UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Renan Lírio de Souza

MLFV: UMA ABORDAGEM CONSCIENTE DO ESTADO DA REDE PARA ORQUESTRAÇÃO DE CADEIAS DE FUNÇÕES DE APRENDIZADO DE MÁQUINA

Santa Maria, RS
2019

Renan Lírio de Souza

MLFV: UMA ABORDAGEM CONSCIENTE DO ESTADO DA REDE PARA ORQUESTRAÇÃO DE CADEIAS DE FUNÇÕES DE APRENDIZADO DE MÁQUINA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação(PGCC) da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Mestre em Ciência da Computação**.

Orientador: Prof. Dr. Alencar Machado

Co-orientador: Prof. Dr. Celio Trois

Santa Maria, RS

2019

Souza, Renan Lírio de
MLFV: Uma Abordagem Consciente do Estado da Rede para Orquestração de Cadeias de Funções de Aprendizado de Máquina / Renan Lírio de Souza.- 2019.
82 p.; 30 cm

Orientador: Alencar Machado
Coorientador: Celio Trois
Dissertação (mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Ciência da Computação , RS, 2019

1. Aprendizado de Máquina 2. Virtualização de Funções de Rede 3. Encadeamento de Funções 4. Computação na Borda I. Machado, Alencar II. Trois, Celio III. Título.

Sistema de geração automática de ficha catalográfica da UFSM. Dados fornecidos pelo autor(a). Sob supervisão da Direção da Divisão de Processos Técnicos da Biblioteca Central. Bibliotecária responsável Paula Schoenfeldt Patta CRB 10/1728.

Renan Lírio de Souza

MLFV: UMA ABORDAGEM CONSCIENTE DO ESTADO DA REDE PARA ORQUESTRAÇÃO DE CADEIAS DE FUNÇÕES DE APRENDIZADO DE MÁQUINA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação(PGCC) da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Mestre em Ciência da Computação**.

Aprovado em 21 de 08 de 2019:

Alencar Machado, Dr. (UFSM)
(Presidente/Orientador)

Tiago Antônio Rizzetti, Dr. (UFSM)

Rodolfo da Silva Villaca, Dr. (UFES)

Santa Maria, RS

2019

AGRADECIMENTOS

Em primeiro lugar agradeço à minha família, em especial aos meus pais, Oscar e Suzana, minha irmã, Rebeca, por estarem ao meu lado durante essa jornada, pelo carinho e por todo o apoio que sempre recebi.

Ao meu orientador Prof. Dr. Alencar Machado, pela confiança, incentivos, ensinamentos e por toda ajuda dispendida antes e durante o desenvolvimento deste trabalho.

Ao Prof. Dr. Magnos Baroni, pelas informações e esclarecimentos na área da Engenharia Civil e pela disponibilização dos dados que foram essenciais para a conclusão deste trabalho.

Ao meu co-orientador, Prof. Dr. Celio Trois, pela dedicação, companheirismo e incansável auxílio em todos os momentos, contribuindo de forma ímpar com o desenvolvimento desta pesquisa.

Aos meus amigos e colegas de mestrado, William, Un Hee, Milene, Maria e Isadora, pela amizade, apoio e por todos os bons momentos de convívio durante esses anos. Aos meus amigos Raul, Henrique, Felipe, Otávio, Talisson, Douglas, pelos momentos de descontração e amizade. Ainda, aos amigos da minha cidade natal que, apesar da distância, sempre estiveram presentes na minha vida de alguma forma.

Agradeço a todos que, direta ou indiretamente, contribuíram para a conclusão desta dissertação. Por fim, agradeço a Deus por toda a sabedoria, proteção, força, e por me abençoar muito mais do que eu mereço.

"Quando uma criatura humana desperta para um grande sonho e sobre ele lança toda a força de sua alma, todo o universo conspira a seu favor."

(JOHANN GOETHE)

RESUMO

MLFV: UMA ABORDAGEM CONSCIENTE DO ESTADO DA REDE PARA ORQUESTRAÇÃO DE CADEIAS DE FUNÇÕES DE APRENDIZADO DE MÁQUINA

AUTOR: RENAN LÍRIO DE SOUZA
ORIENTADOR: ALENCAR MACHADO
CO-ORIENTADOR: CELIO TROIS

As plataformas de Aprendizado de Máquina como Serviço (MLaaS) vem possibilitando a aplicação de técnicas de Aprendizado de Máquina (ML) de qualquer lugar e a qualquer momento. Essas plataformas, em geral, são hospedadas na nuvem e possuem infraestruturas escaláveis com um alto poder de processamento; no entanto, apresentam algumas desvantagens como a necessidade de realizar a transferência dos dados para a nuvem. A aplicação de ML na borda da rede (*Edge Computing*) está surgindo como uma opção para reduzir algumas das limitações impostas por essas plataformas, diminuindo, por exemplo, a latência e o uso da rede; além disso, evita problemas com relação a privacidade e segurança dos dados por mantê-los na rede interna. No entanto, a aplicação de ML na borda ainda apresenta desafios de pesquisa, como, por exemplo, a orquestração de funções de ML levando em consideração o atual estado da rede e as capacidades computacionais dos nodos. Neste sentido, serviços de orquestração de funções conscientes do estado da rede, como os fornecidos por plataformas de Virtualização de Funções de Rede (NFV), podem ser uma abordagem promissora para gerenciar a distribuição das tarefas de ML. Este trabalho propõe o MLFV (*Machine Learning Function Virtualization*), uma abordagem consciente do estado da rede que explora um ambiente de NFV para orquestrar a execução de cadeias de funções de ML; essas cadeias representam o fluxo de execução das funções, podendo ser agrupadas de forma sequencial e/ou paralela. A abordagem de MLFV implementa um modelo matemático para colocação de cadeias de ML que, considerando restrições de CPU, memória, bibliotecas necessárias e sobrecarga de rede, objetiva reduzir o tempo total de execução de todas as funções de uma cadeia. O modelo distribui as funções de forma a reduzir a sobrecarga na rede, o que acarreta na redução no tempo total de execução, principalmente em casos em que a rede apresenta instabilidade. Para avaliar a abordagem de MLFV foi realizado um estudo de caso na área de geotecnia, utilizando dados de sensores para reproduzir o processo de classificação de solos através de duas cadeias de funções de ML; essas cadeias, implementadas pelo MLFV, foram criadas com base no processo de Descoberta de Conhecimento em Base de Dados (KDD). Os experimentos demonstraram que o MLFV obteve, em média, uma redução no tempo de execução de 25% em comparação às abordagens de ML na nuvem (MLaaS) e na borda (Edge) em um cenário com conexão de rede estável. Em um cenário onde os nodos computacionais apresentavam restrições de largura de banda, o MLFV foi capaz de identificar essas limitações, alocando as tarefas de ML nos *hosts* com conexão estável. As outras abordagens não foram capazes de detectar essas instabilidades, acarretando em um aumento de 400% no tempo de execução das cadeias de funções.

Palavras-chave: Aprendizado de Máquina. Virtualização de Funções de Rede. Encadeamento de Funções. Computação na Borda.

ABSTRACT

MLFV: A NETWORK-AWARE APPROACH FOR MACHINE LEARNING FUNCTION CHAIN ORQUESTRATION

AUTHOR: RENAN LÍRIO DE SOUZA
ADVISOR: ALENCAR MACHADO
COADVISOR: CELIO TROIS

Machine Learning as a Service (MLaaS) platforms are allowing the application of Machine Learning (ML) techniques from anywhere, and at any time. These platforms, in general, are hosted in the cloud and have a scalable infrastructure with high processing power; however, they have some disadvantages, such as the need to send data to the cloud. ML on the Edge (Edge Computing) is emerging as an option to tackle some limitations imposed by these platforms, reducing the latency and bandwidth usage; furthermore, it avoids data privacy and security issues by keeping the data on the local network. However, the application of ML on the edge still presents research challenges, such as the orchestration of ML functions considering the network state and the computational capabilities of the nodes. In this sense, network-aware orchestration services, provided by Network Function Virtualization (NFV) platforms can be a promising approach to manage the ML tasks placement. This work proposes the MLFV (Machine Learning Function Virtualization), a network-aware approach that explores the NFV environment to orchestrate the execution of ML function chains; these chains represent the execution flow of the ML functions that can be grouped as sequential and/or parallel activities. The MLFV implements a model for placing chains of ML, considering constraints on CPU, memory, required libraries and the network overload aiming to reduce the overall execution time of all functions in a chain. This model distributes the functions in order to reduce the network overload, the execution time, especially in cases where the network presents some instability. To evaluate the MLFV proposal a case study in the geotechnical area was conducted, using soil data to reproduce the soil classification process through two ML function chains; these chains, implemented by MLFV, were created based on the Knowledge Discovery in Databases process (KDD). The results showed that MLFV achieved, on average, a 25% reduction in the execution time compared to cloud (MLaaS) and edge approaches in a stable network connection scenario. When some computational nodes had bandwidth constraints, MLFV was able to identify these limitations, allocating the ML tasks on hosts with stable connections. The other approaches were unable to detect these instabilities, resulting in a 400% increase in the overall chain execution time.

Keywords: Machine Learning. Network Function Virtualization. Function Chaining. Edge Computing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANN	*Artificial Neural Networks*
CPTu	*Cone Penetration Test*
DM	*Data Mining*
DT	*Decision Tree*
DPI	*Deep Packet Inspection*
IoT	*Internet of Things*
KDD	*Knowledge Discovery in Databases*
k-NN	*k-Nearest Neighbors*
ML	*Machine Learning*
MLaaS	*Machine Learning as a Service*
MLFC	*Machine Learning Function Chain*
MLFV	*Machine Learning Function Virtualization*
MLVF	*Machine Learning Virtual Function*
NFC	*Network Function Chaining*
NFV	*Network Function Virtualization*
NSH	*Network Service Header*
QoE	*Quality of Experience*
RF	*Random Forest*
SBT	*Soil Behavior Type*
SDN	*Software-Defined Networking*
SFC	*Service Function Chaining*
SFF	*Service Function Forwarder*
SQL	*Structured Query Language*
SVM	*Support Vector Machines*
VM	*Virtual Machine*
VNF	*Virtual Network Function*
VXLAN	*Virtual Extensible LAN*

SUMÁRIO

1 INTRODUÇÃO

Com o advento e popularização de tecnologias como *Internet of Things* (IoT), Cidades inteligentes e plataformas de *crowdsourcing*, surgem novas oportunidades para criação e utilização de uma variedade de dispositivos e sensores, permitindo o desenvolvimento de aplicações que gerenciam esses equipamentos, em diferentes domínios (gestão de tráfego, saúde, agricultura e finanças). Além disso, o aumento das capacidades de clientes móveis (*smartphones*) e o desejo dos usuários por conectividade de alta velocidade, sempre ativa e orientada a multimídia, resulta em uma enorme quantidade de dados que deve ser transmitida e processada.

Para suportar esse novo paradigma, onde uma quantidade significativa de dados é gerada diariamente, cientistas e especialistas da área de inteligência artificial utilizam diferentes técnicas para auxiliar no processamento e extração de informações desses dados. O processo de Descoberta de Conhecimento em Banco de Dados (*KDD*), por exemplo, permite a descoberta de informações em base de dados através de uma cadeia de atividades bem definida; esse processo consiste nas atividades de seleção, pré-processamento, transformação e Mineração de Dados utilizando diferentes funções matemáticas, estatísticas e de técnicas de Aprendizado de Máquina, ou *Machine Learning* (ML), para criação de modelos capazes de identificar padrões e extrair conhecimento dos dados.

Neste contexto, as técnicas de ML vem recebendo cada vez mais atenção, de modo que pesquisadores e empresas estão se esforçando para fornecer plataformas de Aprendizado de Máquina como Serviço (*Machine Learning as a Service* (MLaaS)) para auxiliar o processo de criação, implementação e disponibilização desses modelos para extração de conhecimento. As plataformas de MLaaS buscam facilitar o entendimento e acesso a funções e algoritmos de ML através de serviços hospedados na nuvem, permitindo que vários usuários acessem os recursos disponíveis sob demanda, de qualquer lugar e a qualquer momento (RIBEIRO; GROLINGER; CAPRETZ, 2015).

No entanto, como apresentado por ZHAO et al. (2018), existem diversos desafios não abordados pelas plataformas de MLaaS, mas que devem ser considerados durante a utilização de métodos analíticos de dados; por exemplo, a definição dos requisitos como o volume, a variedade e a velocidade. Em alguns casos é impraticável o envio de todos os dados através da rede para que sejam analisados nas plataformas de MLaaS; as vezes devido ao volume de dados elevado (restrição de velocidade) e, em outros casos, devido a urgência no processamento

dessas informações (restrição de latência). Além disso, a privacidade dos dados e restrições na conexão entre os serviços são questões importantes a serem avaliadas.

Parte dos desafios relacionados à conexão e privacidade dos dados podem ser solucionados movendo as plataformas de MLaaS, hospedadas na nuvem, para a borda da rede (ZHAO et al., 2018). Uma vez que o conceito de *Edge Computing* (ou *Fog Computing*) é caracterizado pela proximidade entre os usuários e os dados, isso acaba proporcionando benefícios como menor latência, privacidade e flexibilidade. No entanto, até onde se sabe, nenhuma das propostas existentes na literatura para execução de ML na borda considera as condições atuais da rede[1] para orquestrar a colocação das atividades e serviços de ML. Dada a estreita relação entre latência e volume dos dados, é importante estar ciente do estado atual da rede para escolher os dispositivos que executarão as tarefas de ML; isso é particularmente importante em ambientes com largura de banda de rede limitada, como conexões 3g, com interferências ou de baixa qualidade.

Já os desafios relacionados a organização e orquestração de atividades (ou funções), levando em consideração o estado da rede, podem ser solucionados através de tecnologias já consolidadas, tais como plataformas de *Network Function Virtualization* (NFV) (Matias et al., 2015; DOMINICINI et al., 2017) e *Network Function Chaining* (NFC) (CASTANHO et al., 2018; Medhat et al., 2017). Essas abordagens permitem a alocação, encadeamento e virtualização de funções de rede, possibilitando a criação e implantação de serviços compostos, de forma virtual, em hardwares genéricos. Além disso, habilitam outros recursos importantes como elasticidade, flexibilidade e orquestração inteligente considerando o estado da rede. Com isso, as plataformas de NFV se mostram uma abordagem promissora no desenvolvimento de serviços de ML na borda. O atual estado da arte reporta trabalhos unindo técnicas de ML com NFV, mas somente onde o primeiro é utilizado para melhorar o último; por exemplo, o escalonamento automático de *Virtual Network Function*s (VNFs) em resposta a alguma alteração de tráfego (RAHMAN et al., 2018).

Neste sentido, esse trabalho apresenta a proposta de *Machine Learning Function Virtualization* (MLFV)[2], uma abordagem que explora o uso de uma plataforma de NFV para orquestrar e virtualizar as atividades relacionadas ao processo de ML. Ao invés de cadeias de funções de rede, o MLFV realiza a organização, distribuição, virtualização e execução de cadeias de

[1] Entende-se por condições atuais da rede a condição da conexão entre o prestador de serviço de ML para o cliente e os dispositivos que irão executar o processamento dos dados

[2] O MLFV está disponível em: http://www.inf.ufsm.br/~trois/MLFV/

funções de ML; essas cadeias representam os fluxos de execução das funções, podendo serem agrupadas de forma sequencial e/ou paralela. Através da plataforma de NFV, a abordagem de MLFV é capaz de estar ciente do estado atual da rede dos dispositivos conectados, permitindo a execução dessas cadeias nos dispositivos menos sobrecarregados, e com maior largura de banda disponível. Além disso, essa abordagem leva em consideração restrições computacionais dos dispositivos (CPU e memória) e as bibliotecas de ML instaladas para efetuar o processo de orquestração. O MLFV implementa um modelo matemático que considera essas restrições, tendo como principal objetivo minimizar o tempo total de execução de todas as funções de uma cadeia, reduzindo também a sobrecarga na rede.

Para avaliar a proposta de MLFV, um estudo de caso na área de geotecnia foi conduzido; utilizou-se dados de solo, advindos de coleta de campo por meio de sensores, para realizar a tarefa de classificação das distintas camadas que compõem o subsolo através modelos de ML. Duas cadeias de funções de ML, baseadas no processo de KDD, foram implementadas e utilizadas nesse processo. Por fim, foram conduzidos experimentos, utilizando dados do estudo de caso e as cadeias de funções propostas, para comparar a abordagem de MLFV com as seguintes plataformas: (i) Microsoft Azure MLaaS Studio, executado na nuvem; (ii) Dask.distributed, uma biblioteca que permite a execução distribuída de técnicas de ML na borda da rede; (iii) um único dispositivo (PC) não conectado na rede, executando as principais bibliotecas de ML (Scikit-learn v0.20.0 (PEDREGOSA et al., 2011)) localmente.

Para avaliar o desempenho da solução proposta foram conduzidos dois experimentos. O primeiro experimento avaliou o tempo de execução das abordagens durante requisições únicas; o segundo, avaliou o tempo de execução para múltiplas requisições concorrentes. Em ambos testes, o MLFV reduziu o tempo de execução em torno de 25% em comparação as outras plataformas. O último experimento avaliou as abordagens em um ambiente de rede com restrições na largura de banda; nesse caso, a biblioteca Dask.distributed aumentou em 400% o tempo de execução, enquanto o MLFV teve um aumento de apenas 36%. Essa diferença ocorre devido a capacidade do MLFV em identificar as limitações da rede e dos dispositivos, alocando as funções nos nodos menos ocupados e com maior largura de banda.

1.1 OBJETIVOS E CONTRIBUIÇÕES

Este trabalho apresenta uma abordagem para distribuição, alocação e virtualização de atividades relacionadas ao processo de ML e KDD, levando em consideração algumas restri-

ções, como o estado da rede, requisitos computacionais e bibliotecas de ML disponíveis nos dispositivos. Neste sentido, o trabalho tem como objetivo definir o protótipo de um sistema que considere o estado da rede, a disponibilidade de bibliotecas e recursos computacionais para orquestração e virtualização de cadeias de funções de ML. Portanto, a questão que motiva a presente pesquisa é:

É possível otimizar a distribuição das tarefas de ML, considerando ambientes onde dispositivos computacionais possuem diferentes capacidades computacionais e de comunicação (processador, memória, bibliotecas e disponibilidade de banda), estando esses conectados em uma rede passível de intermitências?

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é modelar, viabilizar e definir o protótipo de um sistema que considere o estado da rede, a disponibilidade de bibliotecas e recursos computacionais para orquestração e virtualização de cadeias de funções de ML.

1.1.2 Objetivos Específicos

A partir do objetivo geral, este trabalho tem a finalidade de:

- Investigar os conceitos relacionados buscando prover o conhecimento necessário para modelar uma arquitetura de sistema;
- Implementar a arquitetura de sistema com base nesses conceitos;
- Determinar as premissas necessárias para o encadeamento de funções de ML;
- Elaborar um modelo matemático para minimizar o tempo de execução e colocação de cadeias funções, considerando restrições computacionais tais como memória e CPU;
- Definir e apresentar estudo de caso que será utilizado na avaliação do sistema;
- Analisar os dados propostos pelo estudo de caso, de forma a verificar sua validade nos experimentos;
- Investigar plataformas com finalidades relacionadas, de forma a quantificar o desempenho da abordagem proposta;

- Analisar os métodos apresentados durante a avaliação, para aceitar ou negar a pergunta de pesquisa.

1.2 ORGANIZAÇÃO DO TEXTO

O presente trabalho é estruturado da seguinte maneira: No Capítulo 2 são expostos os conceitos básicos e fundamentação teórica utilizada como base para elaboração da abordagem proposta. No Capítulo 3 são apresentados trabalhos relacionados à pesquisa. No Capítulo 4 é apresentada a abordagem de MLFV proposta. No Capítulo 5 são apresentados os métodos para avaliação da abordagem, os experimentos e os resultados alcançados. Por fim, o Capítulo 6 apresenta as considerações finais da pesquisa e trabalhos futuros.

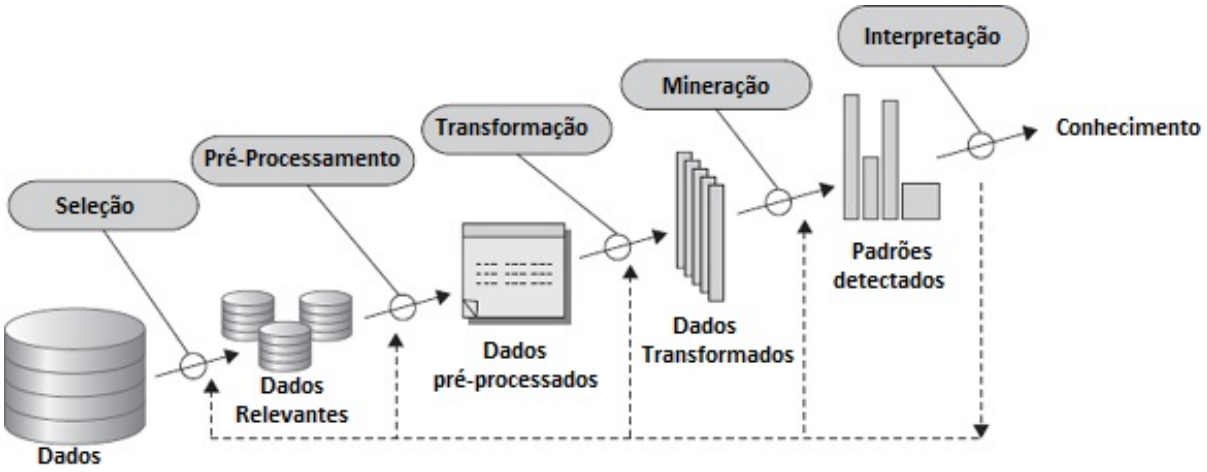
2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta o referencial teórico necessário para compreensão da abordagem proposta neste trabalho. Em um primeiro momento, é apresentado o processo para Descoberta de Conhecimento em Base de Dados e o conceito de Aprendizado de Máquina (ML). Além disso, é introduzido o conceito de Aprendizado de Máquina Como um Serviço (MLaaS), através de plataformas que, hospedadas na Nuvem (Cloud), e na borda da rede (*Edge Computing*), oferecem acesso a técnicas de ML funcionais e prontas para uso. Por fim, são apresentados as abordagens para Virtualização de Funções de Rede (NFV) e Cadeias de Funções de Rede (NFC), as quais são fundamentais para o desenvolvimento da proposta deste trabalho.

2.1 DESCOBERTA DE CONHECIMENTO EM BASE DE DADOS (KDD)

A Descoberta de Conhecimento em Banco de Dados (*Knowledge Discovery in Databases* – KDD) é um processo que introduz todos os passos necessários para realizar a extração de conhecimento de grandes bases de dados (FREIRE, 2016). Em resumo, é um processo não trivial que possui os seguintes objetivos: (i) explorar, organizar e limpar inconsistências de determinado conjunto de dados brutos; (ii) realizar a extração de informações e características dos dados já preparados, utilizando algoritmos inteligentes; (iii) representação do conhecimento resultante da extração (Figura 1).

Figura 1 – Processo de KDD.



