Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Recomendação de algoritmos de detecção de ruído via meta-aprendizado

Pedro Borges Pio

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Orientador
Prof. Dr. Luís Paulo Faina Garcia

Brasília
2023

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Recomendação de algoritmos de detecção de ruído via meta-aprendizado

Pedro Borges Pio

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Prof. Dr. Luís Paulo Faina Garcia (Orientador)
Presidente - CIC/UnB

Prof. Dr. Andre de Carvalho
Membro externo - ICMC/USP

Prof. Dr. Vinicius Borges
Membro Interno - CIC/UnB

Prof. Dr. Thiago Faleiros
Suplente - CIC/UnB

do Programa de Pós-graduação em Informática

Brasília, 24 de Fevereiro de 2023

Dedicatória

Dedico este trabalho a todos os que me apoiaram no processo. Obrigado!

Agradecimentos

Gostaria de começar agradecendo a minha família, que esteve ao meu lado me apoiando sempre que possível. Eles tornaram minha vida mais fácil e suportaram meus momentos de estresse, que eram frequentes durante o período de isolamento da pandemia. Acredito firmemente que só consegui realizar este trabalho devido ao apoio e carinho de todos. Muito obrigado.

Não posso esquecer também dos meus amigos, que mesmo à distância, me ajudaram a continuar com meus estudos e me proporcionaram momentos de descontração e alívio, por meio de conversas, videochamadas e discussões construtivas, que foram fundamentais neste processo.

Por fim, e não menos importante, gostaria de agradecer ao meu orientador, que teve paciência e me guiou durante todo o período de pesquisa. Graças à sua orientação, aprendi muito e consegui concluir este trabalho.

Resumo

Este trabalho apresenta uma solução de recomendação de algoritmos de detecção de ruído por meio de técnicas de Meta-Aprendizado (MtL). Primeiramente, foi realizada uma revisão sistemática da literatura referente ao tema de MtL e recomendação de algoritmos de pré-processamento. Na revisão foram verificadas quais as técnicas de pré-processamento, meta-características, algoritmos de Aprendizado de Máquina (AM) e métricas de desempenho são mais utilizados na área de recomendação de algoritmos de pré-processamento. Em seguida foram implementadas duas abordagens diferentes para a recomendação de filtros de ruído por meio de técnicas de MtL. A primeira é uma abordagem de ranqueamento (MtL-Rank), que realiza a sugestão por meio de regressores com objetivo de prever o valor da métrica *f1-score*. A outra abordagem realiza a recomendação por meio de uma sequência de classificadores encadeados (MtL-Multi). Também foi avaliado o desempenho das abordagens ao realizar a recomendação dos filtros juntamente com seus hiperparâmetros. No total, foram utilizados oito filtros de ruído ou 27 quando consideradas as suas variações de hiperparâmetros, quatro técnicas de AM para se extrair a métrica de desempenho e três meta-ranqueadores ou meta-classificadores para se realizar a sugestão. O sistema é avaliado no nível meta e no nível base. No nível meta é avaliado o desempenho dos algoritmos de recomendação por meio da sua acurácia. Já no nível base é verificado o ganho médio na métrica de desempenho (*f1-score*) ao aplicar cada uma das abordagens. Os resultados mostraram que a abordagem MtL-Rank obteve um ganho médio maior no desempenho, atingindo resultados significativamente melhores que o filtro utilizado como *baseline*. Por outro lado, a abordagem MtL-Multi obteve resultados melhores no nível meta, chegando atingir uma acurácia de 49%. Além disso, foi verificado que a recomendação dos hiperparâmetros em conjunto com o filtro pode gerar um ganho no desempenho da recomendação.

Palavras-chave: Meta-Aprendizado, Aprendizado de Máquina, Pré-processamento, Detecção de Ruído, Filtros de Ruído.

Abstract

This work implements a noise detection algorithm recommendation using meta-learning techniques. First, a systematic review of the literature on the subject of meta-learning for preprocessing algorithm recommendation was performed. The review verified which preprocessing techniques, meta-features, machine learning algorithms and performance metrics are commonly used in the area of recommending preprocessing algorithms. Next, two different approaches were implemented for recommending noise filters using meta-learning techniques. The first is a ranking approach (MtL-Rank), which performs the suggestion using regressors and predicts the value of the performance metric f1-score. The other approach performs the recommendation through a sequence of linked classifiers (MtL-Multi). The performance of the approaches was also evaluated when recommending the filters together with their hyperparameters. In total, we used eight noise filters or 27 when considering their hyperparameter variations, four machine learning techniques to extract the performance metric and three meta-rankers or meta-classifiers to perform the recommendation. The system is evaluated at both the meta and base levels. At the meta level, the performance of a meta-learner is evaluated through their accuracy. At the base level, the average gain in the performance metric (f1-score) is verified. The results showed that the MtL-Rank approach obtained a higher average gain at the base level, obtaining significantly better results than the filter used as baseline. On the other hand, the MtL-Multi approach obtained better results at the meta level, reaching an accuracy up to 49%. In addition, it was verified that the suggestion of hyperparameters together with the noise filter can generate a gain in the performance when compared with only recommending the filter.

Keywords: Meta Learning, Machine Learning, Preprocessing, Noise Detection, Noise Filters.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AENN All-k Edited Nearest Neighbors.

AM Aprendizado de Máquina.

AutoML Automated Machine Learning.

CASH Combined Algorithm Selection and Hyperparameter optimization.

CBF Consistency-based filter.

CFS Correlation-based feature.

CNN Condensed nearest neighbor.

CRISP-DM Cross Industry Standard Process for Data Mining.

DCF Dynamic Classification Filter.

DEF Dynamic Ensemble Filter.

DT Árvore de Decisão.

EDB Edge Boosting Filter.

ENN Edited Nearest Neighbours.

FS Seleção de atributos.

GB Gradient Boosting.

GE Generalized Edition.

HARF High Agreement Random Forest.

HRF Hybrid Repair-Remove Filter.

IBK instance-bases learning with parameter k.

Imb Balanceamento de dados.

InfoGain Information Gain.

IS Seleção de instâncias.

KDD Knowladge discovery in databases.

KNN K-Nearest Neighbor.

KNNI K-Nearest Neighbor Imputation.

MD Mineração de Dados.

MFe Meta-Features.

MIFS Mutual Information based Feature Selection.

MLP Multilayer Perceptrom.

MMI Median or Mode Imputation.

MtL Meta-Aprendizado.

MtL-Multi Meta-aprendizado com múltiplas classificações.

MtL-Rank Meta-aprendizado com ranqueamento.

MVI Imputação de dados faltantes.

NorDis Normalização e discretização.

NPV Negative Predictive Value.

ORB Outlier Removal Boosting Filter.

PART Parcial Decision Tree.

PCA Principal Component Analysis.

PCT Predictive Clustering Trees.

PRISM PReprocessing Instances that Should be Misclassified.

RA Ruídos Aleatórios.

RCA Ruídos Completamente Aleatórios.

RF Random Forest.

RNA Ruídos Não Aleatórios.

RUS Random Under-Sampling.

SEF Static Ensemble filter.

SMOTE Synthetic Minority Over-sampling Technique.

SVM Support Vector Machine.

WSE wrapper subset evaluation.

Capítulo 1

Introdução

Aprendizado de Máquina (AM) pode ser definido como a habilidade de um sistema se adaptar a novas situações, detectar e extrapolar padrões [1]. Estas técnicas já estão presentes no cotidiano da população. Empresas como o Meta e Google aplicam AM para oferecer experiências de usuários mais personalizadas ou melhorar os seus produtos [2] [3].

Nos últimos anos, tem-se percebido um crescimento no uso de técnicas de AM. É difícil atribuir esse aumento de popularidade a um único fator. Neste período, o termo "Big Data" [4] se popularizou, gerando uma procura maior por formas de processar os dados armazenados digitalmente, possibilitando a extração de conhecimento e tomadas de decisão por meio dos dados armazenados. Ao mesmo tempo, várias bibliotecas e ferramentas foram desenvolvidas para facilitar o uso de técnicas de AM, como Weka [5], Scikit-learn [6], H2O [7], Tensorflow [8] e Pytorch [9], que possuem vários algoritmos e funcionalidades já implementadas, prontas para uso, sem a necessidade de configurações complexas. Além disso, o poder computacional dos computadores tem continuado a crescer de acordo com a *Lei de Moore* [10], aumentando o número de pessoas que conseguem executar esses algoritmos em seus computadores pessoais.

