n_vertices	Números de Vértices
	lower_vcost	Menor Custo dos Vértices
	higher_vcost	Maior Custo dos Vértices
	average_vcost	Custo Médio dos Vértices
	standard_vcost	Desvio Padrão dos Custos Vértices
	median_vcost	Mediana dos Custos dos Vértices
Arestas	sum_cost_NN	Soma dos Custos das Arestas para ser Vizinho Mais Próximo
	n_edges	Número de Arestas
	lower_ecost	Menor Custo das Arestas
	higher_ecost	Maior Custo das Arestas
	average_ecost	Custo Médio das Arestas
	standard_ecost	Desvio Padrão dos Custos das Arestas
	median_ecost	Mediana dos Custos das Arestas
	total_lowestEcost	Custo Total dos n menores custos de Arestas
Complexas	average_GeodesicDistance	Distância Geodésica Média
	global_Efficiency	Eficiência Global
	network_Vulnerability	Vulnerabilidade da Rede
	target_Entropy	Entropia de Destino
Total	**18**	

Fonte: Autor (2022)

3.2 MÉTODOS

Como já mencionado anteriormente, os métodos de classificação utilizados neste estudo foram o FA, RNA e kNN treinados e avaliados utilizando a linguagem de programação Python (PYTHON, 2021).

Os códigos utilizados para carregar as *Meta Features* e treinar os modelos foram escritos pelo autor. Para treinar o modelo, utilizamos uma biblioteca de AM escrita em Python, chamada *Scikit-Learn* (PEDREGOSA et al., 2011). Para o kNN foi realizado 6 diferentes análises no qual é variado o parâmetro de pesos, que é o método para atribuir pesos aos vizinhos mais próximos. Neste caso, os pesos serão atribuídos de maneira uniforme e por distância. Outro parâmetro que varia é o número de vizinhos a serem usados, no qual é definido k igual a 3, 5 e 7. No método de MLP foram realizadas três análises, no qual é variado o número de camadas ocultas e a quantidade de neurônios em cada camada ocultada. Na primeira análise foi utilizado uma camada oculta considerando 100 neurônios, na segunda análise foi utilizado três

camadas ocultas seguindo a ordem de 25, 50 e 25 neurônios e na terceira análise foi aplicado duas camadas ocultas com 50 neurônios cada. As três análises utilizaram como função de ativação a Unidade Linear Retificada (em inglês, Rectified Linear Unit - ReLU), usando como taxa de aprendizado inicial de 0.001. Já para a Floresta Aleatória foi adaptado o parâmetro de número de árvores na floresta, no qual foi escolhido os números 50, 100 e 150.

Além de treinar o modelo, é necessario testa-lo. Seria incoerente treinar e testar utilizando a mesma base de dados, pois o algoritmo poderia simplesmente "decorar", todos os parâmetros e não conseguir generalizar o teste para novos dados, sendo este o tipo de comportamento que deve ser evitado (situação cujo nome é overfitting) (HAWKINS, 2004). Desta forma, é preciso dividir os dados em um conjunto treinamento e outro para teste.

3.2.1 Método de *K-fold cross-validation*

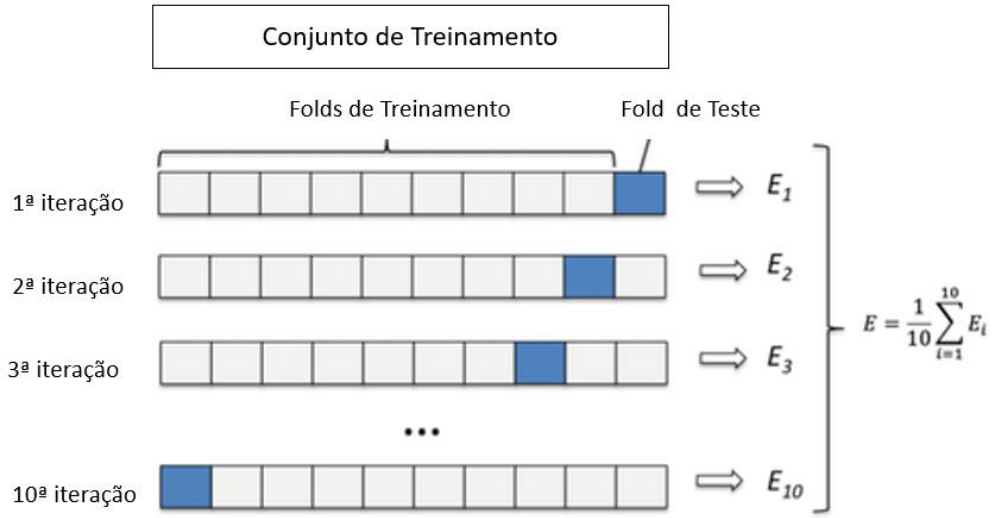
Para particionar os dados e avaliar os modelos, é utilizado o procedimento chamado *K-fold cross-validation* (KOHAVI, 1995). Em *k-fold cross-validation*, o conjunto de dados é particionado em K subconjuntos. Dessa forma, é treinado o modelo utilizando K-1 conjuntos, sendo validado com o conjunto restante, que serve como teste. O processo é repetido K vezes, até que todos os subconjuntos sejam usados como conjunto de teste exatamente uma vez, como mostrado na Figura 13.

A medida de desempenho total é calculada como a média de todas as medidas obtidas em cada iteração. O valor médio E dos resultados do teste da dez pastas é calculado como uma estimativa da precisão do modelo e é usado como um indicador de desempenho para o atual modelo de validação cruzada K-fold. Onde E_i representa o erro de validação cruzada de cada pasta.

Esse processo é suficiente para um modelo cuja proporcionalidade das classes dos dados seja, no mínimo, razoável. Entretanto, como informado anteriormente, o conjunto de dados está mal distribuído. Para resolver este problema, uma variação da técnica *k-fold-cross-validation* é utilizada. *Stratified k-fold-cross-validation* é semelhante ao seu precursor. Entretanto, para solucionar o problema de desbalanceamento de classes, o processo é realizado de forma a manter as proporções das classes igualmente distribuídas em cada fold, ou cada subconjunto.

Finalmente, é utilizado o método *Repeated Stratified k-fold cross-validation* que fornece uma maneira de melhorar o desempenho estimado de um modelo de aprendizado de máquina. Isso envolve simplesmente repetir o procedimento de validação cruzada várias vezes e relatar o resultado médio em todas os folds de todas as execuções. Os principais parâmetros do método são o número de folds, que é o “k” na técnica *k-fold-cross-validation*, o número de repetições e o *RandomState* que controla a geração dos estados aleatórios para cada repetição (PEDREGOSA et al.,

Figura 13 – K-fold cross-validation



Fonte: Adaptado de (BUHAGIAR; AZZOPARDI, 2017)

2011).

Portanto, foi aplicado dois tipos de testes em nosso conjunto de dados. No teste A é utilizado a função *Repeated Stratified k-fold cross-validation* com k igual a 5 para dividir os dados em folds, número de repetições = 2 e *RandomState* = 8, de acordo com as características mencionadas, e foi executado os métodos de AM correspondentes até que cada fold fosse utilizado exatamente uma única vez como conjunto de teste.

No teste B é utilizado a função *Stratified k-fold cross-validation* com k igual a 10 para dividir os dados em folds e *RandomState* = 8. Alem disso, foi aplicado o método SMOTE, de acordo com as características mencionadas, e foi executado os métodos de AM correspondentes até que cada fold fosse utilizado exatamente uma única vez como conjunto de teste.

3.2.2 Avaliação dos Métodos

Para construção de modelos de AM, a avaliação de modelos é uma etapa vital do processo, porque mediante o uso de indicadores, podemos encontrar as taxa de acertos dos modelos. Uma abordagem avaliativa de modelos contribui para a comparação de diferentes algoritmos e técnicas. A métrica utilizada para determinação do “melhor modelo” depende do problema analisado.

Uma maneira de avaliar o desempenho de um classificador é observar a matriz de confusão. A Tabela 4 representa a matriz de confusão que é constituída com verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos (HASTIE; TIBSHIRANI, 2017). Importante notar que a Tabela 4 é valida para problemas

de classificação binários.

Tabela 4 – Matrix de confusão

	Classe prevista	**Classe prevista**
Classe original	*Positivo*	*Negativo*
Positivo	Verdadeiro Positivo (TP)	Falso Negativo (FN)
Negativo	Falso Positivo (FP)	Verdadeiro Negativo (TN)

Fonte: Autor (2022)

Os indicadores para avaliar são derivados da matriz de confusão (SANTOS, 2020a), os quais neste trabalho será considerado a acurácia, sensibilidade, precisão e *F-measure*.