Fonte: Adaptado de (FAYYAD; PIATETSKY; SMYTH, 1996)

Como pode ser observado na Figura 1, Fayyad (FAYYAD; PIATETSKY; SMYTH,

1996) propõe cinco etapas para compor o ciclo de funcionamento do KDD, sendo elas: seleção dos dados; pré-processamento; transformação dos dados; Mineração de Dados; interpretação e avaliação dos resultados . A execução dessas fases ocorre de forma iterativa, havendo uma sequência finita de operações em que o resultado de cada etapa influencia a seguinte, e interativa, o que significa que o usuário pode interferir e tomar decisões em cada uma das etapas.

2.1.1 Seleção dos Dados

A primeira fase do processo de KDD busca compreender o domínio da aplicação, definir os objetivos a serem alcançados e organizar os dados que serão utilizados (FAYYAD; PIATETSKY; SMYTH, 1996). Esta fase também é chamada de integração de dados, uma vez que estes podem vir de uma série de fontes diferentes (*data warehouses*, planilhas, etc.) ou possuírem os mais diversos formatos. O principal objetivo é selecionar e organizar essas fontes de dados em um único conjunto de dados, agrupando todas as informações em um mesmo local. Dessa forma, o acesso fica mais fácil e rápido, gerando ganhos de performance. O conjunto de dados, assim como um banco de dados comum, possui as possíveis variáveis, chamadas de características ou atributos e registros, chamados de casos ou observações (PRASS, 2004).

2.1.2 Pré-processamento

A etapa de pré-processamento consiste em realizar análise e limpeza de dados. Como apresentado por (TAN; STEINBACH; KUMAR, 2009), é praticamente impossível imaginar um conjunto de dados livre de inconsistências, como atributos ou campos incompletos ou em branco, valores duplicados, ruído e *outliers*. Algum erro humano pode ocorrer ou os valores podem ser coletados de forma errada; o usuário pode informar dados inválidos, cometer um engano ou até mesmo ocorrer algum erro inesperado no sistema ao salvar algum registro.

No intuito de minimizar estes problemas são utilizadas técnicas de análise estatística e algoritmos, para coletar as informações necessárias, encontrar as inconsistências citadas e removê-las apropriadamente (FAYYAD; PIATETSKY; SMYTH, 1996).

Todas essas inconsistências existentes nos dados acabam por comprometer o desempenho do processo de extração de conhecimento como um todo. Apesar dos algoritmos de Mineração de Dados tolerarem ruídos nos dados, a presença destes, em excesso, acabam por piorar os resultados. A fase de pré-processamento pode ser dividida em dois momentos: limpeza e seleção de atributos.

A parte da limpeza inicia com a realização de uma análise nos dados. Busca-se fornecer as principais características dos dados, sua disposição e a verificação de *outliers* e/ou ruídos (HAN; KAMBER, 2006). Essa análise é feita através do uso de ferramentas estatísticas como média, mediana, moda, alcance, variância, desvio padrão, dispersão e correlação.

Como resultado desta análise, é apresentada uma avaliação das propriedades dos dados, tanto na forma numérica, quanto através de gráficos de diversos tipos. Os principais tipos de gráficos usados são pizza, histogramas, barras e dispersão. Toda essa primeira parte é também chamada de sumarização. A sua realização identifica as principais inconsistências no conjunto de dados, possibilitando ao usuário decidir quais as ações serão tomadas para realizar a limpeza.

Para solucionar problemas relacionados a valores inconsistentes, a parte de análise, realizada previamente, é de grande importância. A partir dela, é possível detectar valores negativos no atributo "altura" de uma pessoa, por exemplo. Esses erros são facilmente detectados e corrigidos com a ajuda de um especialista da área onde os dados foram coletados (TAN; STEINBACH; KUMAR, 2009).

No caso de dados duplicados, algumas questões devem ser avaliadas antes das ações serem tomadas. Em casos de objetos que realmente representem um único, os valores de seus atributos devem ser verificados; caso estes valores sejam diferentes, a inconsistência deve ser resolvida. Para isso, uma avaliação do especialista da área se faz necessária novamente (TAN; STEINBACH; KUMAR, 2009). Já em casos de objetos com valores em branco ou nulo em algum atributo, as estratégias utilizadas podem variar entre:

Eliminar objeto com valores faltantes. Remove todo o objeto quando algum de seus valores estiver em branco. Apesar de ser uma estratégia interessante, se a maioria dos dados possuir algum valor em branco poucos dados irão restar.

Tratar os valores faltantes. Essa estratégia busca definir algum valor ao atributo vazio. Pode ser um valor predeterminado pelo usuário ou até mesmo uma estimativa usando a interpolação entre os valores de atributos vizinhos. Se o atributo for continuo, o valor médio dos vizinhos mais próximos pode ser usado; caso seja categorizado, então o valor que tem maior ocorrência será atribuído (TAN; STEINBACH; KUMAR, 2009).

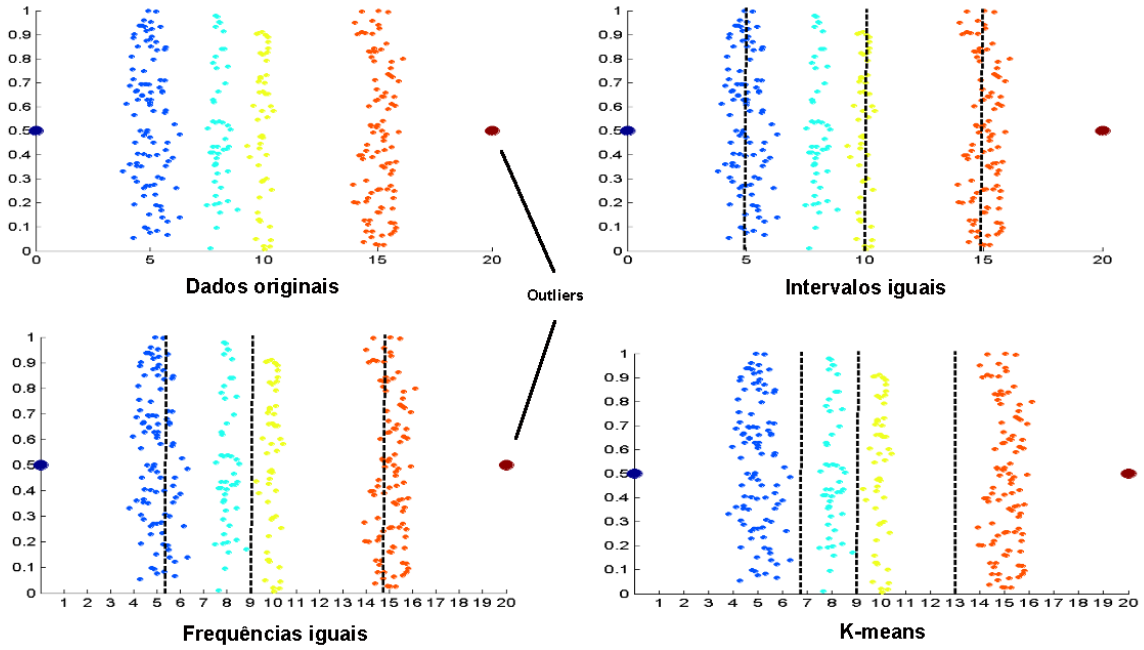
Ignorar os valores faltantes. Outra técnica seria ignorar o atributo que possui muitos valores em branco.

É muito comum, também, os dados apresentarem variações e erros randômicos. Este tipo de problema, conhecido como ruído ou outlier, é um pouco mais complicado de ser corri-

gido. Algumas das técnicas para realizar a "suavização" (*smooth*) dos dados e reduzir o ruído, são:

Discretização (*Binning*). Este método busca suavizar determinado conjunto de dados através da sua reorganização. Para isso, os dados devem ser escolhidos e distribuídos em um número n de grupos. Cada grupo irá representar um tipo, e a partir da realização de cálculos, determinado objeto poderá ser removido de um e adicionado em outro (TAN; STEINBACH; KUMAR, 2009; PRASS, 2004). A distribuição e cálculo dos grupos pode ser feito através da discretização de largura igual, por frequência igual ou usando o algoritmo de *K-Means*, como apresentado na Figura 2 (TAN; STEINBACH; KUMAR, 2009).

Figura 2 – Diferentes técnicas de discretização.



Fonte: Adaptado de (TAN; STEINBACH; KUMAR, 2009)

Regressão Linear. A regressão linear busca descobrir uma função que represente o padrão dos dados, de modo que seja possível usar o valor de um atributo para prever o outro (HAN; KAMBER, 2006).

***Box plot*.** A ferramenta de visualização de *Box plot* é uma forma popular de visualizar a distribuição dos dados (HAN; KAMBER, 2006). Com ela, é possível determinar um limite nos dados onde os valores que o ultrapassarem serão considerados *outliers*.

Após realizar a limpeza nos dados, inicia-se a fase de seleção de atributos. Esse processo é importante devido ao fato de, em geral, os bancos de dados possuírem grande quantidade de

atributos. Reduzir esta quantidade de atributos antes da fase de Mineração de Dados melhora os resultados nos quesitos tempo, custo e qualidade (TAN; STEINBACH; KUMAR, 2009).

Neste contexto, o objetivo é realizar a análise destes atributos, de forma que seja possível: selecionar os melhores, criar novos atributos com a combinação de dois ou mais objetos e/ou descartar atributos que não irão contribuir com o processo de Mineração de Dados. Para isso, são usadas as seguintes técnicas:

Análise de correlação entre atributos. Dados dois atributos, esta técnicas é usada para identificar o quão relacionados eles estão (HAN; KAMBER, 2006). É possível, com isso, identificar se determinados objetos são redundantes, o que acarreta na melhora dos resultados obtidos nas próximas fases.

Agregação e redução de dimensionalidade. Como o próprio nome diz, essa técnica visa combinar dois ou mais objetos em um só. É possível reduzir o número de variáveis e também o conjunto de dados como um todo. Com isso, diminui-se o tempo de processamento e uso de memória. Ademais, essa técnica pode ser usada como ferramenta para mudança de escopo ou escala nos dados, trazendo uma visão de mais alto nível. Seu uso, no entanto, deve feito com cautela, pois se pode perder informações uteis à medida que os dados são reduzidos (TAN; STEINBACH; KUMAR, 2009; HAN; KAMBER, 2006).

2.1.3 Transformação

Na etapa Transformação, os dados devem ser transformados ou consolidados para um formato apropriado ao processo de Mineração de Dados (HAN; KAMBER, 2006). A performance dos algoritmos de classificação está estritamente ligada a sua capacidade de lidar com diferentes tipos de dados, atributos e *datasets* (BHARGAVI; SINGARAJU, 2011). Neste sentido, a tarefa da Transformação é tornar os dados compatíveis com um algoritmo específico de Mineração de Dados e com isso, melhorar seu desempenho. Os métodos comuns utilizados nesta etapa são a normalização, categorização, extração de características e criação de atributos.

Criação de atributos e extração de características. Ao contrário da agregação, esta técnica busca criar novos atributos extraindo características dos originais. Isso ocorre porque muitas vezes os dados possuem as informações necessárias, mas não estão na forma ideal para serem minerados. Por exemplo, em um banco onde se possui a “massa” e o “volume” de determinado objeto, é possível criar outro atributo “densidade” o qual ira capturar as características dos dois atributos. Se o objetivo fosse classificar os objetos dependendo do material o qual é

feito, o uso desse atributo densidade obteria melhor resultado (TAN; STEINBACH; KUMAR, 2009).

Categorização e binarização. Esta forma de discretização busca categorizar os dados para, posteriormente, transformar as categorias em valores binários. Geralmente, essa técnica é usada em algoritmos de classificação ou associação (TAN; STEINBACH; KUMAR, 2009). Para realizar a categorização, tomam-se como exemplo valores numéricos de temperatura em uma sala; destes números, são criadas n categorias que representem estes valores na forma discreta: muito frio, frio, morno, quente e muito quente. Por fim, para realizar a binarização é necessário transformar as categorias, citadas anteriormente, em valores binários (0 e 1) como pode ser visto na Tabela 1.

Tabela 1 – Conversão de um atributo categorizado em três árvores binárias.

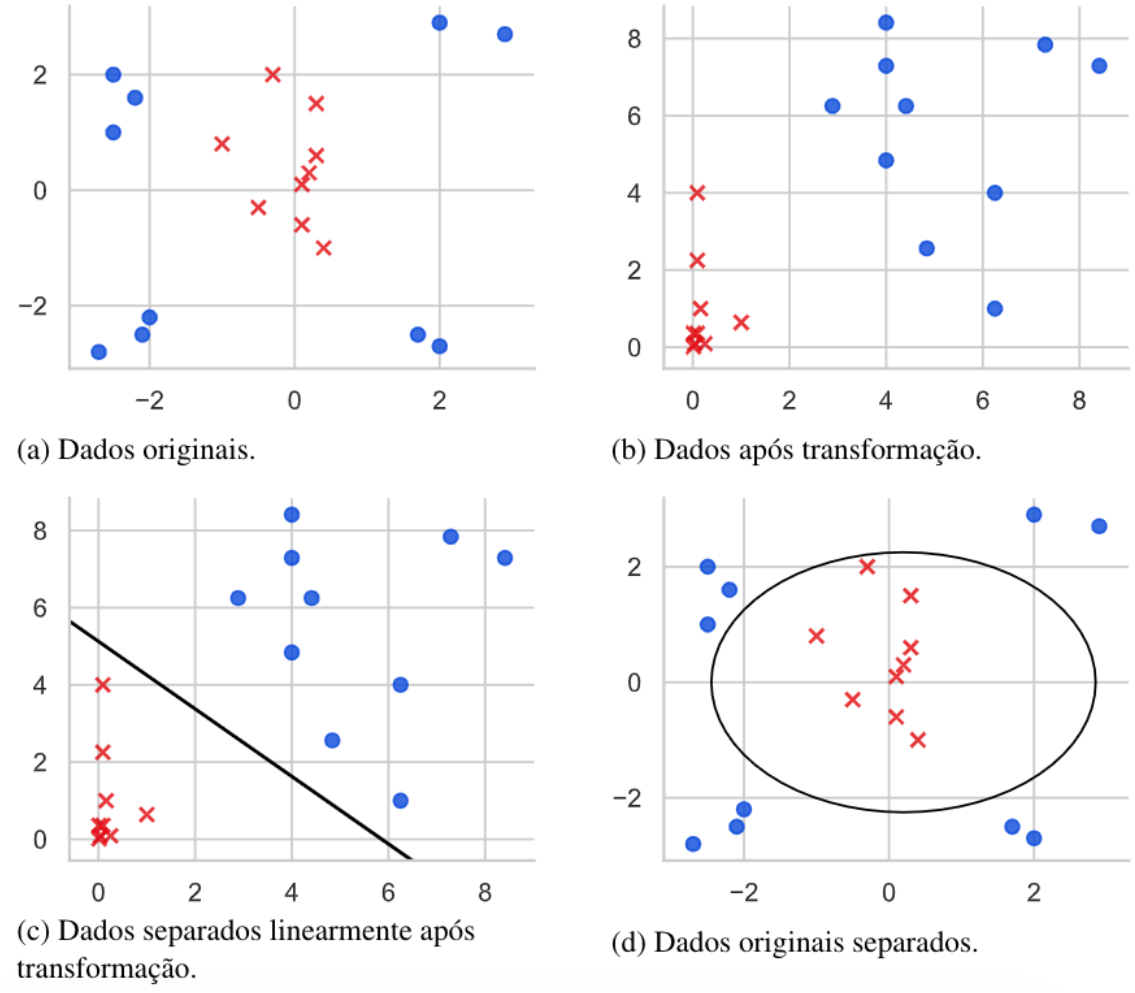
Valor categorizado	Inteiro	x1	x2	x3
muito frio	0	0	0	0
frio	1	0	0	1
morno	2	0	1	0
quente	3	0	1	1
muito quente	4	1	0	0

Fonte: Adaptado de (TAN; STEINBACH; KUMAR, 2009).

Transformação de atributos. A transformação de variáveis ou atributos ocorre quando todos os seus valores são transformados por uma função. Dois tipos destas transformações são as transformações por funções simples e normalização (TAN; STEINBACH; KUMAR, 2009). A transformação por funções simples ocorre quando uma função matemática é aplicada a todos os valores de determinado atributo. As funções matemáticas mais usadas são x^k, $\log^x$, $\exp^x$,$\sqrt{x}$, $\frac{1}{x}$, $\sin x$ ou $||x||$ (TAN; STEINBACH; KUMAR, 2009).

Suponha que determinado conjunto de dados esteja arranjado de uma forma em que não pode ser separados por uma função linear, como pode ser visto na Figura 3. Para resolver este problema é utilizada uma função matemática simples (x^2) nos atributos originais (Figura 3a). O resultado desta transformação é que os dados mudam seu comportamento (Figura 3b). Com isso, é possível utilizar um classificador linear simples para realizar a separação dos dados, como é visto na Figura 3c. Por fim, realizando o processo inverso de transformação, retornando os dados para o espaço original, é possível perceber que a função linear aplicada nos dados transformados torna-se uma função não linear de maior complexidade (Figura 3d).

Figura 3 – Transformação de dados por função simples.



(a) Dados originais.

(b) Dados após transformação.

(c) Dados separados linearmente após transformação.

(d) Dados originais separados.

Fonte: Adaptado de (ABU-MOSTAFA; MAGDON-ISMAIL; LIN, 2012).

Essa técnica, apesar de ser uma opção interessante para auxiliar algoritmos de ML no processo de aprendizado, deve ser usada com cautela. Aplicar uma transformação de forma errada pode comprometer a capacidade de extração de conhecimento e generalização das técnicas de ML devido a alterações nas propriedades e características dos dados, podendo fazer com que os padrão existentes desapareçam (TAN; STEINBACH; KUMAR, 2009; ABU-MOSTAFA; MAGDON-ISMAIL; LIN, 2012).

Normalização. Outra forma de transformação de um atributo é chamada de normalização. O objetivo da normalização é escalar os dados, de determinado atributo, em uma mesma faixa de valor. É possível realizar a transformação de determinado conjunto para que seus valores fiquem dentro dos limites de 0 até 1 ou -1 até 1, como visto abaixo (HAN; KAMBER, 2006).

Tabela 2 – Resultado da normalização de conjunto numérico.

Conjunto original	0	3	6	9	12	15	18	21	24	27
Conjunto normalizado	0	0.11	0.22	0.33	0.44	0.56	0.67	0.78	0.89	1

Fonte: Próprio autor.

A normalização de um conjunto de dados possibilita que atributos com diferente ordem de magnitude possam ser comparados; dados em diferentes escalas possam ser interpretados; atributos, como a cor dos olhos de uma pessoa, por exemplo, são transformados para que possam ser calculados junto de outros atributos numéricos. Além disso, seu uso evita que atributos com valores grandes dominem os resultados dos cálculos executados nos algoritmos de Mineração de Dados (TAN; STEINBACH; KUMAR, 2009). Para realizar a normalização de um conjunto de dados, duas técnicas podem ser usadas:

Normalização Mínimo – Máximo (*Min-Max Normalization/Scaling*). A normalização *Min-Max* é usada para transformar atributos numéricos, de determinado conjunto de dados, para uma mesma faixa de valores. Em geral, este intervalo de valores varia entre 0 e 1 ou -1 e 1. Sua formula é dada pela seguinte expressão:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

onde x é a variável a ser normalizada, x_{norm} é o valor final normalizado, x_{min} o menor valor numérico presente no conjunto de dados e x_{max} o maior valor. O exemplo da Tabela 2 ilustra o a utilização desta fórmula.

Normalização Gaussiana (*Gaussian normalization – Z Score*). Na normalização gaussiana, os valores do conjunto de dados (A) são normalizados baseados na média e desvio padrão deste mesmo conjunto (A). Sua fórmula é dada pela seguinte equação:

$$z = \frac{x - u_A}{\sigma_A}$$

onde x é o valor a ser normalizado, u é a média do conjunto A e σ o desvio padrão do conjunto A.

2.1.4 Mineração de Dados

A etapa de Mineração de Dados tem como objetivo descobrir e extrair padrões e regras de associação presentes no conjunto de dados (HAN; KAMBER, 2006). Isso ocorre através da aplicação, sucessiva e iterativa, de algoritmos de ML (FAYYAD; PIATETSKY; SMYTH, 1996). Estas técnicas buscam correlacionar os atributos presentes nas bases de dados para construir modelos preditivos e generalizar soluções.

Em geral, a etapa de Mineração de Dados desempenha as tarefas de classificação ou agrupamento dos dados (STEINER et al., 2006). Possuem papel de destaque e grande popularidade principalmente nas áreas de banco de dados, inteligência artificial e ML (FAYYAD; PIATETSKY; SMYTH, 1996; HAN; KAMBER, 2006). Na literatura, são apresentados diversos algoritmos de ML, utilizados na etapa de Mineração de Dados, tendo diferentes atribuições; para realizar a tarefa de classificação, os mais populares são os algoritmos de *Decision Tree* (DT), *Artificial Neural Networks* (ANN), *Random Forest* (RF) e *Support Vector Machines* (SVM); para agrupamento, destacam-se os métodos de *k-Nearest Neighbors* (k-NN) e *K-Means*.

Dentre todas as etapas presentes no processo de KDD, a Mineração de Dados é a mais estudada na literatura (STEINER et al., 2006). Apesar disso, alguns erros de interpretação ainda são cometidos, como por exemplo, a diferenciação entre o próprio KDD e os algoritmos de Mineração de Dados; enquanto o KDD compreende o processo geral para descoberta de conhecimento, os algoritmos que compõem a etapa de Mineração de Dados representam apenas uma das atividades necessárias. As etapas adjacentes à Mineração de Dados possuem papel fundamental nesse processo de busca por informações úteis em bases de dados; a aplicação de técnicas ou algoritmos de ML "às cegas" (*Blind Application*), principalmente em dados brutos, leva à descoberta de padrões sem significado (FAYYAD; PIATETSKY; SMYTH, 1996). Outro equívoco, citado por (TAN; STEINBACH; KUMAR, 2009), é que nem toda forma de descoberta de informações deve ser considerada Mineração de Dados. A busca por registros usando um sistema gerenciador de Banco de Dados, por exemplo, através de mecanismos de consulta é chamada de recuperação de dados. Na verdade, a Mineração de Dados busca melhorar estes sistemas de recuperação.

2.2 APRENDIZADO DE MÁQUINA

A criação de sistemas de computadores dotados de inteligência e que possam, assim como o cérebro humano, realizar tarefas de processamento de dados ao mesmo tempo em que aprendem através de experiência já fomentou o desenvolvimento de muitas pesquisas no passado (FACELI; LORENA; CARVALHO, 2011). No entanto, a medida que os estudos avançavam, cientistas acabaram percebendo que a realização dessas atividades, muitas vezes consideradas simples para um ser humano, são complexas de se reproduzir em um robô ou máquina (FACELI; LORENA; CARVALHO, 2011). Ainda assim, vários estudos motivados e inspirados pela analogia entre as células nervosas vivas e processamento eletrônico continuaram a

ser conduzidas (TAFNER; XEREZ; RODRIGUES, 1995).