Infelizmente, mesmo com mais ferramentas que auxiliam e facilitam a implementação de técnicas de AM, criar e utilizar um modelo de AM combinado com técnicas de MD para resolver um problema de ponta a ponta não é uma tarefa simples. O processo envolve diversas etapas mais complexas do que simplesmente executar um algoritmo. Existem diversos modelos que indicam como esses processos devem ser realizados, como, por exemplo, o Cross Industry Standard Process for Data Mining (CRISP-DM) [11] e o Knowladge discovery in databases (KDD) [12]. Ao analisar a semelhança entre ambos os modelos, percebe-se que, além de depender fortemente de dados, nas duas abordagens existem etapas de pré-processamento, modelagem de técnicas de AM e avaliação dos resultados.

Em cada uma dessas etapas, como existem diversas abordagens diferentes, é comum

um processo de escolha dos algoritmos mais adequados para determinado problema. Na etapa de pré-processamento, por exemplo, inicialmente são identificados os problemas presentes no conjunto de dados e, em seguida, é escolhido um algoritmo para solucioná-los. Um processo semelhante ocorre na escolha do modelo, onde é necessário definir qual algoritmo de AM consegue obter o melhor resultado ao ser aplicado a esse conjunto de dados. O processo de escolha de pré-processamentos e modelos não é trivial, além disso, a escolha de um influencia no resultado do outro, ou seja, alguns modelos de AM podem obter melhores resultados quando executados em conjunto com um algoritmo de pré-processamento específico [13].

Existem vários tipos de dados, como imagens, séries temporais, dados tabulares e, para cada um destes, pode-se implementar diversas técnicas de pré-processamentos diferentes. No caso dos dados tabulares, foco deste trabalho, encontram-se técnicas como: detecção de ruído, onde procura-se encontrar erros que podem ocorrer em conjuntos de dados [14]; Seleção de atributos (FS), em que os atributos mais importantes para o problema em questão são escolhidos diminuindo a redundância e dimensão do conjunto [15]; Seleção de instâncias (IS), em que somente algumas instâncias do conjunto são selecionadas [16]; Imputação de dados faltantes (MVI), que é utilizada para tentar preencher as variáveis que possuem dados em falta [17]; e Balanceamento de dados (Imb), onde procura-se equilibrar a distribuição das classes de um determinado conjunto [18].

A grande quantidade de técnicas de tratamento de dados gera o problema de escolher quais devem ser aplicadas. É necessário identificar se um processo será útil ou não, considerando que, quando aplicados de forma errada, um pré-processamento pode piorar o desempenho do processo de AM [19]. O fato de existirem diversos algoritmos propostos para cada pré-processamento aumenta ainda mais o espaço de busca deste problema, o que pode dificultar o trabalho de escolha. Como não existe um algoritmo único que sempre terá o melhor desempenho (*there is no free lunch* [20]), automatizar esse problema pode não só melhorar os resultados dos sistemas de AM, como também diminuir seus custos.

Mesmo não sendo um problema simples, na última década, por motivos como pressão de mercado, a tentativa de automatizar o processo de AM, processo comumente chamado de Automated Machine Learning (AutoML), ganhou força [21]. O campo de AutoML busca automatizar as escolhas de quais algoritmos de AM deve-se utilizar. Idealmente, o usuário simplesmente precisa prover os dados e o sistema determina a melhor abordagem para a situação específica, diminuindo os custos da aplicação de processos de AM e a necessidade de especialistas [22]. Hoje já existem grandes empresas como Google [1], Microsoft [2] e IBM [3], entre outras, com ferramentas próprias de AutoML, indicando uma

[1] `https://cloud.google.com/automl`
[2] `https://azure.microsoft.com/en-us/services/machine-learning/`
[3] `https://www.ibm.com/br-pt/cloud/watson-studio/autoai`

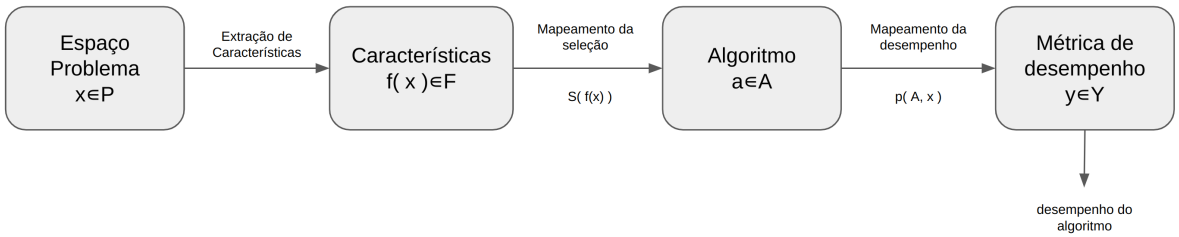

Figura 1.1: Diagrama representando o problema de seleção de algoritmos de Rice.

possível tendência do mercado de investir nesse setor.

Entre os processos mais comuns que um sistema de AutoML procura automatizar estão: pré-processamento de dados e seleção de atributos; seleção de modelos de AM; otimização de hiperparâmetros; busca pela melhor arquitetura; interpretação do modelo; e análise da predição [21]. Porém, não é necessário abordar todas essas etapas, alguns sistemas como Auto-Weka [23] e auto-SKlearn [24] procuram solucionar o problema Combined Algorithm Selection and Hyperparameter optimization (CASH), que engloba a seleção do AM e otimização dos hiperparâmetros. Outros, como o TPOT [25], procuram encontrar o melhor *pipeline*, ou seja, o melhor pré-processamento, modelo e hiperparâmetros para o problema. A forma como os problemas são abordados varia de acordo com a ferramenta e, entre as técnicas mais utilizadas, estão Otimização Bayesiana, Programação Genética e Meta-Aprendizado (MtL) [26].

Naturalmente, o problema de AutoML passa pela escolha do melhor algoritmo, um problema que Rice (1976) [27] foi um dos primeiros a propor uma solução. Nela procura-se o melhor algoritmo em um conjunto de algoritmos A a partir de um espaço de problema P. Essa escolha é feita a partir de um conjunto de atributos F que são extraídos por algum processo computacional aplicado em P. Além disso, cada algoritmo terá seu resultado mapeado no espaço de métricas de desempenho Y. O objetivo principal é, a partir dos atributos, conseguir escolher o algoritmo que tem $y \in Y$ maximizada. A Figura 1.1 representa um diagrama do problema de Rice. A partir de uma instância $x \in P$ são extraídos os atributos tais que $f(x) \in F$, em seguida, é feito um mapeamento dos algoritmos por meio de uma função $S(f(x))$ indicando o algoritmo $a \in A$, por fim, são mapeadas as métricas de desempenho $p(A, x)$ gerando uma métrica $y \in Y$.

Posteriormente, Smith-Miles (2008) [28] expandiu esse conceito de seleção de algoritmos para técnicas de MtL. Este processo, é dividido em três etapas. Na primeira etapa, a partir do conjunto de atributos F e das métricas obtidas pelo algoritmo Y, é gerado um conjunto de dados chamado metadados, também conhecido como meta-base. Na segunda etapa, o objetivo é encontrar relações entre o conjunto F e Y, gerando regras empíricas ou

ranqueamentos[4] que levam à seleção automática do algoritmo. Por fim, na terceira etapa, os resultados obtidos são analisados, estudando como os algoritmos estão desempenhando, e se necessário, realizando alguns ajustes nos algoritmos utilizados.

A atualização do problema proposta por Smith-Miles aplica os conceitos de MtL para a seleção de algoritmos. No entanto, existem outras aplicações de MtL. Vanschoren (2019) [29] apresenta MtL como uma forma de aprender a aprender e indica três possíveis aplicações para MtL: (*i*) aprendizado de avaliações de modelos, com base em avaliações prévias obtidas de tarefas, procura-se prever alguma configuração de hiperparâmetros ou arquiteturas de redes neurais de alguma outra tarefa; (*ii*) aprendizado baseado em características, onde se extrai um conjunto de características que descrevem um problema, chamadas de Meta-Features (MFe), que são utilizadas para sugerir configurações de um problema semelhante, algoritmos, *pipelines*, ou se uma tarefa deve ou não ser realizada; (*iii*) aprendizado por modelos prévios, em que o modelo de aprendizado utiliza a estrutura e parâmetros de modelos semelhantes para treinar outro modelo, sendo comum em treinamentos de redes neurais e *few-shot learning*. Neste trabalho, MtL é utilizado na forma de aprendizado baseado em características onde, considerando as MFe extraídas, recomenda-se um algoritmo de pré-processamento, mais especificamente de redução de ruído, para conseguir um resultado melhor ao aplicar um modelo de AM.

