INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ
IFCE CAMPUS ARACATI
COORDENADORIA DE CIÊNCIA DA COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOYCE QUINTINO ALVES

AVALIAÇÃO DE DESEMPENHO DE ALGORITMOS DE *MACHINE LEARNING* PARA OTIMIZAÇÃO DE SIMULAÇÕES DE REDES DE COMPUTADORES

ARACATI-CE
2019

JOYCE QUINTINO ALVES

AVALIAÇÃO DE DESEMPENHO DE ALGORITMOS DE *MACHINE LEARNING* PARA OTIMIZAÇÃO DE SIMULAÇÕES DE REDES DE COMPUTADORES

Trabalho de Conclusão de Curso (TCC) apresentado ao curso de Bacharelado em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE - Campus Aracati, como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Orientadora: Profa. Dra. Carina Teixeira de Oliveira
Coorientador: Prof. Msc. Silas Santiago Lopes Pereira

Aracati-CE
2019

JOYCE QUINTINO ALVES

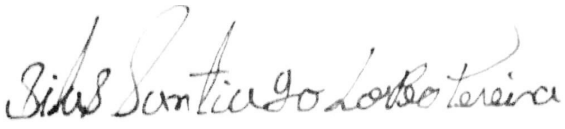
AVALIAÇÃO DE DESEMPENHO DE ALGORITMOS DE *MACHINE LEARNING* PARA OTIMIZAÇÃO DE SIMULAÇÕES DE REDES DE COMPUTADORES

Trabalho de Conclusão de Curso (TCC) apresentado ao curso de Bacharelado em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE - Campus Aracati, como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Aprovada em 03 de outubro de 2019

BANCA EXAMINADORA

Prof. Dra. Carina Teixeira de Oliveira (Orientadora) - IFCE



Prof. Msc. Silas Santiago Lopes Pereira (Coorientador) - IFCE

Prof. Dr. Reinaldo Bezerra Braga - IFCE

Prof. Msc. Raimundo Valter Costa Filho - IFCE

“Não se deve ir atrás de objetivos fáceis,
é preciso buscar o que só pode ser
alcançado por meio dos maiores esforços.”

Albert Einstein

AGRADECIMENTOS

Primeiramente, aos meus pais, Maura Quintino e Evandro Alves que me incentivaram desde criança e fizeram o possível para garantir uma boa formação.

Aos meus irmãos Alan, Ismael, Jamile e Isac pelo amor, apoio e terem tido paciência com minhas brincadeiras nos momentos de procrastinação.

Aos meus sobrinhos, em especial, meu sobrinho irmão Alysson pela companhia.

Ao meu tio querido, Edvaldo Alves pelo o seu amor incondicional e por me proporcionar momentos únicos na sua rápida passagem. Você foi o início dessa jornada.

Aos demais familiares e amigos, que se fizeram presentes, especialmente, Fabrício Ferreira que acompanhou minha trajetória desde o início.

Agradecimento mais que especial ao mestre chefe e amigo, professor Mauro Oliveira por acreditar em mim nos momentos mais difíceis, pelas reuniões, pelos vários repitas e as oportunidades que hoje fazem diferença na minha vida. Às professoras Raquel Silveira e Carina Oliveira pelo incentivo, orientação, atenção, insistência, paciência e carinho.

À orientadora e aos professores Reinaldo Braga e Silas Santiago pelas orientações, sugestões, apoio e desenvolvimento deste trabalho.

À Fundação Cearense de Apoio ao Desenvolvimento Científico e Tecnológico (FUNCAP), no âmbito do Programa de Incentivo à Interiorização e Inovação Tecnológica - BPI, edital número 09/2015 pelo apoio nos estudos.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) através do Programa Institucional de Bolsas de Iniciação em Desenvolvimento Tecnológico e Inovação - PIBITI, edital PIBITI/PRPI/2019 pelo financiamento deste trabalho.

Ao Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE) pelo ensino de qualidade.

À empresa Avicena pelas oportunidades no projeto GISSA.

Ao LAR (Laboratório de Redes e Sistemas Multimídia) pelas oportunidades e transformar as vidas de muitos alunos.

E, por fim, à todos os professores que fizeram parte dessa jornada.

“Que a força esteja com vocês!”

RESUMO

Os profissionais da área de redes de computadores enfrentam diversos desafios ao instalar, configurar e/ou avaliar um ambiente de rede. Dentre eles, está a busca constante pelo alto desempenho e escalabilidade da rede, além do baixo custo. Os simuladores de redes surgem como ferramentas computacionais capazes de auxiliar esses profissionais nesses desafios, possibilitando que eles, por exemplo, desenvolvam, analisem e aperfeiçoem protocolos de comunicação em diferentes cenários antes da implantação. Apesar das vantagens dos simuladores, existem algumas limitações relativas ao seu uso. O tempo de execução das simulações e, consequentemente, os recursos computacionais alocados, crescem à medida que os requisitos de complexidade e precisão aumentam. Neste sentido, destaca-se que as Redes do Futuro tendem a ser cada vez mais heterogêneas, densas e complexas, necessitando de maior poder computacional. Além das limitações de tempo e de recursos computacionais, muitos processos de simulação possuem como única saída *traces* sem a interpretação lógica (ex: arquivos em formato `xml`, `csv`, `dat`, etc), ou seja, que necessitam passar por outras ferramentas para análise dos resultados e, assim, gerar informações relevantes para a tomada de decisão. Como contribuição nesse sentido, este trabalho avalia o desempenho de algoritmos de *Machine Learning* (ML) na otimização de simulações de redes de computadores. O objetivo geral é treinar um conjunto de algoritmos de ML a partir de um *dataset* de resultados de simulações e, então, avaliar se os algoritmos conseguem predizer satisfatoriamente o mesmo comportamento alcançado pelas simulações a partir de um conjunto de discriminantes estatísticos obtidos para cada cenário de rede considerado. Os algoritmos estudados são o *Decision Tree*, a Rede *Multilayer Perceptron*, o *Random Forest* e a *Support Vector Regression*. Em particular, a proposta procura predizer a taxa de entrega de um conjunto de cenários de redes em malha sem fio simulados no *Network Simulator* 3 (NS-3). Como a taxa de entrega tem um comportamento contínuo, o trabalho faz uso da Regressão como tarefa de ML. Os resultados alcançados mostram que alguns algoritmos fazem suposições próximas das simulações reais, tendo alguns modelos de ML com taxa de acerto acima de 90% nas predições da taxa de entrega.

Palavras-chave: Redes de computadores, Simulação, *Machine Learning*

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplo de Tarefa de Classificação. 18
Figura 2 – Exemplo de Tarefa de Regressão. 18
Figura 3 – Estrutura de árvore criada pelo *Decision Tree*. 20
Figura 4 – n árvores geradas através de *Ensemble Learning*. 21
Figura 5 – Margem máxima para SVM. 22
Figura 6 – Exemplo da arquitetura MLP. 24
Figura 7 – Validação cruzada. 27

Figura 8 – Topologia simulada para *Machine Learning* 33
Figura 9 – Topologia simulada para validação 33

Figura 10 –Visão geral da proposta. 38
Figura 11 –Topologia *mesh* configurada para o estudo de caso. 39
Figura 12 –Resultados da Simulações no NS-3. 41
Figura 13 –Exemplo de arquivo `txt` com resultados das simulações. 43
Figura 14 –Diagrama de avaliação dos algoritmos de ML 48

Figura 15 –Simulação x Predição - Modelo: MLP 52
Figura 16 –Simulação x Predição - Modelo: DT 55
Figura 17 –Simulação x Predição - Modelo: RF 58
Figura 18 –Simulação x Predição - Modelo: SVM 60

LISTA DE TABELAS

Tabela 1 – Identificação das colunas da Tabela 1 36
Tabela 2 – Comparação entre os trabalhos relacionados 37

Tabela 3 – Valores dos parâmetros. 39
Tabela 4 – Identificação dos cenários com intervalo de 0.001 42
Tabela 5 – Identificação dos cenários com intervalo de 0.01 42
Tabela 6 – Identificação dos cenários com intervalo de 0.1 42
Tabela 7 – Atributos que compõem os *datasets* 44

Tabela 8 – Métricas de avaliação dos cenários com intervalo entre pacotes de 0.001 para MLP 50
Tabela 9 – Métricas de avaliação dos cenários com intervalo entre pacotes de 0.01 para MLP 51
Tabela 10 –Métricas de avaliação dos cenários com intervalo entre pacotes de 0.1 para MLP 51
Tabela 11 –Métricas de avaliação dos cenários com intervalo entre pacotes de 0.001 para DT 53
Tabela 12 –Métricas de avaliação dos cenários com intervalo entre pacotes de 0.01 para DT 54
Tabela 13 –Métricas de avaliação dos cenários com intervalo entre pacotes de 0.1 para DT 54
Tabela 14 –Métricas de avaliação dos cenários com intervalo entre pacotes de 0.001 para RF 56
Tabela 15 –Métricas de avaliação dos cenários com intervalo entre pacotes de 0.01 para RF 56
Tabela 16 –Métricas de avaliação dos cenários com intervalo entre pacotes de 0.1 para RF 57
Tabela 17 –Métricas de avaliação dos cenários com intervalo entre pacotes de 0.001 para SVM 59
Tabela 18 –Métricas de avaliação dos cenários com intervalo entre pacotes de 0.01 para SVM 59
Tabela 19 –Métricas de avaliação dos cenários com intervalo entre pacotes de 0.1 para SVM 59

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
ML	*Machine Learning*
NS-3	*Network Simulation 3*
ENIAC	*Electronic Numerical Integrator and Computer*
IA	*Inteligência Artificial*
SVM	*Support Vector Machine*
RBF	*Radial Basis Function*
MLP	*Multi Layer Perceptron*
MAE	*Mean Absolute Error*
MSE	*Mean Squared Error*
RMSE	*Root Mean Squared Error*
RF	*Random Forest*
TCP	*Transmission Control Protocol*
RTT	*Round Trip Time*
UDP	*User Datagram Protocol*
XML	*Extensible Markup Language*
TXT	*Text*
PCAP	*Packet Capture Program*
PCA	*Principal Component Analysis*
HTTP	*Hypertext Transfer Protocol*
CP	*Componente Principal*
DT	*Decision Tree*

SUMÁRIO

1 INTRODUÇÃO 12
1.1 Contexto Geral 12
1.2 Motivação 13
1.3 Objetivos 14
1.4 Organização do Trabalho 15

2 FUNDAMENTAÇÃO TEÓRICA 16
2.1 *Machine Learning* 16
2.1.1 Aprendizado Supervisionado 17
2.1.1.1 Decision Tree (DT) 19
2.1.1.2 Random Forest (RF) 20
2.1.1.3 Support Vector Machine (SVM) 21
2.1.1.4 Rede Multilayer Perceptron (MLP) 23
2.1.2 Ajuste de parâmetros dos algoritmos de ML 24
2.1.2.1 Grid search 25
2.1.3 Avaliação de desempenho de modelos preditivos 26
2.1.3.1 Validação Cruzada 26
2.1.3.2 Métricas de erro para modelos de regressão 27
2.1.3.3 R^2 para um modelo de regressão 28
2.2 Rede em Malha Sem Fio 30

3 TRABALHOS RELACIONADOS 32
3.1 Avaliação de desempenho do TCP usando *Machine Learning* em ambientes sem fio 32
3.2 Simulação em ambientes 34
3.3 Simulações de Redes no NS-3 35
3.4 Ferramenta TraceMetrics para simulações no NS-3 35

4 PROPOSTA 38
4.1 Etapa 1: Definição da Configuração da Rede 38
4.2 Etapa 2: Simulação no NS-3 40
4.3 Etapa 3: Preparação de dados 42
4.4 Etapa 4: Treinamento dos algoritmos de ML 45
4.4.1 Definição dos Hiperparâmetros dos Modelos Preditivos 46
4.5 Etapa 5: Avaliação dos modelos 47
4.6 Etapa 6: Comparação das simulações com os modelos 48

5 RESULTADOS 49
5.1 Avaliação de desempenho da MLP 50
5.2 Avaliação de desempenho da DT 53
5.3 Avaliação de desempenho do RF 56
5.4 Avaliação de desempenho da SVM 59

6 CONCLUSÕES E TRABALHOS FUTUROS 62

REFERÊNCIAS 63

1 INTRODUÇÃO

1.1 Contexto Geral

Os simuladores são ferramentas computacionais utilizadas em vários campos da ciência e da engenharia (LEE; SUH; CHO, 2017). Os recursos disponibilizados pelos simuladores permitem a experimentação de determinada solução antes de colocá-la em produção. Tal aspecto reduz os riscos associados à criação de uma nova solução, assim como reduz os riscos de uma mudança em uma solução já existente. Consequentemente, os simuladores possibilitam a redução significativa dos custos e do tempo de uma implantação (PRADELLA, 2013) (ABREU et al., 2017).

A prática de simulação se intensificou entre os anos de 1939 e 1945, durante a II Guerra Mundial. Computadores como o Mark I e o *Electronic Numerical Integrator and Computer* (ENIAC) da Marinha e do Exército Norte-Americano, respectivamente, foram utilizados durante esse período para a realização de cálculos que exigiam conhecimento substancial em matemática com uma maior agilidade. Assim, foram usados, por exemplo, para simular lançamentos e trajetórias táticas de mísseis. Até a década de 70, apenas grandes corporações e universidades possuíam máquinas suficientemente potentes para a execução de simulações. Além dessas limitações, havia a escassez de profissionais qualificados para o desenvolvimento de soluções especialistas. Entretanto, no final da década de 70, linhas de montagem de automóveis já utilizavam simulação para resolver problemas de segurança e otimizar a produção. Nos anos 90, o uso de simuladores tornou-se mais intenso graças ao barateamento de equipamentos, ao aumento da velocidade de processamento das máquinas, além do surgimento de sistemas e ferramentas para execução de simulações (BALADEZ, 2009).

No Brasil, existe o Sistema Nacional de Processamento de Alto Desempenho (SINAPAD), uma rede de centros de computação de alto desempenho, geograficamente distribuídos, instituída pelo Ministério da Ciência, Tecnologia, Inovações e Comunicações (MCTIC). No total, são nove unidades, denominadas Centros Nacionais de Processamento de Alto Desempenho (CENAPADs), que são operadas pelo Laboratório Nacional de Computação Científica (LNCC) e por diversas universidades e institutos brasileiros. A infra-estrutura computacional do SINAPAD é utilizada para simulações de pesquisa de alto nível nas áreas de medicina, química, biologia, computação, matemática, física, dentre outras (SINAPAD, 2019).

Assim, dada a importância dos simuladores, eles hoje são utilizados nos mais

diversos tipos de sistemas e aplicações. Na bioinformática, usa-se simuladores para estudar o comportamento dinâmico de moléculas em um sistema biológico (LEE; SUH; CHO, 2017). Também são usados simuladores para reproduzir o voo de aeronaves de forma precisa, auxiliando pilotos em treinamentos (SANTOS; SILVEIRA, 2019). Outro exemplo é o uso de simuladores no apoio à tomada de decisão de investimentos financeiros. (OLIVEIRA, 2017).

1.2 Motivação

Os simuladores também são bastante explorados na área de redes de computadores. Existem vários simuladores que reproduzem o comportamento de uma rede de comunicação real. Alguns exemplos mais populares são o Cisco *Packet Tracer*[1], OMNet++[2], *Optimized Network Engineering Tool* (OPNET)[3], QualNet[4], *Network Simulator* 2 (NS-2)[5] e *Network Simulator* 3 (NS-3)[6]. Este último é utilizado no estudo de caso deste trabalho. Esses e outros inúmeros simuladores de redes fornecem ambientes e ferramentas para os mais diversos fins, partindo desde o simples auxílio ao estudo de conceitos de redes, muitas vezes disponibilizando interfaces gráficas animadas para entendimento preciso do funcionamento da rede, até o desenvolvimento, implantação e análise de desempenho de complexos protocolos e arquiteturas. No mais, também permitem simular diferentes tipos de aplicações, modelar o tipo de tráfego e a interação entre as diferentes entidades da rede (ex: roteadores, comutadores, pontos de acesso, etc).

Apesar dos vários benefícios do uso de simuladores na área de redes de computadores, eles também apresentam limitações relativas ao seu uso. O tempo de execução das simulações e, consequentemente, os recursos computacionais alocados para execução das simulações crescem à medida que os requisitos de complexidade e precisão da rede aumentam. Nesse contexto, destaca-se que as Redes do Futuro (*Networks of the Future* - NoF) tendem a ser cada vez mais heterogêneas, densas e complexas. Como exemplos de ambiente NoF existem as *smart cities*, Internet das Coisas (IoT), redes *machine-to-machine*, sistemas de transporte inteligente, *green computing*, etc (LOUREIRO, 2016). Portanto, realizar experimentos simulando esses ambientes muitas vezes exige recursos computacionais de processamento e armazenamento que nem sempre estão disponíveis. Quando disponíveis, podem levar

[1] https://www.netacad.com/pt-br/courses/packet-tracer
[2] https://omnetpp.org/
[3] http://opnetprojects.com/opnet-network-simulator/
[4] https://www.scalable-networks.com/qualnet-network-simulation
[5] https://www.isi.edu/nsnam/ns/
[6] https://www.nsnam.org/

muito tempo para simular (horas, dias, meses etc). Além do inconveniente do tempo, também existe o gasto de energia alocado para a realização das simulações.

Além das limitações relativas ao tempo e aos recursos computacionais, muitos processos de simulação de redes possuem como única saída *traces* sem a interpretação lógica (ex: arquivos em formato `xml`, `csv`, `dat`, etc), ou seja, que necessitam passar por outras ferramentas que ajudem na interpretação dos resultados e, assim, gerar informações relevantes para a tomada de decisão. O trabalho proposto analisa a predição da taxa de entrega para auxiliar nas decisões dos especialistas no momento de criação dos cenários.

Como contribuição nesse sentido, este trabalho de TCC avalia o desempenho de algoritmos de *Machine Learning* (ML) na otimização de simulações de redes de computadores. Quatro algoritmos de ML são treinados a partir de um *dataset* de resultados de simulações. Em seguida, é avaliado se os algoritmos conseguem predizer satisfatoriamente o mesmo comportamento alcançado pelas simulações a partir de um conjunto de discriminantes estatísticos obtidos para cada cenário de rede considerado. Os algoritmos treinados são o *Decision Tree*, a Rede *Multilayer Perceptron*, o *Random Forest* e a *Support Vector Regression*.