Donald Hebb (HEBB, 1949), por exemplo, foi o pesquisador que introduziu o primeiro método de treinamento para um neurônio artificial. Em seu trabalho foi demonstrado que a alteração dos pesos (sinapses) dos nodos de entrada reflete no aprendizado do modelo. Já McCulloch e Pitts (MCCULLOCH; PITTS, 1988) foram os pioneiros a estudarem a área de neuro-computação, desenvolvendo o primeiro modelo matemático inspirado em um neurônio humano. Seu trabalho abriu portas para o estudo de ANN. Vale lembrar que o neurônio pensado por McCulloch e Pitts possuía pesos com valores fixos pré-definidos. Este método de aprendizado ficou conhecido como regra de aprendizado de Hebb e foi usado como base para desenvolvimento de outros métodos.

2.2.1 Conceitos Básicos

O conceito de ML foi criado com o intuito de desenvolver técnicas e algoritmos computacionais capazes de adquirir conhecimento de forma automática, possibilitando a criação de sistemas de aprendizado capazes de tomar decisões baseado em experiências acumuladas através da solução bem sucedida de problemas anteriores. Os diversos algoritmos de ML existentes possuem características particulares e comuns que possibilitam sua classificação quanto à linguagem de descrição, modo, paradigma e forma de aprendizado utilizado (MONARD; BARANAUSKAS, 2003). Esses algoritmos foram desenvolvidos através de modelos matemáticos que buscam simular a capacidade de processamento e aprendizado dos seres humanos em computadores. Eles possuem a capacidade de descobrir relacionamentos não-lineares entre diferentes parâmetros ou atributos, aprender por experiência (treinamento), generalizar soluções desconhecidas, e aproximar funções matemáticas utilizando conjuntos de dados como exemplo.

Em geral, todo algoritmo de ML deve possuir parâmetros de entrada (*inputs*) e de saída (*outputs*). As entradas são os valores, geralmente numéricos (*Integer* ou *Double*), que representam variáveis existentes, em um banco de dados, que serão apresentadas ao sistema. Em um problema de classificação de solos, por exemplo, onde busca-se realizar a identificação de um tipo de solo (areia, argila, etc.), os parâmetros de entrada utilizados são as características (ou medidas, coletadas via experimentos) que representam as propriedades de um solo específico; o atributo "razão de atrito" ($Rf\%$) e "resistência da ponta do cone" (qc/Mpa) representam as propriedades de resistência e atrito de um tipo de solo e, com base nessas informações, fica possível descobrir qual solo está sendo avaliado. Os parâmetros de entrada geralmente são re-

presentados pelo símbolo x (x_1, x_2, x_3, x_n). Seguindo essa lógica, no contexto de classificação de solos, a saída de um algoritmo de ML se refere a classe ou objeto a ser aprendido, com base nos atributos de entrada apresentados. Os parâmetros de saída geralmente são representados pelo símbolo y (y_1, y_2, y_n) (SILVA; SPATTI; FLAUZINO, 2010; BRAGA; CARVALHO; LUDERMIR, 1998).

O treinamento ou processo de aprendizado desses algoritmos ocorre, geralmente, através de inferência lógica (indução) em conjuntos de dados (exemplos), que permite obter conclusões genéricas sobre o mesmo. Na indução, um conceito é aprendido efetuando-se inferência indutiva sobre os exemplos apresentados, sendo um dos principais métodos utilizados para derivar conhecimento novo e predizer eventos futuros (MONARD; BARANAUSKAS, 2003). Em outras palavras, o conceito de indução ocorre quando são apresentados dados ao algoritmo, denominados dados de treinamento, que exprimem o comportamento de determinado problema, para que este possa extrair informações e aprender o relacionamento entre os parâmetros ou atributos existentes (SILVA; SPATTI; FLAUZINO, 2010; BRAGA; CARVALHO; LUDERMIR, 1998).

Cada um destes algoritmos pertence a um tipo de paradigma de aprendizado. Dependendo do problema e da disponibilidade ou tipo dos dados existentes, determinado paradigma será mais adequado. Os dois principais paradigmas mais conhecidos são: aprendizado supervisionado e aprendizado não supervisionado (FACELI; LORENA; CARVALHO, 2011).

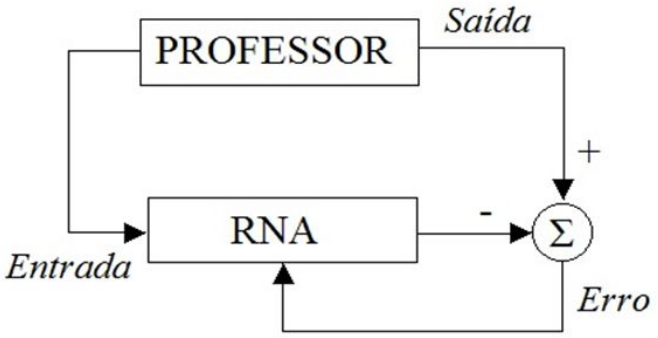
2.2.2 Aprendizado Supervisionado

Este método de aprendizado é o mais conhecido e utilizado. É assim chamado pois as entradas e saídas desejadas são fornecidas por um intermediador, chamado de supervisor externo (**professor**) (BRAGA; CARVALHO; LUDERMIR, 1998). Os dados, chamados de dados de treinamento, são apresentados em pares de entradas e saídas, geralmente estando representados na forma de vetores numéricos. Desta forma, os vetores de entrada x devem possuir seus vetores de saída y correspondentes, sendo necessário, muitas vezes, o desenvolvimento de tabelas (matrizes) com todos estes valores organizados.

À medida que os vetores de valores são apresentados ao algoritmo, ele irá ajustar os seus parâmetros internos baseado na correlação existente entre os dados de entrada e suas respectivas saídas. Isso significa que os dados de entrada x são processados internamente e o resultado produzido na saída é comparado ao vetor y correspondente. Essa comparação resulta em um valor de erro, que servirá como base para ajustar os parâmetros internos durante as pró-

ximas iterações, como pode ser visto na Figura 4. São necessárias inúmeras iterações deste tipo para diminuir o erro na camada de saída até que, por fim, o algoritmo possa ser considerado treinado (BRAGA; CARVALHO; LUDERMIR, 1998).

Figura 4 – Exemplo de funcionamento do aprendizado supervisionado.



Fonte: Adaptado de (BRAGA; CARVALHO; LUDERMIR, 1998).

A Figura 4 representa visualmente o processo descrito anteriormente. Nela, um banco de dados (representado como **professor**), contendo exemplos de um problema, apresenta os valores de entrada para um algoritmo de ML (representados por RNA). A RNA processa essa entrada com o conhecimento atual (parâmetros internos) e retorna, na sua saída, o resultado desse processamento. Esse resultado é então comparado com a saída desejada, fornecida pelo **professor**, e a diferença existente é calculada como erro que realimenta a RNA para atualizar seus parâmetros internos e melhorar sua performance e aprendizado.

2.2.3 Aprendizado Não Supervisionado

Diferente do método anterior, o aprendizado não supervisionado não possui um supervisor externo (**professor**). Neste caso, os dados de treinamento são apenas dados de entrada x, sem suas respectivas saídas. Os vetores de entrada X são apresentados ao algoritmo de ML e este deve aprender qual o relacionamento e padrões presentes nos dados, se auto-organizando para separar classes e/ou subconjuntos.

De acordo com Braga (BRAGA; CARVALHO; LUDERMIR, 1998), a partir do momento em que o algoritmo estabelece uma harmonia com as regularidades estatísticas da entrada de dados, desenvolve-se nele uma habilidade de formar representações internas para codificar características da entrada e criar novas classes ou grupos automaticamente.

2.3 APRENDIZADO DE MÁQUINA COMO SERVIÇO (MLAAS)

Os algoritmos de Aprendizado de Máquina estão cada dia mais se tornando importantes ferramentas em aplicações de softwares inteligentes no mundo real (CHAN et al., 2013). Muitas das tendências tecnológicas dos dias atuais (veículos autônomos, detecção e tradução de fala, robôs inteligentes, sistemas de monitoramento de saúde móvel, recomendação de produtos, etc.) dependem dessas técnicas para realização de diferentes tarefas, como a classificação, regressão, descoberta de conhecimento, detecção de padrões e criação de regras de associação baseada em dados (Bierzynski; Escobar; Eberl, 2017).

Com a popularização de sensores, dispositivos móveis conectados à Internet e o advento do conceito de IoT, houve um aumento gigantesco na quantidade de dados gerados, diariamente, e que necessitam, de alguma forma, serem analisados ou processados (RIBEIRO; GROLINGER; CAPRETZ, 2015). Grandes empresas já estão, há tempos, investindo uma grande quantidade de dinheiro e esforços no desenvolvimento das suas próprias soluções personalizadas, inteligentes e automatizadas, que utilizam algoritmos de ML, para lidar com esses dados. No entanto, enquanto que para essas empresas, com inúmeros funcionários e especialistas na área, o desenvolvimento dessas aplicações é um processo corriqueiro, a aplicação de técnicas de ML continua sendo um desafio para grande parte de desenvolvedores de software e, principalmente, pessoas fora da área de computação.

De acordo com (CHAN et al., 2013), existem diversos desafios relacionados a aplicação dos algoritmos de ML no desenvolvimento de softwares, o que acaba elevando a curva de aprendizado associada à utilização dessas técnicas. Em primeiro lugar, há uma grande variedade de *frameworks* existentes à serem escolhidos, cada um com suas particularidades e curva de aprendizado particular. Além disso, existe uma enorme opção de algoritmos ML para se escolher que, assim como os *frameworks*, possuem suas particularidades, vantagens e desvantagens. Escolher esses algoritmos e, ainda, aprender a ajustá-lo de forma efetiva é um grande desafio. Entender como os algoritmos de ML funcionam e construir aplicações preditivas não é uma tarefa simples e envolve muitas atividades complexas. Além disso, em alguns casos esses algoritmos podem exigir recursos computacionais com custos impraticáveis (RIBEIRO; GROLINGER; CAPRETZ, 2015).

Uma forma de lidar com essas dificuldades foi através da criação de plataformas contendo Serviços de Aprendizado de Máquina funcionais e prontos para uso, chamados de MLaaS.

Essas plataformas buscam facilitar o acesso a técnicas de ML até mesmo para usuários não especialistas (Bierzynski; Escobar; Eberl, 2017). Esses serviços, ofertados por empresas como Google, Amazon e Microsoft, são hospedados na nuvem e fornecem acesso, através de *web services* e interfaces de consulta, a ferramentas para construir, treinar e implantar modelos de ML. O processo de utilização de técnicas de ML é simplificado, através da abstração dos desafios existentes no armazenamento dos dados, treinamento dos modelos e predição (YAO et al., 2017).

O principal objetivo é possibilitar o acesso à diversas soluções, ferramentas e técnicas para criação de modelos de ML, variando entre opções extremamente simples (soluções não paramétricas prontas para uso) ou totalmente customizáveis, que necessitam de conhecimento avançado (sistemas totalmente ajustáveis para desempenho otimizado). Algumas dessas plataformas funcionam como uma "caixa-preta", não revelando ao usuário nem mesmo qual classificador foi usado. Já outras, permitem aos usuários escolher diversas funcionalidades, desde o pré-processamento de dados, seleção de classificadores, seleção de recursos, até o ajuste de parâmetros (YAO et al., 2017).

(CHAN et al., 2013) utilizam o termo "Servidor de ML" (*ML server*) para se referenciar as plataformas de MLaaS. De acordo com o autor, o funcionamento de um Servidor de ML pode ser comparado a um servidor de Banco de Dados comum, tendo como a principal diferença o fato de que o último realiza as tradicionais consultas de *Structured Query Language* (SQL) (busca nos dados) enquanto o primeiro realiza atividades de predição. Pelo fato de ser um servidor de ML e estar hospedado na Nuvem, essas plataformas possuem, teoricamente, um poder de processamento e armazenamento virtualmente ilimitado. Ambas essas capacidades são necessárias para transformar a grande quantidade de dados, gerados por dispositivos IoT, em conhecimento e comandos inteligentes, sendo um dos principais benefícios de se utilizar plataformas de MLaaS atualmente (Bierzynski; Escobar; Eberl, 2017).

No entanto, ainda de acordo com (Bierzynski; Escobar; Eberl, 2017), infraestruturas tradicionais de MLaaS na nuvem possuem algumas restrições, não atendendo alguns requerimentos que as tecnologias de IoT necessitam em diferentes domínios e situações. Os principais problemas se referem principalmente à latência e largura de banda; por estar situado na nuvem, essas plataformas possuem um elevado tempo de envio e recebimento dos dados. Algoritmos e sistema sensíveis à latência são os mais afetados neste sentido, devido ao fato das conexões disponíveis não enviarem os dados para nuvem em um tempo razoável para serem processados,

resultando no atraso na tomada de decisões e perda de informações importantes.

2.3.1 Aprendizado de Máquina na Borda da Rede

A utilização de algoritmos e serviços de ML em conjunto ao conceito de Computação na Borda (*Edge Computing*) vem sendo estudado como um complemento ou extensão as plataformas de MLaaS ofertados na Nuvem, permitindo que o gerenciamento, processamento e execução das atividades relacionadas ocorram perto de onde os dados são coletados (Bitye Dimithe; Reid; Samata, 2018). O objetivo, com isso, é reduzir diversas limitações, restrições e problemas de latência, perda da integridade dos dados, segurança, privacidade e tráfego de dados na rede.

Os primeiros trabalhos relacionados ao tópico de Computação na Borda relataram o uso de dispositivos pequenos e portáteis para realizar processamento e computação de informações em tempo real (HARTMANN, 2018). Para isso, um sistema de computação na borda ideal deve ser composto de uma grande quantidade de nodos e permitir uma baixa latência entre eles. Em resumo, o conceito de ML na borda busca trazer os recursos computacionais, uma vez na nuvem, para a borda da rede. Com isso, elimina-se a necessidade de transferência dos dados para a nuvem, processo o qual se revela caro, demorado e não rápido o suficiente para ser aplicado em aplicativos sensíveis à latência (HARTMANN, 2018). Neste sentido, os principais benefícios dessa abordagem são os seguintes:

Limite de banda e latência. A largura de banda e latência são consideradas as maiores desvantagens presentes nas infraestruturas tradicionais de ML na Nuvem (Bierzynski; Escobar; Eberl, 2017). Uma conexão à Internet de baixa capacidade (comunicação de dados através de telefonia celular) ou qualidade (rede sem fio com sinal fraco), além da necessidade de enviar para a nuvem uma grande quantidade de dados, o que acarreta em problemas de latência e limite de banda de rede, aumentando consideravelmente o tempo de execução de aplicações. Pelo fato de analisar e processar as informações em nodos que estão próximos de onde ela é coletada, executar técnicas de ML na borda possibilita reduzir a latência e o tráfego de dados na rede.

Privacidade e segurança. A crescente conscientização social a respeito da privacidade de dados pessoais impõem diversos desafios aos serviços na Nuvem; em geral, é difícil manter dados privados e, ao mesmo tempo, utilizá-los em análises com técnicas de ML para oferecer bons serviços personalizados. Esses dados podem conter fotos pessoais, históricos, registros de voz e texto, dados sigilosos dentre outras informações, as quais, grande parte dos usuários

não tem interesse em compartilhar com provedores de serviços descoberta de conhecimento. Além disso, existe também o risco destes serviços serem atacados por hackers e, com isso, ter essas informações vazadas. Mover os serviços da nuvem para dispositivos na borda pode manter os dados privados e reduzir efetivamente o custo de comunicação, oferecendo controle absoluto sobre as informações do cliente, não dependendo de terceiros para garantir a sua segurança (ZHAO et al., 2018; RAJ, 2018).

Customização e custo-benefício. A maior parte dos serviços profissionais para criação, treinamento e hospedagem de modelos de ML na nuvem não são gratuitos. A solução econômica com melhor custo benefício seria utilizar dispositivos na borda da rede para realizar esse processo de criação e disponibilização de modelos de ML. Esses dispositivos permitem uma maior customização da forma de funcionamento do serviço ofertado (RAJ, 2018).

Redundância. Em arquiteturas de rede, o conceito de redundância é de extrema importância para superar falhas de comunicação. A falha em um nó na rede pode causar impactos em outros nodos. Os dispositivos na borda da rede podem fornecer um bom nível de redundância; caso algum dos dispositivos (nodos) conectados falhe, outro dispositivo vizinho assume seu lugar temporariamente (RAJ, 2018).

2.4 VIRTUALIZAÇÃO DE FUNÇÕES DE REDES E CADEIAS DE FUNÇÕES

O conceito de NFV consiste em uma abordagem criada para oferecer serviços de rede através de funções virtualizadas; a aplicação da tecnologia de virtualização busca oferecer uma nova forma de projetar, implantar e gerenciar serviços de rede em qualquer hardware, eliminando limitações propostas por tecnologias proprietárias convencionais (MIJUMBI et al., 2016). Dessa forma, as NFVs tendem a migrar aplicações de rede, tradicionalmente implementadas em hardwares específicos, para infraestruturas virtualizadas em servidores de uso geral. Isso tende a diminuir os custos de aquisição e gerenciamento, proporcionando mais flexibilidade e escalabilidade.

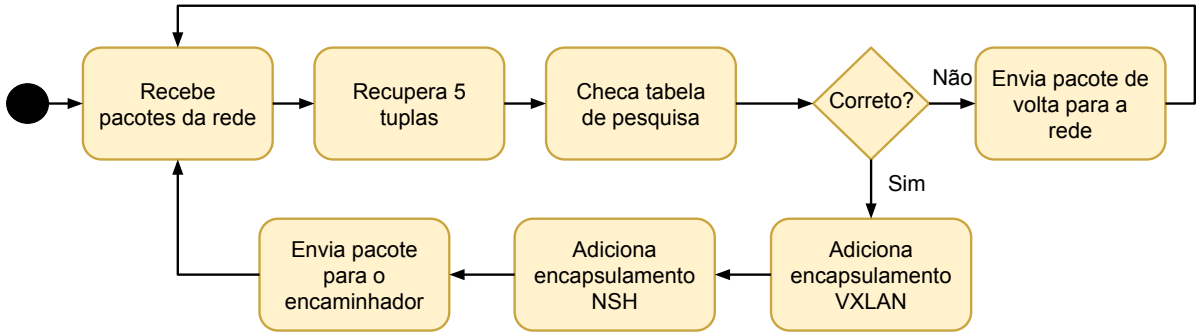
A ideia principal por trás do conceito de NFV é o desacoplamento das funções que são executadas em dispositivos e equipamentos proprietários na rede. Isso significa que uma função de rede (*firewall*, por exemplo) pode ser despachada para um cliente como uma instância de software simples. Em resumo, o conceito de NFV busca, através da virtualização de funções de rede, criar e implantar serviços completos de rede como software para serem executados em qualquer tipo de hardware. Além disso, o conceito de NFV permite a colocação dessas funções

e serviços em clientes levando em consideração diferentes requerimentos de rede (MIJUMBI et al., 2016).

Todo serviço de rede criado por uma plataforma de NFV é composto por um conjunto de *Virtual Network Function* (VNF), implementados como uma instância de *software*. As VNFs são como blocos de construção para criação dos serviços de NFV, sendo organizadas como cadeias sequenciais de atividades para criação e fornecimento dos serviços desejados (HAWILO; JAMMAL; SHAMI, 2019). De acordo com PATTARANANTAKUL et al. (2018), essa forma de organização simplifica a implantação de novas funcionalidades, ajuda a economizar energia, reduzir custos além de ser mais flexível e escalável. Os principais exemplos de funções de rede que são virtualizadas pelo NFV incluem firewalls, balanceadores de carga e inspeção profunda de pacotes de rede (*Deep Packet Inspection* (DPI)).

A interconexão de funções de rede é chamada de Service Function Chaining (SFC) ou, ainda, NFC. Essa interconexão se refere a uma lista organizada de funções à serem executadas em uma determinada ordem, determinando um fluxo de execução para criar serviços de rede compostos (PHAM et al., 2017). Além disso, de acordo com Medhat et al. (2017), o conceito de SFC ainda proporciona um melhor gerenciamento do tráfego de dados, serviços e aplicações, por fornecer ferramentas e soluções que definem o fluxo de execução e as rotas, levando em consideração diversos requisitos como o estado atual da rede e a disponibilidade de banda. A combinação das técnica de NFV e SFC fornecem ferramentas eficientes e efetivas no processo de implantação e orquestração de funções de rede, como pode ser visto na Figura 5.

Figura 5 – Encadeamento de funções para classificação de pacotes de rede.



Fonte: CASTANHO et al. (2018).

A Figura 5, proposta por Castanho (CASTANHO et al., 2018), apresenta essa combinação entre NFV e SFC através de uma cadeia de funções, visando a criação de um classificador

de pacotes de rede. As ações desse classificador podem ser resumidas em três etapas: classificar os pacotes com base nos parâmetros que compõem o conceito de 5-tupla (IP de origem, IP de destino, protocolo de transporte, porta de origem e porta de destino) para determinar o caminho do serviço, encapsular pacotes com *Network Service Header* (NSH) e enviá-los para um *Service Function Forwarder* (SFF). Além do NSH, um encapsulamento de transporte (*Virtual Extensible LAN* (VXLAN)) é adicionado aos pacotes, para permitir a comunicação entre cada elemento na arquitetura (CASTANHO et al., 2018).

3 TRABALHOS RELACIONADOS E MOTIVAÇÃO

Este capítulo tem como objetivo apresentar trabalhos relacionados aos diferentes tópicos abordados neste trabalho, ou seja, Aprendizado de Máquina (ML), ML disponibilizado como serviço, ML executado na borda da rede e Virtualização de Funções de Rede utilizando plataformas de NFV.

3.1 APRENDIZADO DE MÁQUINA

STEINER et al. (2006) explora o uso do processo de KDD, dando enfase na análise exploratória dos dados. Seu objetivo é mostrar a influência dessa análise no desempenho dos algoritmos de ML, comparar o desempenho deles entre si e verificar as técnicas com maior percentual de acerto. Os resultados demonstram que a aplicação da análise exploratória e o processo de KDD trazem importantes melhorias no desempenho dos algoritmos de ML abordados no trabalho, sendo um importante ferramenta para otimizar os resultados finais.

BIONDI et al. (2006) apresenta um sistema chamado de NEURO-CPT que aplica um algoritmo de ML na área de geotecnia, especificamente na classificação de solos buscando automatizar essa tarefa de classificação. O autor utilizou uma ANN com uma arquitetura composta de duas camadas escondidas. A primeira camada com 10 neurônios, a segunda com 150 neurônios e a saída com 12 neurônios artificiais. A rede foi treinada com um banco de dados validado pelo método CPT envolvendo uma matriz de 2 linhas (qc e Rf) e 545 colunas de exemplos representativos das 12 classes distintas de solos a serem tipificados. Os parâmetros utilizados para treinamento da ANN utilizaram um número máximo de épocas de 1000 e uma tolerância do erro de $1e-3$. Os resultados alcançados mostraram porcentagens de acurácia de 96% utilizando o algoritmo de ANN com essas configurações.

Outro sistema que aplica algoritmos de ML na área de geotecnia é chamado de CONCC, apresentado por Bhattacharya e Solomatine (BHATTACHARYA; SOLOMATINE, 2006) como um sistema para automatização do processo de classificação de solo utilizando técnicas de Aprendizado de Máquina. A proposta apresentada foi treinar dois algoritmos de ML para extrair o conhecimento de especialistas da área de Engenharia Civil e realizar as atividades de segmentação e classificação dos dados. A fase de segmentação buscou separar o conjunto de dados em camadas que representem uma variação de tipo de solo, respeitando a restrição de contiguidade

presente. A fase de classificação buscou construir um classificador para atribuir as classes de solos para cada camada criada durante a fase de segmentação. Os algoritmos utilizados foram as DT, ANN e SVM, sendo comparado e apresentado o desempenho de cada um deles. Os resultados demonstraram que a abordagem é capaz de imitar a classificação dos especialistas e automatizar a tarefa de classificação, com acurácia acima de 83% em média.