Neste contexto de aprendizado baseado em características, Brazdil et. al. (2008) [30] define MtL como o estudo de métodos que, por meio de meta-conhecimentos, obtém modelos e soluções eficientes realizando adaptações nos processos de AM e de MD. Pode-se dividir esse processo em dois níveis: o nível base e o nível meta. O nível base é formado pela aplicação direta dos algoritmos de AM para uma tarefa específica. Já o nível meta é formado pela junção de diversas aplicações do nível base, assim, pode-se dizer que no nível meta trabalha-se com os metadados.

Para que o sistema de MtL consiga adquirir o meta-conhecimento, é fundamental a escolha das melhores Meta-Features (MFe) para o problema. As MFe são as características extraídas dos conjuntos de dados e são essenciais no processo de MtL, formando, em conjunto com as métricas de avaliação Y, o conjunto de metadados utilizado para a escolha do algoritmo. As MFe são comumente separadas em grupos que compartilham características entre si, porém, não existe um consenso quanto a eles, sendo encontrados diferentes nomes para descreverem os mesmos grupos [31]. Assim, neste trabalho utiliza-se a classificação proposta por Rivolli et. al. (2022) [31], que é semelhante a apresentada por Brazdil et al. (2008) [30], sendo separados nos seguintes grupos: Simples; Estatísticos; Teoria da informação; Baseados em modelos; e *Landmarking*.

[4]para evitar confusão com a classificação de AM e a classificação em forma de uma lista ordenada, neste trabalho, a palavra ranqueamento ou ranque, será utilizada, em um sentido de *ranking*

Neste projeto, utiliza-se MtL para sugerir o filtro de ruído[5] mais recomendado para um determinado conjunto de dados. Para definir a detecção de ruído como objetivo do estudo, foi realizada uma revisão sistemática no Capítulo 2, onde analisou-se diversas recomendações de técnicas de pré-processamento via MtL e notou-se um ganho considerável de desempenho da aplicação dessas técnicas e uma quantidade menor de trabalhos relacionados ao tema. Sabe-se que as etapas de pré-processamento são uma das que mais demandam tempo para serem realizadas ao aplicar técnicas de AM [32]. Portanto, este trabalho pode ajudar a reduzir o tempo gasto e a necessidade de especialistas em problemas mais simples, ou ser implementado em conjunto com outras ferramentas de AutoML que normalmente não possuem sugestões desse tipo de pré-processamento implementadas.

Além de um estudo extenso na literatura sobre como MtL está sendo utilizado para recomendação de algoritmos de pré-processamento, neste trabalho são avaliadas duas abordagens de recomendação de filtros de ruído: uma realizando um ranqueamento dos algoritmos estudados e outra sendo uma adaptação da abordagem apresentada por Parmezan et. al. (2021) [33], onde utiliza-se uma sequência de classificações para realizar a sugestão do algoritmo. Também são comparados os resultados obtidos ao realizar a recomendação dos hiperparâmetros dos filtros, verificando se o sistema consegue gerar algum ganho ao adicionar essa complexidade à solução.

1.1 Justificativa

Com o aumento da quantidade e importância dos dados, os sistemas de AutoML estão sendo cada vez mais utilizados por empresas e pesquisadores para aumentar a utilização de técnicas de AM e reduzir a necessidade de especialistas na área. Por isso, encontram-se ferramentas de AutoML sendo disponibilizadas comercialmente por empresas [21]. No entanto, técnicas de pré-processamento de dados de forma automatizada ainda não são fortemente exploradas por essas ferramentas [21].

Estima-se que entre 50% e 80% do tempo gasto em aplicações de AM será dedicado ao pré-processamento [32]. No entanto, o maior foco das ferramentas desenvolvidas no mercado ainda está na seleção de modelos e otimização de hiperparâmetros. O desenvolvimento de uma ferramenta de recomendação de filtros de ruídos pode não somente reduzir o tempo gasto na etapa de pré-processamento, como também melhorar o desempenho de algoritmos de AM e diminuir a necessidade de especialistas no processo. Além disso, ferramentas de AutoML, como auto-SKlearn e TPOT, que não possuem processos de detecção de ruído implementados, podem se beneficiar ao incorporar este processo em suas recomendações.

[5]Neste trabalho os algoritmos de detecção de ruído utilizados são sempre filtros de ruído.

Como uma das aplicações de técnicas de MtL é simular o processo de escolha realizado por humanos em computadores por meio de conhecimentos prévios [29], MtL torna-se um candidato promissor para simular o trabalho de especialistas e auxiliar na escolha dos melhores algoritmos de pré-processamento em uma aplicação de AM. Dessa forma, a partir de uma meta-base contendo informações prévias sobre conjuntos de dados, acredita-se ser possível uma recomendação com foco em automatizar a escolha de filtros de ruído, melhorando o desempenho dos algoritmos de AM.

1.2 Objetivo

Este trabalho tem como objetivo principal verificar a viabilidade e a qualidade de técnicas de MtL para gerar um sistema de recomendação de filtros de ruídos, de forma a facilitar a escolha deles para usuários. Para elaborar o sistema de MtL será necessário um estudo sobre o efeito das técnicas de redução de ruídos em relação ao desempenho dos algoritmos de AM. Com o intuito de formar uma meta-base efetiva, também será necessário identificar quais características de um conjunto influenciam mais na escolha do algoritmo.

Duas abordagens diferentes de MtL serão testadas e comparadas para definir qual é a mais adequada para o conjunto de dados em análise. Uma utilizando técnica de ranqueamento e a outra realizando várias classificações em sequência. Em cada uma das abordagens, será verificada qual possui a melhor acurácia na recomendação e maior ganho geral em relação à métrica de desempenho, possibilitando quantificar a qualidade de cada abordagem, além de confirmar se a abordagem é eficiente na recomendação dos filtros.

Ao final deste trabalho, espera-se comprovar que as técnicas de MtL são capazes de recomendar quais filtros de ruído devem ou não ser utilizados e facilitar a escolha dos melhores filtros para conjuntos de dados genéricos com base em seu treinamento prévio, possibilitando assim, auxiliar usuários a aplicar técnicas de AM de uma maneira eficiente e automática.

1.3 Hipótese

O intuito principal deste trabalho é validar a eficiência da recomendação de filtros de ruído por meio de técnicas de MtL extraindo MFe relevantes para o problema, possibilitando adquirir conhecimento antes da execução dos algoritmos. Desta forma, a hipótese principal é que “é possível gerar recomendações de técnicas de filtros de ruído que resultem em um ganho de desempenho para algoritmos de AM utilizando MtL.”. Além disso, há também as seguintes hipóteses secundárias:

- A recomendação de hiperparâmetros em conjunto com o filtro de ruído por meio de técnicas de MtL gera ganhos na métrica de desempenho maiores do que somente recomendando o filtro;
- Utilizar uma série de classificações em sequência como MtL para recomendar filtros de ruído possibilita ganhos na métrica de desempenho de algoritmos de AM;
- Utilizar uma série de classificações em sequência como MtL para recomendar filtros de ruído gera resultados melhores que uma abordagem utilizando ranqueamento.

1.4 Organização do Documento

O restante desse documento será dividido da seguinte forma: No Capítulo 2 é apresentada a revisão sistemática sobre o assunto, verificando o que vem sendo estudado na área e quais as tendências de pesquisa; o Capítulo 3 contém a revisão teórica da área, explicando os conceitos necessários de AM, pré-processamento para detecção de ruído e MtL; o Capítulo 4 introduz a metodologia aplicada nesta pesquisa; Os experimentos realizados estão presentes no Capítulo 5; e as conclusões do trabalho são apresentados no Capítulo 6.

Capítulo 2

Revisão Sistemática

Uma das etapas mais importantes ao realizar uma pesquisa é compreender melhor o estado da arte referente ao tema. Nesse processo procura-se entender melhor o que está sendo ou foi estudado no assunto. Existem diversas formas de se descobrir o estado da arte, neste trabalho o método escolhido foi uma revisão sistemática da literatura [34].