O *Network Simulator* 3 (NS-3) (NS-3, 2019) é utilizado como simulador no trabalho. A rede utilizada como estudo de caso é uma Rede em Malha Sem Fio (*Wireless Mesh Network* - WMN) (AKYILDIZ; WANG; WANG, 2005). Em particular, a proposta tem como foco predizer a taxa de entrega de diferentes cenários de WMN. Esses cenários são gerados a partir da variação de três parâmetros principais: distância entre nós, tamanho do pacote e intervalo entre a geração de pacotes.

O trabalho faz uso da Regressão como tarefa de *Machine Learning* para predizer a taxa de entrega que possui um comportamento contínuo. Os resultados alcançados mostram que os algoritmos fazem suposições próximas das simulações reais, tendo alguns modelos de ML com taxa de acerto acima de 90% nas predições da taxa de entrega.

1.3 Objetivos

Este trabalho tem como objetivo geral treinar um conjunto de algoritmos de *Machine Learning* a partir de um *dataset* de resultados de simulações de redes e, com isso, avaliar se os algoritmos conseguem predizer satisfatoriamente o mesmo comportamento alcançado pelas simulações a partir de um conjunto de discriminantes estatísticos obtidos para cada cenário de rede considerado.

Os objetivos específicos deste trabalho são:

- Planejar, implementar e executar simulações no NS-3 de cenários de WMN variando alguns parâmetros de rede (distância entre nós, tamanho do pacote e intervalo entre a geração de pacotes);
- Tratar os *traces* gerados pelas simulações no NS-3 para composição dos *datasets* de treino e validação;
- Treinar quatro algoritmos de *Machine Learning* a partir do *dataset* para treinamento gerado;
- Avaliar o desempenho dos algoritmos de ML na predição da taxa de entrega de WMN no NS-3.

1.4 Organização do Trabalho

A sequência do trabalho está organizada do seguinte modo: o Capítulo 2 descreve a Fundamentação Teórica, que consiste no detalhamento das técnicas de *Machine Learning* aplicadas na proposta; o Capítulo 3 exibe os trabalhos relacionados; o Capítulo 4 apresenta o detalhamento da proposta de otimização de redes e o processo metodológico adotado para implementação da predição da taxa de entrega; o Capítulo 5 exibe os resultados obtidos e, por fim, o Capítulo 6 apresenta as conclusões e os direcionamentos para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são apresentados os principais conceitos de *Machine Learning* (ML) e redes em malha sem fio necessários para uma melhor compreensão da proposta do trabalho. Além dos conceitos gerais, o capítulo aborda as técnicas de otimização de parâmetros, o uso de validação cruzada e as métricas de avaliação de desempenho dos algoritmos durante a fase de treinamento. O capítulo apresenta também os algoritmos de ML utilizados para atender o domínio em estudo.

2.1 *Machine Learning*

O *Machine Learning* (ML) é um subcampo da Inteligência Artificial (IA) que trabalha com a ideia de que computadores são programados para aprender com base na experiência passada (FACELI et al., 2011). Existem muitas definições de ML. Uma delas é apresentada em Faceli (FACELI et al., 2011): “Algoritmos de *Machine Learning* aprendem a induzir uma função ou hipótese capaz de resolver um problema a partir de dados que representam instâncias do problema a ser resolvido".

Na literatura de Mitchell (MITCHELL, 1997), por exemplo, existem três principais tipos de *Machine Learning*:

- **Aprendizado supervisionado**: refere-se a construção de um modelo (função que mapeia elementos da entrada para as saídas) de ML baseado em um conjunto de treino com dados rotulados (JOSHI, 2017). Se uma hipótese conseguir predizer bem os rótulos do conjunto de treino, provavelmente conseguirá predizer bem para um conjunto do mesmo tipo que tenha um volume maior de dados.
- **Aprendizado não supervisionado**: refere-se ao processo de construção de um modelo de ML com os dados do conjunto de treino não rotulados (JOSHI, 2017). Nesse tipo de aprendizado, busca-se particionar o espaço para agrupar os dados semelhantes. Ele é útil quando é preciso categorizar dados não categorizados.
- **Aprendizado por reforço**: refere-se ao processo de aprender o que fazer e mapear situações a certas ações a fim de maximizar a recompensa. Na maioria dos paradigmas de ML, um agente de aprendizado é informado de quais ações devem ser tomadas para alcançar certos resultados (JOSHI, 2017). Esse tipo de aprendizado é muito usado na robótica.

O objetivo geral do presente trabalho de TCC é fazer uso de *Machine Learning* para predizer os resultados de simulações de redes de computadores, mais

especificamente, os resultados da taxa de entrega de pacotes de uma rede em malha sem fio. Como os resultados da taxa de entrega possuem comportamento contínuo, o aprendizado supervisionado é o tipo de ML mais adequado para atingir o objetivo proposto. O foco do trabalho é baseado no aprendizado supervisionado e, por isso, são apresentados maiores detalhes sobre o aprendizado supervisionado a seguir.

2.1.1 Aprendizado Supervisionado

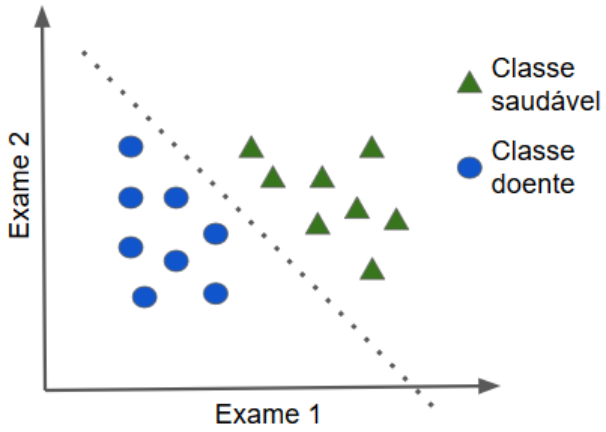
Esta seção apresenta definições das tarefas de Classificação e Regressão que fazem parte do aprendizado supervisionado.

Segundo Faceli *et al.* (FACELI et al., 2011), dado um conjunto de observações de pares $D = (x_i, f(x_i), 1, ..., n)$, em que f representa uma função desconhecida, um algoritmo de ML supervisionado aprende uma aproximação f da função desconhecida f. Essa função aproximada f permite estimar o valor de f para novas observações de x. De acordo com a natureza de f, é comum distinguir duas possíveis situações:

- Classificação: as observações ou exemplos fazem parte de um conjunto de características que podem ser separadas por classes, também chamadas de rótulos, ou seja, existem diferentes características que representam as classes presentes no conjunto. Em um problema de classificação, os classificadores buscam aprender os padrões que diferem as classes e, a partir disso, classificam um novo objeto que tenha características próximas das conhecidas pelos classificadores (FACELI et al., 2011).
- Regressão: diferentemente da classificação, a regressão é usada quando a classe se comporta de maneira contínua e os valores diferem de uma representação binária (também pode ser representada considerando sim ou não). Os algoritmos de regressão aprendem o comportamento contínuo dos dados e retornam respostas numéricas. Geralmente, é utilizada para responder perguntas do tipo: “Quanto custa” ou “Quantos existem” (FACELI et al., 2011).

A Figura 1 exemplifica um problema de classificação que considera dois exames médicos (1 e 2). Com base nas características desses exames, deseja-se classificar se um paciente é saudável ou doente. Assim, o objetivo é encontrar uma fronteira de decisão que separe os exemplos de uma classe dos exemplos da outra. Se os exemplos de uma classe forem linearmente separáveis dos exemplos da outra, como existem dois atributos, a fronteira de decisão pode ser uma reta. Se os exemplos não forem linearmente separáveis, uma combinação de retas é necessária. Se os exemplos possuem mais de dois atributos de entrada, em vez de retas, são utilizados hiperplanos de separação (FACELI et al., 2011).

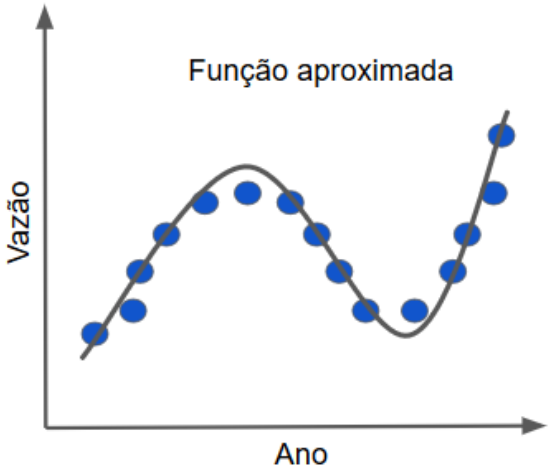
Figura 1: Exemplo de Tarefa de Classificação.



Fonte: adaptada de (FACELI et al., 2011).

Na tarefa de Regressão, o modelo aprende a prever valores numéricos baseados nos dados. A Figura 2 ilustra um caso de regressão em que o objetivo é aprender uma função que relacione um ano à vazão de água de um dado rio nesse ano. Os pontos formados pelos valores dia e vazão geram uma curva e, espera-se que essa curva se aproxime da curva real da função que, dado o valor do ano, retorna o valor correto da vazão (FACELI et al., 2011).

Figura 2: Exemplo de Tarefa de Regressão.



Fonte: adaptada de (FACELI et al., 2011).

Neste trabalho, usa-se a tarefa de regressão definida anteriormente dado o comportamento da métrica estudada neste trabalho (taxa de entrega de uma rede em malha sem fio). A seguir, são apresentados os algoritmos com suas versões para tarefa de regressão.

2.1.1.1 Decision Tree (DT)

Decision Tree é uma técnica de aprendizado supervisionado que tem comportamento recursivo usando a estratégia dividir para conquistar. Formalmente, uma árvore de decisão é um grafo acíclico direcionado em que cada nó ou é um nó de divisão, com dois ou mais sucessores, ou um nó folha. Ela funciona para ambas variáveis categóricas e contínuas de entrada e de saída. Dessa forma, pode ser facilmente utilizada em problemas de regressão.

Para geração de uma árvore de decisão, são necessários os cálculos de entropia e de ganho da informação. A entropia mede a aleatoriedade (dificuldade para predizer) do atributo alvo em bits usando logaritmos na base 2, representando, assim, a falta de informação. A cada nó de decisão, o atributo menos aleatório é utilizado para dividir os dados. A entropia é calculada pela Equação 2.1, sendo S um conjunto de dados, com elementos x (dados que representam o problema existentes em S) pertencentes à classe i (rótulos do problema), com probabilidade de ocorrência p.

$$Entropia(S) = \sum_{k=1}^{n} p(x_i) \log_2 p(x_i) \tag{2.1}$$

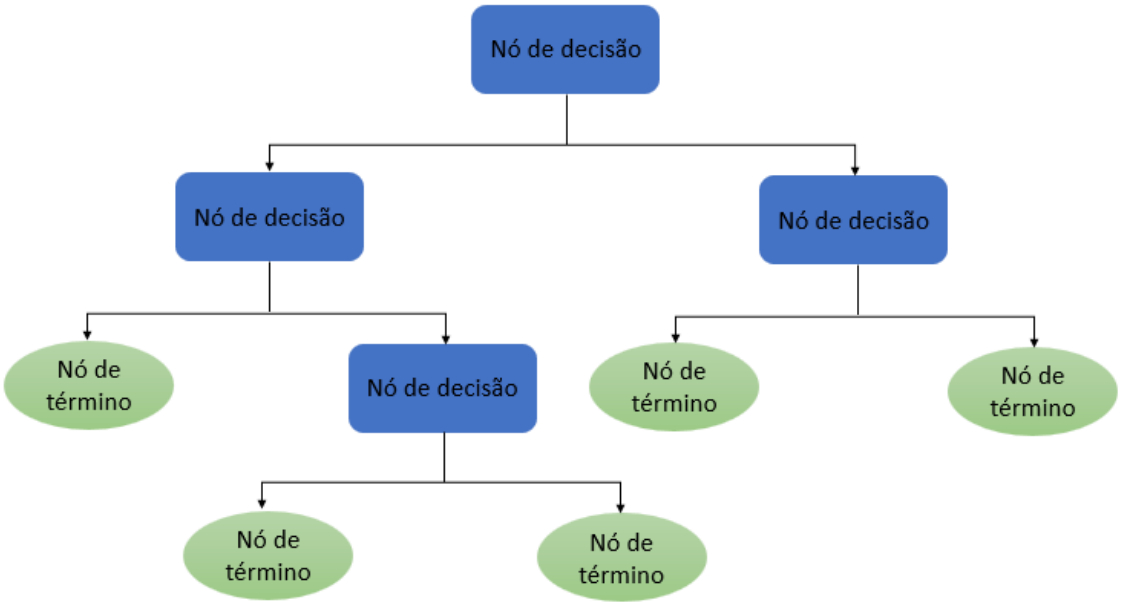
Por outro lado, o ganho da informação define a redução da entropia, ou seja, representa o quanto é preciso para completar a informação faltante. Assim, o ganho da informação é dado pela diferença entre a entropia do conjunto de exemplos S e a soma ponderada da entropia das partições. A Equação 2.2 mostra que $Ganho(S, A)$ significa a redução esperada na entropia de S, ordenado pelo atributo A.

$$Ganho(S, A) = Entropia(S) - \sum_{v \in Values(A)} \frac{\mid S_v \mid}{\mid S \mid} Entropia(S_v) \tag{2.2}$$

O cálculo de $Entropia(S)$ é feito para todos os atributos do conjunto de dados. Em seguida, faz-se o cálculo do $Ganho(S, A)$ da informação para escolha do atributo mais importante para representar o nó raiz da árvore. Após esse processo, os cálculos são realizados novamente para escolha dos ramos da árvore. Por último, são adicionados os nós folhas que representam as classes do conjunto de dados. No caso da regressão em uma árvore de decisão, considera-se a média dos cálculos realizados em cada partição da árvore (FACELI et al., 2011) (SMOLYAKOV, 2017).

A Figura 3 mostra a estrutura de uma árvore construída por uma *Decision Tree* no momento de treinamento da técnica com os dados disponíveis. Na árvore construída, tem-se os nós de decisão que levam aos nós de término, representando a classificação final de cada caminho formado na árvore. E durante o processo de criação da árvore são realizados os cálculos de $Entropia(S)$ e $Ganho(S, A)$, discutidos anteriormente.

Figura 3: Estrutura de árvore criada pelo *Decision Tree*.



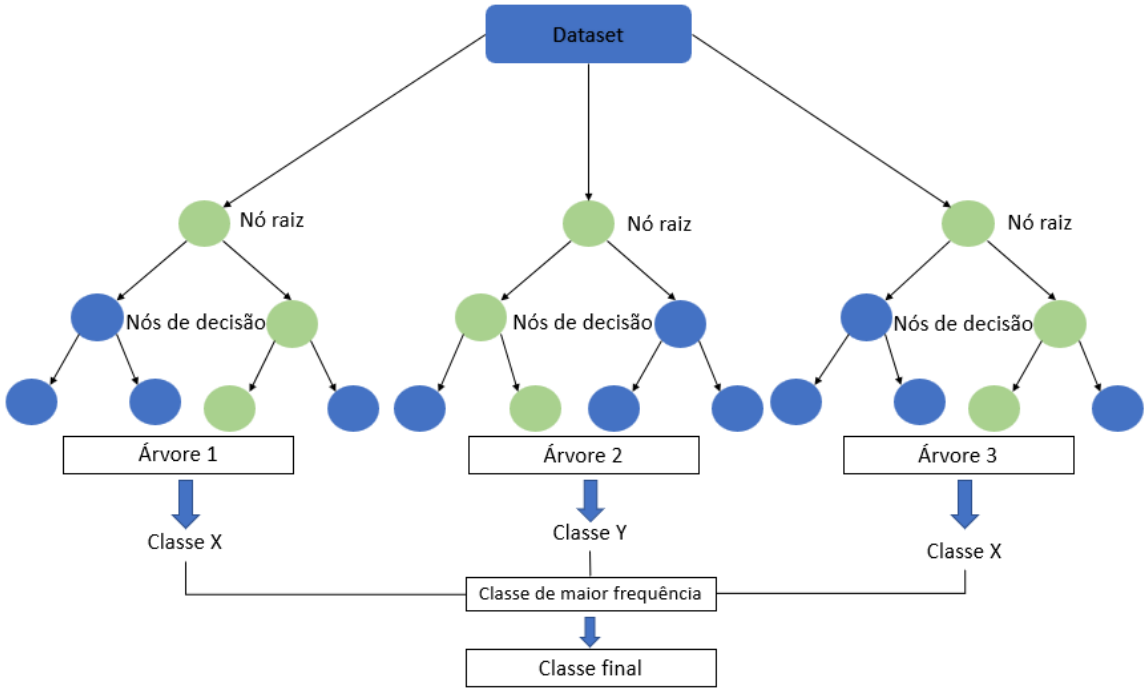
Fonte: elaborada pelo autor

2.1.1.2 Random Forest (RF)

Random Forest segue o mesmo procedimento de construção de uma árvore de decisão descrito anteriormente, sendo uma melhoria da árvore de decisão. Diferentemente da árvore de decisão, esse algoritmo baseia-se na abordagem *Ensemble Learning*, que gera diversas árvores randômicas. Essa abordagem permite fazer a combinação de modelos seguindo os cálculos de $Entropia(S)$ e $Ganho(S, A)$ da informação. Assim, o resultado final é dado a partir da combinação dos resultados de cada árvore. No caso da regressão, considera-se o valor da média das respostas de cada árvore (SMOLYAKOV, 2017).

A Figura 4 esquematiza a geração de árvores criadas durante o treinamento da técnica *Random Forest*. Durante seu treinamento, n árvores são geradas com diferentes partições dos dados para formar uma estrutura que represente bem os dados em estudo, sendo capaz de classificar novos objetos. Como pode ser visto na Figura 4, o *dataset* é dividido no momento de criação das árvores e, ao final de cada árvore, tem-se uma classe que representa a resposta da árvore formada com uma parte dos dados. Após a resposta de cada n árvores, a classe com maior frequência é escolhida como classe final do conjunto de árvores geradas.