3.2 ML DISPONIBILIZADO COMO SERVIÇO

Os Serviços de ML (MLaaS) buscam facilitar o acesso aos algoritmos de ML para simplificar o processo de criação, treinamento e implantação de modelos de ML. Uma das propostas desses serviços, como a própria Microsoft propõe com sua plataforma Azure (MICROSOFT, 2019), é descomplicar o uso dos algoritmos de ML, principalmente para empresas que querem utilizar modelos ML no mercado de forma mais rápida. Em geral, essas plataformas são hospedadas na Nuvem e fornecem ferramentas e estruturas prontas para uso visando facilitar a implantação de modelos de ML.

Ribeiro et al. (RIBEIRO; GROLINGER; CAPRETZ, 2015) propõem um *framework* de uma plataforma escalável, flexível e sem bloqueio que serve como um Serviço de ML, baseado no modelo de Arquitetura Orientada a Serviços (SoA). Essa plataforma fornece ferramentas para facilitar o processo de criação, validação e execução de modelos de ML, além de auxiliar com a coleta e organização de dados de múltiplas fontes e permitir a criação desses modelos utilizando diferentes algoritmos e técnicas de ML.

Yao et al. (YAO et al., 2017) avaliaram, em seu trabalho, a efetividade, performance e complexidade de seis plataformas de MLaaS, que incluem Amazon Machine Learning, Google Prediction API, Microsoft Azure ML Studio, Automatic Businesse Modeler, BigML, PredictionIO e um Cliente local totalmente customizado utilizando bibliotecas de ML. Seus resultados esclarecem a relação custo-benefício entre os sistemas de MLaaS e técnicas de ML aplicadas localmente em apenas um computador. Foi demonstrado que a utilização de sistemas locais permite uma maior e melhor customização das tarefas e técnicas de ML, dependendo apenas dos softwares e hardwares suficientes para execução. Embora tenham avaliado conjuntos de dados de vários tamanhos, eles não apresentaram resultados comparando o tempo de execução de cada plataforma.

Outra plataforma de MLaaS, chamada de Chiron, é apresentada por (HUNT et al., 2018). Chiron é um sistema para criação de modelos de ML que foca na questão de preservação da pri-

vacidade dos dados de clientes. Para isso, o sistema executa os processos de ML que utilizam esses dados numa espécie de 'Caixa Preta', buscando prevenir o vazamento de qualquer informação sobre os dados, os resultados de treinamento ou os modelos de ML. O principal objetivo dessa plataforma é permitir que detentores de dados (*Data holders*) treinem modelos ML em um serviço terceirizado sem revelar seus dados de treinamento.

3.3 ML EXECUTADO NA BORDA DA REDE

O sistema Zoo, apresentado por ZHAO et al. (2018), é uma plataforma que visa auxiliar o processo de composição, construção, e implantação de modelos de ML em dispositivos locais e na Borda (Edge). Esse sistema propõe a ideia de transformar as técnicas de ML em "serviços compostos", onde os usuários podem construir novos serviços baseados nos já existentes e, com isso, contribuir com a comunidade de desenvolvimento. Como grande parte dos usuários não possuem o conhecimento necessário de ML para entender como um modelo funciona e como implantá-lo adequadamente em dispositivos locais, o sistema Zoo busca utilizar um Design mais acessível com esses serviços compostos. Os exemplos de implantação, provam que os serviços de ML são fáceis de compor e implantar utilizando a plataforma Zoo. As avaliações também apresentam o seu desempenho em comparação com plataformas de MLaaS e de ML na borda.

Dask (AUGSPURGER; KOPPULA; ROCKLIN, 2019) fornece uma biblioteca, baseada em Python, que permite a criação de pipelines de funções ML. Esses pipelines podem especificar a execução de tarefas em sequência ou em paralelo, incluindo operações de limpeza de dados, transformação de dados, extração de recursos e características para acelerar estágios de treinamento e inferência, e, ainda, avaliação de modelos e seleção de modelos.

PredictionIO (CHAN et al., 2013) é uma plataforma que busca facilitar a construção de aplicações, "para o mundo real", de ML, possuindo uma interface gráfica para auxiliar os usuários. Essa interface gráfica possui um passo-a-passo que guia os desenvolvedores na criação, avaliação, comparação e implantação de algoritmos de aprendizado escaláveis, dentre outras opções. Para isso, sua arquitetura integra múltiplos processos de ML em um servidor distribuído onde os recursos são acessados através de interfaces de uma Interface de Programação de Aplicações (*Application Programming Interface - API*) e serviços WEB.

SZYDLO; SENDOREK; BRZOZA-WOCH (2018) apresentam uma proposta para utilização dos algoritmos de ML na borda, mais especificamente em dispositivos IoT com baixo poder de processamento. Seu trabalho propõe o conceito de geração do código de fonte de

modelos de ML para ser compilado diretamente no *firmware* desses dispositivos. Ao invés de implementar as bibliotecas de ML tradicionalmente, estas são convertidas em código fonte para facilitar a forma com que dispositivos embutidos (*embedded*) possam ler e interpretar os modelos. Sua abordagem tem a intenção de mover esses algoritmos de ML para a borda da rede, com acesso esporádico à nuvem. Neste sentido, os modelos não são treinados na borda; na realidade, a medida que determinado dispositivo da borda é alimentado (carregado), eles sincronizam com a nuvem, em um servidor, para enviar os históricos de uso e, se necessário, fazer o download de modelos de ML atualizados.

3.4 VIRTUALIZAÇÃO DE FUNÇÕES DE REDE

Diversas propostas vem surgindo com o intuito de realizar a integração do paradigma de NFV com outros serviços, visando auxiliar o processo de programação e planejamento das plataformas de NFVs e das VNF, além de atender aos requisitos altamente dinâmicos dos diversos serviços de rede (Matias et al., 2015; Duan; Ansari; Toy, 2016). Apesar desses trabalhos apresentarem o deslocamento de NFV de um agnóstico de *Software-Defined Networking* (SDN) inicial para uma solução NFV totalmente habilitada para SDN, eles não apresentam implementações reais. Além disso, o estado-da-arte existente apresenta a Orquestração NFV usando switches OpenFlow baseados em hardware para interconectar nodos VNF, fornecendo programação limitada de infraestrutura física.

Outra plataforma de NFV, proposta por DOMINICINI et al. (2017, 2016) e chamada de VirtPhy, implementa um *testbed* para fornecer uma infraestrutura de rede totalmente programável ao orquestrador NFV com base numa topologia de rede centrada no servidor (*server-centric network topology*). Essa proposta permite e, acima de tudo, torna viável a implementação adequada das principais funcionalidades de nossa abordagem de MLFV: orquestração, alocação de recursos e funções de rede em cadeias (SFC), e virtualização de funções (CASTANHO et al., 2018).

Nguyen et al. 2017 formulou um modelo de programação quadrática e propôs uma solução heurística para solucionar problemas de colocação e roteamento de VNF. No entanto, eles não consideram questões como a criação, manutenção e execução de cadeias de VNF e outras dependências nesse contexto.

BACON (HAWILO; JAMMAL; SHAMI, 2019) propôs um algoritmo para otimizar o problema de colocação de VNF em uma *chain*, apresentando modelo para otimização de pro-

gramação linear inteira mista em redes de computadores de pequena e grande escala. Seu trabalho considera restrições de CPU, memória, atrasos de rede e disponibilidade da conexão. Essa proposta, assim como o VirtPhy (DOMINICINI et al., 2017), auxiliou no desenvolvimento das restrições impostas pela abordagem de proposta nessa pesquisa.

OPNET (PHAM et al., 2017) propõe uma abordagem que combina a aproximação de cadeias de Markov com a Teoria do ajuste (*Matching theory*) para otimizar o processo de colocação de cadeias de VNF e, com isso, economizar energia e minimizar o tráfego de dados na rede.

BENKACEM et al. (2018) formulou e apresentou, em seu trabalho, uma nova proposta para realizar a alocação de VNF, utilizando um algoritmo que minimiza o custo e maximiza o *Quality of Experience* (QoE) de serviços de streaming virtuais. Os autores aplicaram a Teoria ou Problema de Barganha (*Bargaining Game Theory*) na solução proposta, buscando alcançar um balanço ideal entre a eficiência de custo e QoE.

3.5 CONTRIBUIÇÕES

Como motivação para o desenvolvimento deste trabalho, tem-se que a utilização de uma plataforma de NFV para gerenciar, virtualizar e realizar a orquestração de atividades relacionadas à construção e utilização de modelos de ML possa tornar essas atividades mais eficientes, reduzindo custos e melhorando o tempo de execução. O estado da arte relata trabalhos conectando algoritmos de ML e NFV apenas com primeiro sendo utilizado para melhorar o segundo; as técnicas de ML, em geral, são vistas como funções intermediárias para auxiliar no controle de tráfego de dados, por exemplo (RAHMAN et al., 2018). No entanto, as plataformas de NFV podem ser uma abordagem promissora para implementação de Serviços de ML na borda da rede, uma vez que permitem a implantação, encadeamento e virtualização de funções de rede em hardwares genéricos (MA; MEDINA; PAN, 2015). As principais características consideradas para desenvolvimento da abordagem proposta nesta pesquisa, com base nos trabalhos apresentados, são as seguintes:

Utilização de plataformas de NFV. O principal diferencial entre as abordagens de ML implementadas nos trabalhos citados e a desenvolvida neste trabalho é a utilização de uma plataforma de NFV. Essa plataforma permite realizar o gerenciamento e orquestração das atividades que compõem o processo para criação e implantação de modelos de ML. Além disso, permite que funções de ML sejam virtualizadas e utilizadas sob demanda, da mesma forma como as

funções de rede (Seção 2.4).

Análise do estado da rede. A utilização da plataforma de NFV permite, além da virtualização de funções, verificar o estado atual da rede entre o Módulo principal e seus Clientes para orquestrar as funções de forma eficiente. Verificar o estado da rede antes de enviar uma função ou dados para serem processados em nodos computacionais evita problemas de como o atraso na execução e congestionamento na rede.

Análise de requisitos computacionais e bibliotecas. A abordagem proposta nesta pesquisa integra, ainda, algoritmos para otimizar a colocação das cadeias de funções virtualizadas com base nos requisitos computacionais dos nodos (Clientes), semelhante ao que propõem (HAWILO; JAMMAL; SHAMI, 2019). A análise desses requisitos evita problemas de incompatibilidade entre Clientes e funções durante a execução das atividades, além de otimizar o processo de distribuição e alocação dessas atividades.

Conforme visto neste capítulo, até a defesa deste mestrado, não foram encontrados na literatura trabalhos que contemplem todos os aspectos propostos nesta pesquisa: Considerar o estado da rede para orquestrar e virtualizar funções de ML na borda. Desta forma, foi desenvolvida uma abordagem chamada MLFV (*Machine Learning Function Virtualization* – Virtualização de Funções de Aprendizado de Máquina), que será apresentada no próximo capítulo.

4 MACHINE LEARNING FUNCTION VIRTUALIZATION

A proposta para Virtualização de Funções de Aprendizado de Máquina (MLFV) é apresentada nesse capítulo. O principal objetivo do MLFV é realizar a orquestração e virtualização de funções de ML utilizando uma plataforma de NFV, considerando o estado da rede. Essa plataforma, como descrito na Seção 2.4, possibilita a orquestração e virtualização de funções levando em consideração diversas restrições, permitindo o encadeamento de funções para realização de tarefas de ML.

O capítulo primeiramente explica como as funções de ML, baseadas no processo KDD foram implementadas. A seguir é demonstrado como essas funções são agregadas em forma de cadeias para que possam ser executadas. Por fim, a arquitetura do MLFV é apresentada, mostrando a interação entre seus módulos, o modelo matemático para colocação das funções nos clientes disponíveis e detalhes da implementação.

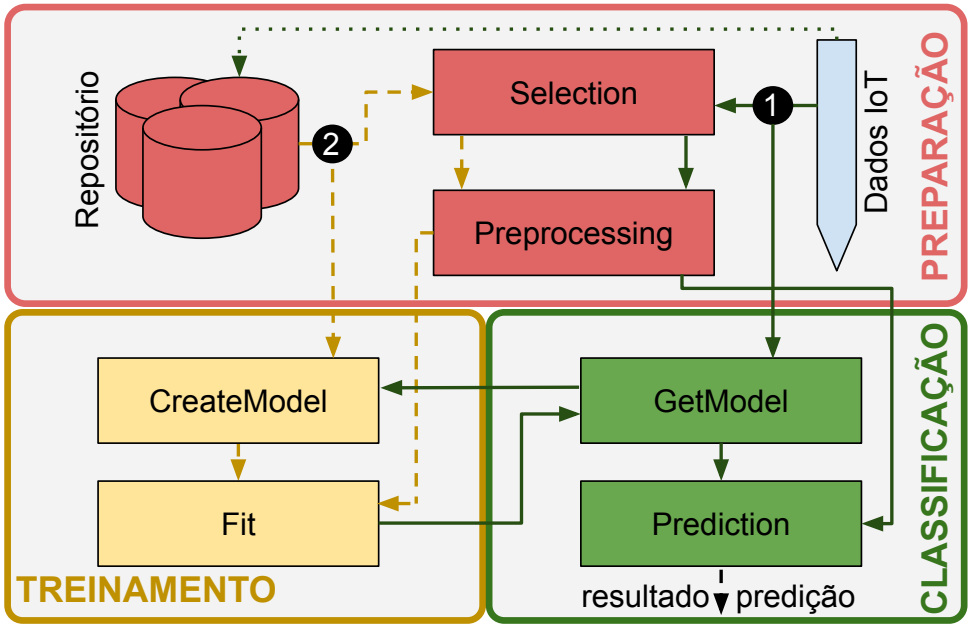
4.1 FUNÇÕES DE ML E O PROCESSO DE KDD

Uma função de ML, no contexto desta pesquisa, se refere a um método, implementado na linguagem Python, para representar as atividades específicas e necessárias no processo de criação, treinamento e utilização de modelos de ML. Como descrito na Seção 2.1, existe uma rotina de diferentes passos e atividades na criação de modelos de ML, essenciais para assegurar que o conhecimento derivado dos dados reflita a realidade e os padrões existentes. Desse modo, a implementação dessas funções de ML tomou como base o processo de KDD, onde as funções propostas nesta pesquisa foi inspirada em uma ou mais etapas propostas por esse processo (Seção 2.1). Reproduzir os passos apresentados pelo KDD garante a padronização e um fluxo de execução de atividades bem definido, mais flexível, eficiente e otimizado. No entanto, é importante destacar que a utilização desse processo não é obrigatória, servindo mais como um guia e uma forma para avaliação das funções propostas. Diferentes processos para desenvolvimento e implementação de funções podem ser aplicados.

Ao total foram desenvolvidas seis (6) funções, divididas em três (3) etapas, como pode ser visto na Figura 6. As etapas servem para organizar as funções de acordo com suas características e representam o objetivo final que deve ser alcançado pela execução das suas funções. Cada função possui um objetivo único que é alcançado através da execução de uma série de

atividades e algoritmos. Os resultados da execução de cada função serve como entrada para outra função, criando um sequência de atividades semelhante aos que é proposto pelo KDD. Essa cadeia de atividades cria um serviço completo e composto, com uma finalidade diferente e um produto final único; neste caso, o processo para (1) **criação e treinamento** de modelos de ML (representado pelas setas de linhas contíguas na Figura 6), ou (2) a **predição de dados** com base nesses modelos (representado pelas linhas tracejadas). A abordagem de MLFV apresenta esse fluxo de execução como cadeias de funções de ML (*Machine Learning Function Chain* (MLFC)), as quais serão descritas na Seção 4.2. A seguir são detalhadas cada etapa e funções apresentadas.

Figura 6 – Visão global das Etapas e Funções de ML propostas



Fonte: Próprio autor.

4.1.1 Etapa de Preparação

A etapa de Preparação agrupa duas funções responsáveis por automatizar o processo de preparação e manipulação de dados brutos. Seu principal objetivo é tornar esses dados brutos, através da sua organização e remoção de inconsistências, num conjunto de dados adequado para ser utilizado por algoritmos de ML. Para isso, esta etapa é composta de duas funções:

Selection. Tem como objetivo agrupar esquemas de bases de dados em um mesmo local e formato. Essa organização permite que o acesso aos dados seja fácil e rápido, otimizando todos os processos seguintes. Compõe a primeira atividade proposta pelo processo *KDD* e pos-

sibilita aumento da velocidade e precisão dos algoritmos. Recebe como parâmetro de entrada conjuntos de dados (*dataset*) e as colunas (*columns*). '*dataset*' se refere aos dados brutos informados, podendo estar armazenados em diferentes formatos, tamanhos ou grupos. '*columns*' se refere as colunas, presentes nessas bases de dados, que serão utilizadas como características (*features*, *inputs*) e classes ou rótulos (*outputs*) pelos algoritmos de ML. A função organiza os dados fornecidos com base no parâmetro '*columns*', selecionando apenas as colunas relevantes (Redução de dimensionalidade), criando um novo banco de dados unificado. Uma vez terminada a função retorna, como resultado, esse novo conjunto de dados.

Preprocessing. Responsável pela aplicação de diversos algoritmos ou filtros para realizar a limpeza ou remoção de inconsistências básicas nos dados. Em geral, valores atípicos e imprecisos como negativos, zeros, em branco (valores faltantes), nulos e valores discrepantes (*outliers*, ruídos) são removidos para preservar a consistência dos dados e aumentar sua qualidade. Além disso, essa função tem como papel realizar o processo de normalização dos dados, o qual se utiliza de algoritmos de transformação como o *"Standard Scaler"* ou "Min-Max"(Seção 2.1.3). Recebe como parâmetro de entrada esperado o '*dataset*' resultante da função de Seleção. Retorna, como resultado, o conjunto de dados da entrada devidamente pré-processado.

4.1.2 Etapa de Treinamento

A etapa de Treinamento é responsável pela criação, treinamento, teste e validação de modelos de ML. Estes modelos são criados através da extração de conhecimento de bases de dados, através da utilização de diferentes algoritmos de ML. Futuramente, estes modelos serão utilizados pela etapa de Classificação, onde o conhecimento adquirido será utilizado para realizar tarefas como predição ou classificação. O ciclo de operação desta etapa consiste na execução de duas (duas) funções, as quais são inspiradas na etapa de Mineração de Dados do processo de *Knowledge Discovery in Databases* (KDD), descritas a seguir.

CreateModel. Função responsável por criar uma instância de um modelo de ML. Para isso, essa função recebe como parâmetros de entrada o nome do classificador ('*classifier*') e as configurações desejadas ('*options*'). Dentre as opções de classificadores disponíveis, destacam-se as ANN, RF, k-NN, dentre outros. O parâmetro de '*options*' varia dependendo do classificador escolhido, podendo conter valores como, por exemplo, Épocas (*Epochs*, ou número de vezes que o algoritmo irá executar no treinamento), Camadas (*layers*), *seed* (estado randômico), dentre outros. Uma vez terminada a função retorna, como resultado, uma instância de

um modelo de ML escolhido.

Fit. Função responsável pelo treinamento do modelo de ML criado, utilizando um conjunto de dados de treinamento (*trainset*). Recebe como parâmetros de entrada duas variáveis: (i) instância de um modelo de ML (*model*), resultado da função de **Criação**; (ii) conjunto de dados (*trainset*), resultante da operação realizada na função de **Preprocessamento**. Todos os dados fornecidos são utilizados como exemplo para ajustar o modelo de ML conforme o padrão existente nestes valores, como descrito na Seção 2.1.4. Ao final desse ajuste, a função retorna a instância de um modelo de ML treinado e ajustado com base nos dados de treinamento fornecidos.

4.1.3 Etapa de Classificação

A etapa de Classificação é responsável pelo processo necessário para realizar a classificação ou predição de classes (rótulos) em determinado conjunto de dados. Como visto na Figura 6, esse processo é composto de duas funções, as quais são descritas a seguir:

GetModel. A função 'GetModel' tem o objetivo de carregar um modelo de ML já existente. Em geral, a função irá recriar esse modelo, o qual já passou pelo processo de treinamento, e repassá-lo para a função 'Prediction' executar a tarefa de predição. Para isso, a função 'GetModel' recebe como entrada o parâmetro '*classifier*', contendo as informações necessárias para realizar a criação de uma nova instância de um modelo de ML já existente. Retorna, quando terminado esse processo, como resultado o modelo para iniciar a tarefa de classificação na função 'Prediction'.

Prediction. Tem como tarefa realizar a classificação ou predição de dados fornecidos. A função 'Prediction' recebe, como entrada, um modelo, carregado pela função 'getModel', e o conjunto de dados à ser classificado. Uma vez finalizada a tarefa de predição, retorna o conjunto de dados, devidamente classificado, ao **Módulo MLFV** finalizando a execução da cadeia de funções.

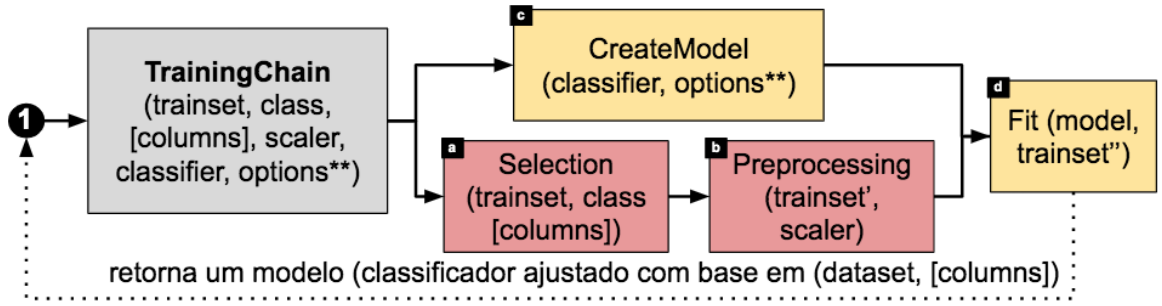
4.2 CADEIAS DE FUNÇÕES DE APRENDIZADO DE MÁQUINA

Quando várias funções são organizadas em um determinada ordem de execução elas formam cadeias de funções, conceito proposto pelas abordagens de SFC e NFC (Seção 2.4). Uma NFC se refere a uma lista de funções de rede a serem executadas em uma determinada

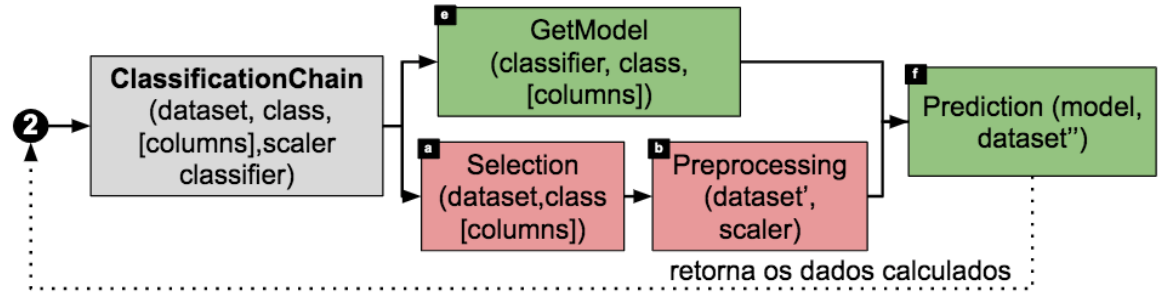
ordem buscando criar, assim, serviços compostos. De acordo com CASTANHO et al. (2018), essas cadeias de funções podem ser descritas como um conjunto de atividades de diferentes naturezas (com diferentes propósitos) organizadas em uma determinada sequência, criando uma atividade ou serviço completo para realizar uma tarefa complexa.