A acurácia é calculada pela proporção dos resultados verdadeiros (positivos e negativos), sobre o total do número de casos previstos e originais (SANTOS, 2020a), a fórmula da acurácia é presentada pela Equação (14):

$$Acuracia = \frac{TP + TN}{TP + TN + FP - FN} \tag{14}$$

Em uma abordagem de dados desbalanceados a métrica de avaliação pode interferir nos resultados, por exemplo quando há uma classe muito desbalanceada, a acurácia não é uma boa métrica a ser usada (BARELLA, 2015). Portanto o modelo tem uma sensação errada que está fazendo a classificação correta. Por exemplo, se temos um conjunto de testes, que somente 10% da população possui câncer e um modelo construído para detectar câncer prever que um paciente não tem câncer 100% das vezes, o modelo possuirá 90% de acurácia, portanto o modelo esta mostrando não ter um bom uso de aprendizagem (OLIVERA, 2016) .

Segundo Santos (2020a) a precisão é definida pela proporção de previsões corretas, sobre os indicados pelo modelo como positivos, formulada pela Equação (15):

$$Precisao = \frac{TP}{TP + FP} \tag{15}$$

O *recall*, também chamado de taxa de sensibilidade, é calculado pela proporção de verdadeiros positivos sobre os verdadeiros positivos e falsos negativos (GERON, 2019), representada pela Equação (16):

$$Recall = \frac{TP}{TP + FN} \tag{16}$$

A métrica *F-measure* considera os valores da precisão e da taxa de sensibilidade (recall) para que então seja calculado a pontuação. Santos (2020a) define que é uma métrica para o teste de precisão, sendo a pontuação uma média harmônica, com resultado entre 0 e 1, onde o seu melhor valor é obtido em 1. O F-measure é

formulado pela Equação (17), onde 0 < β < 1 ou β > 1 (HASTIE; TIBSHIRANI, 2017).

$$F - measure = (1 + \beta^2)\frac{precisao.recall}{(\beta^2.precisao) + recall} \tag{17}$$

Quando se trata de casos multiclasse, o F-Measure, Precisão e Recall devem envolver uma equação para todas as classes (GRANDINI; BAGLI; VISANI, 2020). Para fazer isso, é necessario de uma medida multiclasse.

Precisão, Recall, e F-Measure para cada classe são calculados usando as mesmas fórmulas da configuração binária (GRANDINI; BAGLI; VISANI, 2020), conforme descrito acima. As Equações (18), (19) e (20) representam as três grandezas para uma classe genérica *k*, considerando K classes.

$$Precisao_k = \frac{TP_k}{TP_k + FP_k} \tag{18}$$

$$Recall_k = \frac{TP_k}{TP_k + FN_k} \tag{19}$$

$$F - measure_k = (1 + \beta^2)\frac{precisao_k.recall_k}{(\beta^2.precisao_k) + recall_k} \tag{20}$$

Posteriormente, para obter um valor macro de cada métrica é feito uma média ponderada, multiplicada pelo peso de cada classe w_k, ou seja, a frequência da classe em toda a base de dados. Adicionamos também a soma dos pesos *W* no denominador. As Equações (21), (22) e (23) representam as métricas macro para casos de multiclasse.

$$precisao\ ponderada = \frac{\sum_{k=1}^{K} w_k.precisao_k}{W} \tag{21}$$

$$recall\ ponderada = \frac{\sum_{k=1}^{K} w_k.recall_k}{W} \tag{22}$$

$$F - measure\ ponderada = \frac{\sum_{k=1}^{K} w_k.F - measure_k}{W} \tag{23}$$

Uma vez que cada métrica tenha sido ponderada pela frequência de cada classe w_k, a média da métrica não é mais afeitada pelas classes de baixa frequência: as classes de grande frequência terão um peso proporcional ao seu tamanho e as pequenas terão um efeito redimensionado (GRANDINI; BAGLI; VISANI, 2020). Como cada métrica é ponderada pela frequência de classe do conjunto de dados, a métrica multiclasse pode ser um bom indicador de desempenho quando o objetivo é treinar um algoritmo de classificação com um grande número de classes (GRANDINI; BAGLI; VISANI, 2020). Na verdade, essa métrica permite manter os desempenhos dos algoritmos separados nas diferentes classes, para que possamos rastrear qual classe

causa um desempenho ruim. Ao mesmo tempo, acompanha a importância de cada classe pela frequência.

4 RESULTADOS E DISCUSSÕES

Neste capítulo serão apresentados os resultados obtidos em cada um dos três métodos propostos para treinamento e validação dos dados. Em seguida, os resultados foram analisados de modo a avaliar qual é o método mais adequado para nossa base de dados.

4.1 ANÁLISE DOS RESULTADOS POR MÉTODO

Os 3 modelos de AM com suas respetivas variações de parâmetros foram comparados utilizando as métricas de classificação acurácia, precisão, recall, F-measure. Além disso, foram feito dois tipos de teste, no qual o teste A consiste em uma validação cruzada estratificada repetida (*Repeated Stratified k-fold Cross Validation*) de 5 pastas e 2 repetições, tendo um total de 10 pastas para verificar o quanto os modelos criados são generalizáveis. As 4 métricas de classificação foram calculadas nas 10 pastas para cada um dos modelos de AM e ao final obtivemos a média ponderada e desvio padrão das métricas de classificação nas etapas da validação cruzada.

O teste B consiste em uma validação cruzada estratificada (*Stratified k-fold Cross Validation*) de 10 pastas, ademais da aplicação do método *SMOTE*. Novamente 4 métricas de classificação foram calculadas nas 10 pastas para cada um dos modelos de AM e ao final obtivemos a média ponderada e desvio padrão das métricas de classificação nas etapas da validação cruzada.

4.1.1 Análise das métricas no método kNN

Para o método kNN foi realizado 6 diferentes análises no qual é variado o parâmetro de pesos, que é o método para atribuir pesos aos vizinhos mais próximos. Neste caso, os pesos serão atribuídos de maneira uniforme e por distância. Outro parâmetro que varia é o número de vizinhos a serem usados, no qual é definido k igual a 3, 5 e 7.

Considerando os testes realizados, importante observar cada resultado obtido. Na Tabela 5, Tabela 6 e Tabela 7, é possível observar a aplicação do peso uniforme, considerando k equivalente a 3, 5 e 7. Ao aplicar o teste A, no qual não há aplicação da técnica *SMOTE*, o melhor resultado foi o subconjunto 1 nas quatro métricas mensuradas, independentemente do valor de k aplicado. Quando da aplicação do teste B, na qual foi aplicada a técnica *SMOTE*, o subconjunto 1 também obteve o melhor resultado nas quatro métricas, independentemente do valor de k.

De modo geral, ao observar as tabelas citadas, verifica-se que o teste que

Tabela 5 – Média dos Resultados do Método kNN - Peso Uniforme | k = 3

		Acurácia	Precisão	Recall	F1-score
TESTE A	Subconjunto 1	79.2%±1.4%	78.1%±1.5%	79.2%±1.4%	78.5%±1.4%
	Subconjunto 2	78.2%±1%	77.2%±1.1%	78.2%±1%	77.6%±1%
	Subconjunto 3	78%±1%	77%±1.2%	78%±1%	77.4%±1.1%
	Subconjunto 4	77.8%±1%	76.8%±0.9%	77.8%±1%	77.2%±0.9%
	Subconjunto 5	78.5%±1.6%	77.4%±1.2%	78.5%±1.6%	77.8%±1.4%
TESTE B	Subconjunto 1	87.9%±1.4%	87.9%±1.5%	87.9%±1.4%	87.4%±1.5%
	Subconjunto 2	87.1%±0.9%	87.1%±0.8%	87.1%±0.9%	86.8%±0.9%
	Subconjunto 3	86.6%±1.2%	86.6%±1.2%	86.6%±1.2%	86.3%±1.2%
	Subconjunto 4	73.3%±1.7%	73.3%±1.8%	73.3%±1.7%	73.1%±1.8%
	Subconjunto 5	82.8%±1%	82.7%±1%	82.8%±1%	82.4%±1%

Fonte: Autor (2022)