Figura 4: n árvores geradas através de *Ensemble Learning*.



Fonte: elaborada pelo autor

2.1.1.3 Support Vector Machine (SVM)

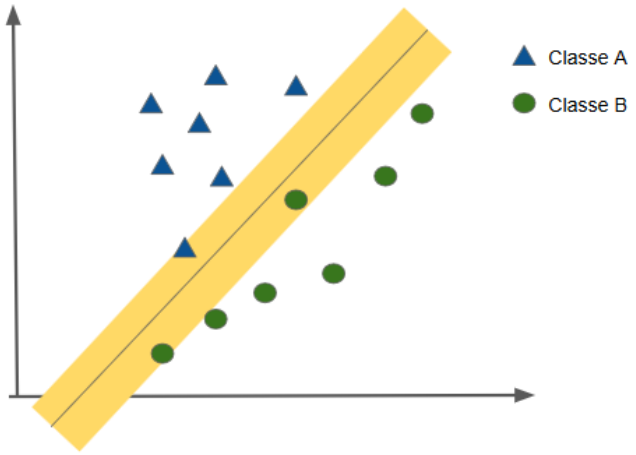
A *Support Vector Machine* (SVM) é baseada na teoria de aprendizado estatístico, desenvolvido por Vapnik (1995) a partir de estudos iniciados em Vapnik e Chervonenkis (1971). Sejam h um classificador e H o conjunto de todos os classificadores que um determinado algoritmo de ML pode gerar, no decorrer do processo de aprendizado, esse algoritmo utiliza um conjunto de treinamento X, composto de n pares (x_i, y_i) para gerar um classificador particular $\widehat{h} \in H$.

Embora a SVM seja bastante aplicada em problemas de classificação, ela também pode resolver problemas de regressão através do algoritmo *Support Vector Regression* (SVR). Uma SVR tem o objetivo encontrar uma função $h(x)$ que produz saídas contínuas para os dados de treinamento. A SVR acaba sendo uma adaptação da SVM para a tarefa de regressão e, para problemas não linearmente separáveis a SVM usa o conceito de *Kernel Trick* que transforma o espaço não linear em linear, facilitando os demais cálculos necessários do algoritmo.

O aprendizado de uma SVM é encontrar um hiperplano com margem máxima representando a melhor reta que separa bem os dados. Além disso, busca encontrar o hiperplano de margem máxima minimizando o erro dos dados em relação ao hiperplano encontrado. A Figura 5 exemplifica um hiperplano com margem máxima para indicar quais características representam um triângulo (Classe A) e quais representam

um círculo (Classe B), assim, classificar um novo objeto como triângulo ou círculo.

Figura 5: Margem máxima para SVM.



Fonte: elaborada pelo autor

Seja X um conjunto de treinamento com n objetos $x_i \in X$ e seus respectivos rótulos $y_i \in Y$, em que X constitui o espaço de entradas e Y são as possíveis classes. A Equação 2.3 ilustra a função da reta usada pela SVM para criação do hiperplano de separação dos dados, em que $w * x$ é o produto escalar entre os vetores w e x, $w \in X$ é o vetor normal do hiperplano e b uma constante $\in \Re$, o objetivo da SVM além de encontrar um hiperplano com o menor erro possível dado os dados e conhecendo a média geral dos *dataset*.

$$y = w * x + b \tag{2.3}$$

Nos cálculos da SVM, existem muitas manipulações algébricas usando os pontos sobre os hiperplanos, essas manipulações permitem a maximização da margem de separação dos objetos em relação $x * x + b$. Essa maximização da margem pode ser obtida através da minimização de $\mid w \mid$. Isso recorre-se ao um problema de otimização expressado por $\frac{1}{2} \mid w \mid^2$.

Uma ideia do SVM é obter o mínimo de erro possível com a Equação 2.4, sendo c uma forma de punição por classificação incorreta. Uma vez que c é um valor baixo, mais erros podem ocorrer. Tendo o cálculo dos erros dos vetores de suporte dado pelo somatório de a_i, o algoritmo SVM atualiza o novo hiperplano sempre que encontra um valor menor que o atual calculado.

$$\frac{1}{2} \mid w \mid^2 + c \sum_i a_i \tag{2.4}$$

A SVM é bastante utilizada para problemas linearmente separáveis, porém, podemos utilizá-la para problemas não linearmente separáveis usando uma técnica chamada de *Kernel Trick*. Ela possibilita ao algoritmo trabalhar com um conjunto de dados não linear transformando-o em um conjunto linear. Existem alguns *Kernels Trick* que podem ser utilizados, como o linear, o polinomial e o *Gaussian Radial Basis Function* (RBF). De acordo com a natureza deste trabalho, os *kernels* mais adequado para o tipo de problema são o RBF e o polinomial, ambos para problemas não lineares (FACELI et al., 2011).

2.1.1.4 Rede Multilayer Perceptron (MLP)

Um Perceptron é um classificador para problemas linearmente separáveis baseado na estrutura de um neurônio humano. Dado um vetor x como entrada, sendo w um vetor de pesos para cada uma das entradas de x e adicionado a um viés (ou bias) b, um Perceptron produz uma única saída com base em uma função de ativação que recebe a $Soma$ mostrada na Equação 2.5.

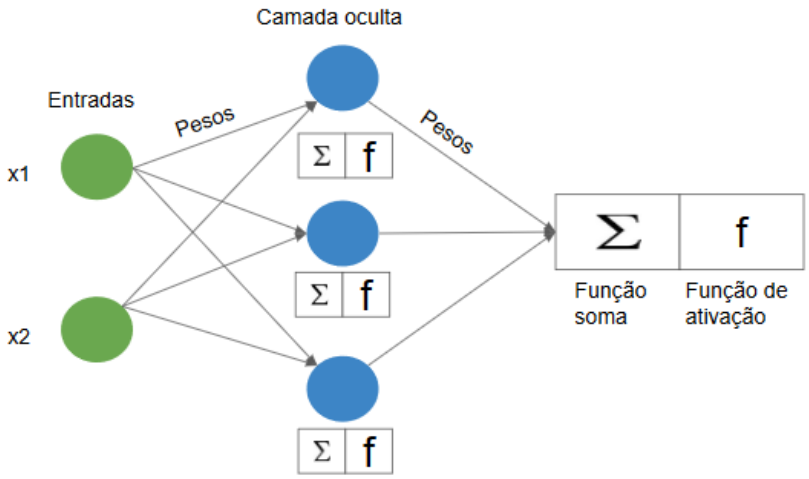
$$Soma = \sum_{i=1}^{n} x_i * w_i + b \tag{2.5}$$

As Redes Multilayer Perceptron são uma abstração da rede neural biológica que lida com problemas não lineares e possui vários Perceptrons conectados. A arquitetura dessa rede neural funciona como uma rede *feedforward*, ou seja, os neurônios de uma camada l estão conectados a todos os neurônios da camada $l + 1$ (FACELI et al., 2011). Ela é composta por uma camada de entrada, uma camada de saída que toma uma decisão sobre a entrada e, entre essas duas camadas, podem existir várias camadas ocultas que são o verdadeiro mecanismo computacional da MLP. Nas camadas ocultas, são utilizadas funções de ativação não lineares (ACADEMY, 2018). A Figura 6 apresenta um exemplo de arquitetura de rede MLP.

A função implementada por um neurônio de uma determinada camada é uma combinação das funções realizadas pelos neurônios da camada anterior que estão conectadas a ele. À medida que o processamento avança de uma camada para outra, esse processamento torna-se mais complexo. Na primeira camada, cada neurônio aprende uma função que define um hiperplano, o qual divide o espaço de entrada em duas partes. Assim, cada neurônio da camada seguinte combina um grupo de hiperplanos definidos pelos neurônios da camada anterior, formando regiões convexas (FACELI et al., 2011).

Em casos de classificação de dados, a rede classifica corretamente um objeto quando o valor de saída mais elevado produzido pela rede é aquele gerado pelo neurô-

Figura 6: Exemplo da arquitetura MLP.



Fonte: elaborada pelo autor

nio de saída que corresponde à classe correta do objeto. Quando nenhum neurônio produz um valor elevado ou o valor elevado é produzido por mais de um neurônio, a rede não tem condições de prever a classe do objeto (FACELI et al., 2011).

Os algoritmos possuem diversos parâmetros de configuração que também influenciam no aprendizado de cada um durante o treinamento. Muitas vezes, esses parâmetros são configurados manualmente, ocorrendo um processo repetitivo no momento de treinamento. Existem técnicas de otimização de parâmetros para escolher os melhores parâmetros de cada algoritmo, evitando que vários experimentos sejam realizados com diferentes parâmetros configurados manualmente para um mesmo algoritmo. Duas dessas técnicas são apresentadas a seguir.

2.1.2 Ajuste de parâmetros dos algoritmos de ML

Nesta seção, são definidos os conceitos de hiperparâmetros e apresentados os conceitos do algoritmo *Grid search* para encontrar os melhores parâmetros de cada algoritmo, esse procedimento também é conhecido na literatura como ajuste ou otimização de parâmetros dos algoritmos de ML.

Os parâmetros de entrada dos algoritmos podem influenciar na precisão no momento de analisar o desempenho. Hiperparâmetro é uma técnica em *Machine Learning* que busca otimizar a procura dos melhores parâmetros para treinar os algoritmos. Como os dados se comportam diferentemente, as configurações dos hiperparâmetros dependem da estrutura organizacional dos dados. Então, para cada *dataset* pode existir n configurações de parâmetros encontradas baseando-se na técnica de hiperparametrização (ZHENG, 2015).

Muitas etapas são realizadas antes de encontrar um modelo ideal para resolver um problema em questão. No entanto, o modelo ideal encontrado ainda pode ser aprimorado testando diferentes configurações de seus hiperparâmetros. Este processo de aprimoramento também é conhecido por *model tuning* que trata da questão de como encontrar os melhores hiperparâmetros para cada modelo de *Machine Learning* utilizado.

Existem algumas formas de encontrar os hiperparâmetros de um modelo de *Machine Learning* comumente utilizadas na literatura: manualmente e usando algoritmos de *tuning*. Ou seja, o primeiro busca testar diferentes parâmetros configurados manualmente, enquanto que o segundo, busca testar diferentes configurações (ZHENG, 2015) dos parâmetros usando algoritmos específicos, por exemplo, o *Grid search* para encontrar essas configurações. Este trabalho usa o ajuste de parâmetros para evitar que diferentes configurações nos algoritmos sejam realizadas manualmente que demanda tempo para análise dos experimentos com os algoritmos. Utilizado-se o *Grid search* como algoritmo de busca de parâmetros para configuração dos algoritmos.

2.1.2.1 Grid search

No *Grid search*, avaliamos o modelo para cada combinação de uma lista predefinida de valores dos hiperparâmetros. Todos os valores são colocados na forma de matriz, semelhante à uma grade. Cada conjunto de parâmetros é considerado e a precisão ou métrica de avaliação, por exemplo, pode ser utilizada para comparação com os demais testes. Uma vez que todas as combinações são avaliadas, o modelo com o conjunto de parâmetros que obtém o melhor resultado da métrica em análise é escolhido. Muito utilizado na literatura e tem melhores resultados quando o *dataset* possui baixa dimensionalidade. O *Grid search* tem um desempenho baixo na busca dos parâmetros quando o número de hiperparâmetros cresce exponencialmente (SENAPATI, 2018).

Uma maneira que o *Grid search* usa para buscar os hiperparâmetros é a validação cruzada apresentada na subseção 2.1.3.1. Em outras palavras, o objetivo do *Grid search* é encontrar a melhor combinação de parâmetros de configuração dos algoritmos de *Machine Learning* para que os mesmos obtenham desempenhos satisfatórios. Esse algoritmo percorre todo espaço de atributos de entrada e encontra uma combinação dependendo da métrica de avaliação definida inicialmente para o *Grid search* (KAPIL, 2019).

2.1.3 Avaliação de desempenho de modelos preditivos

Na aplicação de algoritmos de ML a problemas reais, o conhecimento do domínio sendo investigado é obtido por meio do conjunto de exemplos, a partir do qual a indução de um modelo preditivo é realizada. Existem várias técnicas de ML que podem ser utilizadas na indução de modelos de classificação e/ou regressão a partir de um conjunto de exemplos rotulados. Assim, não é possível estabelecer que determina técnica será melhor na resolução de qualquer tipo de problema.

Em alguns casos, as próprias características das técnicas existentes e do problema abordado podem ser consideradas para auxiliar na escolha da técnica a ser utilizada. Ainda que essas possibilidades possam contribuir na escolha, existe também a necessidade de experimentação. Faceli recomenda seguir procedimentos que garantam a corretude, a validade e a reprodutibilidade dos experimentos realizados (FACELI et al., 2011).

2.1.3.1 Validação Cruzada

A validação cruzada é o processo de treinamento dos modelos (gerados após os algoritmos conhecerem os dados) usando um conjunto de dados e testando-os usando um conjunto diferente. Esse processo estima o erro do método de aprendizado em observações não utilizadas no treino e como o modelo construído se comporta com novos dados.

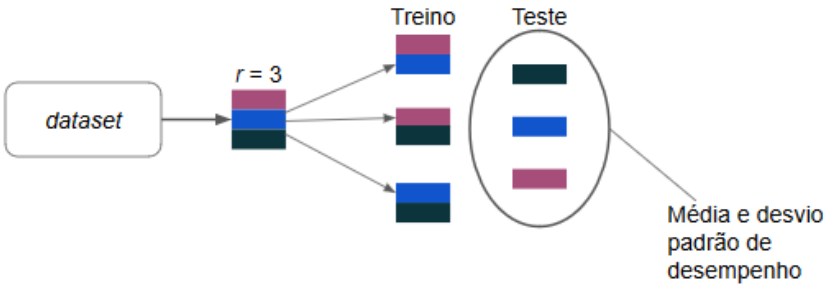
O método de validação cruzada *r-fold cross-validation* consiste em dividir o conjunto de exemplos em r partições aproximadamente igual. Para cada partição, estima-se o método sem presença de uma determinada parte e verifica-se o erro médio na partição não utilizada durante o treino.

Os objetos de *r-1* partições são utilizados no treinamento de um preditor, o qual é testado na partição restante. Esse processo é repetido r vezes, utilizando em cada ciclo uma partição diferente para teste. Esse processo é ilustrado na Figura 7, com $r = 3$ (FACELI et al., 2011).

No caso dos procedimentos que envolvem médias de desempenho, deve-se reportar também os valores de desvio padrão associados. Um alto desvio padrão indica uma alta variabilidade nos resultados, ou seja, uma instabilidade do modelo perante mudanças nos objetos que são todos provenientes de uma mesma distribuição (FACELI et al., 2011).

Um dos desafios em *Machine Learning* é descobrir qual modelo é o melhor. Isso significa que a estimativa do melhor modelo estatístico e do conjunto de trei-

Figura 7: Validação cruzada.



Fonte: adaptada de (FACELI et al., 2011)

namento, capitaliza padrões aleatórios de amostras específicas e associações entre variáveis.

Para identificar que um modelo é melhor que outro, precisamos ser capazes de estimar o erro de teste (estimativa do erro) em um modelo testado em diferentes observações que fazem parte do contexto em estudo.

2.1.3.2 Métricas de erro para modelos de regressão

As métricas *Mean Absolute Error* (MAE), *Mean Squared Error* (MSE) e *Root Mean Squared Error* (RMSE) são abordadas nesta subseção. Elas são comumente utilizadas para avaliar o desempenho de modelos de regressão (FACELI et al., 2011). Na regressão buscamos prever um valor numérico, como, por exemplo, as vendas de uma empresa para o próximo mês.

A utilidade de cada métrica de erro, depende do objetivo e do problema que estamos tentando resolver. Em problemas de regressão, o erro da hipótese $\widehat{y}$ pode ser calculado pela distância entre o valor y_i conhecido e aquele predito pelo modelo $\widehat{y}_i$ (FACELI et al., 2011).

A MAE mede a magnitude média dos erros em um conjunto de previsões, sem considerar sua direção. Essa métrica vista na Equação 2.6 é a média da amostra de teste das diferenças absolutas entre a previsão $\widehat{y}_i$ e a observação real y_i, em que todas as diferenças individuais têm peso igual (WORLD, 2016) (DRAKOS, 2018).

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y}_i| \tag{2.6}$$

A MSE é uma das métricas mais simples e comum para a avaliação de regressão. Ela definida pela Equação 2.7 e mede o erro quadrado médio das previsões

realizadas. Para cada ponto, calcula a diferença quadrada entre as previsões $\widehat{y}_i$ e o valor real y_i, em seguida, calcula a média desses valores (DRAKOS, 2018).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2 \tag{2.7}$$

Quanto maior é o resultado do MSE, pior é o modelo de ML. O resultado dessa métrica nunca é negativo, pois o resultado é elevado ao quadrado. Para um modelo perfeito, o resultado é 0 (DRAKOS, 2018).

A RMSE é definida como a raiz quadrada da distância quadrática média entre a observação real y_i e a previsão $\widehat{y}_i$ (Equação 2.8):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2} \tag{2.8}$$

O RMSE é a raiz quadrada do MSE. A raiz quadrada é calculada para fazer a escala dos erros ser a mesma que a escala dos alvos, ou seja, tornar os valores de erros mais próximos dos possíveis valores reais que esses erros possam ser representados (DRAKOS, 2018).

Atingir a raiz quadrada dos erros quadrados médios tem algumas implicações interessantes para o RMSE. Como os erros são elevados antes da média, o RMSE atribui um peso relativamente alto a erros grandes. Isso significa que o RMSE deve ser mais útil quando erros grandes são particularmente indesejáveis (WORLD, 2016).

Essas métricas podem ser usadas juntas para diagnosticar a variação dos erros em um conjunto de previsões. A RMSE é sempre maior ou igual a MAE. Quanto maior a diferença entre elas, maior a variação dos erros individuais da amostra. Se o RMSE = MAE, então todos os erros são da mesma magnitude (WORLD, 2016).