Uma *Machine Learning Function Chain* (MLFC), diferente de uma NFC, propõem o encadeamento de funções de ML para criar serviços de descoberta de conhecimento e extração de informações em dados; ao invés de uma sequência de funções de redes tradicionais (Figura 5), uma MLFC é composta pela sequência de funções e atividades necessárias para criação e treinamento de modelos de ML ou para realizar a classificação e predição de dados usando esses modelos (Figura 7).

Figura 7 – Cadeias MLFV propostas.



(a) **Cadeia de Treinamento**.



(b) **Cadeia de Classificação**.

Fonte: Próprio autor.

A Figura 7 materializa as duas cadeias MLFC apresentadas na Figura 6. A **Cadeia de Treinamento** (*TrainingChain*, Figura 7a) compreende todo o processo hierárquico de execução das atividades para criação, preparação e treinamento, através de aprendizado supervisionado (Seção 2.2.2), de modelos de ML. A **Cadeia de Classificação** (*ClassificationChain*, Figura 7b), por sua vez, apresenta a sequência de atividades necessárias para realização do processo de

predição, em dados não classificados, utilizando um modelo de ML; esse modelo é resultante da cadeia de Treinamento. É possível notar ainda ambas MLFC permitem que funções sejam organizadas em serial e paralelo, dependendo das suas dependências.

Apesar de possuírem diferentes objetivos e funcionalidades, as cadeias de **Classificação** e **Treinamento** compartilham um conjunto comum de ações executadas como parte do processamento de dados proposto pelo KDD (Seção 2.1.2). Isso inclui as duas funções para Preparação dos dados (*Selection* e *Preprocessing*) devido a constante necessidade da limpeza dos dados, independente do serviço solicitado. Dessa forma, essas funções genéricas são reutilizadas sempre que necessário, sem geração de novo código.

O funcionamento de uma **Cadeia de Treinamento** (Figura 7a) pode ser resumido nos seguintes passos: O **Módulo MLFV** recebe uma requisição de Treinamento contendo um conjunto de dados exemplos (*trainset*), as colunas que são entradas e saídas desejadas (*columns*), o classificador (*classifier*) a ser utilizado e as opções para ajustar esse classificador (*options*). O primeiro passo ocorre com a criação do modelo de ML, pela função *CreateModel*(*c*), utilizando o parâmetro opções. Ao mesmo tempo em que esse modelo é criado, a função de *Selection* (*a*) recebe o conjunto de dados bruto para prepará-lo; o resultado é repassado para a função de *Preprocessing* (*b*), a qual realiza o preprocessamento dos dados. Após o término de préprocessamento, os dados são repassados para a função *Fit* (*f*), junto com o resultado da função *CreateModel* (*c*), que se refere a um modelo de ML sem treinamento. Por fim, a função *Fit* (*f*) executa o processo de treinamento do modelo de ML usando os dados preprocessados. O resultado é retornado ao **Módulo MLFV**.

As ações propostas por uma **Cadeia de Classificação** (Figura 7b) do MLFV podem ser resumidas em 4 passos, funcionando da seguinte forma: O **Módulo MLFV** recebe uma requisição para Classificação contendo um conjunto de dados brutos (*dataset*), nome da coluna de rótulos (*class*), colunas com os parâmetros (*columns*) e um classificador (*classifier*) desejado (ANN, RF, ou outro). Em um primeiro momento, um modelo de ML é carregado pela função *getModel* (*e*) utilizando os parâmetros de classificador, classe e colunas fornecidos. Ao mesmo tempo, o conjunto de dados é processado e preparado pela função de *Selection* (*a*). O resultado alcançado pela função de *Selection* (*a*) é repassado para a função de *Preprocessing* (*b*), para limpar os dados. Por fim, após o término da limpeza, os dados são repassados para a função de *Prediction* (*f*), ao mesmo tempo que o resultado da atividade *getModel* (*e*) também recebido. Com ambas informações (Modelo ML e conjunto de dados preparado) em mãos, a função de

Prediction (f) executa sua tarefa de predição, retornando os resultados para o **Módulo MLFV**.

4.3 ARQUITETURA PROPOSTA

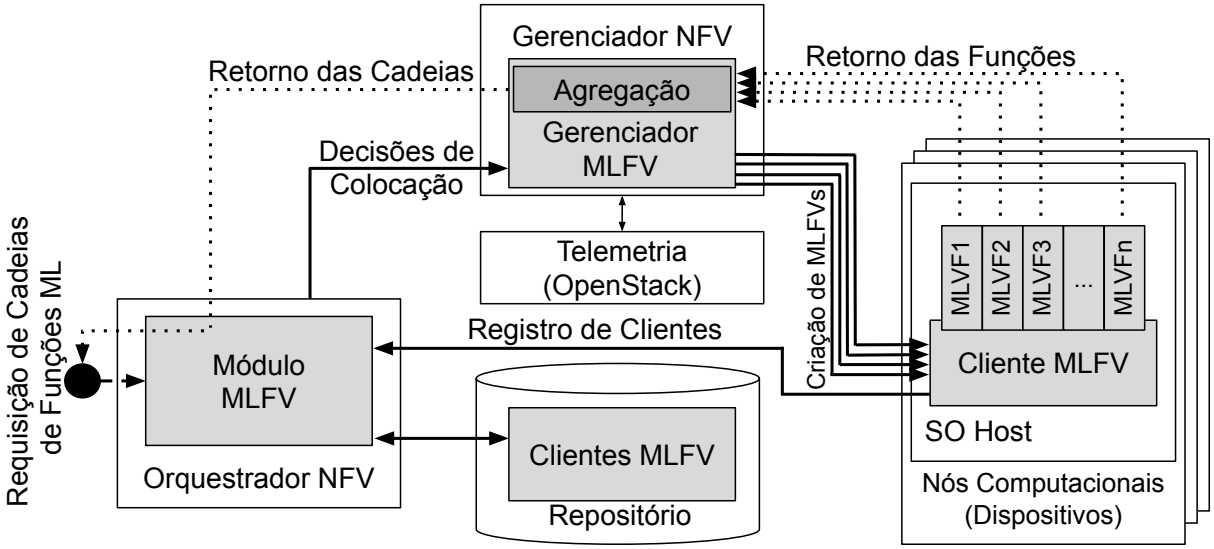
A arquitetura proposta para criação da abordagem de MLFV foi desenvolvida com base no processo de KDD (Seção 2.1), nas cadeias de NFC e plataformas de NFV (Seção 2.4). As seções anteriores apresentaram como as funções de ML podem ser agregadas, paralela ou sequencialmente, através de uma MLFC para formar o processo de KDD. Nessa seção, será descrita a arquitetura de MLFV que, utilizando uma plataforma de NFV, tem como principal objetivo realizar a orquestração e virtualização dessas cadeias de funções em clientes conectados na borda da rede.

A Figura 8 apresenta a arquitetura de MLFV, onde os módulos coloridos na cor branca se referem aos módulos da plataforma NFV, enquanto os módulos coloridos em cinza, desenvolvidos e implementados neste trabalho, representam a proposta de MLFV. A arquitetura do sistema é composta por várias unidades e módulos, sendo eles:

Requisição de Cadeias de ML. A arquitetura de MLFV recebe como entrada MLFC. Uma MLFC é uma lista ou conjunto de funções de ML que podem estar organizadas em série ou em paralelo. A medida que cada função é executada, os resultados retornados por ela são utilizados como entrada para próxima função na sequência de eventos da cadeia. O conceito de MLFC segue a mesma ideia proposta pelas cadeias de funções de rede (NFC) para que sejam interpretadas pelo orquestrador NFV, como apresentado na Seção 2.4. No entanto, ao invés de virtualizar cadeias de funções de rede e criar serviços que auxiliam em atividades relacionadas ao contexto de rede, nossa proposta busca virtualizar cadeias de funções de ML, as quais representam a sequência de atividades propostas pelo processo de KDD para criar serviços de descoberta de conhecimento e predição.

Módulo MLFV. O **Módulo MLFV** funciona em cima de um Orquestrador de NFV que, originalmente, é responsável por centralizar a orquestração e alocação de funções de rede e gerenciar recursos e serviços da rede. O **Módulo MLFV** recebe requisições de MLFC e, para cada função nessa cadeia, consulta o **Repositório de Clientes ML**. Essa consulta busca encontrar um conjunto de **Clientes** que atendam aos requisitos de CPU e memória para cada função da cadeia. Além disso, busca encontrar os **Clientes** que possuam as bibliotecas de ML necessárias para execução de cada uma funções. Dessa forma, esse Módulo associa cada uma das funções de uma MLFC a um ou mais **Clientes MLFV** que atendam a todas essas restrições

Figura 8 – Arquitetura da proposta de MLFV.



Fonte: Próprio autor.

impostas, organizando o envio e execução de cada função. Este módulo também gerencia os pedidos de registros para novos **Clientes MLFV** que queiram fornecer suas máquinas para virtualizar as funções de ML.

Gerenciador MLFV. Recebe, como entrada, uma MLFC junto de uma lista de clientes compatíveis para cada uma das funções dessa MLFC. Em resumo, essas informações se referem as decisões de colocação apresentadas na Figura 8. Antes de iniciar a distribuição das funções para os clientes com base nas decisões de colocação, o Gerenciador MLFV verifica o estado da rede para cada cliente, usando informações de sobrecarga de rede coletadas pelo módulo de *OpenStack Telemetry*[3]. A partir dessas informações o Gerenciador MLFV seleciona o **Cliente** menos sobrecarregado e envia uma das funções da MLFC para ser virtualizada e, posteriormente, executada. Portanto, o objetivo deste módulo é ser responsável por despachar, iniciar a execução, monitorar e receber os valores resultantes de cada Função Virtual de ML (MLVF). Caso uma MLFC especifique **operações de junção**[4], o **Módulo de Agregação** aguarda o retorno de todas as funções envolvidas nessa junção para continuar executando a cadeia. Para determinar qual **Cliente** executará cada função, o gerenciador solicita as informações disponí-

[3] Usamos o OpenStack Telemetry para calcular a subtração da taxa média de entrada (ou saída) de bytes pela largura de banda total da conexão. Os dados são coletados utilizando um *Ceilometer*.

[4] Quando uma função de ML específica depende dos resultados produzidos por outras funções de ML. Por exemplo, a utilização de dados brutos pelas funções de Treinamento ou Classificação só ocorre após a execução da função de pré-processamento, a qual só será executada após a função de seleção. Assim, é criado uma cadeia de eventos.

veis a cerca de cada **Cliente** cadastrado, como, por exemplo, a largura de banda disponível ao módulo de *OpenStack Telemetry*.

Cliente MLFV. Um **Cliente MLFV** é instalado em **dispositivos** que desejam oferecer seus recursos computacionais para processamento de atividades de ML. Para isso, cada **dispositivo** registra seu interesse informando suas capacidades e recursos computacionais (CPU e memória RAM) e quais bibliotecas estão disponíveis; dessa forma, é possível determinar quais funções e tarefas determinado **Cliente** é capaz de realizar. Todas as informações são recebidas pelo **Módulo MLFV** e armazenas no **Repositório de Clientes ML**. Após esse processo de registro, os novos **Clientes MLFV** estão aptos para receber requisições do **Gerenciador MLFV**. A cada nova requisição, esse **Cliente** instância um ambiente virtual para executar as funções de ML e, quando finalizado, retorna o resultado de volta para o **Gerenciador MLFV**.

Repositório de Clientes ML. Mantém as informações sobre os clientes cadastrados, armazenando, para cada **Cliente**, o seu endereço de IP, capacidades computacionais e bibliotecas instaladas. O repositório também mantém um histórico de todas as funções ML enviadas para cada **Cliente**; essa informação também é usada para identificar o cliente menos ocupado.

Esta arquitetura foi desenvolvida com o objetivo de viabilizar o desenvolvimento de sistemas de MLaaS capazes de orquestrar e virtualizar de cadeias de funções de ML considerando restrições como o estado atual da rede, recursos computacionais e capacidades dos dispositivos conectados. Para isso, foram utilizados os diferentes conceitos propostos no Capítulo 2 e nas Seções 4.1 e 4.2.

A seguir será apresentado o modelo matemático que possibilita o funcionamento dessa arquitetura. Esse modelo busca minimizar o custo e tempo para realizar a distribuição, colocação e execução das funções de ML propostas.

4.3.1 Modelo Matemático para Colocação de MLFC

O modelo matemático proposto pela abordagem de MLFV tem como objetivo otimizar o processo de alocação e distribuição de funções de ML nos dispositivos (Clientes MLFV) conectados na rede. Esse modelo foi implementado para aprimorar a orquestração dessas funções no Módulo MLFV, o qual está acoplado à plataforma de NFV.

Para realizar a orquestração e virtualização das funções de uma MLFC, o **Módulo MLFV** considera diversas restrições e atributos dos **Clientes MLFV**, sendo elas: Valores de CPU e memória RAM; sobrecarga da rede e largura de banda disponível; e disponibilidade das

bibliotecas necessárias para execução das funções e atividades. Desse modo, o **Gerenciador MLFV** terá conhecimento de todas as informações necessárias referentes as decisões de colocação das funções, possibilitando um melhor planejamento do local de execução (**Cliente**) de cada função da cadeia.

O desenvolvimento do algoritmo para colocação das cadeias de ML e gerenciamento das atividades do **Módulo MLFV** foi inspirado pelos trabalhos de (DOMINICINI et al., 2017, 2016; HAWILO; JAMMAL; SHAMI, 2019; GOKHALE; M, 2011). Nesta seção será apresentado o modelo matemático utilizado para representar o funcionamento desse algoritmo. A seguir são descritas todas as variáveis utilizadas (Tabela 3) e como o problema foi desenvolvido.

Tabela 3 – Operadores utilizados no modelo matemático.

Nome	**Operador**
Gerenciador MLFV	M
Conjunto de funções ML	F
Função ML	f
Cadeia de funções (MLFC)	$\sum_{f=0}^{F} f, \forall f \in F$
Subconjunto de funções	τ
Operação sequencial	so
Operação paralela	po
Parâmetros de determinada função	P_f
Tamanho dos parâmetros (bytes)	P_f^s
Clientes MLFV disponíveis	C
Cliente único	c
Recursos computacionais (RC)	R
RC único	r
RC utilizado por uma função	r_f
RC de um Cliente	r'_c
Bibliotecas Python	A
Subconjunto de bibliotecas (SB)	L
SB para executar uma função	L_f
SB instaladas em um Cliente	L'_c
Largura de banda entre o MLFV e um Cliente	B_{Mc}
Decisão binária	e_{fc}
Constante	k

Fonte: Próprio autor.

O operador e_{fc}, que representa a decisão binária, recebe 1 caso o cliente c esteja qualificado a executar a função f; caso contrário, o operador binário recebe o valor 0. Já a constante k, representada pelo operador k, serve para correlacionar o tempo de processamento com o tempo de transmissão dos parâmetros através da rede, e tem como objetivo aumentar a importância do tempo de transmissão no cálculo final.

Após apresentar os principais operadores que serão utilizados no modelo matemático, é preciso definir os operadores para representar os diferentes tempos de execução existentes. Uma função f, por exemplo, possui diversos tempos independentes para executar algum tipo de ação, como demonstrado a seguir:

- O tempo que o MLFV leva para receber a requisição e liberar uma função f é caracterizado por rt_f;
- O tempo para transmitir os parâmetros de uma função f para um cliente, na rede, é denotado como tt_{fc};
- O tempo para processar uma função f em um cliente c é dado por pt_{fc};
- O tempo decorrido para receber de volta o resultado de uma função f, executada por um cliente c, é denotado por gt_{fc};
- O tempo para concluir a execução de uma cadeia de funções de ML é dado por CT_f.
- O tempo para concluir uma única função de ML, dentro da cadeia, em um cliente c é denotado por ct_{fc};
- O tempo para processar um conjunto de funções τ de forma sequencial, onde determinada função f pertence a esse conjunto, é dado por TS_f^τ;
- Já o tempo para processar um conjunto de funções τ em paralelo, onde uma função f pertence a esse conjunto, é dado por TP_f^τ.

Se uma função f pertence a um grupo de funções τ em uma operação sequencial, o operador binário so_f é definido como 1; caso contrário, o valor é definido como 0. Se uma função f pertence a um grupo de funções τ em uma operação paralela, o operador binário po_f é definido como 1; caso contrário, o valor é definido como 0.

A função objetivo foi formulada da seguinte forma:

$$Minimizar \sum_{f=0}^{F} CTf - rt_f \; \forall f \in F \tag{4.1}$$

Sujeito a:

Tempo para transmitir os parâmetros da função:

$$tt_{fc} = P_f^s \div B_{Mc}, \forall f \in F, \forall c \in C \tag{4.2}$$

Tempo de processamento:

$$pt_{fc} = r_f \div r_c, \forall f \in F, \forall c \in C \tag{4.3}$$

Tempo de conclusão de uma única função:

$$ct_{fc} = rt_f + tt_{fc} + pt_{fc} + gt_{fc}, \forall f \in F, \forall c \in C \tag{4.4}$$

Tempo de processamento em série:

$$TS^{\tau}_{fc} = \sum_{g=0}^{\tau} ct_{gc} \times so_f, \forall f, g \in F, \forall c \in C \tag{4.5}$$

Tempo de processamento paralelo:

$$P^{\tau}_{fc} = \sum_{g=0}^{\tau} Max(ct_{gc} \times po_f), \forall f, g \in F, \forall c \in C \tag{4.6}$$

Restrições computacionais:

$$\sum_{f=0}^{F} e_{fc} \times r_f \leq C^r_c, \forall r \in R, \forall c \in C \tag{4.7}$$

As restrições de bibliotecas são expressas por:

$$\sum_{f=0}^{F} e_{fc} \times L_f \subset C^L_c, \forall L \in A, \forall c \in C \tag{4.8}$$

A seleção de clientes para executar as funções é dado por:

$$\sum_{f=0}^{F} \sum_{c=0}^{C} Min(tt_{fc} \times k + pt_{fc}), \forall f \in F, \forall c \in C \tag{4.9}$$

O tempo de conclusão de uma cadeia de funções é calculado como:

$$\sum_{f=0}^{F} TS_f + TP_f \; \forall f \in F \tag{4.10}$$

A função objetivo, apresentada na Equação 4.1, minimiza o tempo de conclusão de todas as funções em uma cadeia de funções ML. Como o tempo do MLFV para disponibilizar uma função não pode ser reduzido, ele foi desconsiderado do tempo total. A Restrição (4.2) é usada para derivar o tempo para transmitir os parâmetros de uma determinada função. Esse valor é calculado dividindo o tamanho total dos parâmetros pela largura de banda disponível entre o **Gerenciador MLFV** e o **Cliente MLFV**.

De forma similar, a Restrição (4.3) é dada através da divisão dos recursos computacionais (CPU, RAM) de uma função pelos recursos computacionais do **Cliente MLFV** que irá executar essa função; essa restrição indica que, quanto mais recursos recursos computacionais o **Cliente** possuir, menor será o tempo de processamento das atividades nele executadas.

O tempo para concluir a execução de uma função é dado pela Restrição (4.4), onde todos os tempos necessários para realizar as tarefas propostas são somados. A Restrição (4.5) efetua o cálculo do tempo total para uma cadeia de funções em série executar suas operações, através da soma dos tempos de todas as funções envolvidas nesse processo. Já a Restrição (4.6) descreve o tempo total para uma cadeia de funções em Paralelo executar suas operações, o qual é representado pelo maior tempo de execução apresentado por uma das funções existentes. Em outras palavras, como as funções são executadas ao mesmo tempo, o tempo de execução da função que finalizar sua tarefa por último representa o tempo total da cadeia.

A Restrição (4.7) garante que o **Cliente MLFV** possua os recursos computacionais necessários para executar determinada função de ML, enquanto a Restrição (4.8) garante que esse **Cliente** possua as bibliotecas necessárias. A forma de selecionar qual cliente irá executar determinada função é dada pela Restrição (4.9); ela obtém o valor mínimo considerando o tempo de execução, mais o tempo de transmissão multiplicado por uma constante que aumenta a importância desse tempo no cálculo final.

Por fim, o tempo total para concluir a execução de uma cadeia é derivado na Restrição (4.10), onde o tempo de conclusão de todas as operações (seriais e paralelas) da cadeia são somados. Os detalhes da implementação desse modelo matemático e cada módulo da abordagem de MLFV são discutidos na próxima seção.

4.3.2 Implementação

A implementação da proposta de MLFV foi desenvolvida em Python, na sua versão 2.7 (ROSSUM, 1995). Esta escolha se justifica, principalmente, pela variedade de bibliotecas de métodos disponíveis para implementação de algoritmos de ML (Scikit-Learn (v0.20.0)), cálculos matemáticos (numpy (v1.15.2)), manipulação de dados e análises estatísticas (Pandas (v0.23.4))[5]. Essa seção tem como objetivo abordar a forma como foram implementadas as principais classes, métodos e funções, para criação da abordagem de MLFV, além de apresentar as principais bibliotecas utilizadas nesse processo.

[5] https://scikit-learn.org | https://pandas.pydata.org | http://www.numpy.org

A biblioteca utilizada para realizar a comunicação entre os usuários, clientes, serviço de MLFV e o orquestrador de NFV é chamada de *Remote Python Call library* (RPyC v4.0.2)[6]. Ela permite a chamada simétrica de procedimentos e métodos Python remotos, agrupamento de atividades e computação distribuída. A biblioteca RPyC permitiu tornar o **Módulo MLFV** em um servidor que recebe, processa e repassa requisições de cadeias de funções para que o **Orquestrador NFV** possa realizar o processo de distribuição e virtualização das funções, listadas na cadeia, nos **Clientes MLFV**.

A Figura 9 apresenta como o **Módulo MLFV** foi desenvolvido utilizando a biblioteca RPyC. Essa biblioteca, além de permitir o desenvolvimento do servidor que recebe e processa requisições de MLFC, possibilitou implementar o processo para transformar, compactar e enviar funções e todos os seus parâmetros através do **Orquestrador NFV** para serem executadas virtualmente nos **Clientes MLFV**.

Figura 9 – Classe Python para implementar o servidor requisições do Módulo MLFV.

```
1  import rpyc
2  port=8888
3
4  class MLFV_Service(rpyc.Service):
5     def on_connect(self,x):
6         print "Connection received"
7     def on_disconnect(self,x):
8         print "Connection ended"
9     def exposed_exec_chain(self, c, p):
10        MLFV.receive_chain(c, p)
11
12 if __name__ == "__main__":
13     rpyc.core.protocol.DEFAULT_CONFIG['allow_pickle'] = True
14     from rpyc.utils.server import ThreadedServer as Server
15     import MLFV_Module as MLFV
16     server = Server(MLFV_Service, port=port, backlog=10, protocol_config=rpyc.core.
           protocol.DEFAULT_CONFIG)
17     server.start()
```

Fonte: Próprio autor.