Tabela 6 – Média dos Resultados do Método kNN - Peso Uniforme | k = 5

		Acurácia	Precisão	Recall	F1-score
TESTE A	Subconjunto 1	80.1%±1.3%	78.8%±1.3%	80.1%±1.3%	79.2%±1.3%
	Subconjunto 2	79%±1.2%	77.7%±1.3%	79%±1.2%	78%±1.2%
	Subconjunto 3	78.7%±1.1%	77.4%±1.1%	78.7%±1.1%	77.8%±1%
	Subconjunto 4	78.1%±1.1%	77.2%±0.9%	78.1%±1.1%	77.3%±1%
	Subconjunto 5	79.4%±1.5%	78.2%±1.2%	79.4%±1.5%	78.5%±1.3%
TESTE B	Subconjunto 1	86.9%±1.4%	87.1%±1.4%	86.9%±1.4%	86.4%±1.5%
	Subconjunto 2	86.2%±1.4%	86.4%±1.4%	86.2%±1.4%	85.8%±1.5%
	Subconjunto 3	86%±1.6%	86.1%±1.7%	86%±1.6%	85.7%±1.7%
	Subconjunto 4	75.4%±1.5%	75.5%±1.4%	75.4%±1.5%	75.2%±1.5%
	Subconjunto 5	82.2%±1.6%	82.3%±1.5%	82.2%±1.6%	81.8%±1.6%

Fonte: Autor (2022)

Tabela 7 – Média dos Resultados do Método kNN - Peso Uniforme | k = 7

		Acurácia	Precisão	Recall	F1-score
TESTE A	Subconjunto 1	80.6%±1.6%	79.3%±1.5%	80.6%±1.6%	79.7%±1.5%
	Subconjunto 2	79.5%±1.4%	78.1%±1.4%	79.5%±1.4%	78.5%±1.3%
	Subconjunto 3	79.4%±1.5%	77.9%±1.5%	79.4%±1.5%	78.3%±1.4%
	Subconjunto 4	77.8%±0.6%	77.3%±0.5%	77.8%±0.6%	77.2%±0.5%
	Subconjunto 5	80.1%±1.8%	78.9%±1.6%	80.1%±1.8%	79.1%±1.6%
TESTE B	Subconjunto 1	85.9%±1%	86.3%±1%	85.9%±1%	85.4%±1.1%
	Subconjunto 2	84.9%±1.5%	85.1%±1.4%	84.9%±1.5%	84.5%±1.6%
	Subconjunto 3	85.2%±1.3%	85.4%±1.2%	85.2%±1.3%	84.8%±1.3%
	Subconjunto 4	76.4%±1.8%	76.8%±1.8%	76.4%±1.8%	76.1%±1.8%
	Subconjunto 5	81%±1.6%	81.4%±1.5%	81%±1.6%	80.5%±1.6%

Fonte: Autor (2022)

obteve melhor resultado em todos os subconjuntos é o teste B. Contudo, ao observar detalhadamente o teste A, no subconjunto 4, foi aquele que obteve melhor resultado. Quando da análise do parâmetro k, no teste B, o k de melhor resultado foi aquele equivalente a 3. Quando observado o teste A, o k de melhor resultado é aquele igual a 7.

Tabela 8 – Média dos Resultados do Método kNN - Peso por Distância | k = 3

		Acurácia	Precisão	Recall	F1-score
TESTE A	Subconjunto 1	77.7%±1.5%	77.2%±1.8%	77.7%±1.5%	77.3%±1.6%
	Subconjunto 2	76.6%±1.5%	76.3%±1.6%	76.6%±1.5%	76.4%±1.5%
	Subconjunto 3	76.6%±1.5%	76.3%±1.6%	76.6%±1.5%	76.4%±1.6%
	Subconjunto 4	76.5%±1.3%	76.3%±1.2%	76.5%±1.3%	76.3%±1.2%
	Subconjunto 5	77.4%±1.6%	77%±1.6%	77.4%±1.6%	77%±1.5%
TESTE B	Subconjunto 1	88.8%±1.1%	88.8%±1.2%	88.8%±1.1%	88.4%±1.1%
	Subconjunto 2	87.7%±1.1%	87.6%±1%	87.7%±1.1%	87.5%±1.1%
	Subconjunto 3	87.9%±0.5%	87.8%±0.4%	87.9%±0.5%	87.6%±0.4%
	Subconjunto 4	75.4%±1.9%	75.4%±2%	75.4%±1.9%	75.3%±1.9%
	Subconjunto 5	84.4%±1.1%	84.2%±1.1%	84.4%±1.1%	84.1%±1.1%

Fonte: Autor (2022)

Tabela 9 – Média dos Resultados do Método kNN - Peso por Distância | k = 5

		Acurácia	Precisão	Recall	F1-score
TESTE A	Subconjunto 1	78.6%±1.4%	77.9%±1.5%	78.6%±1.4%	78.1%±1.4%
	Subconjunto 2	77.3%±1.5%	76.8%±1.5%	77.3%±1.5%	77%±1.5%
	Subconjunto 3	77.4%±1.4%	76.8%±1.6%	77.4%±1.4%	77%±1.5%
	Subconjunto 4	77%±1.3%	76.7%±1.2%	77%±1.3%	76.8%±1.2%
	Subconjunto 5	78.4%±1.7%	77.8%±1.7%	78.4%±1.7%	77.9%±1.5%
TESTE B	Subconjunto 1	88%±0.9%	88.1%±0.8%	88%±0.9%	87.6%±0.9%
	Subconjunto 2	87.6%±0.7%	87.6%±0.6%	87.6%±0.7%	87.4%±0.7%
	Subconjunto 3	87.3%±0.9%	87.3%±0.8%	87.3%±0.9%	87.1%±0.9%
	Subconjunto 4	77.4%±1.7%	77.5%±1.6%	77.4%±1.7%	77.3%±1.6%
	Subconjunto 5	84.3%±1.2%	84.3%±1.2%	84.3%±1.2%	84%±1.3%

Fonte: Autor (2022)

Nas comparações das Tabela 8, Tabela 9 e Tabela 10, nas quais foi aplicado o peso por distância, verifica-se que os valores de k correspondem a 3, 5 e 7. Assim, no teste A, que não há aplicação de *SMOTE*, o melhor resultado foi o subconjunto 1 nas quatro métricas mensuradas, independentemente do valor de k aplicado. Quando da aplicação do teste B, na qual foi aplicada a técnica *SMOTE*, o subconjunto 1 também obteve o melhor resultado nas quatro métricas, independentemente do valor de k. Nessas tabelas, os resultados do teste B também se sobressaem em relação aos resultados do teste A. Ademais, importante mencionar que as melhores performances nas Tabela 5, Tabela 6 e Tabela 7, se assemelham aos resultados das Tabela 8, Tabela 9

Tabela 10 – Média dos Resultados do Método kNN - Peso por Distância | k = 7

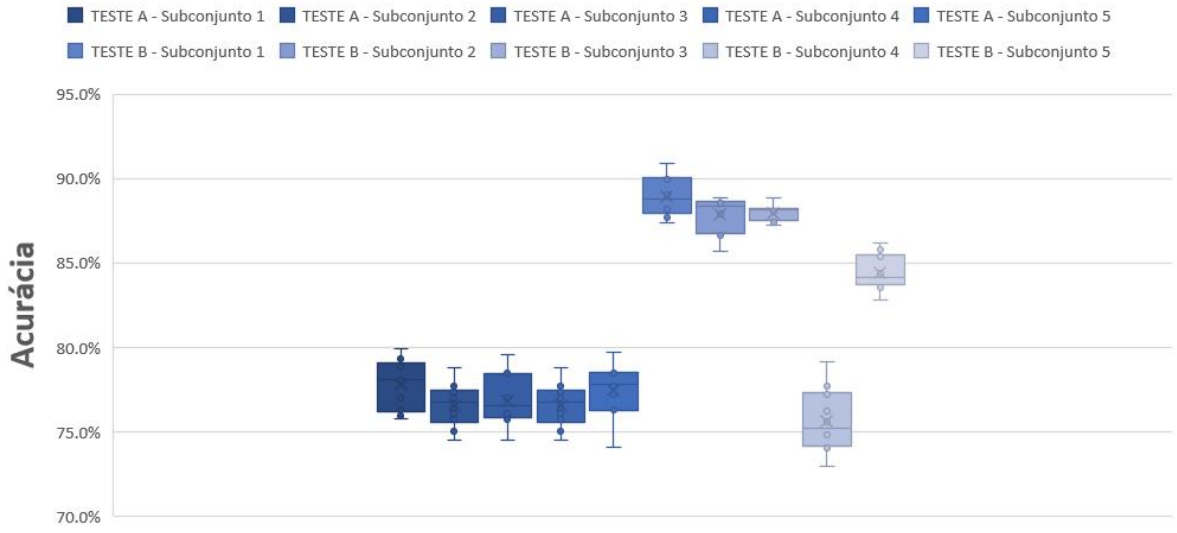
		Acurácia	Precisão	Recall	F1-score
TESTE A	Subconjunto 1	79%±1.6%	78.4%±1.5%	79%±1.6%	78.5%±1.5%
	Subconjunto 2	77.8%±1.6%	77.2%±1.8%	77.8%±1.6%	77.3%±1.6%
	Subconjunto 3	77.8%±1.5%	77.2%±1.7%	77.8%±1.5%	77.3%±1.5%
	Subconjunto 4	77.1%±1.3%	76.9%±1.3%	77.1%±1.3%	76.9%±1.3%
	Subconjunto 5	78.7%±1.7%	78.1%±1.8%	78.7%±1.7%	78%±1.6%
TESTE B	Subconjunto 1	87.4%±0.8%	87.5%±0.9%	87.4%±0.8%	87%±0.8%
	Subconjunto 2	87.2%±1.1%	87.1%±1%	87.2%±1.1%	86.9%±1%
	Subconjunto 3	87%±1.1%	86.9%±1.1%	87%±1.1%	86.7%±1.1%
	Subconjunto 4	76.5%±1.8%	76.6%±1.8%	76.5%±1.8%	76.3%±1.8%
	Subconjunto 5	84.3%±1.6%	84.3%±1.6%	84.3%±1.6%	84%±1.6%

Fonte: Autor (2022)

e Tabela 10.