Devido ao sucesso obtido nas pesquisas de medicina [34], outras disciplinas como estudos sociais, educação, sistema de informações e engenharia de software [35] começaram a aplicar a revisão sistemática de uma forma mais recorrente. Apresentando assim, uma maior replicabilidade, transparência a pesquisa, sendo necessário, por exemplo, que o autor especifique os motivos pelo qual os artigos foram selecionados para a revisão [34].

Pode-se dizer que uma revisão sistemática é composta por três etapas distintas: planejamento, condução e apresentação da revisão. Na etapa de planejamento, são definidas as perguntas da pesquisa e os protocolos da revisão e validação. Em seguida, na etapa de condução, são selecionadas as pesquisas, verifica-se sua qualidade e extraem-se e analisam-se os dados relevantes encontrados. Por fim, os resultados da revisão são apresentados [35].

Neste capítulo, será apresentada a revisão sistemática aplicada nesta pesquisa. O processo utilizado foi baseado na abordagem de uma revisão sistemática em dinâmica da digitação apresentada em [36], porém, adaptada para o tema de MtL e pré-processamento. Na Seção 2.1 é descrito o planejamento da revisão, em seguida, a Seção 2.2 introduz a etapa de condução, mostrando como a pesquisa foi elaborada, por fim, na Seção 2.3 é mostrado o que foi encontrado com a revisão sistemática.

2.1 Planejamento

Como primeira etapa do planejamento, verificou-se a necessidade de realizar uma revisão sistemática sobre o tema, pois é possível que existissem algumas publicações que suprissem

a necessidade desta pesquisa. Caso a revisão fosse encontrada, seria possível utilizá-la para responder as perguntas e auxiliar a condução desse projeto, evitando assim um retrabalho. Porém, não foi encontrada nenhuma referente a MtL e pré-processamento, o que torna a realização desta revisão fundamental. É importante ressaltar que essa busca foi limitada a pesquisas publicadas em inglês.

Em seguida, para guiar melhor o resultado da pesquisa e entender o estado da arte da utilização de MtL quando aplicada para recomendação de técnicas de pré-processamento, elaborou-se as seguintes perguntas que deveriam ser respondidas com a revisão:

- Quais são os algoritmos de pré-processamento mais aplicados em recomendações? Como esses algoritmos são classificados?
- Quais são os algoritmos de AM mais aplicados na etapa de recomendação? E na etapa de classificação utilizada para extrair as métricas formar a meta-base?
- Existe um conjunto de MFe frequentemente utilizado para a recomendação do pré-processamento?
- Quais são os métodos de avaliação mais utilizados para realizar as recomendações?
- Existe alguma vantagem em aplicar MtL para recomendação de algoritmos de pré-processamento?
- Existe um conjunto de dados padrão que é utilizado para treinar e avaliar os sistemas?

Tendo os objetivos definidos com as perguntas de pesquisa, procurou-se encontrar os termos que seriam mais adequados para a criação da *string* de pesquisa que seria utilizada para a busca dos artigos. Após um breve estudo sobre a área e as técnicas de pré-processamento, a seguinte *string* de pesquisa foi elaborada. Note que alguns termos, como *metalearning*, aparecem mais de uma vez pois, dependendo do artigo, a grafia pode variar:

(metalearning OR "meta learning" OR meta-learning OR meta-features OR "meta features" OR "characterization measures") AND (pre-processing OR "pre processing" OR preprocessing OR postprocessing OR "feature selection" OR "noise reduction" OR "data preparation" OR "data reduction" OR "instance selection" OR "noise detection" OR "outlier detection" OR "unbalanced" OR "imbalanced" OR "missing data" OR "missing values")

A *string* foi aplicada em quatro base de dados de pesquisas diferentes, Scopus[1], Web of Science[2], IEEE Explorer[3], ACM Digital Library[4].

Por fim, resta planejar a etapa de seleção e avaliação dos trabalhos encontrados. A seleção é realizada em duas etapas: Inicialmente são lidos os resumos de todas as referências encontradas, as que não mencionarem a utilização de técnicas de MtL para seleção de algoritmos serão descartadas; em seguida, os artigos que restaram passam pela leitura da introdução e conclusão, onde são selecionadas as pesquisas que utilizam MtL para a recomendação de algoritmos de pré-processamento. Dessa forma, espera-se a identificação das publicações que realmente abordam o tema de interesse da pesquisa.

Os artigos selecionados na segunda etapa ingressam na etapa de avaliação, onde são lidos para responder de forma binária (sim ou não) as seguintes questões:

1. É possível replicar[5] o experimento?
2. Os conjuntos de dados foram apresentados e podem ser reutilizados?
3. São apresentados os algoritmos de pré-processamento que foram utilizados?
4. Foram apresentadas as MFe utilizadas?
5. Foi explicado o motivo da escolha das MFe?
6. O método de MtL é explicado claramente?
7. Os resultados foram comparados com pesquisas anteriores?
8. É mostrada a vantagem da a abordagem escolhida?
9. O artigo realizou análise na meta-base?
10. O artigo realizou análise no nível meta?
11. O artigo realizou análise no nível base?

Para cada pergunta o artigo recebe uma nota 0 ou 1, onde 0 representa que o artigo não satisfaz o que a pergunta sugere e 1 que satisfaz completamente. Este processo é importante para avaliar a qualidade das pesquisas que estão sendo publicadas. Além dessas perguntas, quando possível, foram utilizadas perguntas complementares para realizar uma análise mais ampla:

[1] `http://www.scopus.com/`
[2] `http://www.webofscience.com/`
[3] `http://ieeexplore.ieee.org/`
[4] `http://dl.acm.org/`
[5] Um trabalho é considerado replicável se contém todas as informações necessárias para implementar o que está sendo proposto e obter resultados equivalentes

1. Quais MFe foram utilizadas na pesquisa?
2. Quais são os métodos de previsão utilizados?
3. Quais algoritmos de pré-processamento foram utilizados?
4. Qual métrica é utilizada para predição do meta-aprendizado?

Diferentemente das primeiras perguntas, para avaliar os artigos, dessa vez o resultado é armazenado em forma de lista, onde são anotados quais os algoritmos, características e métodos mais utilizados em cada implementação, o que possibilita uma análise mais objetiva sobre o que é utilizado com mais frequência nas pesquisas.

2.2 Condução

A etapa de condução começa com a procura das referências nas quatro plataformas de busca Scopus, Web of Science, IEEE Explorer, ACM Digital Library. Como planejado, a *string* de busca foi adaptada para que funcione em cada plataforma. A busca foi realizada em agosto de 2022 e foram encontrados 320 trabalhos no Scopus, 96 no Web of Science, 90 no IEEE Xplore e 121 referências no Digital Library, totalizando 627 quando somadas. Em seguida, as referências foram baixadas e adicionadas à plataforma Mendeley[6], um software para gerenciar referências de pesquisas. Após filtrar as que apareciam em mais de uma base de dados, restaram um total de 450 referências.

Depois da leitura dos resumos de todos os 450 trabalhos encontrados, foram selecionados os que realizavam recomendação de algoritmos por meio de MtL, restando um total de 77 trabalhos. Estes passaram para a segunda etapa de seleção, onde, após a leitura da introdução e conclusão dos artigos, as pesquisas com foco em recomendação de algoritmos de pré-processamento utilizando MtL foram selecionadas, resultando em um conjunto de 37 artigos para a leitura e análise completa de acordo com as perguntas apresentadas na etapa de planejamento.

O gráfico apresentado na Figura 2.1 mostra a distribuição dos trabalhos com relação ao ano de publicação, bem como a quantidade de pesquisas selecionadas para a avaliação após a realização das etapas de seleção. Ao analisá-lo, é possível perceber que a quantidade de publicações obtidas a partir da *string* de pesquisa cresceu consideravelmente nos últimos anos, passando de 35 em 2019 para 103 em 2021. Porém, a quantidade de artigos selecionados para a leitura não seguiu o mesmo padrão, estando quase constante desde 2016. Além disso, vale ressaltar que as publicações de 2022 só estão catalogadas até agosto, o que explica a baixa quantidade de resultados encontrados neste ano.