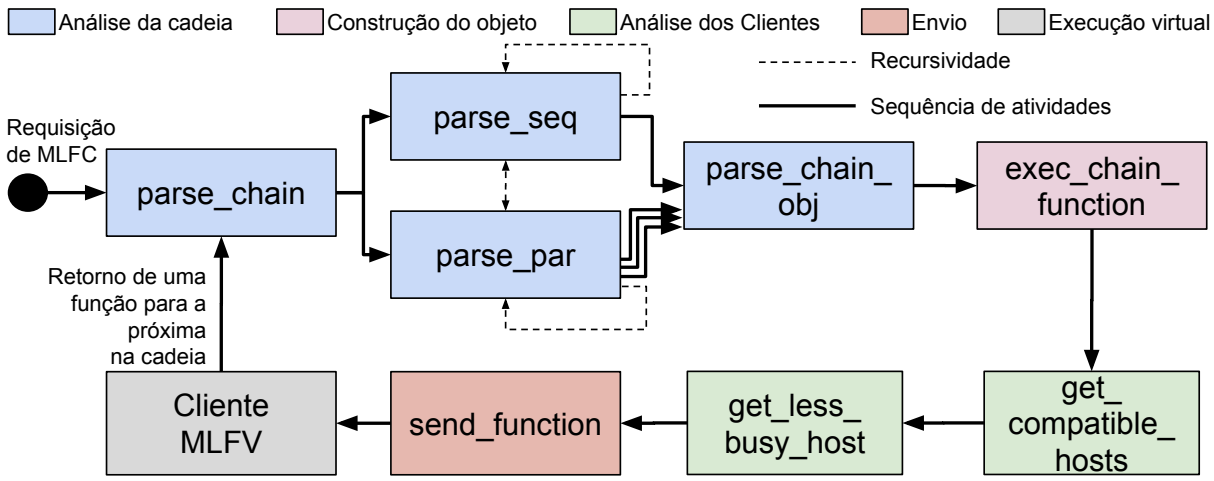
O código da Figura 9 apresenta a classe 'MLFV_Service' (linha 4), composta por métodos pré-definidos para iniciar, finalizar e tratar as requisições recebidas pelo **Módulo MLFV**. Dentre os métodos existentes destaca-se o 'exposed_exec_chain' (linha 9) que, em resumo, fornece o acesso remoto a um dos métodos do **Módulo MLFV** utilizando as ferramentas fornecidas pela biblioteca RpyC, as quais permitem expor diversos métodos privados. Na classe principal ainda é realizado um teste 'if' (linha 12), para que o **Módulo MLFV** possa diferenciar a execução dessa classe principal entre um cliente normal, que busca realizar uma requisição,

[6] https://rpyc.readthedocs.io

ou a máquina que será usada para levantar esse serviço; caso seja a própria máquina executando, o ‘if’ será verdadeiro e o servidor será levantado, importando as bibliotecas necessárias (linha 14 e 15) e criando a instância do servidor (linha 16) com os parâmetros necessários.

Como observado na Figura 9, o **Módulo MLFV** não está explicitamente implementado no código apresentado; na realidade, a implementação desse Módulo é invocado através da importação da classe MLFV_Module (linha 15). Em outras palavras, essa classe é a transposição direta do Módulo MLFV, apresentado na Figura 8 e explicado na seção 4.3, em código Python, implementando as funcionalidades descritas para criação da abordagem de MLFV.

Figura 10 – Atividades e principais funções do Módulo MLFV.



Fonte: Próprio autor.

A Figura 10, apresenta os principais métodos que compõem a classe do **Módulo MLFV**. Como observado, essa classe é composta de uma sequência de métodos para realizar o processamento de uma MLFC (‘parse_chain’), os parâmetros de cada função (‘parse_chain_obj’), preparar cada função (‘exec_chain_function’) de acordo com as restrições propostas (‘get_compatible_hosts’ e ‘get_less_busy_host’) e enviá-la para o cliente escolhido (‘send_function’). O fluxo de execução dos métodos propostos são representados pelas setas de linhas contíguas, enquanto as setas com linhas tracejadas representam a recursividade existente entre alguns dos métodos.

O primeiro método ‘parse_chain’ possui o papel de analisar a sequência de uma MLFC. Como cada MLFC é criada como um objeto do tipo *String*, esse método realiza a análise de cada caractere que compõe o objeto. Dependendo do tipo de caractere encontrado, ele executa outro

método: Caracteres do tipo Tupla representam uma sequência de funções a serem executadas de forma sequencial, sendo repassados para o método ‘parse_seq’; Caracteres do tipo Lista representam uma sequência de funções a serem executadas em paralelo, sendo repassados para o método ‘parse_par’; Caracteres que representem uma função (’f1’,’f2’), são repassados para o método ‘parse_chain_obj’.

Os métodos ‘parse_seq’ e ‘parse_par’ funcionam de forma recursiva (linhas tracejadas), analisando cada um dos caracteres de uma MLFC. Por exemplo, caso o primeiro caractere, analisado inicialmente no método ‘parse_chain’, de uma MLFC seja uma tupla, o método ‘parse_seq’ é chamado para executar a análise do próximo caractere. Se esse caractere for uma função, ele envia para o método ‘parse_chain_obj’ e espera o retorno da execução da função e executa a próxima análise; caso o próximo caractere seja uma Lista, ele chama o método ‘parse_par’ que realiza o mesmo processo, porém executando as funções dessa lista em paralelo. Todo esse processo se repete até que toda a cadeia seja analisada e as funções executadas.

O método ‘parse_chain_obj’ realiza a análise e transformação dos parâmetros de determinada função. Assim como as funções de uma MLFC são objetos *String*, os parâmetros são armazenados em forma de dicionário *String* e precisam ser organizados para construir a função e seus parâmetros.

O método ‘exec_chain_function’ recebe a função e seus parâmetros para enviar para um Cliente. Desse modo, ele invoca outros métodos para analisar o melhor Cliente possível, dos disponíveis, para executar essa função.

O método ‘get_compatible_hosts’ analisa a lista de **Clientes MLFV** disponíveis e seleciona os que são compatíveis com determinada função, de acordo com as Restrições 4.7 e 4.8 já mencionadas; clientes com potencial de processamento elevado realizam tarefas como o treinamento de modelos, enquanto clientes com menor capacidade de processamento realizam atividades que exigem menor poder computacional, como a separação, organização e limpeza de dados, por exemplo.

O método ‘get_less_busy_host’ analisa os Clientes compatíveis para verificar quais destes está menos sobrecarregado de atividades e possui uma melhor conexão, também levando em consideração o modelo matemático (Seção 4.3.1) e a Restrição 4.9.

Por fim, o método ‘send_function’ realiza o envio de uma função para o cliente escolhido com base em todas condições e restrições verificadas pelos métodos apresentados. Após a execução da função no cliente escolhido, o resultado retorna para o método ‘parse_chain’ sendo

utilizado para alimentar a próxima função na cadeia MLFC.

A Figura 11 e a Figura 12 apresentam como são implementadas as duas cadeias de ML (**TrainingChain** e **ClassificationChain**, Figura 7) propostas pela abordagem de MLFV. Essas cadeias de funções são especificadas utilizando variáveis, objetos e vetores nativos da linguagem Python, não sendo necessário instalação de bibliotecas específicas; ou seja, um usuário que deseja executar serviços de ML usando o MLFV não necessita instalar bibliotecas adicionais.

Figura 11 – Implementação da requisição de uma cadeia de funções para realizar o Treinamento de um modelo de ML.

```
1 f1="cla=mod.CreateModel(classifier,options**)"
2 f2="selected=sel.Selection(dataset,columns**)"
3 f3="preprocessed=prep.Preprocessing(selected)"
4 f4="fit=fit.Fit(cla,preprocessed)"
5 chain=([f1,(f2,f3)],f4)
```

Fonte: Próprio autor.

Figura 12 – Implementação da requisição de uma cadeia de funções para realizar a Classificação de dados usando um modelo de ML treinado.

```
1 f1="cla=mod.GetModel(classifier,columns**)"
2 f2="selected=sel.Selection(dataset,columns**)"
3 f3="preprocessed=prep.Preprocessing(selected)"
4 f4="pred=pred.Prediction(cla,preprocessed)"
5 chain=([f1,(f2,f3)],f4)
```

Fonte: Próprio autor.

O processo de implementação de um MLFC, retratado pelas Figura 11 e Figura 12, ocorre através da utilização de 4 variáveis: f1,f2,f3 e f4, que representam cada uma das funções da cadeia MLFC; e o vetor 'chain', que organiza essas funções na sequência desejada. Cada uma das variáveis que representam as funções (linha 1,2,3 e 4) declaram, no formato 'String', um código que será executado pelo **Módulo MLFV** através dos métodos de 'parse_chain' (Figura 10), para invocar os métodos de cada função (Figura 7), implementados nesse módulo; a função 'f1', por exemplo, contem a 'String' com a semântica necessária para instanciar o resultado da execução do método 'GetModel' no atributo 'cla', que irá armazenar o classificador ou modelo ML. O vetor 'chain' organiza a forma de execução das funções citadas; as funções que são executadas em série são definidas como Tuplas em Python (funções separadas por vírgula entre parênteses), enquanto as funções executadas em paralelo são especificadas como listas (funções separadas por vírgulas entre colchetes).

Para que uma requisição de Classificação ou Treinamento seja enviada e processada pelo **Módulo MLFV** é necessário, além de especificar o vetor 'chain', apresentado anteriormente, a implementação de um objeto para armazenar os dados que serão utilizados como parâmetro. Esse objeto é criado como um dicionário (dict), representado na Figura 13, para organizar e armazenar os parâmetros de cada função.

Figura 13 – Implementação e organização dos parâmetros das funções que compõem uma MLFC.

```
1  p = {}
2  columns = ['qt','fs', 'u0', 'sv0']
3  clss = 'class'
4  scaler = 'Standard'
5  p['dataset_T'] = np.asarray(pd.read(train_dataset))
6  p['dataset_C'] = np.asarray(pd.read(input_dataset))
7  p['classifier'] = 'RNA'
8  p['cla_options'] = [solver='lbfgs',activation='tanh',layers=(150,15),iter=200,random=1]
9  p['sel_options'] = [Columns = columns, Class = 'class']
10 p['pp_options'] = [Scaler = 'Standard']
```

Fonte: Próprio autor.

A variável 'p' (linha 1) é o dicionário mencionado, e é uma abreviação de *parameters*, ou parâmetros. A variável 'columns' (linha 2), é um vetor que armazena as *features* (parâmetros de entrada), definidos pelo usuário, que serão utilizados para treinar o modelo de ML. A variável 'clss' (linha 3) armazena a classe ou rótulos utilizados para treinar o modelo de ML, através de Aprendizado Supervisionado (Seção 2.2.2). Ambas as variáveis 'column' e 'clss' representam, na realidade, o nome das colunas do conjunto de dado de treinamento (linha 5) ou classificação (linha 6) que serão utilizadas, como já apresentado na Seção 2.2. O classificador ('classifier') escolhido é armazenado como uma 'String' com o nome do classificador (linha 7). As opções para ajustar o classificador são armazenados como 'cla_options' (linha 8). Na linha 9, os parâmetros da função de Seleção são armazenados como 'sel_options'. As opções (parâmetros) para a função de Pré-processamento são armazenadas como 'pp_options' (linha 10).

A Figura 14 apresenta a forma como são implementadas cada uma das funções de ML (Figura 7), que serão virtualizadas e enviadas para os clientes MLFV executarem. Elas são representadas através de classes Python e compartilham da mesma lógica de construção, possuindo objetos, atributos e métodos semelhantes.

Como é possível observar, cada classes deve possuir três características imutáveis: (i) o atributo *constr* (linha 3), que especifica os requerimentos básicos (bibliotecas e recursos com-

Figura 14 – Implementação genérica das funções propostas.

```
1 from mlfv_constraints import *
2 class Training(object):
3     constr = MLFVConstraits("numpy,sklearn.ensemble",1000,2,10)
4     def __init__(self, pars):
5         #constructor
6     def run(self):
7         #do activities
8         return results
```

Fonte: Próprio autor.

putacionais (CPU e RAM)) de execução de cada função; (ii) um método construtor (linha 4), que recebe e armazena os parâmetros de entrada em atributos específicos da classe; (iii) e o método de *run* (linha 6), que, ao ser invocado, inicia a execução das atividades específicas de determinada função, utilizando os atributos criados no construtor.

A Tabela 4 apresenta as informações e características, mantidas no Repositório de Clientes MLFV (Figura 8), a respeito de cada cliente MLFV cadastrado e disponível. A primeira coluna (ip) guarda o endereço de IP de cada cliente; a segunda (port) guarda a porta que está disponível para conexão; a terceira (libs) armazena as bibliotecas disponíveis (habilitadas); a coluna 'cpu' armazena a velocidade (em Mhz) do processador; 'mem' significa a memória RAM do cliente (em GB); 'net' é a conexão disponível entre um cliente e o serviço de MLFV (em Mb/s). Com esses dados, o orquestrador usa o modelo apresentado na Seção 4.3.1 para identificar o cliente ideal para realizar diferentes atividades de ML; clientes com baixo poder de processamento, por exemplo, realizam tarefas mais simples, como a seleção dos dados e pré-processamento; clientes com uma conexão pobre recebem funções com menos quantidade de dados para processar, como para criar ou carregar um modelo de ML.

Tabela 4 – Registro das informações dos Clientes MLFV.

ip	port	libs	cpu	mem	net
10.1.2.35	18810	os,numpy,pandas,sklearn,sklearn.preprocessing	3000	8	94
10.1.2.250	18810	os,numpy,pandas,sklearn,sklearn.preprocessing	3000	8	94
10.1.2.251	18810	os,numpy,pandas,sklearn,sklearn.preprocessing	3000	8	94
10.1.2.51	18810	os,numpy,pandas,sklearn,sklearn.preprocessing	3000	8	94

Fonte: Próprio autor.

É importante ressaltar que, apesar da informação sobre a conexão do cliente ('net') estar salva de forma estática no repositório, a plataforma de NFV realiza, em tempo real, a análise do estado da rede de cada cliente antes de enviar uma função. Por fim, para acessar o repositório de

clientes, o **Módulo MLFV** utiliza o método ‘read_hosts’ (Figura 10), já apresentado anteriormente. Por fim, após apresentar as Funções de ML, Cadeias de Funções, arquitetura de MLFV, e a implementação dos algoritmos e métodos que compõem a abordagem proposta, o próximo capítulo tem como objetivo apresentar os experimentos realizados para certificar a viabilidade da proposta deste trabalho.

5 EXPERIMENTOS E AVALIAÇÃO DOS RESULTADOS

Este capítulo apresenta o processo de avaliação da abordagem de MLFV utilizando como estudo de caso a área de Geotecnia. Diversos experimentos foram conduzidos buscando avaliar diferentes aspectos da arquitetura proposta. Os resultados, obtidos em cada uma das fases previstas, são descritos a seguir.

O capítulo introduz uma visão geral do estudo de caso, onde são apresentadas as definições do contexto de aplicação da abordagem de MLFV na área de Engenharia Civil, seguido pela descrição dos dados empregados nesta pequisa. Os experimentos utilizados para avaliar a abordagem proposta são detalhados juntamente com os passos necessários para avaliar a qualidade dos dados, algoritmos e processos empregados nessa avaliação. Em um primeiro momento, é feita uma avaliação dos dados, além da precisão dos algoritmos de ML escolhidos e a eficiência das cadeias de funções de ML (MLFC) propostas. A segunda parte busca avaliar a abordagem de MLFV de forma a verificar sua performance em comparação ao estado da arte.

5.1 ESTUDO DE CASO: GEOTECNIA E A CLASSIFICAÇÃO DOS SOLOS

A classificação de distintas camadas que compõe o solo é uma das tarefas mais importantes para garantir a segurança na construção civil. Conhecer o subsolo da região onde uma obra será executada é um pré-requisito (BHATTACHARYA; SOLOMATINE, 2006). Para isso são necessários parâmetros geotécnicos confiáveis, os quais são coletados através de investigações de campo e laboratório (FREIRE, 2016).

O ensaio de *piezocone* (CPTU) é uma das ferramentas utilizadas para realizar a coleta desses parâmetros, o qual consiste da cravação contínua no solo utilizando uma ponteira de forma cônica. Durante a penetração, as forças medidas pela ponta e pelo atrito lateral variam em função das propriedades dos materiais atravessados. Os principais registros coletados pelos sensores da ponteira cônica do ensaio de CPTU são resistência de ponta (qc), atrito lateral local (fs) e poropressão (u); a partir dessas informações, é possível determinar o comportamento de diferentes tipos de solo e classificá-los conforme a sua composição (FREIRE, 2016).

De acordo com Hegazy (HEGAZY; MAYNE, 2002), a determinação ou classificação de solos utilizando dados de *piezocone* ocorre, principalmente, através da interpretação dos parâmetros coletados aplicando uma das seguintes abordagens: (1) Análise dos dados brutos da

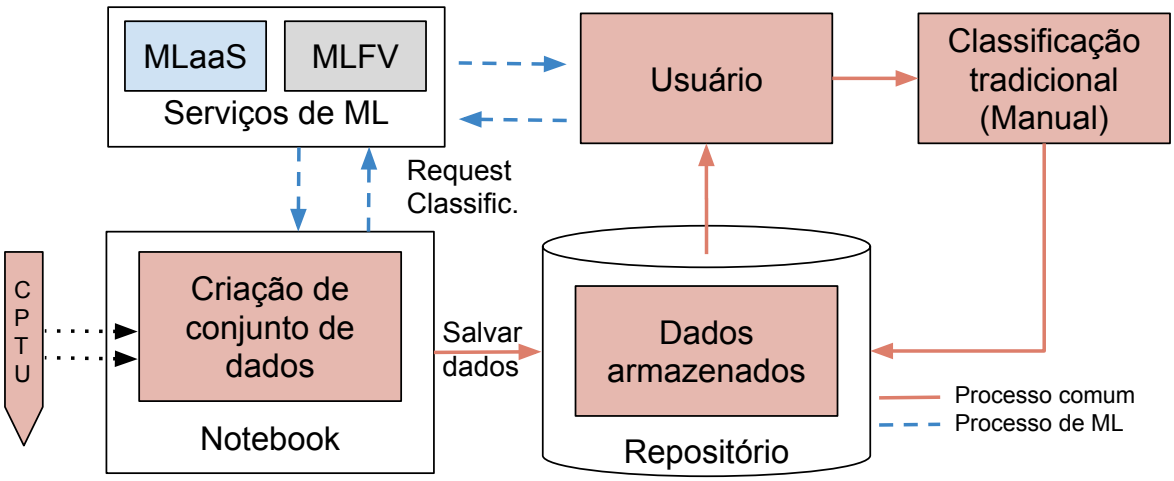
coleta, que demanda tempo e experiência do engenheiro; (2) Gráficos para classificações empíricas, nomeados ábacos para identificação do Tipo de Comportamento do Solo (*Soil Behavior Type*, SBT) (ROBERTSON; G. CAMPANELLA, 2011)(SCHNAID; ODEBRECHT, 2012); (3) Métodos estatísticos (CETIN; OZAN, 2009; HEGAZY; MAYNE, 2002; WANG; HUANG; CAO, 2013); (4) Técnicas de Mineração de Dados e algoritmos de classificação (BHATTACHARYA; SOLOMATINE, 2006; BIONDI et al., 2006; CHAGAS et al., 2007; FREIRE, 2016).

Dentre os métodos citados, as abordagens tradicionais (1 e 2) são executadas de forma manual, demandando muito tempo e experiência dos engenheiros envolvidos sendo, consequentemente, passíveis de erros e incertezas (REALE et al., 2018; BHARGAVI; SINGARAJU, 2011). Além disso, as abordagens tradicionais que utilizam ábacos (2) para classificação possuem diversas variações, criadas por diferentes autores, como as propostas por Robertson (ROBERTSON; G. CAMPANELLA, 2011), Schnaid (SCHNAID; ODEBRECHT, 2012) ou Ganju (GANJU; PREZZI; SALGADO, 2017); avaliam o solo de forma diferente, baseado na experiência de cada autor. O ideal seria realizar a interpretação utilizando o máximo de abordagens possível para decidir qual se encaixa melhor com o tipo de solo estudado, o que acaba demandando mais tempo ainda (WANG; HUANG; CAO, 2013).

Motivado pela necessidade de reduzir custos, tempo e criar formas para automatizar o processo de classificação utilizando os ábacos SBT, muitos estudos usando diferentes abordagens de ML (3) para auxiliar especialistas em Geotecnia foram conduzidos (REALE et al., 2018; BHARGAVI; SINGARAJU, 2011; BIONDI et al., 2006; BHATTACHARYA; SOLOMATINE, 2006). Em sua maioria, esses trabalhos focam na criação de um algoritmo de ML para generalizar os padrões de um ábaco, não apresentando uma solução completa na criação e implantação de serviços de ML (HEGAZY; MAYNE, 2002; FREIRE, 2016; BIONDI et al., 2006; BHATTACHARYA; SOLOMATINE, 2006); a utilização de técnicas de ML não é uma tarefa simples nem mesmo para especialistas da área de Computação (FAYYAD; PIATETSKY; SMYTH, 1996), tornando serviços de ML (MLaaS) uma opção mais viável, simples, de fácil entendimento, e aplicação para usuários comuns.

Como pode ser visto na Figura 15, o processo comum para coleta e classificação de solos é resumido em várias etapas: Os sensores de *piezocone* (CPTU) coletam os dados e enviam para um dispositivo (Notebook), que organiza e armazena os dados em um repositório. Após essa fase, o usuário realiza a classificação dos dados utilizando uma das abordagens citadas por (HEGAZY; MAYNE, 2002) e armazena os resultados novamente no repositório. Com

Figura 15 – Estudo de caso



Fonte: Próprio autor.

um Banco de Dados possuindo dados de solo rotulados, é possível iniciar a automatização da classificação tradicional por meio dos algoritmos de ML. Porém, para isso, o usuário precisa instalar, na sua máquina, as bibliotecas e pacotes de ML necessários além de realizar todos os passos necessários para configurar e treinar os modelos de ML.

Como os dispositivos que recebem as informações do *piezocone* possuem, em geral, baixo poder de processamento, além de não possuírem as bibliotecas necessárias para realizar tarefas de ML, a utilização de algoritmos de ML é viável apenas no dispositivo do usuário. Para facilitar a aplicação desses técnicas, uma opção seria a utilização de serviços de MLaaS, onde qualquer dispositivo pode enviar dados para serem processados em tempo real, sem se preocupar com poder de processamento ou bibliotecas instaladas. O usuário pode acessar esse serviço de qualquer local para customizá-lo. Além disso, o usuário pode criar um modelo de ML para cada tipo de classificação existente e realizar múltiplas predições simultâneas, sem instalar nada em sua máquina.