Ao comparar os dois pesos, uniforme e por distância, aquele que obteve melhor resultado foi o peso uniforme quando da aplicação do teste A. Entretanto, no teste B, o peso de melhor resultado foi o peso por distância. Isso posto, para obtenção de melhores resultados no método kNN, é necessário aplicar o peso por distância com k equivalente a 3, utilizando a técnica *SMOTE* por meio do subconjunto 1, que obteve média de 88,8% na acurácia, na precisão e no recall, e 88,4% no f1-score, com desvio padrão aproximado de 1,1% nas quatro métricas. Em razão das análises realizadas em ambos os testes, A e B, no método kNN, podemos verificar que todos os resultados ultrapassaram o percentual de 75%.

Figura 14 – Diagrama de Boxplot no Método kNN - Peso por Distância | k = 3



Fonte: Autor (2022)

Na Figura 14 temos a demonstração de um diagrama de Boxplot no método kNN, aplicando peso por distância com k igual a 3, no qual podem ser observadas as variações dentro dos subconjuntos de dados. Assim, verifica-se que na técnica sem *SMOTE*, a variação é menor entre os subconjuntos. Ao aplicar a técnica com *SMOTE* no subconjunto de dados, se obteve os melhores resultados, no entanto, as variações são maiores.

4.1.2 Análise das métricas no método de Redes Neurais Artificiais com arquitetura MLP

Para o método RNA com arquitetura MLP foram realizadas três análises, no qual é variado o número de camadas ocultas e a quantidade de neurônios em cada camada ocultada. Foram realizados dois testes, A e B, na mesma forma que realizado para o método kNN, conforme ilustram-se das Tabela 11,Tabela 12 e Tabela 13.

Tabela 11 – Média dos Resultados do Método RNA com arquitetura MLP

		Acurácia	Precisão	Recall	F1-score
TESTE A	Subconjunto 1	75.7%±7%	77%±3.3%	75.7%±7%	74.3%±6.4%
	Subconjunto 2	77.5%±2.6%	73.2%±2.2%	77.5%±2.6%	74.5%±1.1%
	Subconjunto 3	74.5%±4.3%	71.6%±2%	74.5%±4.3%	71.3%±4.1%
	Subconjunto 4	37.7%±11.7%	23.1%±15.1%	37.7%±11.7%	25.2%±10%
	Subconjunto 5	73.9%±4.9%	78.2%±3.2%	73.9%±4.9%	73.4%±5.7%
TESTE B	Subconjunto 1	74.3%±4.6%	76.5%±3.6%	74.3%±4.6%	73%±5.5%
	Subconjunto 2	69.2%±3.9%	73.1%±6.8%	69.2%±3.9%	66.1%±5%
	Subconjunto 3	78.2%±3.5%	80.1%±2.1%	78.2%±3.5%	77.2%±4.8%
	Subconjunto 4	34.1%±3.4%	17.5%±7.3%	34.1%±3.4%	19.7%±2.2%
	Subconjunto 5	65.7%±5%	65.7%±9.5%	65.7%±5%	62.1%±7.5%

Fonte: Autor (2022)

Tabela 12 – Média dos Resultados do Método RNA com arquitetura MLP

		Acurácia	Precisão	Recall	F1-score
TESTE A	Subconjunto 1	79.5%±1.4%	78.2%±3.7%	79.5%±1.4%	77.2%±1.9%
	Subconjunto 2	76.4%±5.4%	75.4%±4.2%	76.4%±5.4%	73.7%±4.9%
	Subconjunto 3	76.5%±5%	74.8%±3.9%	76.5%±5%	73.8%±4.4%
	Subconjunto 4	42.3%±5.7%	22%±9.9%	42.3%±5.7%	27.5%±7.1%
	Subconjunto 5	77.6%±3.6%	75.7%±5.2%	77.6%±3.6%	75.4%±3.3%
TESTE B	Subconjunto 1	77%±3.4%	78.6%±3%	77%±3.4%	76.5%±3.7%
	Subconjunto 2	74.8%±6.4%	78.3%±4.1%	74.8%±6.4%	72.3%±8.2%
	Subconjunto 3	75.5%±4.2%	76.9%±5.7%	75.5%±4.2%	73.6%±5.6%
	Subconjunto 4	25.7%±4.8%	15.5%±12.5%	25.7%±4.8%	13.9%±4.8%
	Subconjunto 5	62.7%±8.8%	67.4%±10.5%	62.7%±8.8%	59.2%±11.4%

Fonte: Autor (2022)

Assim, na Tabela 11 é possível conferir a aplicação de uma camada oculta

Tabela 13 – Média dos Resultados do Método RNA com arquitetura MLP

		Acurácia	Precisão	Recall	F1-score
TESTE A	Subconjunto 1	75.4%±6.2%	74.3%±3.2%	75.4%±6.2%	72.4%±5.9%
	Subconjunto 2	72.9%±4.5%	76.3%±4.2%	72.9%±4.5%	72%±4.7%
	Subconjunto 3	74.4%±4.7%	75%±3.5%	74.4%±4.7%	72.8%±3.7%
	Subconjunto 4	42.9%±1%	21.6%±10.4%	42.9%±1%	26.5%±3%
	Subconjunto 5	72.7%±8%	77.4%±4.5%	72.7%±8%	71%±7.9%
TESTE B	Subconjunto 1	74.2%±3.7%	77.3%±2.4%	74.2%±3.7%	73.3%±4%
	Subconjunto 2	77.3%±4.1%	77.5%±6.2%	77.3%±4.1%	75.8%±5.8%
	Subconjunto 3	78.3%±3.6%	78.7%±5.6%	78.3%±3.6%	76.6%±5.3%
	Subconjunto 4	30.2%±7.3%	15.9%±7.3%	30.2%±7.3%	17.5%±6.7%
	Subconjunto 5	59.9%±3.6%	59.6%±11.1%	59.9%±3.6%	54.9%±5.9%

Fonte: Autor (2022)

que contêm 100 neurônios. Na Tabela 11 observa-se que ao aplicar o teste A, técnica sem *SMOTE*, o melhor resultado foi obtido no subconjunto 1. Já para o teste B, que aplica a técnica *SMOTE*, o subconjunto 3 foi o que obteve os melhores resultados. Ao compará-los os testes, o teste A obteve melhores resultados em sua maioria. Contudo, o subconjunto 3, que contém *Features* de Vértices e Arestas, o teste B foi o que se sobressaiu com média de 78,2% na acurácia e no recall, 80,1% na precisão, e 77,2% no f1-score. Ao comparar o desvio padrão nos métodos kNN e MLP, observa-se que no MLP houve um aumento, inclusive, ao aplicar o subconjunto 4, verifica-se desvio maiores que 15%.

Na Tabela 12 foi aplicado o método RNA com arquitetura MLP, com três camadas ocultas seguindo a ordem de 25, 50 e 25 neurônios. No teste A, no qual não é aplicado a técnica *SMOTE*, verifica-se que o subconjunto 1 obteve melhores resultados, já para o teste B, com *SMOTE*, o subconjunto 1 também obteve melhores resultados. Quando comparamos os dois testes, identifica-se que todos os subconjuntos obtiveram os melhores resultados sem aplicar a técnica do *SMOTE*.