[6] `https://www.mendeley.com/`

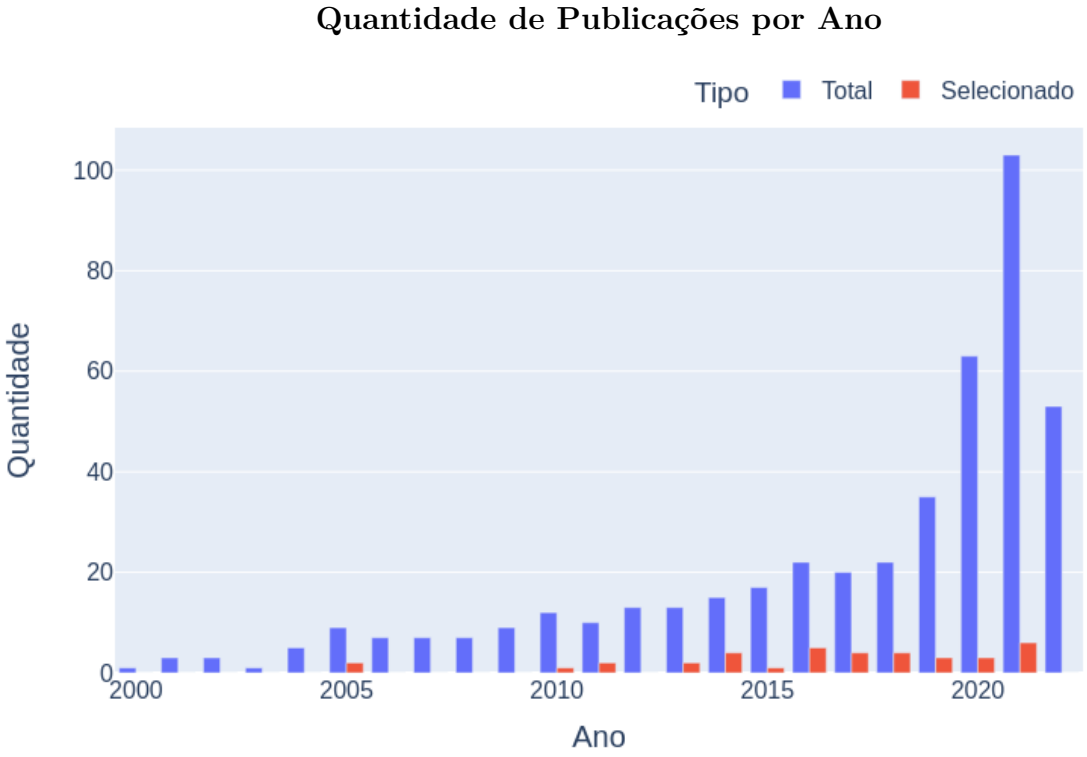

Figura 2.1: Gráfico contendo a quantidade de publicações encontradas e selecionadas para a leitura agrupadas pelo ano da publicação.

2.3 Apresentação

Após a avaliação, iniciou-se a etapa de extração de informações. A partir das notas obtidas nas perguntas, foi possível verificar quais trabalhos eram mais completos. A Figura 2.2 e a Figura 2.3 mostram, respectivamente, um gráfico de porcentagens e um histograma das notas dos artigos. Neles, nota-se que poucos artigos apresentam notas acima de nove e que a maioria (54%) dos artigos obtiveram uma nota total entre 5 e 7. Além disso, somente dois artigos satisfizeram todas as perguntas e obtiveram a nota 11. Similarmente, poucos artigos obtiveram uma nota muito baixa, o que era esperado considerando que as etapas de filtragem aplicadas para chegar ao número reduzido de artigos envolviam a leitura da introdução e conclusão das pesquisas.

Também foi possível visualizar quais perguntas eram frequentemente respondidas ou não. A Figura 2.4 mostra a porcentagem de artigos que satisfizeram cada pergunta. Nota-se que poucos trabalhos apresentam uma comparação com trabalhos anteriores (Q7), o que pode ser justificado por uma série de fatores que dificultam a comparação, como a falta de conjuntos de dados de *benchmark*, os diferentes conjuntos de algoritmos de pré-processamento utilizados ou até mesmo o alto custo computacional que é necessário para

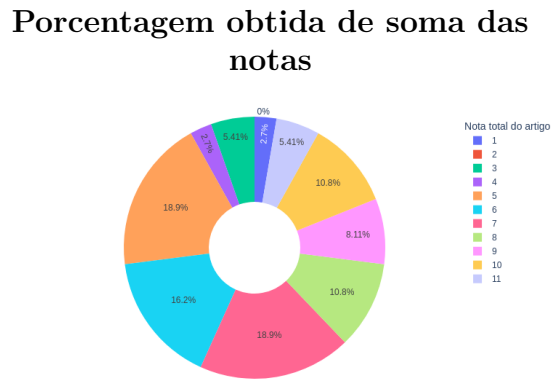

Figura 2.2: Gráfico apresentando a porcentagem de artigos que obtiveram determinada soma das notas na avaliação realizada.

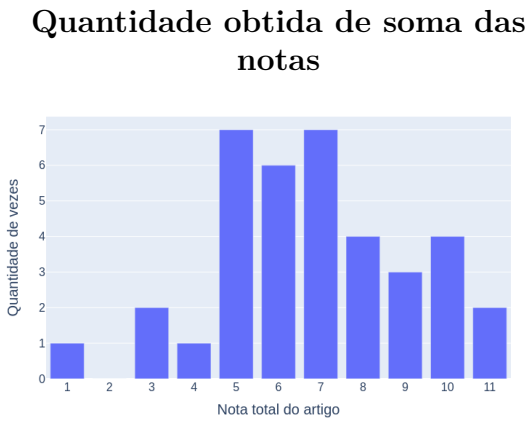

Figura 2.3: Gráfico de barra contendo a quantidade de artigos que obtiveram determinada soma das notas na avaliação realizada.

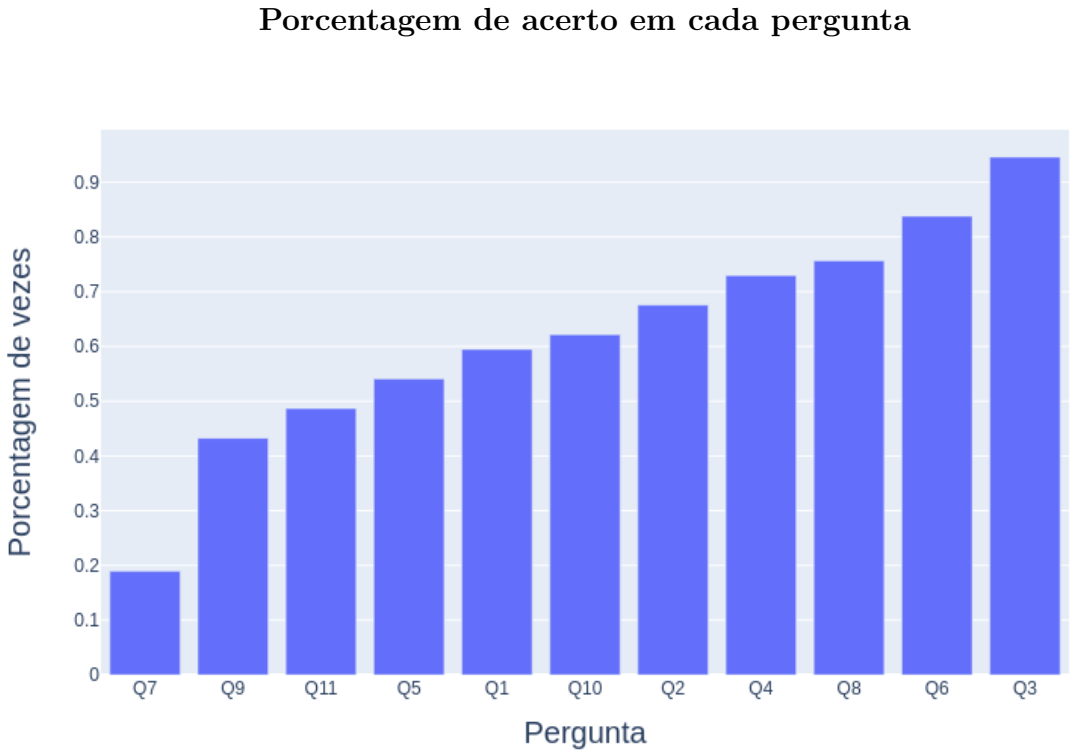

Figura 2.4: Gráfico contendo a porcentagem de vezes que cada pergunta foi satisfeita, a equivalência da pergunta com seu respectivo número pode ser encontrada na Seção 2.1.