Além do erro calculado para avaliação do quanto um algoritmo erra com relação a média do *dataset*, observa-se também o ajustamento ou qualidade do modelo de *Machine Learning* para a tarefa de regressão. Esse ajuste é explicado pela métrica R^2 detalhada a seguir.

2.1.3.3 R^2 para um modelo de regressão

O R^2, ou simplesmente *score*, é uma métrica comumente utilizada para indicar a qualidade do ajustamento dos dados em relação a linha de regressão, formalmente conhecido por coeficiente de determinação. Temos que R^2 mede em termos percentuais como a variável independente x explica a variável dependente y. Essa métrica

fornece um valor situado entre 0 e 1, que também pode ser interpretada em termos percentuais, ou seja, valores entre 0% e 100%.

Tendo uma reta com x e y representando os pontos de um dado conjunto, a melhor reta selecionada depende do valor de R^2, ou seja, a reta possui a menor distância quadrada entre o ponto e a reta propriamente dita. Usa-se a Equação 2.9 para descobrir a inclinação da reta e realização dos próximos cálculos.

$$y = mx + b \tag{2.9}$$

Dado um vetor com os elementos $y_1, y_2, ..., y_n$, a Equação 2.10 representa o erro quadrado da reta em relação ao x.

$$SE_{reta} = (y_1 - (mx_1 + b))^2 + (y_2 - (mx_2 + b))^2 + ... + (y_n - (mx_n + b))^2 \tag{2.10}$$

Já a Equação 2.11 representa o erro quadrado médio em relação a média do conjunto, expressada por $\widehat{y}$.

$$SE_{\widehat{y}} = (y_1 - \widehat{y})^2 + (y_2 - \widehat{y})^2 + ... + (y_n - \widehat{y})^2 \tag{2.11}$$

Como o objetivo do R^2 é calcular o quanto x consegue explicar y, a Equação 2.12 resulta no valor que explica o quanto da variação total não é descrita pela reta e, a partir desse valor, pode-se encontrar o quanto da variação total em y é correspondida pela variação em x. Para isso, basta realizar o procedimento da Equação 2.13.

$$\frac{SE_{reta}}{SE_{\widehat{y}}} \tag{2.12}$$

$$R^2 = 1 - \frac{SE_{reta}}{SE_{\widehat{y}}} \tag{2.13}$$

A partir do cálculo feito pela Equação 2.13, tem-se o valor para R^2 que explica a qualidade do modelo e sua capacidade de ajustar os dados. Quando mais próximo de 1, o valor de R^2 estiver, mais ajustado é o modelo de *Machine Learning* para explicar novos dados.

O valor mais baixo que R^2 pode obter é 0, quando isso acontece, usa-se a média do conjunto para representar sua qualidade. Porém, se R^2 resultar em um valor negativo, isso significa que a linha de regressão é pior que o valor médio (NERDY, 2018).

2.2 Rede em Malha Sem Fio

O presente trabalho usa como estudo de caso um conjunto de cenários de Redes em Malha Sem Fio (*Wireless Mesh Network* - WMN) (AKYILDIZ; WANG; WANG, 2005). Assim, essa seção apresenta de forma breve as principais definições e características dessas redes para facilitar o entendimento da proposta no Capitulo 4, assim como para ajudar na compreensão dos resultados no Capitulo 5.

As WMNs são consideradas um tipo de Rede do Futuro (*Network of the Future* - NoF) e têm atraído cada vez mais atenção por causa de seu baixo custo, facilidade de implantação, robustez e cobertura de serviço confiável. Estas redes são capazes de oferecer uma cobertura sem fio em banda larga para grandes áreas (como *smart cities*), sem exigências em termos de infraestrutura, ao mesmo tempo em que são capazes de garantir a conectividade em ambientes com dinamicidade das condições do meio sem fio e com usuários móveis (SILVA et al., 2018).

As WMNs são padronizadas pelo IEEE como uma solução de rede em malha para difusão (*broadcast*) e entrega de pacotes *unicast* ao longo de uma topologia de múltiplos saltos auto-configurável. O padrão é chamado de IEEE 802.11s *Mesh Networking* (IEEE 802.11s, 2011). Ele propõe, entre outros serviços de malha, a seleção de caminhos e o encaminhamento com competências de roteamento na camada MAC, interoperabilidade com redes externas e soluções de segurança.

As WMNs são compostas por roteadores sem fio, chamados roteadores mesh, interligados para formar um backbone de múltiplos saltos. Esses roteadores mesh são dinamicamente auto-organizáveis, auto-configuráveis e auto-curáveis, sem qualquer controle centralizado. Essas características permitem que uma WMN seja implantada incrementalmente, um nó de cada vez, de acordo com a demanda. Além disso, os roteadores mesh são geralmente estacionários e, consequentemente, podem ser permanentemente alimentados com energia elétrica. Assim, os roteadores mesh podem se beneficiar de recursos como memória, processamento e energia. Para estender o acesso à rede com fio além do alcance de transmissão de um único ponto de acesso, os roteadores mesh são interconectados através do meio sem fio para estabelecer e manter a conectividade da malha. Eles encaminham, através de múltiplos saltos, o tráfego recebido das estações, bem como o tráfego recebido de outros roteadores mesh. Portanto, o backbone mesh sem fio combina as vantagens de ambos os modos infraestruturado e ad hoc.

Roteadores mesh também podem desempenhar o papel de *gateway*/*bridge*. Esta funcionalidade permite a integração da WMN com diferentes redes cabeadas e sem fio, como Ethernet, celulares, Wi-Fi e sensores. Alguns dos roteadores mesh também podem atuar como *gateways* em direção à Internet via enlaces cabeados de

alta velocidade. Assim, através de uma WMN integrada, os usuários dessas redes cabeadas e sem fio podem se beneficiar de serviços que, de outra forma, seriam impossíveis de acessar.

3 TRABALHOS RELACIONADOS

Neste capítulo, são apresentados trabalhos que usam ferramentas no auxílio de cálculos das medidas de desempenho em cenários de redes e outros, na extração dos resultados das simulações e que fazem uso de *Machine Learning* (ML) na tomada de decisão.

3.1 Avaliação de desempenho do TCP usando *Machine Learning* em ambientes sem fio

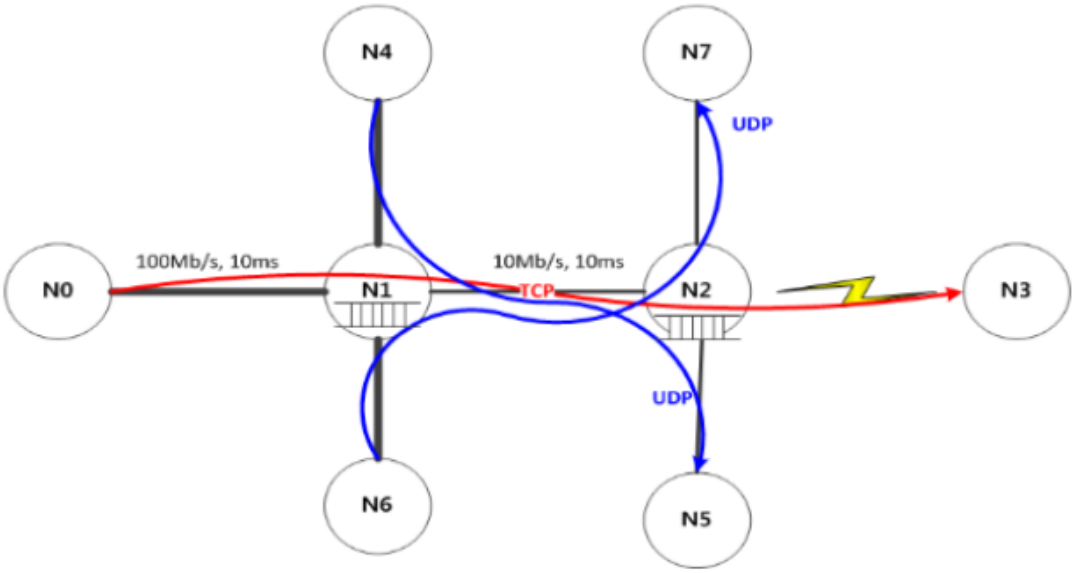
Hwang *et al.* (HWANG; LEE; KIM, 2017) propõem uma solução para a causa da perda de pacotes em redes com e sem fio (usando o Protocolo de Controle de Transmissão (TCP) *NewReno* na camada de Transporte) com base no algoritmo de *Machine Learning Single-layer Perceptron*. Essa solução é aplicada no algoritmo de controle de congestionamento quando a causa da perda de pacotes é prevista como um congestionamento de rede. Se previsto como um erro aleatório, o pacote perdido é retransmitido. A solução proposta resultou em um algoritmo chamado de controle de congestionamento da perda de pacotes com e sem fio usando *Machine learning* (ML-TCP).

Segundo o trabalho, o TCP é o protocolo de transmissão mais utilizado. Ele usa uma janela deslizante para controlar o fluxo e o congestionamento na transmissão de dados entre dispositivos. É também um protocolo que garante a transmissão de dados através da retransmissão de um segmento danificado. Porém, os autores abordam o fato do TCP ser adequado apenas para uma rede com fio e com baixa taxa de erro de bit. Diante disso, os autores ressaltam a importância de melhorias no desempenho do protocolo.

Assim, o trabalho desenvolve o algoritmo ML-TCP com o intuito de auxiliar no desempenho do TCP. No treinamento do *Single-layer Perceptron*, os dados utilizados são fornecidos por simulações executadas no *Network Simulator 3*. A configuração da rede é mostrada na Figura 8, tal configuração possui os tamanhos das filas dos nós N1 e N2 igual a 100. Os nós N4 e N6 geram aleatoriamente tráfego UDP para causar uma perda devido ao congestionamento no nó N1. Além disso, gera-se um erro entre os nós N2 e N3.

O *dataset* para treinamento do ML-TCP possui os seguintes parâmetros de entrada: o intervalo ACK recebido, o último RTT (*Round Trip Time* - Latência), o RTT mínino, o número de pacotes enviados, mas não recebidos e o ACK. Especificamente,

Figura 8: Topologia simulada para *Machine Learning*

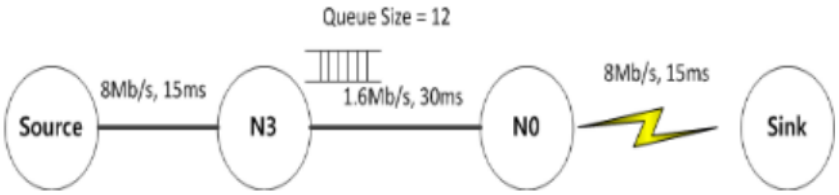


Fonte: elaborada por (HWANG; LEE; KIM, 2017).

o ML-TCP usa dados oriundos de tráfego com perda devido ao congestionamento da rede e com perda de informação devido ao erro na rede sem fio.

Para validação da proposta, os autores fazem uma comparação de desempenho entre os protocolos TCP NewReno e o TCP Veno usando o ML-TCP no ambiente de rede sem fio simulado no NS-3, seguindo a topologia de rede mostrada na Figura 9.

Figura 9: Topologia simulada para validação



Fonte: elaborada por (HWANG; LEE; KIM, 2017).

O TCP Veno tem um bom comportamento no ambiente sem fio com pouca perda aleatória. Por outro lado, mostra uma taxa de precisão menor quando a taxa de perda de pacotes aumenta de 10^3 para 10^2, descobrindo erroneamente a perda devido a perda aleatória e a perda devido ao congestionamento da rede. Os resultados da simulação mostram que o ML-TCP possui maior precisão para prever a causa da perda de pacotes em diferentes ambientes.

Como trabalhos futuros, os autores pretendem aplicar o algoritmo proposto em um ambiente real e avaliar o desempenho do TCP. Além disso, pretendem também

aplicar outros algoritmos de *Machine Learning* para aprimorar o ML-TCP, por exemplo, um algoritmo baseado em *Deep Learning*.

3.2 Simulação em ambientes

Lee *et al.* (LEE; SUH; CHO, 2017) apresentam um simulador capaz de auxiliar diversos ambientes. O simulador proposto faz carregamento automático dos resultados de simulações para um banco de dados e utilizam esses resultados para predizer o fluxo de ar ao redor da asa de um avião usando ML. Esse carregamento automático é feito através de um formulário padrão para um banco de dados *MongoDB*. Os autores também propõem a redução de cálculos desnecessários, evitando a execução redundante de simulações.

Para predição dos resultados das simulações, são utilizados algoritmos de ML disponíveis no ambiente de programação *R*, tais como: *Multiple Linear Regression*, *Support Vector Machine* (SVM), *Local Regression* (LR), *Random Forest* (RF), *K-Nearest Neighbour regression* (K-NN), *Multilayer Perceptron*, dentre outros. Após um usuário solicitar o resultado de determinada simulação, a ferramenta verifica a existência desse resultado no banco de dados. Caso não exista, a predição do novo resultado é realizada. Em seguida, o novo resultado é armazenado no banco. Para treinamento dos algoritmos, são utilizados 4 atributos de entrada e 5 de saída de uma amostra de dados sobre o fluxo de ar ao redor da asa de um avião produzidos pelo programa de simulação KFLOW, disponível na plataforma EDISON. Com base em uma amostra de 7.680 instâncias, os *datasets* de treinamento e teste possuem um total de 6.200 e 1480 instâncias, respectivamente.

São realizados 5 experimentos, um para cada parâmetro de saída. O primeiro experimento realiza a predição do parâmetro coeficiente de sustentação e tem o RF como melhor algoritmo por obter 0.9 de taxa de erro. O segundo experimento realiza a predição do parâmetro coeficiente de arrasto total e tem os algoritmos RF e o LR com 1.6 de taxa de erro. O terceiro experimento realiza a predição do parâmetro coeficiente de resistência à pressão e tem os algoritmos RF e o LR com 1.1 de taxa de erro. O quarto experimento realiza a predição do parâmetro coeficiente de resistência ao atrito da pele e tem como melhores resultados os algoritmos RF e LR com 1.1 de taxa de erro e o K-NN com 2.0 de taxa de erro. Por fim, o quinto experimento realiza a predição do parâmetro coeficiente do momento de lançamento. Nesse caso, os algoritmos RF e LR obtiveram taxa de erro entre 7.4 e 7.5, respectivamente.

Entretanto, os autores não aplicam em um cenário de redes e também não consideram outras métricas de avaliação para os algoritmos.

3.3 Simulações de Redes no NS-3

O trabalho de Silva (SILVA, 2014) apresenta uma ferramenta de apoio à realização de análise de *traces* de simulações realizadas no NS-3 a fim de calcular medidas de desempenho em redes de computadores. A ferramenta desenvolvida utiliza o *Node.js* no *back-end* e o *Chart.js* para visualização gráfica dos resultados em forma de relatório para uma melhor tomada de decisão.

A partir dos resultados das simulações extraídos dos *traces*, a ferramenta faz o cálculo das medidas e geração de gráficos para extração de informações relevantes e comparação entre as simulações. Um exemplo de simulação é implementado no NS-3 como estudo de caso para avaliar os resultados de saída da ferramenta. Esse exemplo de simulação é composto por um fluxo com dois nós (origem e destino) e um roteador fazendo a interligação desses nós. São feitas variações para os seguintes protocolos TCP: *New Reno*, *Tahoe* e o *Westwood*. Primeiramente, é realizado um pré-processamento nos *traces* para remover eventos ocorridos no roteador, em seguida, a simulação é executada. Após esse processo, a análise dos resultados dos *traces* é realizada.

Por fim, o trabalho ressalta a importância da implementação de um formato de *traces* padrão que possa ser utilizado em qualquer simulação do NS-3 com o propósito de que apenas eventos ocorridos na origem e destino (por fluxo) sejam considerados.

3.4 Ferramenta TraceMetrics para simulações no NS-3

Já Saggioro *et al.* (SAGGIORO; GONZAGA; RIBEIRO, 2012) propõem uma ferramenta, denominada *TraceMetrics* como uma alternativa para a obtenção de medidas de interesse em simulações realizadas no NS-3. A *TraceMetrics* analisa cada linha do *trace* gerado pelo simulador e obtém, a partir dele, medidas de interesse. Os autores destacam que a principal vantagem está na flexibilidade, uma vez que se tem o registro de tudo o que aconteceu na simulação, pacote a pacote, pode-se calcular qualquer medida desejada.

A ferramenta também oferece suporte a algumas medidas de acordo com os tipos definidos: medidas calculadas por nó na simulação e calculadas por fluxo. Como exemplos de medidas suportadas pela *TraceMetrics*, tem-se: quantidade e tamanho médio dos pacotes enviados, recebidos e descartados; quantidade de dados enviados, recebidos e descartados; *Throughput* e *Goodput*; atraso fim-a-fim por pacote, média e variância.

Para obtenção dos resultados, uma topologia é definida com quatro nós para

calcular as medidas dentre o tempo de simulação estabelecido de 1.000 segundos. No tempo 0, é criado um fluxo UDP com taxa de 500 Kbps do nó 0 para o nó 3; no tempo de 500 segundos, é iniciada uma conexão TCP, com taxa de 450 Kbps do nó 1 para o nó 3. Segundo o trabalho, as taxas contemplam apenas os dados, desconsiderando cabeçalhos. Cada conexão envia pacotes com 1000 *Bytes* de dados mais *overhead* de cabeçalhos, que para o UDP somam 30 *Bytes* (8 UDP + 20 IPv4 + 2 *Point to Point*), e para o TCP somam 42 *Bytes* (20 TCP + 20 IPv4 + 2 Point to Point). O arquivo *trace* possui 276,338 *MBytes* e sua análise pela ferramenta leva 1 minuto e 34 segundos para a topologia simulada.

Para verificar a corretude dos resultados, o *script* de simulação é configurado para fazer uma simulação de 1.000 segundos. Contudo, os autores ressaltam que a desvantagem está no desempenho, pois são necessários acessos à disco e armazenamento de informações em memória. Como trabalhos futuros, os autores propõem a geração de gráficos e uma maneira de construir a topologia por meio do arquivo de *trace*.