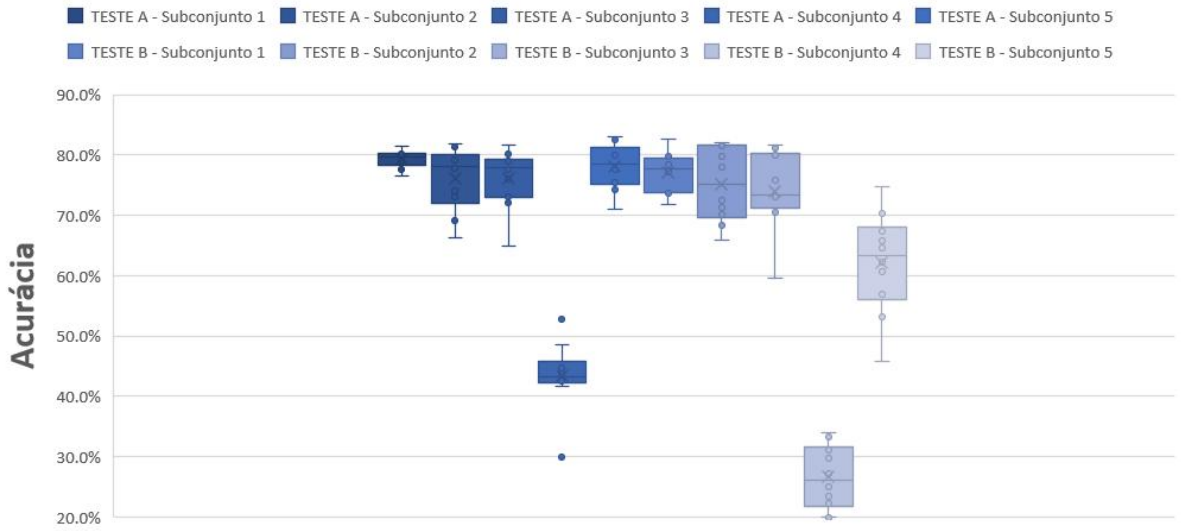
Por fim e ainda no método RNA com arquitetura MLP, ao observar a Tabela 13, com aplicação de duas camadas ocultas com 50 neurônios cada, temos o subconjunto 1 no teste A com melhores resultados. Já no teste B, os melhores resultados foram observados no subconjunto 3. Ao comparar os testes, o teste A obteve os melhores resultados com os subconjuntos 1, 4 e 5, e o teste B obteve os melhores resultados com os subconjuntos 2 e 3.

Com as análises das Tabela 11, Tabela 12 e Tabela 13, podemos concluir que, para obter o melhor resultado com o método RNA com arquitetura MLP, sugere-se a aplicação de três camadas ocultas com 25, 50 e 25 neurônios para o subconjunto 1 do teste A.

Os resultados obtidos no método RNA com arquitetura MLP foram bem

diferentes daqueles realizados no método kNN, isso porque no método RNA com arquitetura MLP os resultados foram piores. Ao observar os dois testes detalhadamente, em geral, o teste A obteve os melhores resultados. No entanto, o melhor resultado foi aquele obtido no teste B, quando da aplicação do subconjunto 3, no qual se obteve 78,2% na acurácia e no recall, 80,2% na precisão e 77,2% no f1-score.

Figura 15 – Diagrama de Boxplot no Método RNA com arquitetura MLP - três camadas ocultas com 25, 50 e 25 neurônios



Fonte: Autor (2022)

Na Figura 15 temos a demonstração de um diagrama de Boxplot no método RNA com arquitetura MLP, aplicando três camadas ocultas com 25, 50 e 25 neurônios respectivamente, no qual podem ser observadas as variações dentro dos subconjuntos de dados. Assim, verifica-se que na técnica sem *SMOTE*, a variação é menor entre os subconjuntos, excepto no subconjunto 4 que contém Features Complexas. Ao aplicar a técnica com *SMOTE* no subconjunto de dados, se obteve variações maiores, e no subconjunto 4 tem valores abaixo da média. Talvez pelo fato do subconjunto 4 possuir só features complexas, o resultado não seja satisfatório nesse subconjunto dados.

4.1.3 Análise das métricas no método Floresta Aleatória

Para o método de FA foram realizadas três análises, no qual é variado o número de árvores de decisão. Foram realizados dois testes, A e B, na mesma forma que realizado para o método anteriores, conforme extrai-se das Tabela 14,Tabela 15 e Tabela 16.

Na Tabela 14 é possível identificar a aplicação do método FA que contêm 100 árvores. Nessa tabela observa-se que ao aplicar o teste A, técnica sem *SMOTE*, os melhores resultados foi obtido no subconjunto 1 e no subconjunto 2, que corresponde a Features de Vértices e Features de Arestas respetivamente. Já para o teste B, que

Tabela 14 – Média dos Resultados do Método Floresta Aleatória com 100 árvores

		Acurácia	Precisão	Recall	F1-score
TESTE A	Subconjunto 1	78.8%±1.5%	77.9%±1.6%	78.8%±1.5%	77.9%±1.5%
	Subconjunto 2	78.8%±1.5%	77.9%±1.6%	78.8%±1.5%	77.9%±1.5%
	Subconjunto 3	78.7%±1.8%	77.8%±1.9%	78.7%±1.8%	77.9%±1.9%
	Subconjunto 4	78.4%±1.3%	77.9%±1.4%	78.4%±1.3%	77.9%±1.3%
	Subconjunto 5	78.6%±1.6%	77.9%±1.7%	78.6%±1.6%	77.9%±1.7%
TESTE B	Subconjunto 1	88.7%±1.3%	88.7%±1.3%	88.7%±1.3%	88.5%±1.3%
	Subconjunto 2	89.5%±0.7%	89.5%±0.7%	89.5%±0.7%	89.3%±0.8%
	Subconjunto 3	90%±0.6%	90.1%±0.5%	90%±0.6%	89.8%±0.6%
	Subconjunto 4	87.5%±1.2%	87.5%±1.2%	87.5%±1.2%	87.3%±1.2%
	Subconjunto 5	89.7%±0.7%	89.8%±0.7%	89.7%±0.7%	89.6%±0.7%

Fonte: Autor (2022)

Tabela 15 – Média dos Resultados do Método Floresta Aleatória com 50 árvores

		Acurácia	Precisão	Recall	F1-score
TESTE A	Subconjunto 1	78.6%±1.8%	77.6%±1.9%	78.6%±1.8%	77.7%±1.8%
	Subconjunto 2	79.1%±1.8%	78.2%±2%	79.1%±1.8%	78.2%±1.7%
	Subconjunto 3	78.5%±1.4%	77.5%±1.5%	78.5%±1.4%	77.6%±1.4%
	Subconjunto 4	78.4%±1.5%	77.8%±1.5%	78.4%±1.5%	77.8%±1.5%
	Subconjunto 5	78.6%±1.5%	77.7%±1.7%	78.6%±1.5%	77.7%±1.6%
TESTE B	Subconjunto 1	88.5%±0.6%	88.6%±0.5%	88.5%±0.6%	88.4%±0.6%
	Subconjunto 2	89.3%±0.9%	89.4%±0.9%	89.3%±0.9%	89.2%±0.9%
	Subconjunto 3	89.8%±0.9%	89.9%±0.8%	89.8%±0.9%	89.7%±0.9%
	Subconjunto 4	87.8%±0.4%	87.9%±0.4%	87.8%±0.4%	87.7%±0.4%
	Subconjunto 5	89.8%±0.8%	89.9%±0.8%	89.8%±0.8%	89.7%±0.8%

Fonte: Autor (2022)

Tabela 16 – Média dos Resultados do Método Floresta Aleatória com 150 árvores

		Acurácia	Precisão	Recall	F1-score
TESTE A	Subconjunto 1	78.8%±1.6%	77.9%±1.7%	78.8%±1.6%	77.9%±1.6%
	Subconjunto 2	78.9%±1.9%	78.2%±2.2%	78.9%±1.9%	78.1%±1.9%
	Subconjunto 3	78.3%±2%	77.4%±2.2%	78.3%±2%	77.5%±2.1%
	Subconjunto 4	78%±1.2%	77.6%±1.2%	78%±1.2%	77.5%±1.2%
	Subconjunto 5	78.5%±2%	77.8%±2.1%	78.5%±2%	77.8%±2%
TESTE B	Subconjunto 1	89.2%±0.9%	89.3%±0.9%	89.2%±0.9%	89%±0.9%
	Subconjunto 2	89.5%±0.7%	89.6%±0.6%	89.5%±0.7%	89.4%±0.7%
	Subconjunto 3	89.8%±0.7%	89.9%±0.7%	89.8%±0.7%	89.7%±0.8%
	Subconjunto 4	87.5%±0.9%	87.4%±0.9%	87.5%±0.9%	87.3%±0.9%
	Subconjunto 5	90.3%±0.6%	90.4%±0.7%	90.3%±0.6%	90.1%±0.6%

Fonte: Autor (2022)

aplica a técnica *SMOTE*, o subconjunto 3 foi o que obteve os melhores resultados. Ao compará-los, o teste B obteve melhores resultados. Contudo, o subconjunto 3, que contém Features de Vértices e Arestas, no teste B foi o que se sobressaiu com média de 90% na acurácia e no recall, 90,1% na precisão, e 89,8% no f1-score, com um desvio padrão aproximado de 0,6%.