Ano	FS	Imb	MVI	NorDis	Ruído	IS	Total de publicações
2005	1 [40]	1 [40]	1 [41]	0	0	0	2
2010	0	0	0	0	0	1 [42]	1
2011	1 [43]	0	0	0	0	1 [44]	2
2013	1[45]	0	0	0	0	1 [46]	2
2014	1 [47]	0	0	0	1[48]	3[49, 50, 48]	4
2015	1 [51]	0	0	0	0	0	1
2016	2 [52, 53]	0	2 [54, 52]	1[52]	2 [55, 56]	0	5
2017	2 [57, 58]	1 [59]	0	0	0	0	4 [60]
2018	4 [61, 62, 63, 64]	0	2 [62, 63]	2 [62, 63]	0	0	4
2019	1 [39]	1 [65]	1 [39]	1 [39]	0	1 [66]	3
2020	2[67, 68]	1[67]	1[69]	1 [67]	0	0	3
2021	3[70, 71, 33]	2[72, 73]	1 [73]	1 [73]	0	1 [74]	6

Tabela 2.1: Tabela contendo a quantidade de artigos selecionados em cada ano separado pelo tipo de pré-processamento

replicar os experimentos. O segundo aspecto que é menos trabalhado nas pesquisas é a análise da meta-base (Q9), que talvez possa ser justificado por limitações de espaço em artigos, já que normalmente é uma análise extensa devido à quantidade de MFe e de algoritmos presentes nos experimentos. Por outro lado, percebe-se que mais de 90% dos artigos apresentaram os algoritmos de pré-processamento que foram utilizados (Q3), algo analisado com mais detalhes nas perguntas complementares apresentadas na Seção 2.1, e que mais de 80% explicam claramente o processo de MtL utilizado (Q6), algo que também é fundamental no processo. Uma tabela contendo detalhe a nota de cada um dos artigos avaliados está presente no Anexo I.

Após a análise das perguntas, realizou-se a avaliação a partir dos dados coletados nas anotações com o objetivo de encontrar quais eram os métodos de AM, MFe e algoritmos mais utilizados nas pesquisas. O primeiro passo foi definir como classificar os tipos de pré-processamentos. Neste trabalho, utilizou-se uma classificação dos pré-processamentos semelhante à apresentada em [37], porém, ao invés de dividir em grupos, os algorítimos são separados somente pelo tipo das técnicas. Também foram feitas duas modificações nos seguintes tipos: (i) as técnicas de Normalização e discretização (NorDis) formaram um único grupo, como apresentado no trabalho de Alexandropoulos et. al. (2019) [38]; (ii) como tanto as transformações de espaço e a quanto FS reduzem o número de atributos no conjunto de dados, técnicas de transformação de espaço, como PCA, foram consideradas como parte do grupo de FS, uma abordagem que foi encontrada em [39]. Assim, resultando nas seguintes classes de pré-processamento: Seleção de atributos (FS), Balanceamento de dados (Imb), Imputação de dados faltantes (MVI), Normalização e discretização (NorDis), Detecção de ruído e Seleção de instâncias (IS).

A Tabela 2.1 mostra quantas referências foram encontradas de cada pré-processamento quando separadas pelo ano da publicação e o tipo de pré-processamento. É possível perce-

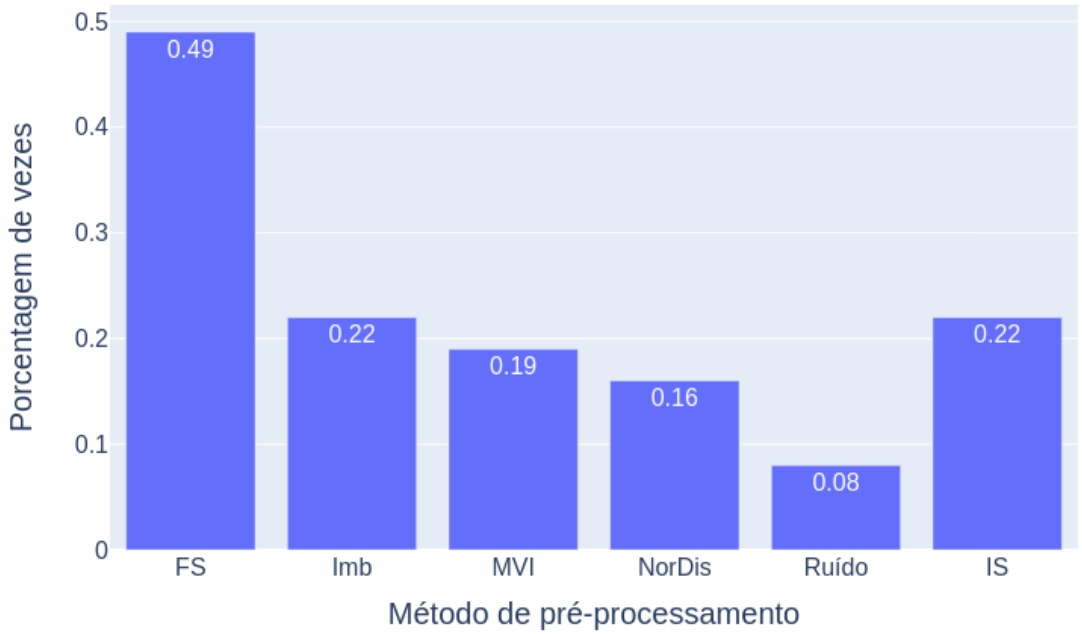

Figura 2.5: Gráfico contendo a porcentagem de vezes que cada método de pré-processamento foi utilizado nos trabalhos.

ber uma maior ocorrência de trabalhos relacionados com FS, sendo o ano de 2010 o único em que não foi encontrada nenhuma publicação sobre o tema. Percebe-se também uma tendência mais recente de utilizações de técnicas de Imb, MVI e NorDis e uma tentativa de retomada de estudos em IS. Além disso, as técnicas de detecção de ruído são as menos pesquisadas. A Figura 2.5 representa bem a distribuição dos tipos de pré-processamento que são recomendados nos trabalhos, do total de 37 artigos, aproximadamente de metade aborda recomendação FS enquanto somente 8% com Ruído.

Depois de entender quais tipos de algoritmos de pré-processamento estão sendo recomendados, realizou-se a análise das anotações feitas de acordo com as quatro perguntas complementares apresentadas na Seção 2.1. Primeiramente foram verificadas quais as MFe mais utilizadas. A Figura 2.6 apresenta as MFe que apareceram ao menos três vezes em trabalhos distintos e sua respectiva classe de pré-processamento, quando possível foi utilizada a nomenclatura referente as MFe presente em [31]. É possível perceber que não é tão comum a repetição dos conjuntos de MFe em trabalhos distintos, principalmente quando se contabiliza as características mais computacionalmente custosas. De fato, todas as MFe que aparecem ao menos oito vezes pertencem ao grupo de características

simples, de teoria da informação, ou estatístico[7]. Somente quatro características foram utilizadas em todas as classes de pré-processamento, e oito foram encontradas em pelo menos quatro das cinco classes, o que indica que cada classe de pré-processamento possui um conjunto de características diferente para se obter um bom desempenho. Além disso, nota-se que não existe um consenso sobre quais MFe devem ser utilizadas para se realizar as recomendações.