Apesar da *TraceMetrics* realizar vários cálculos para métricas de desempenho, não são gerados gráficos com base na análise realizada. A ferramenta disponibiliza apenas os dados em forma de tabela para utilização de outra ferramenta.

Diferentemente dos trabalhos relacionados apresentados, o presente trabalho propõe um mecanismo de otimização de simulações que visa utilizar diferentes algoritmos de *Machine Learning* na predição de resultados das seguintes métricas de desempenho dos cenários de redes: taxa de entrega, vazão, *jitter* e atraso. A partir disso, o mecanismo pode ser capaz de evitar que simulações desnecessárias sejam executadas. Esse mecanismo também pretende fornecer respostas mais precisas aos usuários para criação dos cenários de redes. Além disso, este trabalho propõe disponibilizar o mecanismo como serviço para administradores de redes, salas de suporte técnico, meio acadêmico e demais usuários.

A Tabela 1 mostra os significados de A, B e C presentes na Tabela 2.

Tabela 1: Identificação das colunas da Tabela 1

Identificador	Significado
A	resultados simulados x preditos
B	visualização dos resultados de forma gráfica
C	*Machine Learning* na tomada de decisão

Fonte: elaborada pelo autor.

Na Tabela 2, esquematiza-se um comparativo entre os trabalhos relacionados apresentados e o presente trabalho.

Tabela 2: Comparação entre os trabalhos relacionados

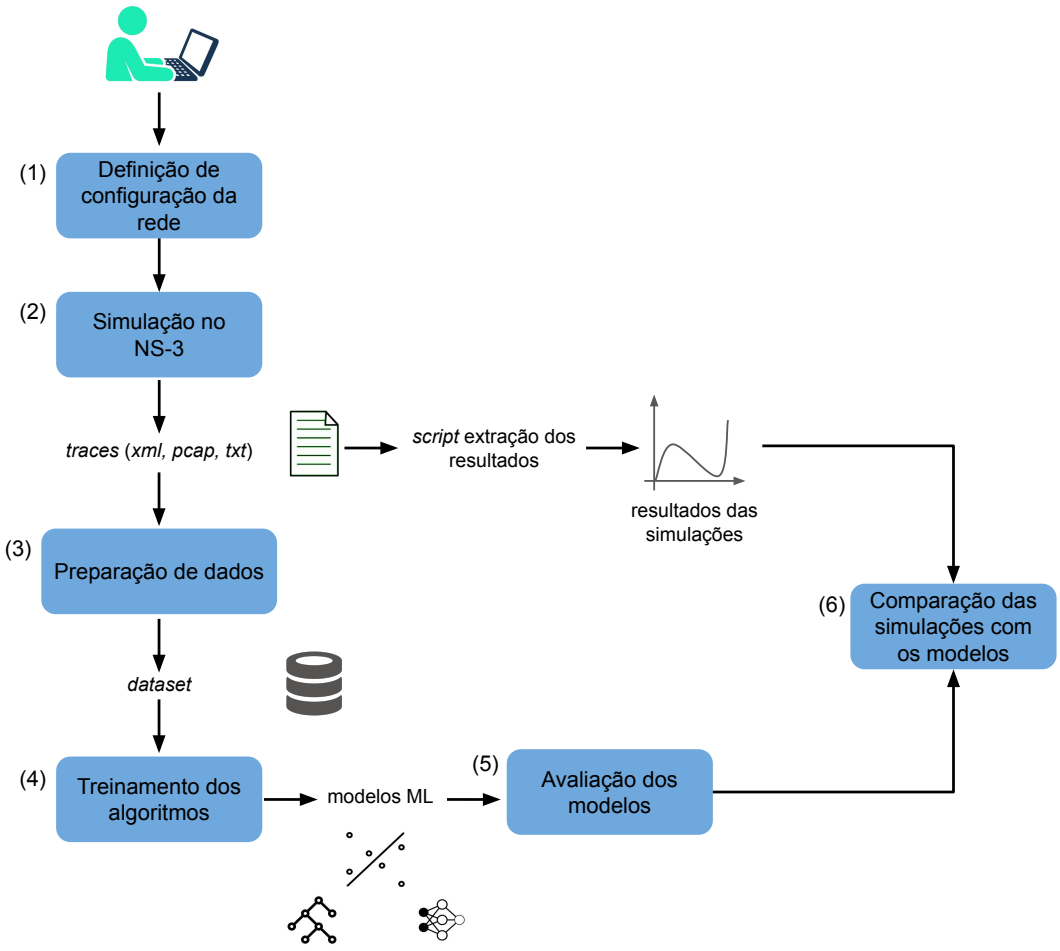
	A	B	C
Simulação em ambientes		X	X
Avaliação de desempenho do TCP usando ML	X		X
Simulações de Redes no NS-3	X	X	
TraceMetrics	X	X	
O presente trabalho	X	X	X

Fonte: elaborada pelo autor.

4 PROPOSTA

Este capítulo detalha todo o processo metodológico de implementação da solução de otimização de simulações de redes de computadores através de algoritmos de *Machine Learning*. A Figura 10 ilustra as principais etapas de tal processo, que são apresentadas e discutidas nas seções a seguir.

Figura 10: Visão geral da proposta.

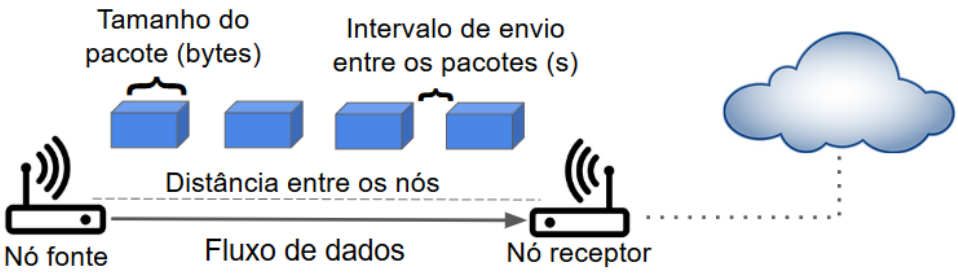


Fonte: elaborada pelo autor.

4.1 Etapa 1: Definição da Configuração da Rede

Conforme citado anteriormente, o estudo de caso deste trabalho é baseado em uma Rede em Malha Sem Fio (*Wireless Mesh Network* - WMN) (AKYILDIZ; WANG;

Figura 11: Topologia *mesh* configurada para o estudo de caso.



Fonte: elaborada pelo autor.

WANG, 2005), cujas definições e características foram detalhadas na Seção 2.2. Mais especificamente, utiliza-se uma topologia *mesh* composta por um fluxo de dados *User Datagram Protocol* (UDP) entre dois nós, sendo um deles o gerador de tráfego e o outro o receptor (*gateway*), conforme ilustra a Figura 11. A camada física utiliza na sua configuração o padrão IEEE 802.11n a uma frequência de 5 giga-hertz (GHz).

A taxa de entrega da rede *mesh* é a métrica em questão no estudo, ou seja, é ela que os algoritmos de ML tentarão predizer. A taxa de entrega é uma porcentagem calculada pela razão entre o número de pacotes recebidos pelo destino e o número de pacotes enviados pela fonte, multiplicada por 100, conforme mostra a Equação 4.1.

$$Taxa\ de\ Entrega = \frac{QtePacotesRecebidos}{QtePacotesEnviados} * 100 \tag{4.1}$$

Baseado na topologia da Figura 11, este trabalho tem como foco o estudo de três parâmetros da redes que impactam diretamente na taxa de entrega: a distância entre nós, o tamanho dos pacotes e intervalo de envio entre os pacotes. A Tabela 3 detalha as variações adotadas para cada parâmetro para criação de diferentes cenários *mesh*.

Tabela 3: Valores dos parâmetros.

Parâmetro	**Valores**
Distância entre os nós	50 a 110 metros
Tamanhos do pacote	256, 512, 1024 bytes
Intervalo de envio entre os pacotes	0.1, 0.01, 0.001 segundos

Fonte: elaborada pelo autor.

4.2 Etapa 2: Simulação no NS-3

Definidas as configurações da rede a serem estudadas, os cenários *mesh* foram simulados no *Network Simulator* 3 (NS-3) (NS-3, 2019). É importante destacar que o simulador já disponibiliza a implementação do padrão de WMN IEEE 802.11s (ANDREEV; BOYKO, 2010).

No total, foram realizadas 16200 rodadas de simulações para obtenção dos resultados a seguir. A Equação 4.2 justifica esse total, sendo $valoresDist$ os 60 valores de distância utilizados, $valoresTamPacote$ os 3 valores de tamanho de pacote e $valoresIntPacotes$ os 3 valores de intervalo de envio entre os pacotes, conforme apresentado anteriormente na Tabela 3. Além disso, cada combinação de parâmetro foi simulada 30 vezes para considerar as variações presentes nas amostras (JAIN, 1991). Esse processo de execução das simulações é o maior responsável pelo longo tempo de resposta do simulador, o grande consumo de processamento e de energia.

$$TotalRodadasNS3 = valoresDist \cdot valoresTamPacote \cdot valoresIntPacotes \cdot 30 \quad (4.2)$$

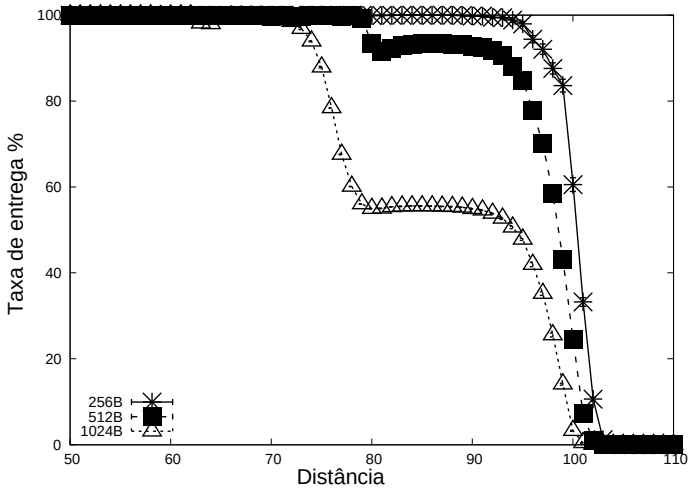
Neste trabalho, a fim de ilustrar os resultados que se espera que os algoritmos repliquem, os *traces* gerados pelas simulações foram submetidos a um processo de extração de resultados referentes à taxa de entrega. As Figuras 12a, 12b e 12c apresentam os resultados da taxa de entrega em função da distância para três diferentes tamanhos de pacotes e para os intervalos de envio entre os pacotes de 0.1s, 0.01s e 0.001s, respectivamente. Foi necessário desenvolver um *script* específico para extração dos valores da taxa de entrega dos *traces* e para o cálculo do intervalo de confiança da taxa de entrega. Também foi preciso usar o programa gnuplot[1] para plotar os gráficos, já que o NS-3 não disponibiliza esse tipo de funcionalidade para representação dos fluxos.

As figuras mostram claramente o comportamento da taxa de entrega à medida que os dois nós são distanciados. Em todos os casos, a taxa de entrega é de 100% para curtas distâncias e tende a diminuir até alcançar 0%. Nota-se que para pacotes de maior tamanho (como 1024B) e menores intervalos de envio entre os pacotes (como 0.001s), a queda da taxa de entrega ocorre mais rapidamente. É importante lembrar que quanto maior a distância entre dois nós, menor é a relação sinal-ruído (*Signal-to-Noise Ratio* - SNR) e, portanto, maior é a taxa de erro de bit (*Bit Error Rate* - BER).

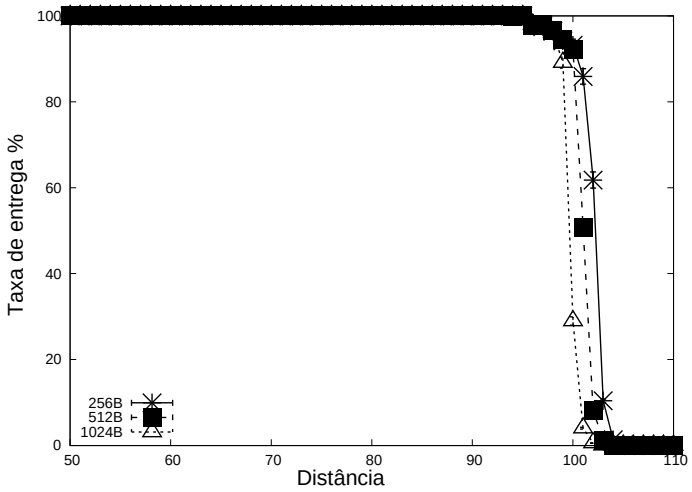
[1] http://gnuplot.sourceforge.net/

Figura 12: Resultados da Simulações no NS-3.

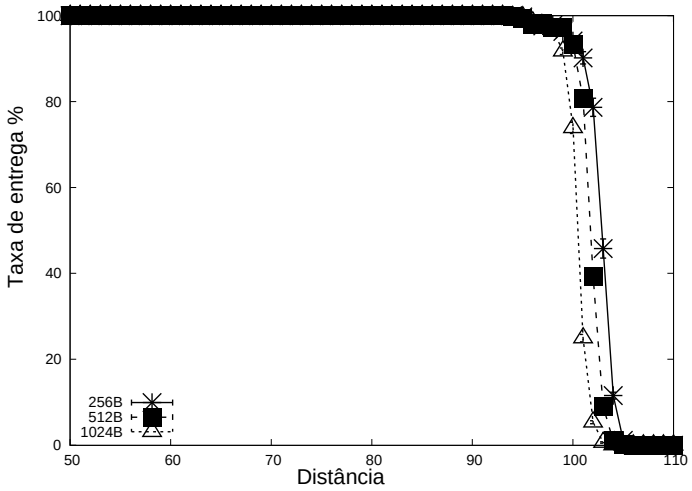
(a) Intervalo de envio entre os pacotes de 0.001s



(b) Intervalo de envio entre os pacotes de 0.01s



(c) Intervalo de envio entre os pacotes de 0.1s



Fonte: elaborada pelo autor.

4.3 Etapa 3: Preparação de dados

Na etapa de preparação, os cenários utilizados na validação do trabalho são numerados para facilitar a identificação no capítulo de resultados. Considerando todos os fatores de definição de um cenário, o trabalho conta com 9 cenários distintos que sofrem mudanças em 2 dos principais parâmetros de entrada, o intervalo entre os pacotes e o tamanho do pacote. As Tabelas 4, 5 e 6 exibem as particularidades de cada cenário para o estudo de caso com a taxa de entrega.

Tabela 4: Identificação dos cenários com intervalo de 0.001

Cenário	Intervalo entre pacotes (segundos)	Tamanho dos pacotes (*bytes*)
Cenário 1	0.001	256
Cenário 2	0.001	512
Cenário 3	0.001	1024

Fonte: elaborada pelo autor.

Tabela 5: Identificação dos cenários com intervalo de 0.01

Cenário	Intervalo entre pacotes (segundos)	Tamanho dos pacotes (*bytes*)
Cenário 1	0.01	256
Cenário 2	0.01	512
Cenário 3	0.01	1024

Fonte: elaborada pelo autor.

Tabela 6: Identificação dos cenários com intervalo de 0.1

Cenário	Intervalo entre pacotes (segundos)	Tamanho dos pacotes (*bytes*)
Cenário 1	0.1	256
Cenário 2	0.1	512
Cenário 3	0.1	1024

Fonte: elaborada pelo autor.

Definidos os 9 cenários de estudo, os dados são extraídos e passam pelas etapas de limpeza e transformação de dados. Na extração desses dados, por padrão, os simuladores de redes exportam os resultados das simulações em arquivos chamados de *traces*. Para este trabalho, o NS-3 gera os *traces* como saída contendo os resultados das simulações em diversos formatos, como `xml`, `pcap`, `txt`, etc. Os *traces* exportados contêm várias informações não estruturadas para os algoritmos, dificultando o entendimento e o carregamento padrão das informações.

A fim de facilitar a leitura dos resultados das simulações, as informações contidas nos *traces* de interesse são transferidas para arquivos no formato `txt`, criados após o final de cada simulação. A Figura 13 mostra um exemplo de arquivo `txt`

com os resultados de uma das simulações. Destaca-se que esses arquivos possuem características importantes para avaliação dos cenários de redes (Taxa de entrega, Vazão, Atraso, *Jitter*, etc). Em outras palavras, caracterizam o fluxo de transmissão dos pacotes na rede. Após o processo de extração de dados, as características se transformam em atributos do *datasets* criados.

Figura 13: Exemplo de arquivo `txt` com resultados das simulações.

```
Node is at (0, 50)
Printing mesh point device #0 diagnostics to mp-report-0.xml
Printing mesh point device #1 diagnostics to mp-report-1.xml
  Flow 1 (10.1.1.2 -> 10.1.1.1)
  DeliveryRate:                 100 %
  Throughput:                  1.08316 Mbps
  TxBytes:                     1419716
  RxBytes:                     1419716
  TxPackets:                   4999
  RxPackets:                   4999
  LostPackets:                 0
  DelaySum:                    1.10236 s
  DelayMean:                   0.000220516 s
  JitterSum:                   0.471814 s
  JitterMean:                  9.44006e-05 s
  TimeFirstTxPacket:           0.0175 s
```

Fonte: elaborada pelo autor.

Um ponto importante nesta etapa é quando os nós da rede estão distantes um do outro ao ponto de não conseguirem se comunicar mais, o fluxo não é iniciado. Na ocorrência de fluxo não iniciado, os pacotes não chegam ao destino. Esse fator causa o não preenchimento das informações de fluxo, gerando *traces* vazios e, consequentemente, arquivos `txt` vazios. Para evitar a exclusão de informações e prejudicar o aprendizado dos algoritmos, realiza-se um tratamento de dados específico para os casos de fluxo não iniciado. Basicamente, quando é identificado que um fluxo não é iniciado, um segundo arquivo `txt` é carregado com todas as informações dos resultados da simulação preenchidas com 0.

Os algoritmos precisam que os dados estejam em uma estrutura que permita um rápido processamento, facilidade no acesso aos dados e de fácil entendimento. Assim, torna-se necessário o uso de uma estrutura padrão para extração das informações relevantes dos *traces* para treinamento dos algoritmos. Para isso, utiliza-se um objeto do tipo *dataframe* que contém os dados das simulações, facilitando o processo de tratamento dos dados e etapas posteriores.