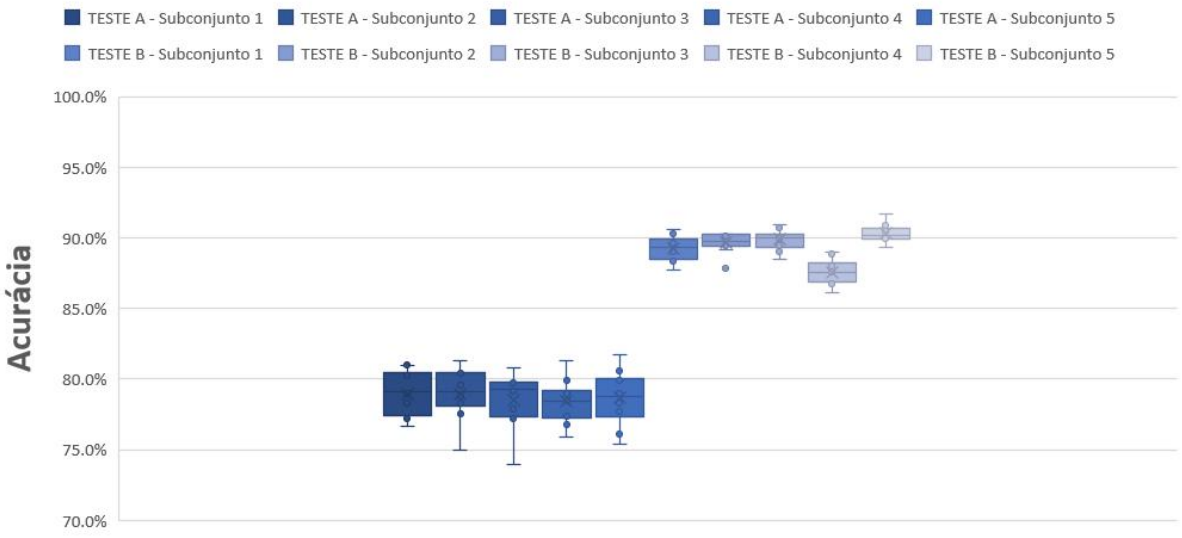
Na Tabela 15 foi aplicado o método FA que contêm 50 árvores. No teste A, no qual não é aplicado a técnica *SMOTE*, verifica-se que o subconjunto 2 obteve melhores resultados, já para o teste B, com *SMOTE*, o subconjunto 3 e o subconjunto 5 obteve os melhores resultados. Quando comparamos os dois testes, identifica-se que todos os subconjuntos obtiveram os melhores resultados ao aplicar a técnica do *SMOTE*.

Por fim e ainda no método FA, ao observar a Tabela 16, com 150 árvores, temos o subconjunto 2 no teste A com melhores resultados. Já no teste B, os melhores resultados foram observados no subconjunto 5. Ao comparar os testes, o teste B obteve os melhores resultados em todos os subconjuntos.

Com as análises das Tabela 14, Tabela 15 e Tabela 16, podemos concluir que, para obter o melhor resultado com o método FA, sugere-se a aplicação do método com 150 árvores, com o subconjunto 5 do teste B.

Ao observar os dois testes detalhadamente, em geral, o teste B obteve os melhores resultados. O melhor resultado foi aquele obtido no método com 150 árvores no teste B, quando da aplicação do subconjunto 5, no qual se obteve 90,3% na acurácia, 90,4% no recall, 90,3% na precisão e 90,1% no f1-score. Quanto à comparação em relação aos outros métodos realizados, os resultados obtidos na floresta aleatória foram maiores, considerando que não se obteve resultado abaixo de 77,9%.

Figura 16 – Diagrama de Boxplot no Método FA com 150 árvores



Fonte: Autor (2022)

Na Figura 16 tem-se a ilustração de um diagrama de Boxplot no método FA,

aplicando 150 árvores de decisão, no qual podem ser observadas as variações dentro dos subconjuntos de dados. Assim, verifica-se que na técnica sem *SMOTE*, a variação é maior entre os subconjuntos. Ao aplicar a técnica com *SMOTE* no subconjunto de dados, se obteve variações menores, e resultados melhores.

4.2 ANÁLISE FINAL DOS RESULTADOS

Como é possível observar, o modelo que obteve melhores resultados para todas as métricas, foi o que utilizou o algoritmo de Floresta Aleatória (FA), seguido pelos algoritmos K-Nearest Neighbors (kNN) e Redes Neurais Artificiais com arquitetura MLP aplicando os parâmetros do teste B, principalmente o método *SMOTE*. A maioria das métricas ficaram próximas ou acima de 80% para os 3 modelos mencionados, inclusive as métricas de acurácia, precisão, recall e F-measure. Isso quer dizer que a taxa de acerto dos modelos de AM são satisfatórios.

Com isso, concluímos que dentre os modelos de AM para classificação gerados, o modelo de *Machine Learning* que utilizou o método de FA no teste é o melhor, pois este obteve os melhores valores nas 3 métricas de classificação utilizadas, embora as diferenças entre o RNA, kNN sejam poucas significativas.

5 CONCLUSÕES

Neste trabalho foi avaliada uma abordagem para construir um sistema que pudesse prever a melhor meta-heurística de otimização a ser usada em uma determinada instância de TSP usando *Meta Learning*.

Inicialmente destacou-se sobre o uso comum das ferramentas computacionais que utilizam métodos heurísticos e meta-heurísticos, já que esses tipos de método trazem soluções apropriadas em um tempo menor para a necessidade de uma empresa. Como com o aumento da complexidade das instâncias de TSP vê-se a necessidade de procurar melhores soluções, foram abordados os conceitos principais e para adentrado nas meta-heurísticas. Durante o desenvolvimento, foram tratadas ainda cinco heurísticas, quais sejam: Tabu Search, Simulated Annealing, GRASP, Algoritmos Genéticos e Ant Colony Optimization. Discorreu-se brevemente sobre cada uma delas.

O conjunto de Meta Dados foi composto por por instâncias de TSP totalmente conectadas obtidas por Gastineau (2018), avaliadas por cada uma das cinco heurísticas: Tabu Search, Simulated Annealing, GRASP, Algoritmos Genéticos e Ant Colony Optimization. Para posteriormente complementar anexando com os dados gerados pelas *Meta Features* de cada instância de TSP. Ao longo do processo de pesquisa, muitas desafios foram encontrados, principalmente nas áreas de implementação de features e o desbalanceamento de classes.

Após breve explicação sobre o Aprendizado de Máquina, escolheu-se os seguintes algoritmos de AM de classificação para criar nosso metamodelo de *Meta Learning*: (1) Rede Neural Artifical; (2) k Nearest Neightbors; e, (3) Floresta Aleatória. Cada um dos algoritmos foi explanado durante o desenvolvimento. Para a comparação dos 3 modelos de aprendizado de máquina, foram utilizados quatro métricas de classificação, as quais foram: acurácia, precisão, recall, F-measure. Considera-se ainda, dois tipos de teste, os dois utilizando processo de validação cruzada estratificada de 10 etapas onde em cada etapa as quatro métricas foram calculados e ao final foi calculada a média e o desvio padrão de cada uma, e outro especificamente utilizando o método de *SMOTE*.

Após as análises realizadas, o modelo de AM que obteve os melhores resultados foi o que utilizou o algoritmo de Floresta Aleatória que ficou com acurácia média de 90%, precisão média de 90%, recall médio de 90% e F-measure médio de 90%. Os valores obtidos demonstram que o modelo de Aprendizado de Máquina é pouco suscetível a erros por falso negativo e falso positivo. Além disso, consegue ainda separar bem as meta-heurísticas adequadas para cada instância de grafo.

Uma sugestão de abordagem para trabalhos/pesquisas futuras seria identificar

novas features que possam auxiliar na precisão do modelo e avaliar quais features são mais relevantes para se obter um melhor desempenho na recomendação de meta-heurísticas.

Outro ponto interessante de melhoria é a utilização de outros algoritmos de AM para obter melhores resultados e ampliar o tamanho da bases de dados do treinamento para avaliar o seu impacto no desempenho final da recomendação.