Mapa de calor contendo a quantidade de vezes que cada MFe foi encontrada

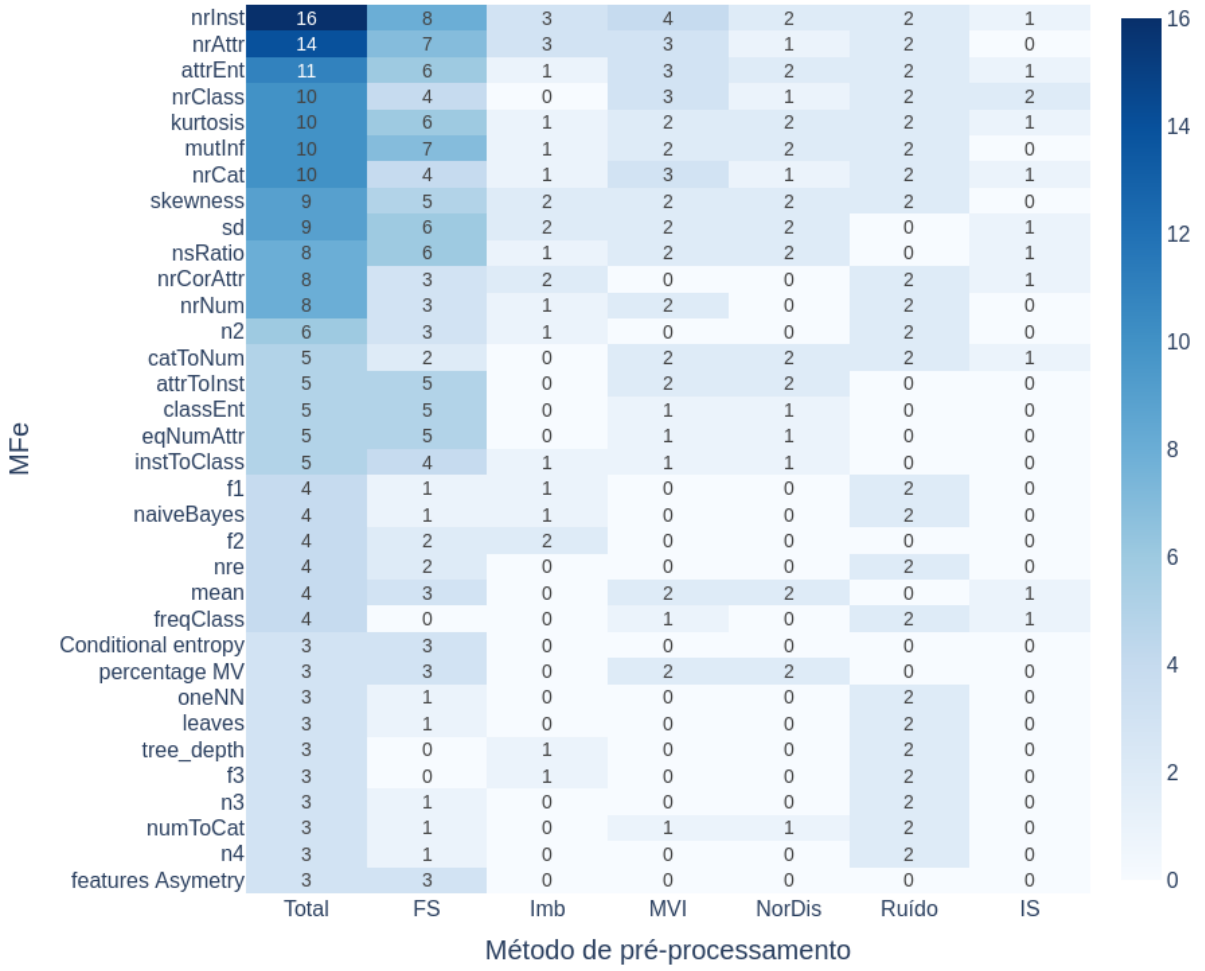

Figura 2.6: Mapa de calor representando a lista das MFes mais utilizadas nos trabalhos juntamente com a classe de pré-processamento que foi utilizada.

É importante ressaltar que, para evitar que um único conjunto de publicações realizadas pelo mesmo grupo de autores influencie na análise, as características presentes em trabalhos contendo o mesmo conjunto de MFe e os mesmos autores foram contabilizadas

[7]Para mais informações sobre os tipos de MFe, favor consultar a Seção 3.3.4

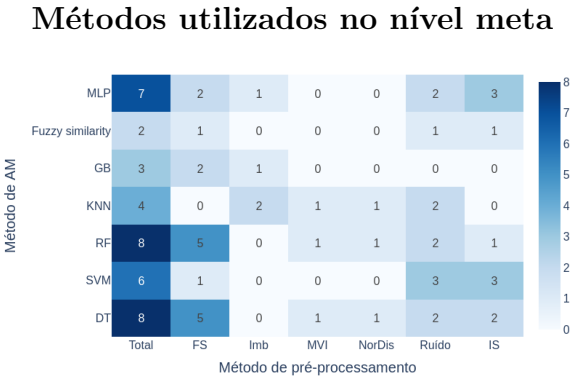

Figura 2.7: Métodos de AM utilizados no nível meta

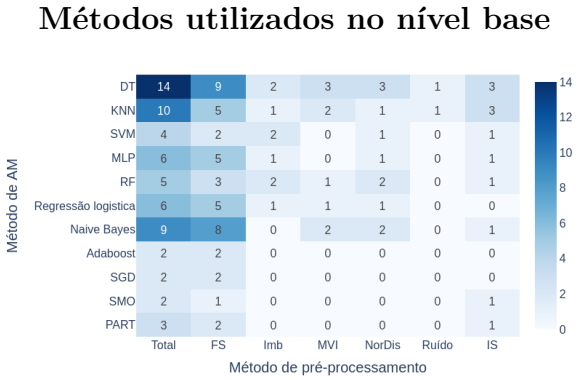

Figura 2.8: Métodos de AM utilizados no nível base

somente uma vez, como no caso das publicações de Bilalli [60, 39, 63]. O mesmo critério foi utilizado nas análises das outras três questões complementares.

Em seguida, foram analisados quais os algoritmos de AM foram mais utilizados no nível meta e no nível base, ou seja, para se realizar a recomendação de algoritmos de pré-processamento e para a criação da meta-base. As Figuras 2.7 e 2.8 apresentam um mapa de calor contendo os métodos de AM encontrados mais de uma vez nas publicações selecionadas no nível meta e base, respectivamente. Nota-se diferentes tendências nas escolhas dos algoritmos para o nível meta quando comparado com o nível base, no primeiro caso, são mais frequentes métodos com um maior custo computacional, como MLP ou RF, já no caso do nível base encontram-se mais frequentemente DT, KNN e *Naive bayes*. Esta diferença de escolhas pode ser justificada pela quantidade de vezes que cada algoritmo é executado, pois os métodos de AM serão treinados múltiplas vezes para se formar uma meta-base, tornando natural que, devido ao alto custo computacional desse processo, se escolha algoritmos mais simples que serão treinados mais rapidamente. Por outro lado, no nível meta, somente uma base é utilizada para se treinar os algoritmos, o que facilita a utilização de algoritmos com um custo mais elevado. Durante a avaliação dos artigos, também notou-se que era mais frequente a utilização de mais algoritmos no nível base do que no nível meta e que em ambos os casos as DT foi o método mais utilizado, porém, assim como no caso da análise das MFes, não existe um algoritmo que seja selecionado na maioria dos trabalhos.

A terceira análise envolveu verificar quais algoritmos de pré-processamento são mais utilizados. Como mostrado na Figura 2.5, existe um desbalanceamento nas classes de pré-processamento, por isso escolheu-se analisar cada um dos tipos de pré-processamentos separadamente. As Figuras 2.9, 2.10, 2.11 e 2.12 apresentam em quantos trabalhos cada algoritmo de FS, Imb, IS e detecção de ruído, respectivamente, faz parte do conjunto de al-

goritmos recomendados. Como o intuito desta análise era descobrir quais algoritmos eram aplicados com mais frequência, somente foram considerados os algoritmos encontrados em pelo menos dois estudos.

Sendo a classe mais comum, era esperado que fossem encontrados um maior número de algoritmos de FS. Na Figura 2.9 percebe-se que dois algoritmos se destacam: o ReliefF e o Correlation-based feature (CFS), ambos sendo utilizados seis vezes, o que corresponde a 33% de todas as publicações referentes a FS. Outro método constantemente aplicado é o Consistency-based filter (CBF), que foi encontrado quatro vezes. O restante dos algoritmos aparece três vezes, como no caso do Principal Component Analysis (PCA), ou duas vezes, como no caso do Mutual Information based Feature Selection (MIFS).