No tratamento de dados, são excluídos caracteres adicionados durante a simulação. A exclusão desses caracteres é necessária para facilitar o processamento dos algoritmos na busca por padrões nos dados.

Para uma melhor avaliação dos algoritmos, são utilizados diferentes partições do *dataset* de simulações através de um processo de validação cruzada detalhado no

capítulo de fundamentação teórica, na Seção 2.1.3.1. A divisão consiste em dividir o *dataset* na proporção 80/20 a cada iteração da validação cruzada. Esse processo acaba treinando e testando os algoritmos com diferentes partes do *dataset*.

Na realização dos experimentos utilizando a topologia de avaliação, cria-se um *dataset* com 24 colunas, sendo 23 delas atributos que implicam na avaliação de desempenho em redes e 1 coluna que representa o atributo alvo para os algoritmos. O *dataset* inicialmente criado passa por uma seleção de atributos, na qual os atributos não importantes apontados por um especialista no assunto são retirados, diminuindo a dimensionalidade dos dados. Após a seleção de atributos, o *dataset* definitivo para os experimentos de ML possui 4 características, sendo 3 parâmetros de entrada (distância entre os nós, intervalo entre os pacotes e tamanho do pacote) e 1 parâmetro de saída (taxa de entrega), representando o atributo de interesse para os algoritmos. Os *traces* de simulações geraram um total de 60 pontos para avaliação dos algoritmos. A Tabela 7 exibe os atributos do *dataset* criado e o significado de cada um.

Tabela 7: Atributos que compõem os *datasets*

Nome do atributo	Definição
DistanceBetweenNode	distância entre os nós
PacketInterval	intervalo entre os pacotes
PacketSize	tamanho do pacote
DeliveryRate	taxa de entrega

Fonte: elaborada pelo autor.

Os dados passam também por um escalonamento que trata de deixar todos os atributos da base de dados em uma mesma escala de valor, por exemplo, valores entre 0 e 1. O escalonamento de dados é um passo importante por colocar todos os valores presentes no *dataset* em um mesmo padrão, diminuindo as chances dos algoritmos obterem suposições erradas sobre os dados.

Na padronização dos atributos preditores do *dataset* de simulação, utiliza-se a classe *StandardScaler* disponível na biblioteca *Scikit-Learn* (PEDREGOSA et al., 2011). Esse método subtrai cada valor de observação da média geral calculada de acordo com o *dataset* e divide pelo desvio padrão geral. O cálculo realizado segue a Equação 4.3, sendo x o valor da observação ou exemplo, u a média geral e s o desvio padrão, todos do *dataset* em estudo.

$$z = \frac{x - u}{s} \tag{4.3}$$

Outra modificação feita no *dataset* é a aplicação de Análise de Componentes Principais (PCA), um método que reduz a dimensionalidade dos dados de tal maneira que novas variáveis são ortogonais entre si (ou seja, são independentes ou não estão correlacionadas). O PCA é indicado quando possuímos um conjunto de variáveis contínuas muito amplo. Após a análise, cada componente principal representa uma combinação linear das variáveis originais. Os componentes principais agrupam elementos que variam num mesmo sentido, o que facilita a compreensão de como as variáveis interagem entre si (JOLLIFFE, 2011).

Neste trabalho, PCA é aplicado para facilitar a visualização dos resultados por meio de gráficos. O *dataset* criado para treinamento dos algoritmos que possui 3 atributos independentes (distância entre os nós, intervalo entre pacotes e tamanho do pacote) e 1 atributo dependente (taxa de entrega) após a aplicação de PCA, recebe uma nova configuração nos valores das variáveis. Os 3 atributos independentes são transformados em um Componente Principal (PC), representando o eixo x nos gráficos apresentados no próximo capítulo (5).

4.4 Etapa 4: Treinamento dos algoritmos de ML

A etapa de treinamento dos algoritmos de ML consiste em submeter o *dataset* representando o contexto investigado aos algoritmos, que aprendem o comportamento dos dados e buscam padrões neles. Como já mencionado, este trabalho usa diferentes algoritmos de ML adaptados para problemas de regressão com diferentes abordagens. Cada um busca diferentemente os padrões nos dados. Assim, cada um possui sua particularidade e, portanto, são configurados de forma diferente.

O *dataset* é dividido em treino e teste na proporção 80/20, ou seja, 80% da base para treinamento dos algoritmos e 20% para teste. Com as proporções dos *datasets* divididas, são executadas 30 rodadas com o método de validação cruzada com $r = 10$ e a cada execução, faz-se uma troca dos *datasets* de treinamento e teste com diferentes dados das diferentes partições. Ao final de cada rodada são armazenados os valores do R^2 ou *score* e do erro das suposições dos algoritmos para que seja possível a comparação entre os modelos na etapa de análise de desempenho dos algoritmos.

Na configuração dos parâmetros dos algoritmos, define-se uma lista com os possíveis parâmetros. Em seguida, ocorre a aplicação da técnica de hiperparâmetro *Grid search* através de um método disponibilizado pela *Scikit-learn* chamado *gridsearchCV* (Seção 2.1.2.1) que recebe a lista de parâmetros de cada algoritmo e faz a combinação entre eles, retornando uma combinação de parâmetros como resultado. Os melhores parâmetros são selecionados e cada algoritmo recebe sua nova configu-

ração para iniciar o treinamento.

4.4.1 Definição dos Hiperparâmetros dos Modelos Preditivos

A MLP implementada neste trabalho possui uma camada oculta com uma quantidade x de neurônios. A função de ativação é utilizada para decidir a classificação ou o valor (no caso da regressão) de saída de um neurônio após o somatório dos pesos pelas entradas. A taxa de aprendizagem define quão rápido o algoritmo pode aprender. Assim, conforme o valor da taxa de aprendizagem aumenta, o algoritmo processa mais rapidamente as informações. A taxa de aprendizagem é iniciada com 0.001, valor padrão recomendado pela *Scikit-learn*.

Geralmente, as redes neurais fazem uso de otimizadores para ajudar no seu processamento. A funcionalidade de um otimizador é dada pela importância de encontrar os melhores resultados dentro de um intervalo de tempo, por exemplo. O *lbfgs* obteve resultados melhores quando comparado com o otimizador *adam*. O *alpha* é um parâmetro de penalidade na hora dos ajustes dos pesos de cada neurônio. A função de ativação *tanh* não é linear e possui semelhança com a função *sigmóide*. A *tanh* é simétrica em relação à origem e varia de -1 a 1.

Nos cenários 1 e 2 com intervalo de 0.001, a MLP é configurada para ativar os neurônios com a função de ativação *tanh*, já o cenário 3, o *grid search* retornou *logistic* como função de ativação. O *alpha* recebe 0.1, padrão para todos os cenários. O otimizador(*lbfgs*) também é o mesmo para todos os cenários. Uma particularidade da aplicação da MLP nos diferentes cenários é a quantidade de neurônios na camada oculta, pois para cada cenário, tem-se uma quantidade x de neurônios.

Nos cenários 1, 2 e 3 com intervalos entre os pacotes de 0.01 e 0.1, a MLP é configurada para ativar os neurônios com a função de ativação *tanh*. O *alpha* também recebe 0.1, padrão para todos os cenários. O otimizador(*lbfgs*) também é o mesmo para todos os cenários. E para cada cenário, uma quantidade x de neurônios é utilizada na camada oculta.

A *Decision Tree* possui uma configuração considerada simples com apenas os parâmetros *criterion* e o *splitter*. O primeiro significa que a métrica para organização da formação da árvore é baseada no resultado da MSE ou MAE. Já o segundo parâmetro significa como árvore busca dividir seus nós da melhor forma possível, no caso do valor ser *best*. Existe outra abordagem que é a forma aleatória, mas este trabalho usa a estratégia da melhor divisão dos nós. A *Decision Tree* é configurada igualmente para todos os cenários. O parâmetro *criterion* é utilizado para criação da árvore seguindo os resultados da MAE nas iterações do algoritmo. O *splitter* é configurado como *best*.

O *Random Forest* também possui o parâmetro *criterion* configurado que também pode receber a MSE e a MAE como critério de avaliação dos nós no momento de formação das árvores. Um caso particular desse algoritmo é a quantidade de árvores que ele deve receber para criação de n árvores que representam sua forma de classificar um novo objeto ou predizer um valor numérico. Esse parâmetro chama-se *n_estimators* na *Scikit-learn*.

As configurações do *Random Forest* variam bastante para os diferentes cenários em comparação aos demais algoritmos. As duas métricas de verificação de erro foram utilizadas para criação da árvore. O *n_estimators* também variou bastante sob diferentes condições impostas. O RF é treinado com o *criterion* recebendo MSE com a maioria dos cenários com os pacotes de tamanhos 512 e 1024.

A SVM busca encontrar um hiperplano que melhor representa os dados. Para isso, uma função linear ou não linear também chamada de kernel da SVM é usada na busca desse hiperplano. Especificamente neste trabalho, a função selecionada por *grid search* é a *rbf*. Existe também um parâmetro c para a função de custo no momento de buscar a margem máxima dos hiperplanos e cálculo do erro. O c pode ser interpretado como uma punição por resultar em um valor incorreto de predição durante os testes. Já o parâmetro *gamma* representa o coeficiente da função de busca dos hiperplanos. Os valores dos parâmetros para este algoritmo são iguais para todos os cenários testados. O c tem valor 1.0 e o *gamma* é igual a 10. Um problema desse algoritmo em relação aos demais é o seu alto custo de processamento mesmo com o *dataset* considerado pequeno.

Com base nas configurações apresentadas anteriormente, os algoritmos recebem o *dataset* de treino e dependendo da abordagem matemática por trás de cada um, os resultados das predições são obtidos. A partir desses resutados, o erro (seguindo a métrica RMSE) e o *score* de cada algoritmo são calculados. Além disso, calcula-se também o intervalo de confiança da base de validação, possibilitando uma verificação da aproximação entre os valores preditos pelos algoritmos e os valores reais da simulação.

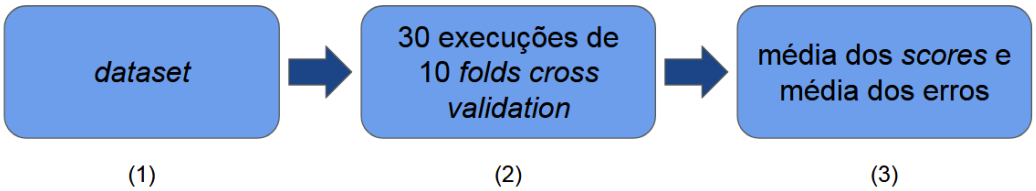
4.5 Etapa 5: Avaliação dos modelos

Neste trabalho, avalia-se os modelos preditivos observando o desempenho e a qualidade do modelo. Essa qualidade serve para verificar o quanto o modelo consegue generalizar ao receber novos dados. Durante essa etapa, calcula-se os valores de *score* que representa a qualidade dos modelos preditivos e os valores para a métrica RMSE (Equação 2.8) como forma de verificar o erro. A avaliação desses modelos permite escolher qual deles melhor se ajusta aos dados e consegue obter bons re-

sultados quando colocado em produção. Como meio de validar a implementação do melhor modelo no serviço de predição, o valor do *score* é utilizado para verificar se o determinado modelo em produção possui um estado adequado para novas predições. Assim, um baixo *score* indica que o modelo precisa voltar ao treinamento.

A Figura 14 mostra o diagrama que ilustra a avaliação de desempenho dos modelos preditivos. No passo 1, o *dataset* representa os dados de treinamento. No passo 2, as 30 execuções de 10 *folds cross-validation* (validação cruzada) são processadas. Enquanto que no passo 3, os valores médios do R^2 e do erro são calculados e guardados para comparação dos resultados de cada algoritmo treinado.

Figura 14: Diagrama de avaliação dos algoritmos de ML



Fonte: elaborada pelo autor.

Em seguida, os modelos preditivos com bons resultados de R^2 e baixos valores de erro são selecionados para produção e interação com os usuários. Por isso, é importante realizar um bom treinamento dos algoritmos, uma boa validação e ajustes dos modelos antes da produção. Isso evita que os modelos forneçam para os usuários predições erradas ou distantes de um possível valor real.

4.6 Etapa 6: Comparação das simulações com os modelos

Os resultados comparativos entre as simulações realizadas na Etapa 2 e os modelos gerados pela Etapa 4 e 5 são apresentados no próximo capítulo.

5 RESULTADOS

Este capítulo apresenta os resultados dos experimentos (treinamentos) realizados com os quatro algoritmos de *Machine Learning* detalhados na Seção 2.1 (*Decision Tree* - DT, *Random Forest* - RF, *Support Vector Machine* - SVM e Rede *Multilayer Perceptron* - MLP), que são comparados com os resultados das simulações (Figura 12). Conforme citado anteriormente, esses resultados são referentes à taxa de entrega de pacotes em uma Rede em Malha Sem Fio.

Os algoritmos são avaliados em termos do *Root Mean Squared Error* - RMSE (Seção 2.1.3.2) e do R^2 (Seção 2.1.3.3) dado o cenário em estudo. Cada cenário é composto por 60 pontos (que representam os valores da taxa de entrega), sendo variados os parâmetros de distância entre os nós da rede, tamanho do pacote e intervalo de envio dos pacotes entre os nós, conforme Tabela 3. Um fator importante para um bom desempenho dos algoritmos é obter um baixo erro de predição e um R^2 entre 0 e 1. Quanto mais perto de 1, melhor a qualidade do modelo.

Uma outra contribuição importante deste trabalho é a exibição gráfica dos resultados, que possibilita uma melhor visualização entre os resultados da predição de cada algoritmo (*modelo*) e os resultados da simulação (*traces*). Assim, as Figuras 15, 16, 17 e 18 mostram as curvas geradas pelos algoritmos com suas suposições sobre os dados de validação dos modelos, além da comparação com as simulações. É possível visualizar que a curva predita em alguns casos, está de acordo com o intervalo de confiança dado pelos valores da simulação. Como os cenários estão divididos por diferentes tamanhos de pacote e diferentes valores para o intervalo entre os pacotes enviados na rede, os resultados são apresentados separadamente para cada cenário e para cada algoritmo. Ao longo do capítulo, os resultados são discutidos e analisados.

5.1 Avaliação de desempenho da MLP

A MLP trata de encontrar os pesos adequados, formando uma arquitetura capaz de predizer novos valores para a taxa de entrega. A Figura 15 mostra uma comparação entre as predições com base nos dados de treinamento e os valores reais da simulação sob os diferentes cenários. Os valores preditos estão dentro do intervalo de confiança estabelecido a partir dos valores da simulação. Porém, isso não significa que o algoritmo tem capacidade de generalizar para todos os cenários, sendo necessário a avaliação das métricas R^2 e RMSE.

Nos cenários com o intervalo entre os pacotes de 0.001, a MLP possui baixos erros de predição e resultados significativos em termos de qualidade dos modelos, ou seja, bons valores para R^2. A Tabela 8 exibe os resultados das métricas de avaliação de desempenho para a MLP. Pode-se observar (Figura 8) que o algoritmo obtém ótimas aproximações dos resultados das simulações.

No cenário 1 - referente ao cenário com 256 *bytes* para o tamanho de pacote, o algoritmo obtém 96% de R^2, e 0.17 de erro estimado pela RMSE. O valor de erro é considerado baixo e como pode ser visto na Figura 15a, a curva gerada pela MLP se aproxima da curva real.

No cenário 2 - referente ao cenário com 512 *bytes* para o tamanho de pacote, o algoritmo obtém 97% de R^2, um bom valor para essa métrica, e 0.16 de erro estimado pela RMSE. O valor de erro também é considerado baixo e como pode ser visto na Figura 15a, a curva gerada pela MLP também se aproxima da curva real.

Já no cenário 3 - referente ao cenário com 1024 *bytes* para o tamanho de pacote, o algoritmo obtém 94% de R^2, que também é um bom valor para essa métrica, e 0.23 para RMSE. O valor de erro também é considerado baixo, no entanto, a MLP tem um desempenho mais baixo nesse tipo de cenário. A Figura 15a, mostra a curva gerada pela MLP se aproximando da curva real com um pouco de dificuldades no ajustamento dos dados.

Tabela 8: Métricas de avaliação dos cenários com intervalo entre pacotes de 0.001 para MLP

Métrica	Cenário 1	Cenário 2	Cenário 3
R^2	96%	97%	94%
RMSE	0.17	0.16	0.23

Fonte: elaborada pelo autor.

Para os cenários com o intervalo entre os pacotes de 0.01, a MLP também obtém bons resultados em termos de R^2 e RMSE. A Figura 15b mostra uma aproxi-

mação entre a curva com os valores estimados pelo algoritmo e os valores somente simulados. A Tabela 9 exibe os resultados da MLP nesses cenários.

Tabela 9: Métricas de avaliação dos cenários com intervalo entre pacotes de 0.01 para MLP

Métrica	Cenário 1	Cenário 2	Cenário 3
R^2	93%	94%	94%
RMSE	0.24	0.23	0.22

Fonte: elaborada pelo autor.

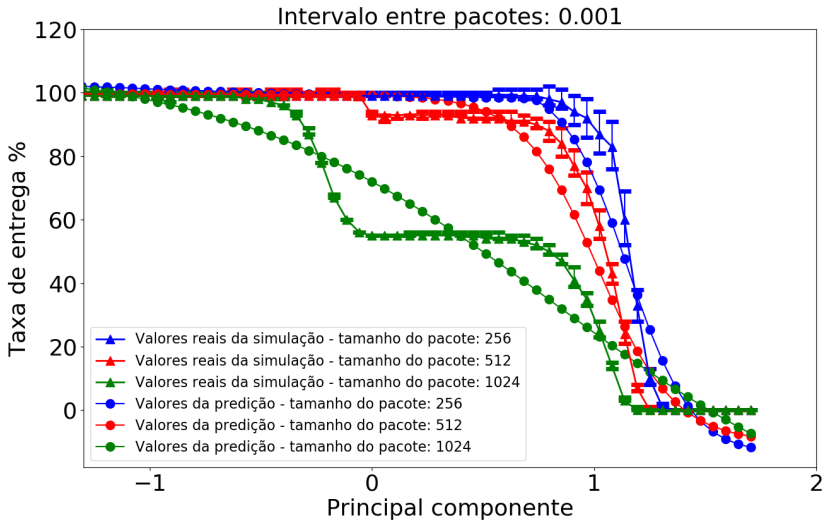
Os resultados também são satisfatórios para os cenários com o intervalo entre pacotes de 0.1. A Figura 15c mostra uma aproximação entre a curva com os valores estimados pelo algoritmo e os valores da simulação. A Tabela 10 exibe os resultados das métricas de avaliação.