Para finalizar, um ponto interessante para ser explorado em trabalhos futuros é considerar uma base de dados com características diferentes, já que os dados disponibilizados por Gastineau (2018) são unicamente grafos totalmente conectados e encontrar diferentes técnicas para calibrar os parâmetros das MH .

REFERÊNCIAS

ALPAYDIN, E. Introduction to machine learning. **MIT Press**, p. 9–10, 2010.

BAILER-JONES, D. M.; BAILER-JONES, C. A. Modelling data: Analogies in neural networks, simulated annealing and genetic algorithms. **New York: Kluwer Academic/Plenum Publishers**, 2002.

BALLOU, R. H. Gerenciamento da cadeira de suprimentos/logística empresarial. **Bookman**, Rio de Janeiro, 2006.

BANG-JENSEN, J.; GUTIN, G.; YEO, A. When the greedy algorithm fails. **Discrete Optimization**, v. 1, n. 2, p. 121–127, 2004.

BARBOZA, A. O. **Simulação e técnicas da computação evolucionária aplicadas a problemas de programação linear inteira mista**. Tese (Doutorado em Engenharia Elétrica e Informática Industrial) — Universidade Tecnológica Federal do Paraná, Curitiba, 2005.

BARELLA, V. H. **Técnicas para o problema de dados desbalanceados me classificação hierárquica**. Dissertação (Mestrado em Ciências da Computação e Matemática Computacional) — Universidade de São Paulo, São Carlos, 2015.

BASAVARAJU, A. et al. A machine learning approach to road surface anomaly assessment using smartphone sensors. **IEEE Sensors Journal**, p. 1–1, 2019.

BASU, S. Tabu search implementation on traveling salesman problem and its variations: A literature survey. **American Journal of Operations Research**, v. 02, 2012.

BEKTAS, T. The multiple traveling salesman problem: an overview of formulations and solution procedure. **Omega**, v. 34, n. 3, p. 209–219, 2006.

BERLANGA, V.; RUBIO, M.; VILÀ, R. Cómo aplicar árboles de decisión en spss. **REIRE. Revista d'Innovació i Recerca en Educació**, ICE Universitat de Barcelona, v. 6, n. 1, p. 65–79, 2013.

BETTINGER, P. et al. Eight heuristic planning techniques applied to three increasingly difficult wildlife planning problems. **Silva Fennica**, v. 36, p. 561–584, 2002.

BIANCHI, L.; GUTJAHR, W. J. A survey on metaheuristics for stochastic combinatorial optimization. **Natural Computing**, v. 8, n. 3, p. 239—-287, 2009.

BODIN, L. Routing and scheduling of vehicles and crews. **Pergamon Press**, v. 10, n. 2, 1983.

BONDY, J. A.; MURTY, U. S. **Graph theory with applications**. 2nd. ed. USA: The Macmillan Press, 1976.

BRAZDIL, P.; SOARES, C.; VILALTA, R. Metalearning: Applications to data mining. **Springer**, Berlin, 2009.

BUHAGIAR, J.; AZZOPARDI, G. **Automatic Segmentation of Indoor and Outdoor Scenes from Visual Lifelogging**. [S.l.], 2017.

BZDOK, D.; KRZYWINSKI, M.; ALTMAN, N. Machine learning: Supervised methods. **Nature Methods**, v. 15, 2018.

CHAWLA, N. et al. SMOTE: Synthetic minority over-sampling technique. **Journal of Artificial Intelligence Research**, AI Access Foundation, v. 16, p. 321–357, 2002.

CHENG, D. et al. knn algorithm with data-driven k value. Springer International Publishing, p. 499–512, 2014.

COOPER, J.; NICOLESCU, R. The hamiltonian cycle and travelling salesman problems in cp systems. **Fundamenta Informaticae**, v. 164, n. 2-3, p. 157–180, 2019.

DAVIS, L. Genetic algorithms and simulated annealing. **Pitman**, p. 216, 1987.

DIETTERICH, T. G. Ensemble learning.the handbook of brain theory andneural network. Malta, v. 2, p. 110–125, 2002.

DIMITRIADIS, S.; LIPARAS, D. How random is the random forest? rf algorithm on the service of structural imaging biomarkers for ad: from adni database. **Neural Regeneration Research**, v. 13, p. 962–970, 2018.

DORIGO, M.; GAMBARDELLA, L. Ant colony system: a cooperative learning approach to the traveling salesman problem. **IEEE Transactions on Evolutionary Computation**, v. 1, n. 1, p. 53—66, 1997.

DORIGO, M.; STUTZLE, T. Ant colony optimization: Overview and recent advances, handbook of metaheuristics. **Springer, Cham**, v. 272, 2019.

DORIGO, M.; STUTZLE, T.; BIRATTARI, M. Ant colony optimization. **IEEE Computational Intelligence Magazine**, v. 1, n. 4, p. 28–39, 2006.

FEO, T. A.; RESENDE, M. G. C. **Greedy Randomized Adaptive Search Procedures**. [S.l.], 1995. v. 6, n. 2, 109—133 p.

FESTA, P.; RESENDE, M. **Grasp: An Annotated Bibliography, Essays and Surveys in Metaheuristics**. 1. ed. Boston: Springer, 2002.

FONSECA, T. H. **Heurística para sintonização de meta-heurísticas aplicada ao Problema de FlowShop Permutacional**. Dissertação (Mestrado em Ciências da Computação) — Universidade Federal do Maranhão, São Luís, 2018.

FREDDO, A. R.; BRITO, R. C. Implementação da metaheurística grasp para o problema do caixeiro viajante simétrico. **Universidade Federal do Paraná, Tópicos em Inteligência Artificial**, 2007.

GABOW, H. et al. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. **Combinatorica**, Springer, v. 6, n. 2, p. 109–122, 1986.

GAMARRA, M.; VERA, H. Desarrollo de un software para el análisis socio económico de los estudiantes de la upac. Universidad Peruana Austral del Cusco, 2018.

GASTINEAU, A. Tsp meta-learning using deep neural networks. **Rose-Hulman Institute of Technology Computer Science and Mathematics Departments**, Terre Haute, 2018.

GERON, A. **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems**. 2. ed. [S.l.]: O'Reilly Media, Inc., 2019.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. **Comput. Open Res.**, v. 13, n. 3, p. 533–549, 1986.

GLOVER, F.; LAGUNA, M. Tabu search. **Boston: Kluwer academic publishers.**, v. 22, 1997.

GLOVER, F.; WERRA, D. A user's guide to tabu search. **Annals of Operations Research**, v. 41, n. 1, p. 1—28, 1993.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. 13. ed. Boston: Addison-Wesley Professional, 1988.

GRANDINI, M.; BAGLI, E.; VISANI, G. Metrics for multi-class classification: an overview. **arXiv preprint arXiv:2008.05756**, 2020.

HASTIE, T.; TIBSHIRANI, R. The elements of statistical learning. p. 12–19, 2017.

HAWKINS, D. M. **The problem of overfitting**. [S.l.], 2004. v. 44, n. 1, 1–12 p.

HAYKIN, S. **Neural Networks: A Comprehensive Foundation**. 2nd. ed. USA: Prentice Hall PTR, 1998.

HILLIER, F. S.; LEIBERMAN, G. J. **Introdução a pesquisa operacional**. Sao Paulo, 2010.

HILLIER, F. S.; LIEBERMAN, G. J. **Introdução à pesquisa operaciona**. 8th. ed. São Paulo: The McGraw-Hill Companies, 2010.

HOLLAND, J. H. Genetic algorithms and the optimal allocation of trials. **SIAM Journal on Computing**, v. 2, n. 2, p. 88—-105, 1973.

IGNACIO, A. A. V.; GALVãO, R. D. Métodos heurísticos num entorno paralelo. **Simpósio Brasileiro de Pesquisa Operacional**, v. 32, p. 769–788, 2000.

ISMAIL, A. Domino algorithm: a novel constructive heuristics for traveling salesman problem. **IOP Conference Series: Materials Science and Engineering**, v. 528, n. 1, may 2019.

JANKOWSKI, N.; GRABCZEWSKI, K.; WŁODZISŁAW, D. Meta-learning in computational intelligence. Springer-Verlag Berlin Heidelberg, 2011.

JORDAN, M. I.; BISHOP, C. M. Neural networks. **Computer Science Handbook**, Boca Raton, v. 2, 2004.

KALOUSIS, A. **Algorithm Seletion via Meta Learning**. Tese (Doutorado em Ciências de Computação) — Centre Universiteire d'Informatique, Université de Genève, 2002.

KANDA, J.; CARVALHO, A. de; SOARES, C. Selection of algorithms to solve traveling salesman problems using meta-learning. **International Journal of Hybrid Intelligent Systems**, v. 8, p. 1—-13, 2011.