5.1.1 Dados

Para avaliar a proposta desse trabalho com o estudo de caso apresentado na Seção anterior (5.1) foram utilizados dados de ensaios de CPTU realizados por (BARONI, 2016), oriundos de depósitos de solos sedimentares, denominados argilas moles, na zona Oeste da cidade do Rio de Janeiro. Este tipo de solo existe em grande parte do litoral brasileiro e compõem o subsolo dos bairros da Barra da Tijuca e Recreio dos Bandeirantes, ambos localizados na zona Oeste

da cidade do Rio de Janeiro, local onde os ensaios foram realizados. Baroni (BARONI, 2016) realizou uma pesquisa bastante abrangente das obras realizadas nestes solos, consultando pesquisadores e empresas responsáveis pela realização de ensaios geotécnicos em argilas moles, buscando reunir as informações necessárias.

Neste processo, foram coletados os resultados de pesquisas de campo em 24 diferentes locais, onde foram efetuadas 67 ensaios de CPTU. Depois de compilados, todos os resultados obtidos foram organizados em planilhas com o aplicativo Microsoft Office Excel, e posteriormente em um banco de dados, da mesma forma como apresentado pela Figura 15. Tanto nas planilhas Excel ou no banco de dados criado, as colunas representam os parâmetros CPTU coletados pelos ensaios. Os parâmetros armazenados estão sempre na mesma ordem, sendo os seguintes: $Profundidade$, qt, fs, $u2$, gw, $gnat$, $u0$, $sv0$, $s1v0$. Além dos ensaios CPTU, foram realizadas análises granulométricas nos locais de coleta; essa análise informa a porcentagem de grãos de areia, argila e silte do solo, podendo ser comparado aos parâmetros de entrada do CPTU pra determinar o tipo de solo (BARONI, 2016). A classificação de solos utilizando fórmulas criadas por Robertson (ROBERTSON; G. CAMPANELLA, 2011) e (SCHNAID; ODEBRECHT, 2012) também foram realizadas. O banco de dados possui uma coluna para cada tipo de classificação, sendo a seguintes: $class\_Rob$, $class\_Schn$.

Os 67 ensaios disponíveis representam um total de cerca de 51.000 linhas com medidas de CPTU e os seus respectivos rótulos de tipo de solo de Robertson ($class\_Rob$) e Schnaid ($class\_Schn$). A próxima seção detalha o processamento desses dados por cada uma das função propostas pelo MLFV (Figura 6), utilizando os parâmetros e rótulos de classes apresentados para avaliar o seu desempenho.

5.2 AVALIAÇÃO DAS FUNÇÕES DE ML PROPOSTAS

Nesta seção será apresentado o funcionamento de cada uma das funções de ML que compõem as cadeias implementadas para avaliar a proposta (Figura 11 e Figura 12). O objetivo é avaliar cada uma dessas funções separadamente, buscando comprovar sua eficácia para então realizar a avaliação final da abordagem de MLFV e responder a pergunta de pesquisa proposta neste trabalho.

Selection. A função *Selection*, como apresentada na Seção 4.1.1, deve receber como parâmetros de entrada um ou mais conjuntos de dados (*dataset*) e outros argumentos com as opções de ajuste. Neste sentido, para avaliar o funcionamento dessa função os seguintes parâ-

metros foram informados:

- *datasets*: Dos 24 locais de ensaio descritos na Seção 5.1.1, foram selecionados randomicamente 4 amostras de dados (conjunto de dados) (Tabela 5); A soma dos arquivos de cada um dos conjuntos de dados apresentados totaliza em 1.8 MB de dados;
- *Columns*: Os parâmetros de coleta CPTU escolhidos: 'qt', 'fs', 'u2' e 'u0'. A escolha desses parâmetros se baseia no conhecimento de especialistas das área e trabalhos relacionados que utilizaram essas mesmas informações (FREIRE, 2016; ROBERTSON; G. CAMPANELLA, 2011; SCHNAID; ODEBRECHT, 2012; BIONDI et al., 2006);
- *Class*: A classe 'Class_Rob', que foi escolhida randomicamente entre as duas opções de rótulos disponíveis e descritos na Seção 5.1.1.

A primeira atividade executada pela função de *Selection* é o agrupamento dos conjuntos de dados, criando um conjunto unificado (Seção 2.1.1) chamado de '*Selection*' na Tabela 5. Ele armazena todas as informações dos outros conjuntos de dados apresentados na Tabela 5, totalizando 9340 linhas (*rows*) de dados. O conjunto 'Resultante' gerado pelo agrupamento possui um tamanho estimado de 1,8 MB.

Tabela 5 – Agrupamento dos locais de ensaio selecionados.

Conjunto de dados	Quantidade de dados
CM1	1588
GlebaF(15)	867
Rio Mais(21)	3815
Solleto (SOP3)	3070
Selection	9340

Fonte: Próprio autor.

Após essa atividade, a função *Selection* realiza o processo de redução de dimensionalidade (Seção 2.1.2), selecionado apenas as colunas relevantes. Como observado na Tabela 6, o conjunto de dados original possui doze (12) colunas das quais apenas cinco (5), como informado através dos parâmetros *Columns* e *Class*, serão utilizadas.

O resultado esperado pelo processo de seleção é apresentado na Tabela 7. Além de remover as colunas indesejadas, esse processo altera o nome da classe escolhida para 'class' e move essa coluna para a primeira posição, facilitando a manipulação dos dados pelos algoritmos de ML. A redução do número de colunas diminuiu a quantidade de dados desnecessários

Tabela 6 – Cabeçalho dos conjuntos de dados originais.

Prof.	qt	fs	u2	gw	gnat	u0	sv0	s1v0	class_Rob	class_Schn
0,7	29,4	1,4	12,8	9,8	12,2	4,1	8,6	4,6	1	2

Fonte: Próprio autor.

armazenado e, dessa forma, o tamanho do conjunto de dados 'Resultante' original (1,8 MB) também foi reduzido, para 1,2 MB. A redução do tamanho do arquivo diminui o tempo de transferência de dados e execução das atividades em geral.

Tabela 7 – Cabeçalho do conjunto de dados resultante.

class	qt	fs	u2	u0
1	29,4	1,4	12,8	4,1

Fonte: Próprio autor.

Preprocessing. A função *Preprocessing*, apresentada na Seção 4.1.1, deve receber como parâmetros de entrada um conjunto de dados e os argumentos com opções para o ajuste específico dessa função. Neste sentido, para avaliar o seu funcionamento os seguintes parâmetros foram informados:

- *dataset*: Conjunto de dados resultante da função *Selection* ('Selection' na Figura 5);
- *scaler*: Algoritmo para transformação de dados *Standard Scaler*.

A primeira atividade executada pela função de *Preprocessing* foi remover as inconsistências existentes no *dataset*, como apresentado na Tabela 8.

Tabela 8 – Processo de limpeza dos dados.

Original		*Preprocessing*				Resultado
Conjunto de dados	Total	Negativo	Nulo	Duplicado	Outlier	Total
dataset	9340	725	527	173	326	7589
	100%	7,76%	5,64%	1,85%	3,49%	81,25%

Fonte: Próprio autor.

Das 9340 linhas do *dataset* apresentado, a função *Preprocessing* identificou e removeu: 725 (7,76% do total) linhas com valores negativos; 527 (5,64%) linhas com valores nulos; 173 (1,85%) linhas com valores duplicados; 326 (0,62%) de linhas com *outliers*. Como resultado, o *dataset* passou de 9340 linhas de dados para 7589 linhas (81,25% do valor orginal). O tamanho em MB do arquivo orginal diminuiu, passando de 1,2 MB para 1MB após a limpeza. Após esse processo, a função realizou a transformação das colunas (Tabela 7) do *dataset*.

***Fit* e *Prediction*.** As função *Fit* e *Prediction*, como apresentado na Seção 4.1, recebem como parâmetros de entrada um modelo de ML e um conjunto de dados para ajustar um modelo de ML, através de aprendizado supervisionado, ou realizar a predição de dados. Para avaliar o funcionamento dessas duas funções, as funções anteriores (*CreateModel* e *GetModel*) foram necessárias. Os seguintes parâmetros foram utilizados:

- *classifier*: Dois classificadores foram utilizados para realizar o processo de classificação de dados de solo: (i) algoritmo de ANN, escolhido com base no trabalho de (BIONDI et al., 2006) que utilizou esse modelo para automatizar a classificação de solos; (ii) algoritmo de RF, escolhido aleatoriamente dentre outras opções existentes (Seção 2.2).

- *options***: As opções de configuração dos classificadores variaram de acordo com cada um deles: (i) ANN foi configurado da mesma forma que BIONDI et al. (2006) propôs, utilizando 2 camadas ocultas; a primeira camada com 14 neurônios e a segunda com 120 neurônios. O método de *Levenberg-Marquardt* (BIONDI et al., 2006) foi utilizado para treinar esse classificador; (ii) Para configurar o algoritmo de RF, foi utilizado o algoritmo *Grid Search* (PEDREGOSA et al., 2011). Os parâmetros de entrada e saída utilizados são advindos do processo de *Selection* e *Preprocessing*.

- *model*: Os dois modelos (ANN e RF) com as configurações informadas.

- *trainset*: Conjunto de dados de treinamento fornecidos pela função de *Preprocessing*.

Tabela 9 – Dados e tipos de solos utilizados pela função *Fit*.

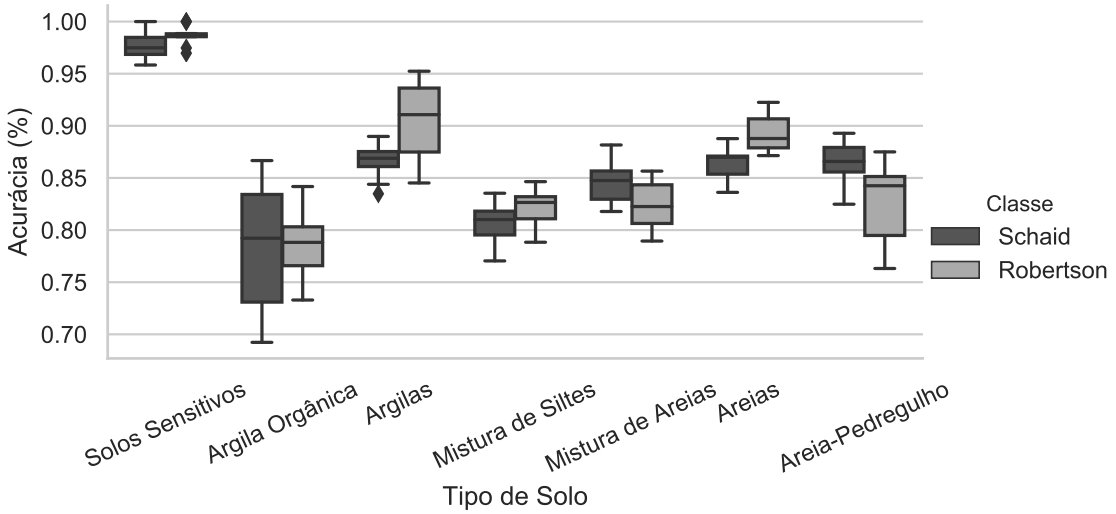
Classes de tipos de solo	Quantidade dados
Solos Sensitivos	808
Argila Orgânica	452
Argilas	1750
Mistura de Siltes	1099
Mistura de Areias	1285
Areias	1755
Areia-Pedregulho	440
Total	7589

Fonte: Próprio autor.

Para avaliar o funcionamento de cada função com dados de solo, foram utilizadas duas classes (rótulos) nos conjuntos de treinamento (*trainset*): $class\_Rob$ e $class\_Schn$, apresentadas ao final da Seção 5.1.1. Além disso, foi utilizado o algoritmo de *Cross Validation* para avaliar o desempenho de cada um dos modelos de ML (ANN e RF) com essas duas classes de solo.

A Figura 16 apresenta os resultados da função *Fit* durante o treinamento utilizando o algoritmo de ANN e *Cross Validation*. O algoritmo de *Cross Validation* criou 10 novos conjuntos de dados diferentes (*10-fold Cross Validation*) com base no conjunto de dados total (Tabela 9); cada um desses conjuntos utilizou diferentes fatias do conjunto total para realizar o treinamento (60%), teste e validação (40%) dos algoritmos de ML com os dados de solo. Desse modo, a variação de acurácia apresentada na Figura 16 consiste no resultado de acurácia de cada um dos 10 conjuntos criados.

Figura 16 – Resultado da função *fit* utilizando *10-fold Cross-Validation*



Fonte: Próprio autor.

Como observado, a maior variação de acurácia ocorre na classe de Argila Orgânica utilizando o rótulo de $Schnaid$. Dos dados de solos utilizados, essa classe possui uma das menores quantidade de valores disponíveis comparado com as outras, como pode ser visto na Tabela 9; isso pode justificar essa variação e também uma menor porcentagem de acurácia. Em geral, quanto maior a quantidade de dados disponível para cada classe maior a probabilidade de se encontrar um padrão a ser extraído (Seção 2.2). Para as outras classes, a variação na acurácia, em geral, não se mostrou tão grande demonstrando a eficácia das funções propostas no contexto de classificação de solos e a qualidade dos dados utilizados.

Já os resultados apresentados pela função *Predict* são apresentados na Tabela 10. Foram utilizados 3 conjuntos de dados diferentes dos disponibilizados para o treinamento dos modelos de ML; eles foram escolhidos randomicamente dos 24 disponíveis, excluindo os 4 já utilizados para treinamento (Tabela 5). Como pode ser observado na Tabela 10, em geral, o algoritmo de RF teve uma pequena vantagem de acurácia em comparação ao algoritmo de ANN; nas classes com menor quantidade de dados essa vantagem se mostrou superior. No total, ambos algoritmos atingiram uma acurácia média entre 85% e 89% neste teste de estresse, demonstrando a eficácia das funções e cadeias MLFC mesmo utilizando uma quantidade de dados razoável, possuindo classes desbalanceadas, durante o treinamento. Por fim, após a avaliação de todas as funções propostas nesta pesquisa, a próxima Seção busca avaliar a abordagem de MLFV.

Tabela 10 – Resultado da Predição

Classes de tipos de solo	% de acurácia predição			
	Robertson		Schnaid	
	ANN	RF	ANN	RF
Solos Sensitivos	0,98	0,95	0,97	0,95
Argila Orgânica	0,81	0,92	0,78	0,86
Argilas	0,90	0,89	0,86	0,88
Mistura de Siltes	0,82	0,82	0,80	0,85
Mistura de Areias	0,82	0,82	0,84	0,88
Areias	0,89	0,89	0,86	0,90
Areia-Pedregulho	0,81	0,92	0,86	0,94
Total	0,86	0,89	0,85	0,89

Fonte: Próprio autor.

5.3 AVALIAÇÃO DA ABORDAGEM MLFV

Neste seção será apresentado o funcionamento e avaliação da abordagem de MLFV. O objetivo é verificar a performance da abordagem de MLFV em comparação a outras 3 plataformas semelhantes: Microsoft AzureMLaaS Studio, Dask.distributed e um ambiente local, utilizando apenas um PC para executar as funções de ML propostas. Três diferentes experimentos foram conduzidos, sendo realizados em dois ambientes de teste diferentes: (i) o primeiro *testbed* foi conduzido no laboratório do GMOB na UFSM; (ii) o segundo ocorreu em um ambiente OpenStack com várias máquinas virtuais disponíveis. A seguir são descritos os ambientes de teste citados e os resultados alcançados em cada um dos três experimentos.

Ambiente de Teste (a), Laboratório UFSM. Os testes no laboratório da UFSM utili-

zaram cinco (5) computadores iMac 12.1, com processadores Intel quad-core 2.5Ghz, 12GB RAM, 500GB HD, com sistema operacional MacOS Yosemite (v10.10.5), todos conectados em uma mesma rede *Ethernet* de 100Mb. Nesse teste o MLFV não recebeu informações sobre a rede para realizar suas atividades. O primeiro e segundo experimentos, reportados abaixo, foram executados na UFSM.

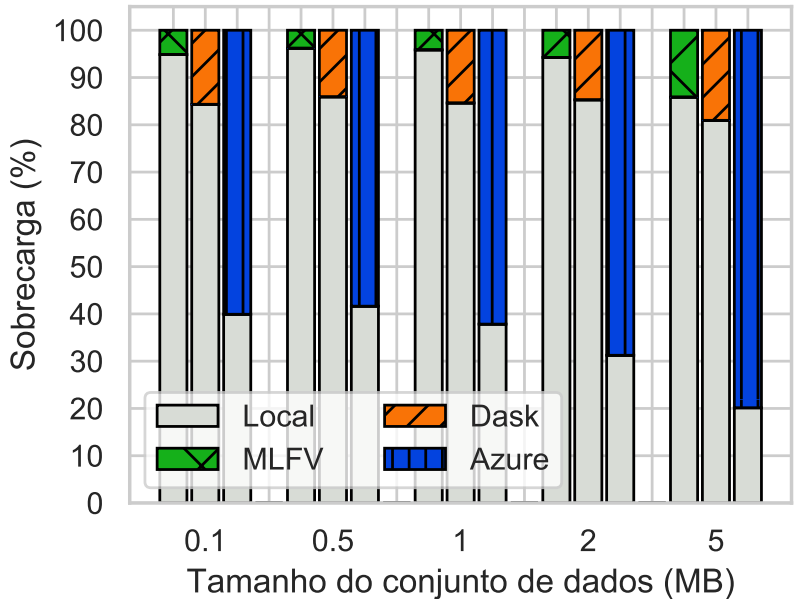
Ambiente de Teste (b), OpenStack. Orquestrado pelo OpenStack Rocky, contendo oito máquinas virtuais com 3Ghz vCPU, 1 GB de RAM, 50 GB de HD, rodando o Ubuntu 16.04.3. Neste *testebed*, o MLFV recebeu as informações da rede através de um módulo *Ceilometer*[7], mais especificamente, através da leitura dos parâmetros de **network.incoming.bytes.rate** e **network.outcoming.bytes.rate**. O terceiro experimento foi executado neste ambiente de testes.

Resultados dos testes e experimentos. Como já mencionado, foram realizados três experimentos em dois ambientes de testes (*testbeds*), comparando a proposta de MLFV com outras três diferentes abordagens semelhantes. A primeira dessas abordagens é o Microsoft Azure MLaaS Studio[8], o qual utiliza a nuvem do Azure para executar a criação, treinamento e implantação de modelos de ML. A segunda é a biblioteca Python Dask.distributed (v1.26.1)[9], que apresenta uma abordagem semelhante à nossa onde um servidor central, chamado de **dask-scheduler**, coordena as ações de diversos trabalhadores (escravos), chamados de **dask-workers**, que, espalhados por vários computadores, realizam as tarefas que compõem o processo de ML. Por fim, a terceira abordagem se refere a funções de ML criadas localmente, à serem executadas em apenas um cliente (PC), utilizando bibliotecas padrões de Python, como Scikit-Learn, Pandas e etc. Todos os experimentos relatados nesta seção foram executados 30 vezes, onde os gráficos, para cada experimento, apresentam a média e o desvio padrão desses valores.

O primeiro dos três experimentos (Figura 17) apresenta a diferença do tempo de execução (sobrecarga) das abordagens de MLFV, Microsft Azure e Dask em comparação com um computador utilizando as funções e bibliotecas de ML em um ambiente local (reportado, nessa seção, como **Local**). Como todas as abordagens testadas, excluindo a **Local**, utilizam a rede para transmitir todos os seus parâmetros (informações, chamadas, conjunto de dados) para realizar a tarefa de classificação, esse primeiro experimento tem como objetivo avaliar o impacto dessa transmissão no tempo de execução, das funções de ML, de cada abordagem.

[7] https://docs.openstack.org/ceilometer
[8] https://azure.microsoft.com/en-us/services/machine-learning-studio
[9] http://distributed.dask.org

Figura 17 – Sobrecarga imposta pelas abordagens em comparação com um ambiente Local.



Fonte: Próprio autor.

A Figura 17 apresenta os resultados alcançados pelo primeiro experimento, onde o eixo Y mostra a sobrecarga imposta pelas abordagens de MLFV, Dask e Microsoft Azure em comparação a abordagem **Local** para realizar a cadeia de classificação descrita, previamente, pela Figura 7. Já o eixo X determina o tamanho dos conjuntos de dados, utilizados para treinar o modelo de ML, variando nos seguintes tamanhos: 0.1MB, 0.5MB, 1MB, 2MB, 5MB.

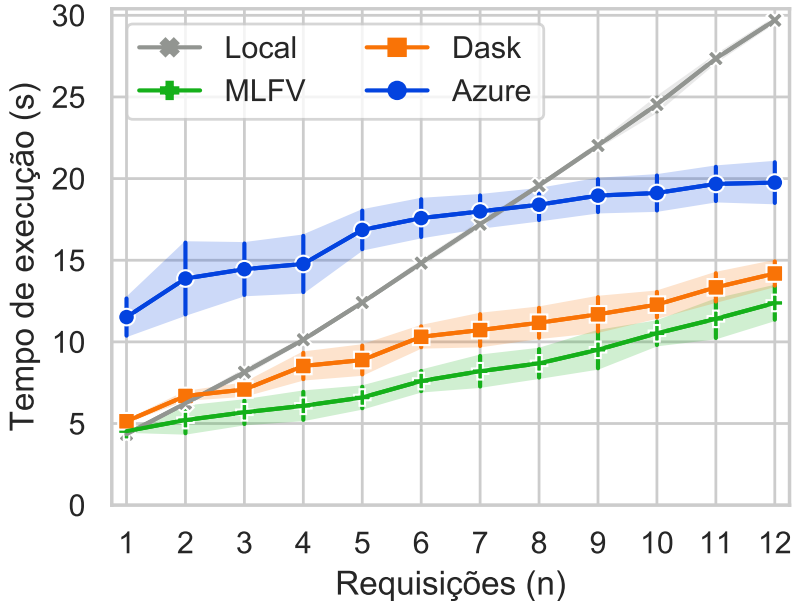
O tempo médio para finalizar a execução completa da cadeia proposta, utilizando os diferentes conjuntos de dados citados, no ambiente **Local** foi de 4.27, 4.34, 4.46, 4.50, e 4.69 segundos, respectivamente. A plataforma Azure impôs a maior sobrecarga (*overhead*) (de 58% até 84%), devido a necessidade de envio dos conjuntos de dados até a nuvem, pela Internet; além disso, essa plataforma apresentou um atraso de execução próximo a 8 segundos entre a conclusão do envio do conjunto de dados e o início do seu processamento.

A menor diferença de sobrecarga foi observada pelo MLFV utilizando os menores conjuntos de dados (0.5MB - 3.8%, 1MB - 4.1%, e 2MB - 5.7%); contudo, a medida que o tamanho aumentou, essa sobrecarga cresceu consideravelmente. Isso ocorre pelo fato de que o **MLFV Manager** envia uma função com todos seus parâmetros para um **Cliente MLFV** e espera seu retorno (mais parâmetros e dados trafegando na rede). Após receber essas informações, ele repete o processo de envio de dados para um próximo **Cliente MLFV** que irá executar a próxima função existente na cadeia. Diferente da aborgagem de MLFV, a plataforma Dask identifica quando uma função depende do resultado de outra e envia elas para um mesmo **dask-worker**

(Cliente) com todos os parâmetros e dados de uma só vez, reduzindo o tráfego na rede e o tempo de execução quando os conjuntos de dados são maiores.