Tabela 10: Métricas de avaliação dos cenários com intervalo entre pacotes de 0.1 para MLP

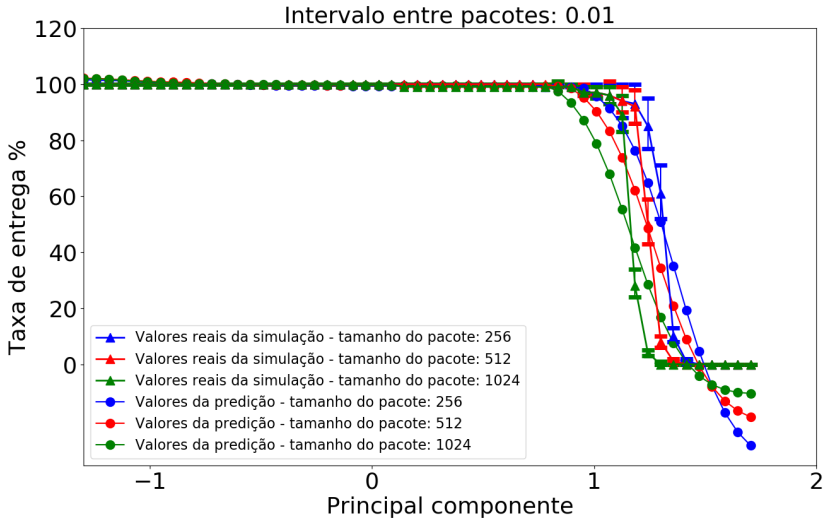
Métrica	Cenário 1	Cenário 2	Cenário 3
R^2	93%	94%	95%
RMSE	0.26	0.23	0.21

Fonte: elaborada pelo autor.

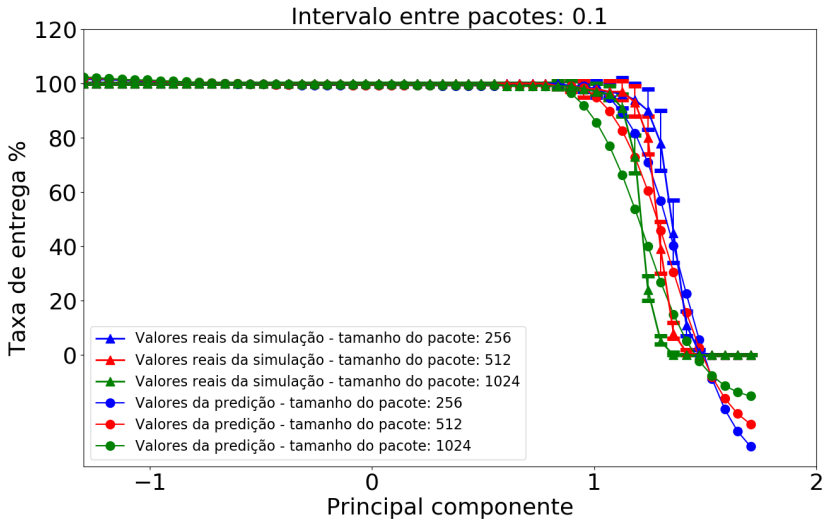
A análise mostra que a MLP consegue melhores resultados em termos de R^2 e RMSE Os resultados mostram que a MLP treinada com os dados dos cenários referentes ao intervalo de 0.001 tem bons valores para R^2 e baixos erros de predição. Especificamente, os resultados são melhores para os cenários com tamanho de pacote igual a 512 *bytes*. Essa arquitetura de MLP treinada é capaz de fornecer boas suposições sobre cenários com características semelhantes aos cenários usados para treinamento. Já para os demais cenários, a MLP também obtém bons resultados, sendo capaz de generalizar.



(a)



(b)



(c)

Figura 15: Simulação x Predição - Modelo: MLP

5.2 Avaliação de desempenho da DT

A DT busca encontrar a árvore que melhor se ajusta aos dados para tomar decisões da taxa de entrega. A Figura 16 mostra uma comparação entre as predições e os valores reais da simulação sob os diferentes cenários.

Diferente dos outros algoritmos, a DT possui resultados bem próximos para os diferentes cenários de redes, obtendo 99% de R^2 para todos os cenários e erros baixos. Os valores preditos pela DT, vistos na Figura 16, estão dentro do intervalo de confiança, assim, mostra a eficiência da DT na predição da taxa de entrega.

No cenário 1 - referente ao cenário com 256 *bytes* para o tamanho de pacote, o algoritmo obtém 99% de R^2 e 0.02 de erro estimado pela RMSE. Na Figura 16a, observa-se que a curva de predição da DT se aproxima da curva real da simulação.

No cenário 2 - referente ao cenário com 512 *bytes* para o tamanho de pacote, o algoritmo também obtém 99% de R^2 e 0.05 de erro estimado pela RMSE. O valor de erro também é considerado baixo e como pode ser visto na Figura 16a, a curva gerada pela DT também se aproxima da curva real.

Já no cenário 3 - referente ao cenário com 1024 *bytes* para o tamanho de pacote, o algoritmo obtém 99% de R^2, e 0.06 para RMSE. A Figura 16a, mostra a curva gerada pela DT procurando se aproximar da curva real.

Tabela 11: Métricas de avaliação dos cenários com intervalo entre pacotes de 0.001 para DT

Métrica	Cenário 1	Cenário 2	Cenário 3
R^2	99%	99%	99%
RMSE	0.02	0.05	0.06

Fonte: elaborada pelo autor.

Os resultados mostram que a DT tem excelentes resultados nos cenários com intervalo de 0.001 explorados neste trabalho. Observando os resultados obtidos, a DT tem o melhor desempenho em termos de RMSE no cenário 1.

A Tabela 12 exibe os resultados das métricas de avaliação para os cenários com o intervalo entre os pacotes de 0.01. O mesmo ocorreu para esses cenários, o algoritmo obtém 99% de R^2 para todos os cenários e erros baixos de predição.

No cenário 1 - referente ao cenário com 256 *bytes* para o tamanho de pacote, o algoritmo obtém 99% de R^2 e 0.08 de erro estimado pela RMSE. A Figura 16b mostra a curva de predição gerada pela DT e seus valores se aproximam da curva real.

No cenário 2 - referente ao cenário com 512 *bytes* para o tamanho de pacote, o algoritmo obtém 99% de R^2, e 0.03 de RMSE. Como pode ser visto na Figura 16b,

a curva gerada pela DT também se aproxima da curva real.

No cenário 3 - referente ao cenário com 1024 *bytes* para o tamanho de pacote, o algoritmo obtém 99% de R^2 e 0.01 de RMSE. A Figura 16b mostra o melhor desempenho da DT. Nota-se que o algoritmo também tem um desempenho satisfatório nesse cenário.

Tabela 12: Métricas de avaliação dos cenários com intervalo entre pacotes de 0.01 para DT

Métrica	Cenário 1	Cenário 2	Cenário 3
R^2	99%	99%	99%
RMSE	0.08	0.03	0.01

Fonte: elaborada pelo autor.

Os cenários com o intervalo entre os pacotes de 0.1, a DT apresentam baixos erros de predição. Em termos de qualidade dos modelos, os resultados são totalmente favoráveis. Na Tabela 13, são apresentados os valores para as métricas R^2 e RMSE.

No cenário 1 - referente ao cenário com 256 *bytes* para o tamanho de pacote, o algoritmo obtém 95% de R^2 e 0.20 de erro estimado pela RMSE. A Figura 16c exibe o comportamento da DT na predição da taxa de entrega em relação aos valores reais.

No cenário 2 - referente ao cenário com 512 *bytes* para o tamanho de pacote, o algoritmo obtém 99% de R^2 e 0.07 de erro estimado pela RMSE. A curva predita desse cenário é mostrada na 16c.

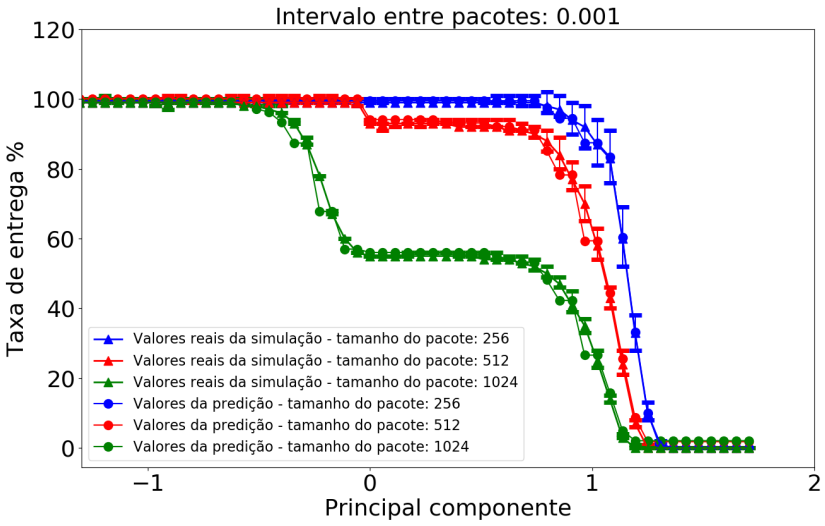
Já no cenário 3 - referente ao cenário com 1024 *bytes* para o tamanho de pacote, o algoritmo obtém 99% de R^2 e 0.02 para RMSE.

Tabela 13: Métricas de avaliação dos cenários com intervalo entre pacotes de 0.1 para DT

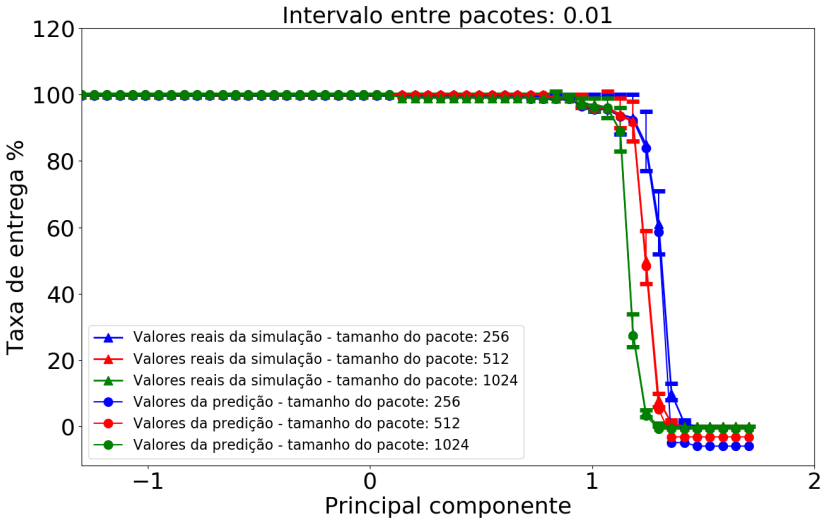
Métrica	Cenário 1	Cenário 2	Cenário 3
R^2	95%	99%	99%
RMSE	0.20	0.07	0.02

Fonte: elaborada pelo autor.

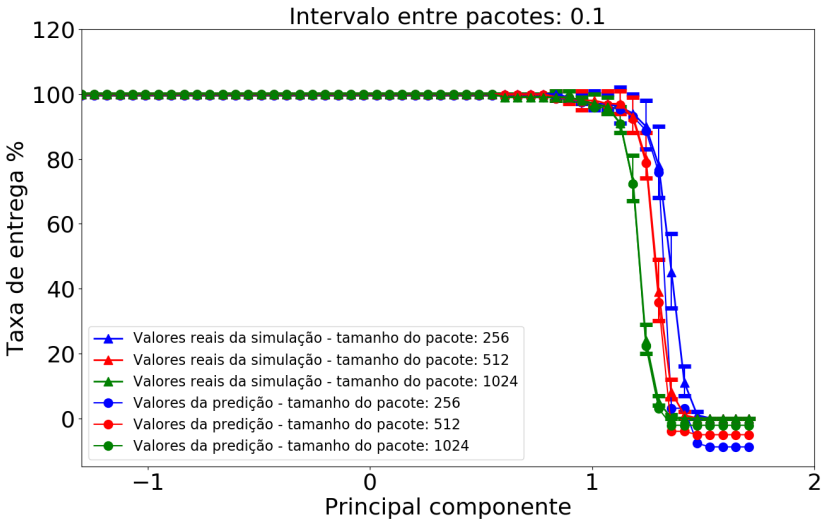
A DT tem resultados próximos quando aplicados em diferentes cenários de redes, ou seja, existe um comportamento padrão deste algoritmo. Os resultados mostram que a DT tem um bom desempenho em todos os cenários avaliados.



(a)



(b)



(c)

Figura 16: Simulação x Predição - Modelo: DT

5.3 Avaliação de desempenho do RF

É um dos modelos mais simples que geralmente obtém resultados satisfatórios. O RF separa os atributos do *dataset* para formar um conjunto de árvores através da abordagem *Ensamble Learning* e a partir dessa separação, o algoritmo consegue classificar um novo objeto. A Figura 17 mostra o comportamento desse algoritmo nos cenários em estudo. A Tabela 14 mostra os resultados das métricas de avaliação.

No cenário 1 - referente ao cenário com 256 *bytes* para o tamanho de pacote, o algoritmo obtém 99% de R^2 e um valor de 0.08 de RMSE. Esse algoritmo tem boas suposições nos seus resultados. Na Figura 18a é possível visualizar a curva de predição gerada pelo RF.

No cenário 2 - referente ao cenário com 512 *bytes* para o tamanho de pacote, o algoritmo obtém um excelente resultado, com 98% de R^2 e com um valor de erro de 0.12. A Figura 18a mostra a curva de predição gerada pelo RF que também se aproxima da curva real.

Enquanto no cenário 3 - referente ao cenário com 1024 *bytes* para o tamanho de pacote, o algoritmo obtém 99% de R^2 e 0.05 de RMSE. Esses resultados também indicam um bom comportamento desse algoritmo nesse cenário.

Tabela 14: Métricas de avaliação dos cenários com intervalo entre pacotes de 0.001 para RF

Métrica	Cenário 1	Cenário 2	Cenário 3
R^2	99%	98%	99%
RMSE	0.08	0.12	0.05

Fonte: elaborada pelo autor.

A Tabela 9 exibe esses valores, já a Figura 18b exibe um comportamento ajustado do RF nos cenários 1, 2 e 3.

Tabela 15: Métricas de avaliação dos cenários com intervalo entre pacotes de 0.01 para RF

Métrica	Cenário 1	Cenário 2	Cenário 3
R^2	99%	99%	99%
RMSE	0.05	0.06	0.06

Fonte: elaborada pelo autor.

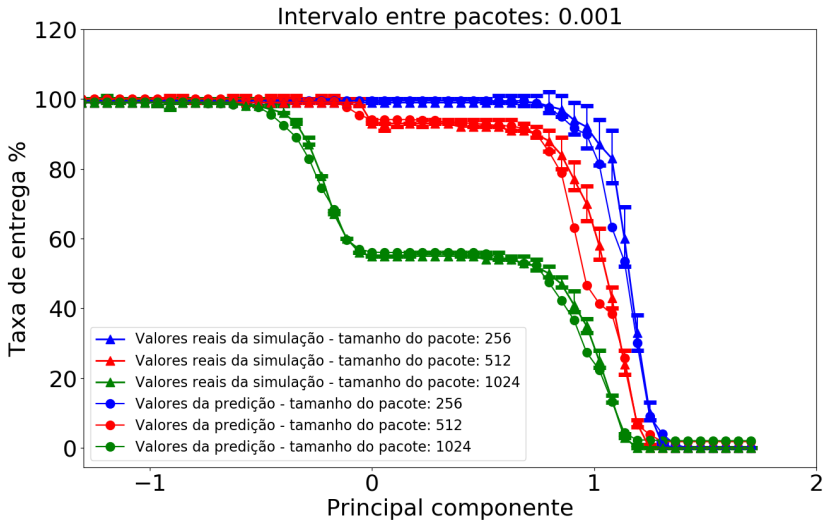
O mesmo ocorre com os cenários com intervalo entre pacotes de 0.1, o RF consegue bons resultados de R^2 e RMSE para todos os cenários. A Tabela 16 exibe os valores calculados por R^2 e RMSE. O comportamento do algoritmo pode ser visualizado na Figura 17.

Tabela 16: Métricas de avaliação dos cenários com intervalo entre pacotes de 0.1 para RF

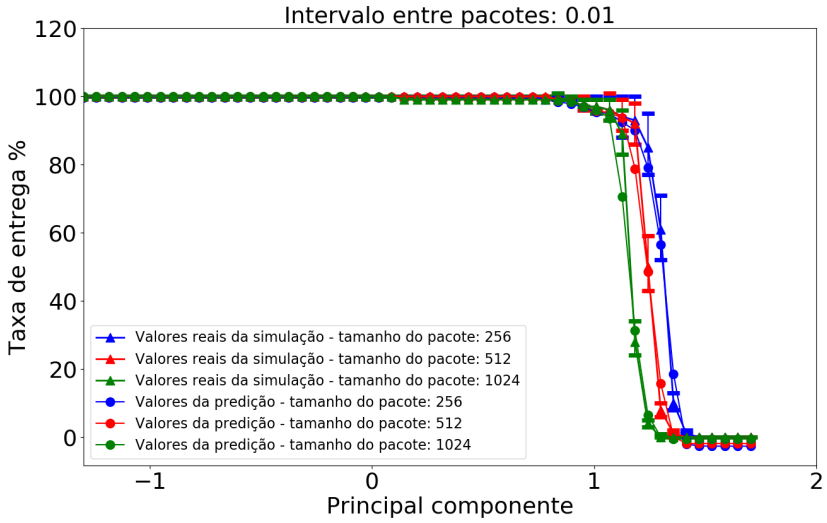
Métrica	Cenário 1	Cenário 2	Cenário 3
R^2	98%	99%	99%
RMSE	0.13	0.06	0.07

Fonte: elaborada pelo autor.