KANDA, J.; SOARES, C.; BRAZDIL, P. Meta-learning to select the best meta-heuristic for the traveling salesman problem: A comparison of meta-features,. **Neurocomputing**, v. 205, p. 393–406, 2016.

KIRKPATRICK, S.; VECCHI, M. P. Optimization by simulated annealing. **Science**, v. 220, n. 4598, p. 671—-680, 1983.

KOHAVI, R. **A study of cross-validation and bootstrap for accuracy estimation and model selection**. [S.l.], 1995. v. 14, n. 2, 1137–1145 p.

KRAMER, R. H. **uma abordagem heurística, para o Pollution-Routing Problem**. Dissertação (Mestrado em Engenharia de Produção) — Universidade Federal de Paraíba, João Pessoa, 2014.

LIAKOS, K. G. et al. Machine learning in agriculture: A review. **Sensors**, v. 18, n. 8, 2018.

MEDOZA, J. J. **Diseño de Algoritmos Heurísticos y Metaheurísticos eficientes para resolver el Problema del Agente Viajero**. Tese (Doutorado em Matemática Aplicada) — Universidad Nacional Autónoma de Nicaragua, 2017.

METROPOLIS, N. Equation of state calculations by fast computing machines. **The Journal of Chemical Physics**, v. 21, n. 6, 1953.

MICO, M. L. **Algoritmos de búsqueda de vecinos más próximos en espacios métricos**. Tese (Doutorado em Ciências de Computação) — Universidad Politécnica de Valencia, 1996.

MITCHELL, M. **An Introduction to Genetic Algorithms**. [S.l.]: The MIT Press, 1998.

MITCHELL, T. **Machine Learning**. [S.l.]: McGraw-Hill, 1997. (McGraw-Hill International Editions).

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. **Computers Ops. Research**, v. 4, n. 11, p. 1097–1100, 1997.

NEMHAUSER, G. L.; WOLSEY, L. A. **Integer and combinatorial optimization.** New York: Wiley, 1988. 785 p.

OCAMPO, E.; SANTA, J.; GRANADA, M. Solución del problema de ruteamiento de vehículos en la distribución de papa en colombia. **Scientia et Technica**, v. 18, n. 1, 2013.

OLIVEIRA, M.; PEREIRA, T.; SOUZA, J. ComparaÇÃo entre os mÉtodos simulated annealing e luus jaakola na resoluÇÃo do problema de estabilidade termodinÂmica. **Revista Mundi Engenharia, Tecnologia e Gestão (ISSN: 2525-4782)**, v. 3, 05 2018.

OLIVERA, A. R. **Comparação de algoritmos de aprendizagem de máquina para construção de modelos preditivos de diabetes não diagnosticado**. Dissertação (Mestrado em Ciências da Computação) — Universidade de Rio Grande do Sul, Porto Alegre, 2016.

OREA, S.; VARGAS, A.; ALONSO, M. Minería de datos: predicción de la deserción escolar mediante el algoritmo de árboles de decisión y el algoritmo de los k vecinos más cercanos. **Ene**, v. 779, n. 73, p. 33, 2005.

OSMAN, I. H.; KELLY, J. P. Meta-heuristics: Theory and applications. **Kluwer Academic**, 1996.

PACIELLO, J. et al. Algoritmos de optimización multi-objetivos basados en colonias de hormigas. 2022.

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

PRUDENCIO, R.; SOUTO, M. de; LUDERMIR, T. Selecting machine learning algorithms using the ranking meta-learning approach. **Meta-Learning in Computational Intelligence**, 2011.

PYTHON, . S. 2021. Disponível em: www.python.org. Acesso em: 18 jul. 2021.

RABELO, R. T. **Arquitetura de hardware dedicada de uma rede neural perceptron para reconhecimento de terreno aplicado a robótica móvel**. Monografia submetida ao curso de graduação em (Engenharia Eletrônica) — Universidade Federal de Brasília, Brasília, 2014.

RAMOS, J. M. **Implementação e Análise do Problema do Caixeiro Viajante usando uma nova abordagem através dos Algoritmos Genético e Simulated Annealing**. Dissertação (Mestrado) - Universidade Federal de Santa Catarina, 2001.

RODRIGUES, F. L. et al. Metaheurística simulated annealing para solução de problemas de planejamento florestal com restrições de integridade. **Revista Árvore**, v. 28, n. 2, p. 247–256, 2004.

ROSENKRANTZ, D.; STEARNS, R.; LEWIS, P. An analysis of several heuristics for the traveling salesman problem. **SIAM journal on computing**, v. 6, n. 3, p. 563–581, 1977.

SALAMON, P.; SIBANI, P.; FROST, R. **Facts, Conjectures, and Improvements for Simulated Annealing**. [S.l.]: Society for Industrial and Applied Mathematics, 2002.

SALTOS, D.; VILLACIS, O. Implementación de machine learning en el área de ventas de la empresa zapec sa. Universidad Católica de Santiago de Guayaquil, 2022.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. **IBM Journal of Research and Development**, v. 3, n. 3, p. 210–229, 1959.

SANTOS, I. C. **Utilização da meta-heurística simulated annealing para otimização da programação de turnos dos funcionários de uma loja varejista**. Trabalho de Conclusão de Curso (Graduação em Engenharia de Produção) — Universidade Federal de Ouro Preto, Ouro Preto, 2016.

SANTOS, M. S. **Modelos de aprendizagem de máquina para identificar o risco do trabalho escravo contemporâneo em cidades brasileiras**. Dissertação (Mestrado o Profissional em Computação Aplicada) — Universidade de Brasília,Brasília, 2020.

SANTOS, V. G. **Um Ensemble baseado em Árvores de Decisão para Predizer a Ocorrência de Aglomerados de Ônibus**. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Campina Grande, 2020.

SARUBBI, J.; LUNA, H. **A New Flow Formulation for the Minimum Latency Problem**. [S.l.], 2007.

SCHRIJVER, A. **On the history of combinatorial optimization**. Amsterdã, Holanda, 2000.

DA SILVA, D. J. **Algoritmos culturais com abordagem memética e multipopulacional aplicados a problemas de otimização**. Tese (Doutor em Engenharia Elétrica) — Universidade Federal do Para, Belém, 2012.

SILVA, J. C. **Um modulo inteligente baseado em aprendizado de máquina para treinamento de estudantes de medicina no doctraining**. Dissertação (Mestrado em Ciência da Computação) — Universidade do Estado do Rio Grande do Norte, 2020.

SINGH, A. A review on algorithms used to solve multiple travelling salesmanproblem. **International Research Journal of Engineering and Technology (IRJET)**, v. 3, n. 4, 2016.

SMARANDACHE, F. **Neutrosophic Sets and Systems, vol. 15/2017 (journal)**. [S.l.: s.n.], 2017.

SOUSA, M. M.; OCHI, L. S. Uma heuristica multi-start local search para o problema do caixeiro viajante multiplo com seleÇÃo de hoteis. **Simpósio Brasileiro de Pesquisa Operacional**, 2016.

SOUZA, B. F. **Meta-aprendizagem aplicada à classificação de dados de expressão gênica**. Tese (Doutorado em Ciências de Computação e Matemática Computacional) — Universidade de São Paulo, 2010.

SPOLTI, A. M. **ClassiĄcação de vias através de imagens aéreas usando Deep Learning**. Trabalho de Conclusão de Curso em (Bacharel de Sistemas de Informação) — Universidade Federal de Uberlândia, Uberlândia, 2018.

SUTARIA, V. Hamiltonian and eulerian cycles. **International Journal of Trend in Research and Development**, v. 3, p. 208–212, 2016.

WOZNIAK, M.; POLAP, D. On some aspects of genetic and evolutionary methods for optimization purposes. **International Journal of Electronics and Telecommunications**, v. 61, p. 7–16, 2015.

XING, L.; CAI, H. A hybrid approach combining an improved genetic algorithm and optimization strategies for the asymmetric traveling salesman problem. **Engineering Applications of Artificial Intelligence**, v. 21, n. 8, p. 1370—-1380, 2008.

XUAN-SHI, Y.; YUN, O. TSP solving utilizing improved ant colony algorithm. **Journal of Physics: Conference Series**, IOP Publishing, v. 2129, n. 1, p. 012026, 2021.

YOUSSEF, H.; SAINT, S. M. Evolutionary algorithms, simulated annealing end tabu search: a comparative study. engineering aplications of artificial inteligence. v. 14, p. 167–181, 2001.

ZAPELINI, C. Z. **Um Estudo Abrangente sobre Metaheurística, incluindo um Histórico**. Universidade de São Paulo - Instituto de Matemática e Estatística, 2009.