O segundo experimento buscou analisar como as diferentes abordagens (**Local**, MLFV, Dask, e Azure) lidam com múltiplas requisições simultâneas de cadeias de classificação. Para esse teste, o tamanho do conjunto de dados foi fixado em 1MB e o número das requisições simultâneas alternando entre 1 até 12 chamadas (Eixo x - Figura 18). Por fim, foi medido o tempo de execução (Eixo y - Figura 18) de cada abordagem para realizar a cadeia de classificação completa, tendo os resultados apresentados pela Figura 18.

Figura 18 – Múltiplas requisições simultâneas.



Fonte: Próprio autor.

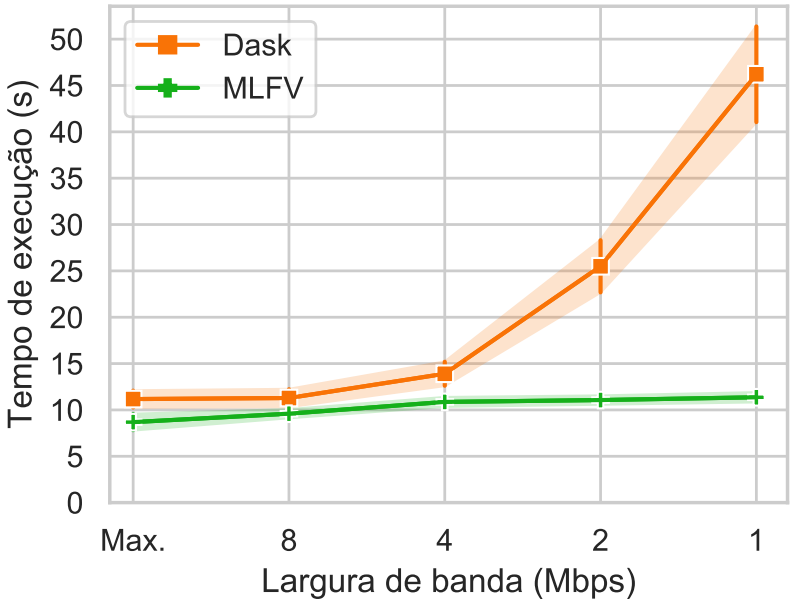
Como a cadeia de classificação especifica duas operações em paralelo, o ambiente de teste **Local** (Processador com 4 cores) apresenta uma piora no tempo de execução à partir de duas solicitações simultâneas que, como esperado, tende a crescer bastante à medida que o número de requisições aumenta. Para as outras três abordagens, o tempo de execução não aumenta tão abruptamente, tendo em vista que as execuções concorrentes são distribuídas entre vários *Hosts* (**Clientes** ou **workers**) com várias CPUs disponíveis para processamento.

A abordagem de MLFV obteve, em média, uma vantagem de 2 segundos (25%) em comparação ao Dask. Também pode ser observado que, embora o tempo para o Azure concluir a execução seja consideravelmente maior, sua curva de crescimento é menos pronunciada do que a Dask e MLFV, indicando que, caso não sejam adicionados mais CPUs (**Clientes** ou **workers**), em algum momento o seu tempo de execução será menor que ambos Dask e MLFV.

O terceiro experimento busca analisar como uma conexão de rede ruim (baixa qualidade, sinal fraco, baixo limite de banda), pode afetar negativamente o tempo de execução das abordagens de ML na borda. Para isso, foi criado um cenário simulado onde a conexão entre alguns dos **Clientes** foi limitada, variando entre 1,2,4,8 e Max (Nenhum limite de banda). Além disso, o número de requisições de cadeias de classificação simultâneas foi fixado em 8 chamadas, utilizando um conjunto de dados com 1MB de tamanho total.

Os resultados desse experimento, apresentados pela Figura 19, demonstram um aumento considerável no tempo de execução da plataforma Dask a medida que a qualidade da conexão entre os **Clientes** diminui, enquanto o MLFV mantém um tempo de execução estável. Isso ocorre porque o **dask-scheduler** não é capaz de identificar que alguns dos seus **dask-workers** possuem uma conexão limitada e continua mandando funções para eles, aumentando consideravelmente o tempo total de execução. Por outro lado, o **MLFV Manager** recebe informações do módulo de Telemetria OpenStack sobre a qualidade das conexões entre os **Clientes MLFV** e, dessa forma, identifica qualquer tipo de problema evitando o envio de uma função que necessite de alta largura de banda para um **Cliente** sobrecarregado ou com conexão limitada. No pior dos cenários, o Dask aumentou o tempo de execução acima de 400%.

Figura 19 – Simulação com largura de banda restrita.



Fonte: Próprio autor.

Por fim, como ultima avaliação, foram desinstaladas algumas das bibliotecas necessárias para execução da funções de ML de alguns dos clientes MLFV e **dask-workers** para verificar como ambas plataformas lidam com **Clientes** desatualizados. Neste sentido, foi constatado que

o Dask não verifica se as instância dos seus **dask-workers** possuem as bibliotecas necessárias e, neste experimento, concluiu a execução prematuramente levantando uma exceção Python **ImportError**. Como a nossa abordagem verifica as bibliotecas necessárias, os clientes desatualizados foram descartados pelo **MLFV Manager** e a execução da cadeia de funções foi concluída com sucesso nos **Clientes** habilitados.

Este capítulo apresentou os principais resultados alcançados pela abordagem de MLFV. Os experimentos conduzidos utilizaram um estudo de caso na área de geotecnia para comparar a abordagem proposta com outras plataformas semelhantes. No entanto, a aplicação da abordagem de MLFV não se limita a essa área de estudo; poderiam ter sido utilizados dados em outro contexto. Por fim, os resultados apresentados neste capítulo respondem a pergunta de pesquisa, comprovando que é possível otimizar a distribuição e virtualização de funções de ML, levando em consideração diversas restrições, utilizando a abordagem de MLFV.

6 CONCLUSÃO

O presente trabalho teve como objetivo o desenvolvimento de uma abordagem para virtualização e orquestração de cadeias de funções de ML. Essa abordagem, chamada de MLFV, explora um ambiente de NFV para criação de uma plataforma proeminente na implantação de cadeias de funções de ML. O MLFV é capaz de realizar a orquestração e virtualização de cadeias de funções levando em consideração diversas restrições, como o estado da rede, requisitos computacionais e bibliotecas de ML disponíveis nos dispositivos. Um modelo matemático foi desenvolvido para minimizar o tempo de execução dessas cadeias de funções com base nessas restrições.

Para realizar a definição e desenvolvimento dessa abordagem, primeiramente foi realizado um estudo sobre os seguintes tópicos: Descoberta de Conhecimento em Banco de Dados (KDD), Aprendizado de Máquina, Aprendizado de Máquina como Serviço (MLaaS), Aprendizado de Máquina na Borda da Rede, Virtualização de Funções de Rede (NFV) e Cadeias de Funções de Rede (NFC). Posteriormente, foram apresentados os trabalhos, encontrados na literatura, relacionados aos tópicos utilizados para definição da presente proposta. Como no atual estado da arte não são encontrados trabalhos que abordem todos os tópicos citados, foram apresentados os trabalhos que apresentaram maior semelhança com a proposta desta pesquisa.

Para avaliar a proposta de MLFV foi utilizado um estudo de caso na área de Geotecnia. A abordagem de MLFV utilizou dados de coletas CPTu para realizar a tarefa de classificação de solos através das cadeias de funções de ML propostas. Foram conduzidos dois experimentos diferentes, utilizando dados de solos fornecidos: (1) avaliar o funcionamento das funções e cadeias de funções propostas; (2) avaliar a execução da abordagem de MLFV em diferentes cenários, comparando os resultados com plataformas similares. As plataformas utilizadas nesse segundo experimento foram a plataforma de MLaaS Microsoft Azure, hospedada na nuvem; a biblioteca Dask.distributed, que realiza o processo de ML de forma distribuída na borda da rede; e em um cliente local (PC) utilizando funções de ML, chamado de **Local**.

Os resultados apresentados durante a avaliação das cadeias de funções demonstraram a eficiência do processo proposto. Cada função funcionou como previsto, alcançando o resultado esperado. Como resultado final, a acurácia média entre os dois algoritmos utilizados para classificação (RF e ANN) ficou entre 85% e 89% durante esse teste.

O segundo experimento consistiu de três testes, executados em dois ambientes de teste.

Os resultados durante a execução de uma única (tarefa simples) cadeia de classificação, demonstraram que o MLFV obteve melhores resultados dentre as plataformas analisadas. Para múltiplas requisições simultâneas de cadeias de ML, o MLFV reduziu o tempo de execução em aproximadamente 25%, superando todas as outras plataformas. No último teste, quando as conexões entre os clientes possuíam algum tipo de limite de banda, a abordagem Dask teve um aumento no tempo de execução de até 400%. Em contraste, o MLFV foi capaz de detectar as restrições de banda e conexão, evitando enviar funções para os clientes sobrecarregados resultando no menor tempo de execução em comparação ao Dask.

Neste sentido, acredita-se que os experimentos apresentados respondam positivamente a questão de pesquisa apresentada junto aos objetivos deste trabalho. Os resultados demonstram a viabilidade da utilização de plataformas de NFV na implementação de serviços de ML virtualizados, principalmente pela sua capacidade de orquestração de atividades estando ciente do estado da rede.

A principal contribuição deste trabalho foi a disponibilização da abordagem desenvolvida, validada através do estudo de caso apresentado, que pode ser aplicada em diferentes domínios. O MLFV permite a criação e implementação de diferentes cadeias de funções para disponibilização de serviços completos de ML. Além disso, é capaz de realizar a distribuição eficiente e virtualização das atividades relacionadas ao processo de ML, levando em consideração diversas restrições. Ademais, este trabalho tem como contribuição a disponibilização do código desenvolvido para criação das cadeias de funções e da abordagem de MLFV. Isso permite que as funções e cadeias disponibilizadas possam ser aprimoradas ou, até mesmo, que novas possam ser desenvolvidas.

6.1 TRABALHOS FUTUROS

Como oportunidade para trabalhos futuros, sugere-se investigar diferentes maneiras para colocação de funções que estão agrupadas sequencialmente. Atualmente, o orquestrador MLFV não verifica a dependência entre funções sequenciais para realizar a sua distribuição. Com isso, funções como a **Selection** e **Preprocessing**, por exemplo, podem ser alocadas diferentes clientes; quando a função de **Selection** termina seu processamento, ela retorna os dados para o **Gerenciador MLFV** que, por sua vez, envia esses dados pela rede para o próximo **Cliente** que irá executar a função **Preprocessing**. Alocar essas funções (**Selection** e **Preprocessing**) em um mesmo cliente elimina a necessidade de retornar os valores para o Gerenciador MLFV,

reduzindo o tempo de execução e a quantidade dados trafegados na rede. Também pretende-se implementar um sistema de cache para armazenar o retorno de funções. Um exemplo disso pode ser o armazenamento do resultado de treinamento de um modelo de ML; a medida que novas requisições dependentes desse modelo são recebidas, o sistema simplesmente busca ele na cache ao invés de realizar todo o treinamento novamente. Questões relacionadas à segurança de uma forma geral também são oportunidades de trabalhos futuros. Por fim, pode-se aplicar heurísticas no modelo matemático proposto, visando otimizar a função que minimiza o tempo de execução de todas as funções de uma cadeia.

6.2 PUBLICAÇÕES

Souza, Renan L.; Trois, Celio; Turchetti, Rogério; Martinello, Magnos; Correa, João H. G.; Mafioletti, Diego R.; Bona, Luis C. E.; Lima, João C. D. e Machado, Alencar. (2019) *MLFV: Network-aware Orchestration for Placing Chains of Virtualized Machine Learning Functions*. 2019 IEEE Global Communications Conference (GLOBECOM). (ACEITO - Qualis: A1).

REFERÊNCIAS

ABU-MOSTAFA, Y. S.; MAGDON-ISMAIL, M.; LIN, H.-T. **Learning From Data**. [S.l.]: AMLBook, 2012.

AUGSPURGER, T.; KOPPULA, R.; ROCKLIN, M. Scalable Machine Learning with Dask. In: AnacondaCON. **Anais...** [S.l.: s.n.], 2019. p.1–8.

BARONI, M. **Comportamento geotécnico de argilas extremamente moles da baixada de Jacarepaguá, RJ**. 2016. Tese (Doutorado em Ciência da Computação) — UFRJ/ COPPE/ Programa de Engenharia Civil, Rio de Janeiro.

BENKACEM, I. et al. Optimal VNFs placement in CDN slicing over multi-cloud environment. **IEEE Journal on Selected Areas in Communications**, [S.l.], v.36, n.3, p.616–627, 2018.

BHARGAVI, P.; SINGARAJU, J. Soil Classification Using Data Mining Techniques: a comparative study. **International Journal of Engineering trends and Technology(IJETT)**, [S.l.], v.2, n.1, p.55–59, 2011.

BHATTACHARYA, B.; SOLOMATINE, D. Machine learning in soil classification. **Neural Networks**, [S.l.], v.19, p.186–195, 2006.

Bierzynski, K.; Escobar, A.; Eberl, M. Cloud, fog and edge: cooperation for the future? In: SECOND INTERNATIONAL CONFERENCE ON FOG AND MOBILE EDGE COMPUTING (FMEC), 2017. **Anais...** [S.l.: s.n.], 2017. p.62–67.

BIONDI, L. et al. Classificação de solos usando-se redes neurais artificiais. **Engevista (UFF)**, [S.l.], v.8, n.1, p.37–48, 2006.

Bitye Dimithe, C. O.; Reid, C.; Samata, B. Offboard Machine Learning Through Edge Computing for Robotic Applications. In: SOUTHEASTCON 2018. **Anais...** [S.l.: s.n.], 2018. p.1–7.

BRAGA, A. d. P.; CARVALHO, A. P.; LUDERMIR, T. B. **Fundamentos de Redes Neurais Artificiais**. Rio de Janeiro, RJ: Editora Rio, 1998.

CASTANHO, M. et al. PhantomSFC: a fully virtualized and agnostic service function chaining architecture. In: IEEE SYMPOSIUM ON COMPUTERS AND COMMUNICATIONS (ISCC), 2018. **Anais...** [S.l.: s.n.], 2018. p.354–359.

CETIN, K. O.; OZAN, C. CPT-Based Probabilistic Soil Characterization and Classification. **Journal of Geotechnical and Geoenvironmental Engineering**, [S.l.], v.135, n.1, p.84–107, 2009.

CHAGAS, C. S. et al. Utilização de redes neurais artificiais para predição de classes de solo em uma bacia hidrográfica no Domínio de Mar de Morros. **XIII Simpósio Brasileiro de Sensoriamento Remoto**, [S.l.], p.2421–2428, 2007.

CHAN, S. et al. PredictionIO: a distributed machine learning server for practical software development. In: ACM CIKM, 22. **Proceedings...** [S.l.: s.n.], 2013. p.2493–2496.

DOMINICINI, C. et al. VirtPhy: a fully programmable infrastructure for efficient nfv in small data centers. In: IEEE CONFERENCE ON NETWORK FUNCTION VIRTUALIZATION AND SOFTWARE DEFINED NETWORKS (NFV-SDN). **Anais...** [S.l.: s.n.], 2016. p.81–86.

DOMINICINI, C. K. et al. VirtPhy: fully programmable nfv orchestration architecture for edge data centers. **IEEE Transactions on Network and Service Management**, [S.l.], v.14, n.4, p.817–830, 2017.

Duan, Q.; Ansari, N.; Toy, M. Software-defined network virtualization: an architectural framework for integrating sdn and nfv for service provisioning in future networks. **IEEE Network**, [S.l.], v.30, n.5, p.10–16, Sep. 2016.

FACELI, K.; LORENA, A. C.; CARVALHO, A. C. P. L. F. **Inteligência Artificial**: uma abordagem de aprendizagem de máquina. Rio de Janeiro, RJ: Editora LTC, 2011.

FAYYAD, U.; PIATETSKY, G.; SMYTH, P. From Data Mining to Knowledge Discovery in Databases. **AI Magazine**, [S.l.], v.17, n.3, p.91–96, 1996.

FREIRE, F. C. **Análise das propriedades geotécnicas do solo mole do cluster/ Suape-PE**. 2016. Tese (Doutorado em Ciência da Computação) — Universidade Federal de Pernambuco. CTG. Programa de Pós-Graduação em Engenharia Civil.

GANJU, E.; PREZZI, M.; SALGADO, R. Algorithm for generation of stratigraphic profiles using cone penetration test data. **Computers and Geotechnics**, [S.l.], v.90, p.73–84, 2017.

GOKHALE, R.; M, M. Scheduling identical parallel machines with machine eligibility restrictions to minimize total weighted flowtime in automobile gear manufacturing. **The International Journal of Advanced Manufacturing Technology**, [S.l.], v.60, 06 2011.

HAN, J.; KAMBER, M. **Data Mining**: concepts and techniques. 500 Sansome Street, Suite 400, San Francisco, CA: Morgan Kaufmann Publishers, 2006.

HARTMANN, M. **Energy Efficient Machine Learning-Based Classification of ECG Heartbeat Types**. 2018. Tese (Doutorado em Ciência da Computação) — UNIVERSITY OF OKLAHOMA, Oklahoma.

HAWILO, H.; JAMMAL, M.; SHAMI, A. Network Function Virtualization-Aware Orchestrator for Service Function Chaining Placement in the Cloud. **IEEE Journal on Selected Areas in Communications**, [S.l.], 2019.

HEBB, D. O. **The organization of behavior**: a neuropsychological theory. Wiley, NJ: [s.n.], 1949.

HEGAZY, Y. A.; MAYNE, P. W. Objective Site Characterization Using Clustering of Piezocone Data. **Journal of Geotechnical and Geoenvironmental Engineering**, [S.l.], v.128, n.12, p.986–996, 2002.

HUNT, T. et al. Chiron: privacy-preserving machine learning as a service. **arXiv preprint arXiv:1803.05961**, [S.l.], 2018.

MA, W.; MEDINA, C.; PAN, D. Traffic-aware placement of NFV middleboxes. In: IEEE GLOBAL COMMUNICATIONS CONFERENCE (GLOBECOM), 2015. **Anais...** [S.l.: s.n.], 2015. p.1–6.

Matias, J. et al. Toward an SDN-enabled NFV architecture. **IEEE Communications Magazine**, [S.l.], v.53, n.4, p.187–193, April 2015.

MCCULLOCH, W. S.; PITTS, W. Neurocomputing: foundations of research. In: ANDERSON, J. A.; ROSENFELD, E. (Ed.). . Cambridge, MA, USA: MIT Press, 1988. p.15–27.

Medhat, A. M. et al. Service Function Chaining in Next Generation Networks: state of the art and research challenges. **IEEE Communications Magazine**, [S.l.], v.55, n.2, p.216–223, February 2017.

MICROSOFT. **Serviço do Azure - Machine Learning**. Acessado em: 2019-01-22, https://azure.microsoft.com/pt-br/services/machine-learning-service/.

MIJUMBI, R. et al. Network Function Virtualization: state-of-the-art and research challenges. **IEEE Communications Surveys Tutorials**, [S.l.], v.18, n.1, p.236–262, 2016.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos Sobre Aprendizado de Máquina. In: **Sistemas Inteligentes Fundamentos e Aplicações**. 1.ed. Barueri-SP: Manole Ltda, 2003. p.89–114.

NGUYEN, T.; FDIDA, S.; PHAM, T. A comprehensive resource management and placement for network function virtualization. In: IEEE CONFERENCE ON NETWORK SOFTWARIZATION (NETSOFT), 2017. **Anais...** [S.l.: s.n.], 2017. p.1–9.

PATTARANANTAKUL, M. et al. NFV Security Survey: from use case driven threat analysis to state-of-the-art countermeasures. **IEEE Communications Surveys Tutorials**, [S.l.], v.20, n.4, p.3330–3368, 2018.

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

PHAM, C. et al. Traffic-aware and energy-efficient vnf placement for service chaining: joint sampling and matching approach. **IEEE Transactions on Services Computing**, [S.l.], 2017.

PRASS, F. S. **Estudo comparativo entre algoritmos de análise de agrupamentos em data mining**. 2004. Tese (Doutorado em Ciência da Computação) — Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-graduação em Ciência da Computação.

RAHMAN, S. et al. Auto-Scaling VNFs Using ML to Improve QoS and Reduce Cost. In: INT. CONFERENCE ON COMMUNICATIONS (ICC). **Anais...** [S.l.: s.n.], 2018. p.1–6.

RAJ, B. **Deep Learning on the Edge - An overview of performing Deep Learning on mobile and edge devices**. Accessed: 2019-01-21, https://towardsdatascience.com/deep-learning-on-the-edge-9181693f466c.

REALE, C. et al. Automatic classification of fine-grained soils using CPT measurements and Artificial Neural Networks. **Advanced Engineering Informatics**, [S.l.], v.36, 04 2018.

RIBEIRO, M.; GROLINGER, K.; CAPRETZ, M. A. M. MLaaS: machine learning as a service. In: IEEE 14TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING AND APPLICATIONS (ICMLA), 2015. **Anais...** [S.l.: s.n.], 2015. p.896–902.

ROBERTSON, P.; G. CAMPANELLA, R. Interpretation of Cone Penetration Tests - Part I (Sand). **Canadian Geotechnical Journal**, [S.l.], v.20, p.718–733, 01 2011.

ROSSUM, G. van. **Python tutorial**. Amsterdam: Centrum voor Wiskunde en Informatica (CWI), 1995. (CS-R9526).

SCHNAID, F.; ODEBRECHT, E. **Ensaios de campo e suas aplicações à engenharia de fundações**. 2nd.ed. São Paulo, SP: Oficina de textos, 2012.

SILVA, I. N.; SPATTI, D. H.; FLAUZINO, R. A. **Redes Neurais Artificiais**: para engenharia e ciências aplicadas. São Paulo, SP: Editora Artliber, 2010.

STEINER, M. et al. Abordagem de um problema médico por meio do processo de KDD com ênfase à análise exploratória dos dados. **Gestão e Produção**, São Carlos, SP, v.13, n.2, 2006.

SZYDLO, T.; SENDOREK, J.; BRZOZA-WOCH, R. Enabling Machine Learning on Resource Constrained Devices by Source Code Generation of the Learned Models. In: COMPUTATIONAL SCIENCE – ICCS 2018, Cham. **Anais...** Springer International Publishing, 2018. p.682–694.

TAFNER, M.; XEREZ, M.; RODRIGUES, I. W. **Redes Neurais Artificiais**: introdução e princípios de neurocomputação. Rio de Janeiro, RJ: Blumenau, SC, 1995.

TAN, P.; STEINBACH, M.; KUMAR, V. **Introdução ao Datamining – Mineração de Dados**. Rio de Janeiro, RJ: Editora Ciência moderna Ltda, 2009.

WANG, Y.; HUANG, K.; CAO, Z. Probabilistic identification of underground soil stratification using cone penetration tests. **Can. Geotech**, [S.l.], v.50, p.766–776, 2013.

YAO, Y. et al. Complexity vs. Performance: empirical analysis of machine learning as a service. In: ACM Internet Measurement Conference (IMC). **Anais...** [S.l.: s.n.], 2017.

ZHAO, J. et al. Privacy-Preserving Machine Learning Based Data Analytics on Edge Devices. In: CONFERENCE ON AI, ETHICS, AND SOCIETY. **Anais...** ACM, 2018. p.341–346. (AIES).