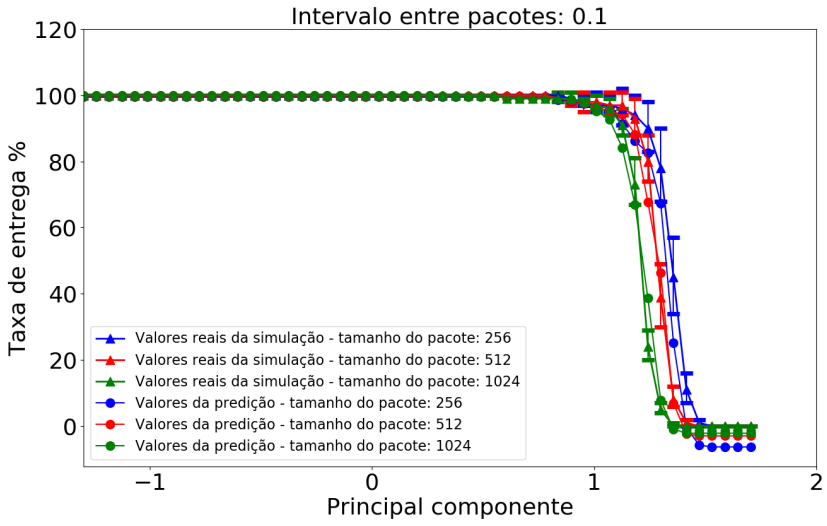
O *Random Forest* (RF) também tem bons resultados de desempenho para os diversos cenários de redes em estudo. É um algoritmo com boas referências de resultados satisfatórios. Os resultados obtidos pelo RF mostram que o algoritmo tem capacidade de trabalhar com as simulações e fornecer boas suposições.



(a)



(b)



(c)

Figura 17: Simulação x Predição - Modelo: RF

5.4 Avaliação de desempenho da SVM

Os resultados mostram que a SVM não se adapta aos cenários de redes, se comportando de maneira inversa aos demais algoritmos que se ajustam melhor aos cenários. A Tabela 17 mostra valores não satisfatórios para as métricas de avaliação dos cenários com o intervalo entre os pacotes de 0.001. A Figura 18 exibe o comportamento do algoritmo para esse cenário e, acaba mostrando que a curva predita não acompanha a curva real.

Tabela 17: Métricas de avaliação dos cenários com intervalo entre pacotes de 0.001 para SVM

Métrica	Cenário 1	Cenário 2	Cenário 3
R^2	-0.002%	-0.0009%	-0.0002%
RMSE	1.0	1.0	1.0

Fonte: elaborada pelo autor.

Diferentemente dos cenários anteriores, a SVM apresenta resultados mais precisos para os demais cenários, cujos valores para intervalo entre pacotes são 0.01 e 0.1. As Tabelas 18 e 19 exibem os resultados obtidos pelo algoritmo nos cenários com 0.01 e 0.1 para intervalo entre os pacotes.

Tabela 18: Métricas de avaliação dos cenários com intervalo entre pacotes de 0.01 para SVM

Métrica	Cenário 1	Cenário 2	Cenário 3
R^2	96%	95%	93%
RMSE	0.18	0.22	0.25

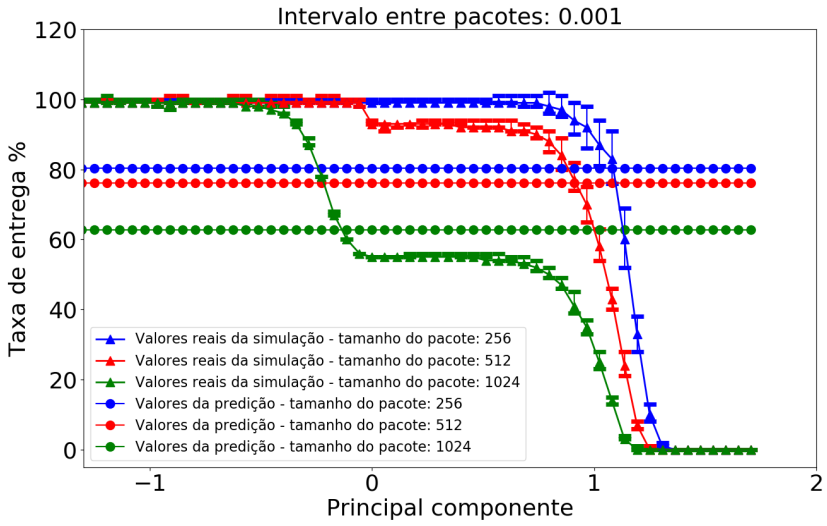
Fonte: elaborada pelo autor.

Os melhores resultados da SVM são para o intervalo de confiança de 0.1, pois o algoritmo obtém os erros de predição mais baixos em comparação aos outros cenários, fazendo com que o algoritmo tenha boas suposições com novos dados desse tipo de cenário.

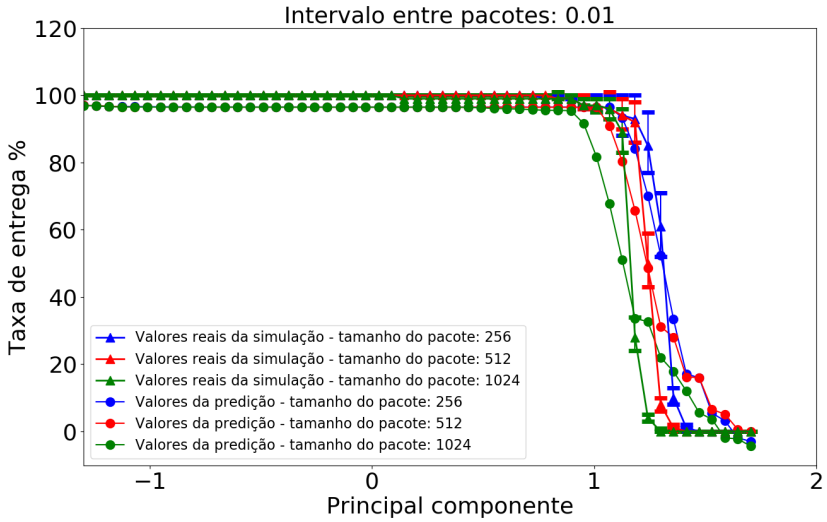
Tabela 19: Métricas de avaliação dos cenários com intervalo entre pacotes de 0.1 para SVM

Métrica	Cenário 1	Cenário 2	Cenário 3
R^2	97%	96%	95%
RMSE	0.17	0.18	0.21

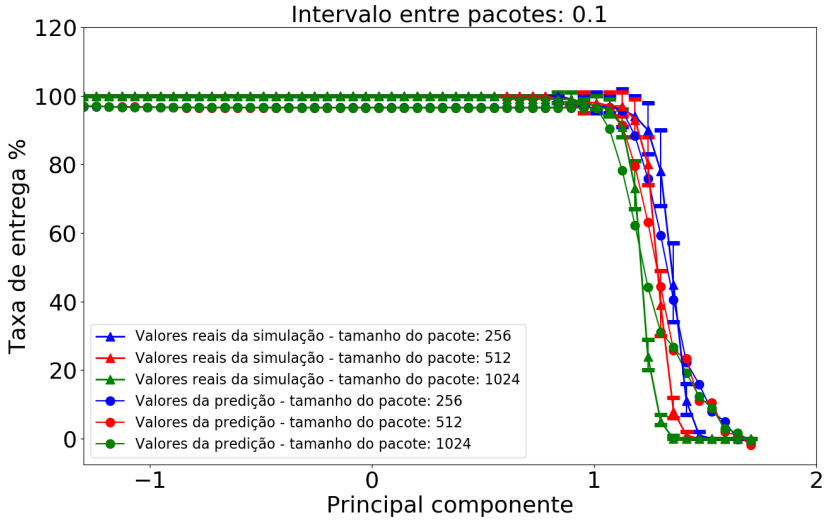
Fonte: elaborada pelo autor.



(a)



(b)



(c)

Figura 18: Simulação x Predição - Modelo: SVM

Fazendo uma análise geral de todos os experimentos e abordagens usadas neste trabalho, os algoritmos obtiveram resultados significativos em termos de desempenho para trabalhar com cenários de redes. Apesar de algumas configurações dos algoritmos serem padrão da biblioteca em uso ou consideradas simples, os modelos estão próximos do esperado. Porém, o SVM tem resultados bem diferentes dos outros algoritmos.

Alguns modelos não se ajustaram aos dados, causando resultados imprecisos como a SVM no cenário 1, 2 e 3, cujo o intervalo entre pacotes é de 0.001. Enquanto os demais algoritmos possuem resultados mais precisos e são capazes de generalizar sob novos dados. Tanto o DT quanto o RF, os resultados são satisfatórios e mostram a eficiência desses algoritmos que possuem características semelhantes.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou uma avaliação de algoritmos de *Machine Learning* na otimização de simulações de redes de computadores a fim de reduzir o tempo e recursos computacionais das simulações. Essa avaliação foi possível por meio de experimentos realizados com os algoritmos de *Machine Learning* com o intuito de analisar o desempenho desses algoritmos nos diferentes cenários de redes. Para isso, cenários de redes foram definidos e simulados no NS-3 para coleta dos dados. Os algoritmos foram treinados a partir dos *datasets* gerados após as simulações de redes. O trabalho teve como estudo de caso uma rede em Malha Sem Fio e a taxa de entrega como medida de desempenho de redes de computadores. Apesar da topologia ser considerada simples, três dos algoritmos utilizados tiveram resultados satisfatórios em relação ao desempenho observado nas predições e se mostraram capazes de auxiliar nas simulações de redes de computadores.

Para realização dos experimentos de *Machine Learning*, quatro algoritmos de diferentes abordagens e voltados para a tarefa de regressão foram configurados de acordo com os melhores parâmetros retornados por uma técnica de otimização de parâmetros usada neste trabalho. O uso de uma técnica de otimização de parâmetros evita a configuração manual dos algoritmos e a realização de experimentos, tornando um processo repetitivo. Com a aplicação do *Grid search*, técnica de otimização de parâmetros utilizada, experimentos repetitivos de *Machine Learning* foram evitados, por exemplo, configuração manual de diferentes parâmetros para os algoritmos.

Os algoritmos *Decision Tree*, *Random Forest*, SVM e a Rede *Multilayer Perceptron* foram analisados por meio das métricas R^2 e *Root Mean Squared Error*. Dentre os algoritmos utilizados, a *Decision Tree* teve resultados mais concretos e próximos dos reais com até 99% de taxa de acerto na maioria dos cenários. Os resultados evidenciam que os algoritmos de ML são capazes de predizer os valores da taxa de entrega com precisão. Contudo, a SVM teve resultados precisos para alguns dos cenários e resultados imprecisos para os cenários com o intervalo entre pacotes de 0.001. Com uma série de novos experimentos e novos cenários de redes, os resultados dos algoritmos podem melhorar e, consequentemente, o comportamento de cada algoritmo em relação aos dados também.

Em relação às perspectivas de trabalhos futuros, destaca-se a criação de novos *datasets* baseados em outros cenários de redes, realização de novos experimentos de *Machine Learning*, estudo de caso usando outras medidas de desempenho de redes, seleção de outros parâmetros de configuração dos algoritmos, separação dos *datasets* em treino, teste e validação para uma melhor avaliação.

REFERÊNCIAS

ABREU, C. E. M. et al. Indústria 4.0: Como as empresas estão utilizando a simulação para se preparar para o futuro. *Revista de Ciências Exatas e Tecnologia*, 2017. Citado na página 12.

ACADEMY, D. S. *Deep Learning Book*. Data Science Academy, 2018. Disponível em: <http://www.deeplearningbook.com.br>. Citado na página 23.

AKYILDIZ, I. F.; WANG, X.; WANG, W. Wireless mesh networks: a survey. *Computer Networks*, v. 47, n. 4, p. 445 – 487, 2005. ISSN 1389-1286. Citado 3 vezes nas páginas 14, 30 e 39.

ANDREEV, K.; BOYKO, P. Ieee 802.11s mesh networking ns-3 model. In: *Workshop on ns-3 (WNS3)*. [S.l.: s.n.], 2010. Citado na página 40.

BALADEZ, F. O passado, o presente e o futuro dos simuladores. *Periódico Eletrônico da FATEC-São Caetano do Sul - Fasci-Tech*, 2009. Citado na página 12.

DRAKOS, G. *How to select the Right Evaluation Metric for Machine Learning Models: Part 1 Regression Metrics*. 2018. Disponível em https://towardsdatascience.com/. Citado 2 vezes nas páginas 27 e 28.

FACELI, K. et al. *Inteligência artificial: uma abordagem de aprendizado de máquina*. Grupo Gen - LTC, 2011. ISBN 9788521618805. Disponível em: <https://books-.google.com.br/books?id=4DwelAEACAAJ>. Citado 8 vezes nas páginas 16, 17, 18, 19, 23, 24, 26 e 27.

HWANG, A.; LEE, J.; KIM, B. Design and performance evaluation of tcp performance enhancement algorithm with machine learning in wireless environments. *International Journal of Applied Engineering Research*, v. 12, p. 14370–14376, 01 2017. Citado 2 vezes nas páginas 32 e 33.

IEEE 802.11s. *IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 10: Mesh Networking, IEEE Std.* 2011. Citado na página 30.

JAIN, R. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* [S.l.]: Wiley, 1991. I-XXVII, 1-685 p. (Wiley professional computing). ISBN 978-0-471-50336-1. Citado na página 40.

JOLLIFFE, I. Principal component analysis. In: ____. *International Encyclopedia of Statistical Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 1094–1096. ISBN 978-3-642-04898-2. Disponível em: <https://doi.org/10.1007/978-3-642-04898-2_455>. Citado na página 45.

JOSHI, P. *Artificial Inteligence with Python - Build real-world Artificial Intelligence applications with Python to intelligently interact with the world around you*. [S.l.]: Packt Publishing Ltd, 2017. ISBN 9781786464392. Citado na página 16.

KAPIL, D. *Hyperparameter Search: Techniques to pick the most Optimal Set*. 2019. Disponível em https://medium.com/analytics-vidhya/hyperparameter-search-part-1-2b67fd7a71d8. Citado na página 25.

LEE, K. Y.; SUH, Y.-K.; CHO, K. W. Development of a simulation result management and prediction system using machine learning techniques. *International Journal of Data Mining and Bioinformatics*, 2017. Citado 3 vezes nas páginas 12, 13 e 34.

LOUREIRO, B. P. S. L. A. M. S. C. S. F. S. C. J. B. B. B. S. P. M. A. M. V. L. F. M. V. O. N. G. A. A. F. nternet das coisas: da teoria à prática. *Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)*, 2016. Citado na página 13.

MITCHELL, T. M. *Machine Learning*. 1. ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN 0070428077, 9780070428072. Citado na página 16.

NERDY, F. *What Is R Squared And Negative R Squared*. 2018. Disponível em http://www.fairlynerdy.com/what-is-r-squared/. Citado na página 29.

NS-3. *Network Simulator 3*. 2019. Disponível em: <https://www.nsnam.org/>. Acesso em: 25 ago. 2019. Citado 2 vezes nas páginas 14 e 40.

OLIVEIRA, B. T. D. *Um protótipo de simulador empresarial acadêmico para o setor de atacado*. 87 p. Monografia (Graduação) — Bacharelado em Ciência da Computação, Instituto Federal de Minas Gerais, Minas Gerais, 2017. Citado na página 13.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011. Citado na página 44.

PRADELLA, M. *A importância do uso de simulações no mundo corporativo*. 2013. Disponível em http://www.informamidia.com.br/artigo-a-importancia-do-uso-de-simulacoes-no-mundo-corporativo/. Citado na página 12.

SAGGIORO, L. F. Z.; GONZAGA, F. B.; RIBEIRO, R. Tracemetrics - uma ferramenta para a obtenção de medidas de interesse no ns-3. *XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, 2012. Citado na página 35.

SANTOS, V. C. dos; SILVEIRA, G. A. da. A efetividade dos simuladores de voo no treinamento de pilotos para tarefas processuais específicas e aquisição de habilidades. *Revista Conexão SIPAER*, v. 10, n. 1, p. 15–30, 2019. Citado na página 13.

SENAPATI, D. *Grid Search vs Random Search*. 2018. Disponível em https://medium.com/@senapati.dipak97/grid-search-vs-random-search-d34c92946318. Citado na página 25.

SILVA, C. et al. Performance evaluation of wireless mesh networks in smart cities scenarios. In: *Proceedings of the Euro American Conference on Telematics and Information Systems, EATIS 2018, Fortaleza, Brazil, November 12-15, 2018*. [s.n.], 2018. p. 7:1–7:7. Disponível em: <https://doi.org/10.1145/3293614.3293615>. Citado na página 30.

SILVA, E. A. D. *Uma Ferramenta para Extração de Medidas de Desempenho no Simulador NS-3*. 56 p. Monografia (Graduação) — Bacharelado em Sistemas de Informação, Universidade Federal de Lavras, Minas Gerais, 2014. Citado na página 35.

SINAPAD. *Sistema Nacional de Processamento de Alto Desempenho*. 2019. Disponível em: <AbnTeX>. Acesso em: 25 ago. 2019. Citado na página 12.

SMOLYAKOV, V. *Ensemble Learning to Improve Machine Learning Results*. 2017. Disponível em https://blog.statsbot.co/ensemble-learning-d1dcd548e936. Citado 2 vezes nas páginas 19 e 20.

WORLD, H. in a M. *MAE and RMSE — Which Metric is Better - Mean Absolute Error versus Root Mean Squared Error*. 2016. Disponível em https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d. Citado 2 vezes nas páginas 27 e 28.

ZHENG, A. *Evaluating Machine Learning Models - A Beginner's Guide to Key Concepts and Pitfalls*. [S.l.]: O'Reilly, 2015. ISBN 9781491932469. Citado 2 vezes nas páginas 24 e 25.