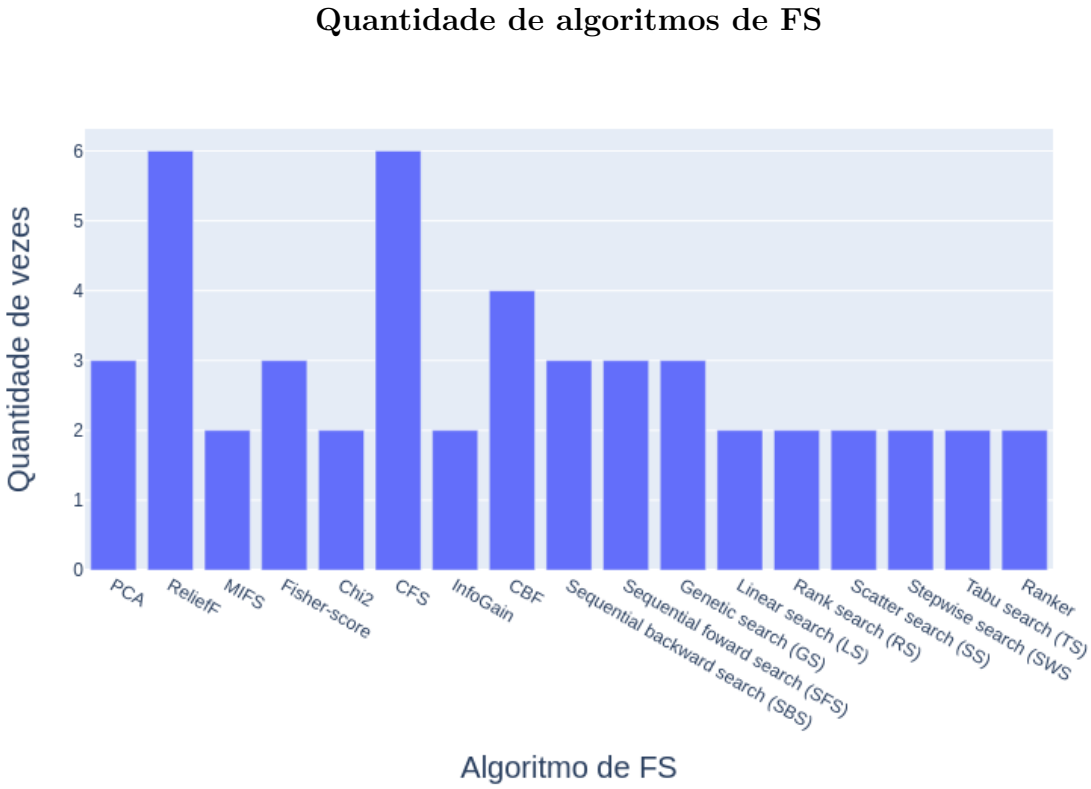

Figura 2.9: Gráfico contendo quantas vezes os algoritmos de FS foram encontrados nos trabalhos.

As outras duas classes mais frequentes são a de Imb e de IS, apresentadas respectivamente pelas Figuras 2.10 e 2.11. As técnicas de balanceamento mais comuns foram a Random Under-Sampling (RUS) seguida pelo Edited Nearest Neighbours (ENN) e pelos algoritmos Condensed nearest neighbor (CNN), All-k Edited Nearest Neighbors (AENN) e Synthetic Minority Over-sampling Technique (SMOTE). Porém, destes, somente o RUS e o SMOTE são técnicas utilizadas somente para balanceamento dos dados. Isso ocorre pois os algoritmos ENN, CNN e AENN também são encontrados para IS ou para detecção

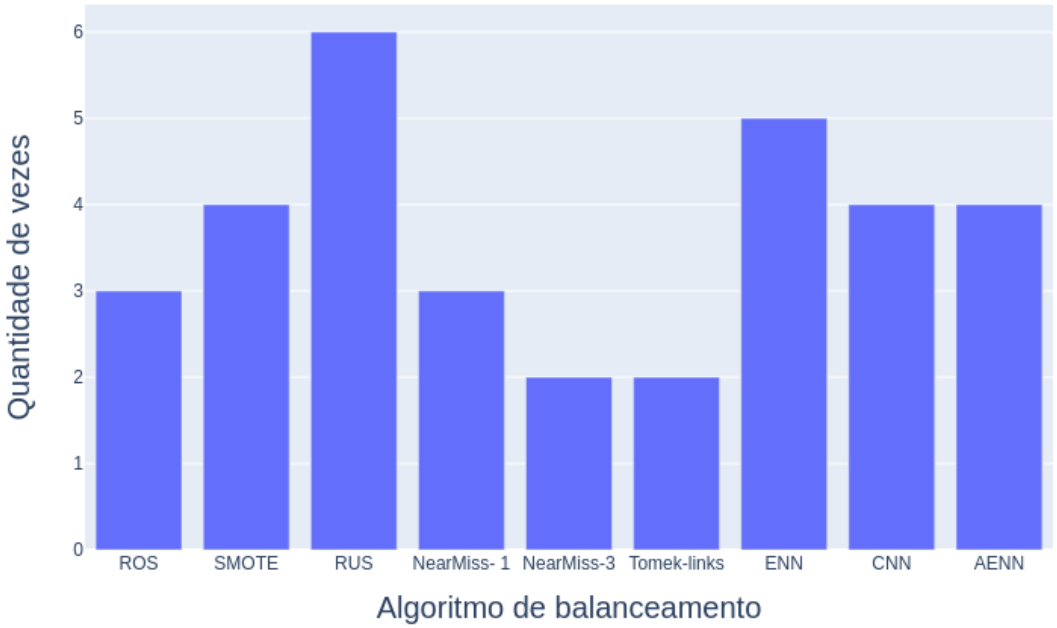

Figura 2.10: Gráfico contendo quantas vezes os algoritmos de Imb foram encontrados nos trabalhos.

de ruído, o que eleva número de ocorrências. Assim, não coincidentemente, estes três também estão presentes na Figura 2.11 sendo o ENN o mais utilizado. Entre os algoritmos aplicados unicamente como técnicas de IS, o Cpruner, IB3 e o ICF foram os mais frequentes, aparecendo em três trabalhos diferentes. Já para os algoritmos de detecção de ruído, Figura 2.12, os algoritmos High Agreement Random Forest (HARF), Static Ensemble filter (SEF), Dynamic Ensemble Filter (DEF) e Pruned SF apareceram em dois trabalhos, enquanto o AENN novamente está presente, sendo encontrado em quatro referências.

Nas outras classes de pré-processamento não foram encontrados muitos algoritmos que atingissem o critério de duas ocorrências. Para MVI, dois algoritmos apareceram duas vezes, o K-Nearest Neighbor Imputation (KNNI) e o Median or Mode Imputation (MMI). Considerando NorDis, foram encontrados três algoritmos, todos utilizados três vezes, estes foram a transformação nominal para binária, a normalização dos dados e a estandardização.

Na análise dos algoritmos, notou-se dois fatores que podem influenciar a escolha do conjunto. O primeiro seria a maior disponibilidade do algoritmos em um *framework* de pré-processamento ou de AM. Notou-se que técnicas presentes em *frameworks* como Weka, scikit-learn, KEEL ou imbalanced-learn estão mais presentes nos trabalhos. O segundo

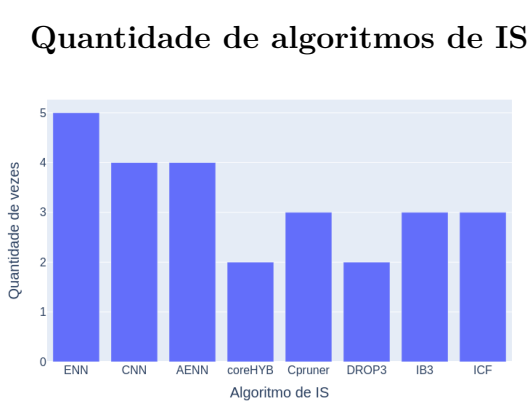

Figura 2.11: Gráfico contendo quantas vezes os algoritmos de IS foram encontrados nos trabalhos.

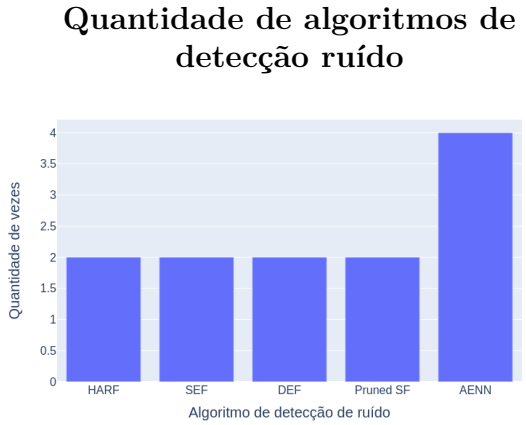

Figura 2.12: Gráfico contendo quantas vezes os algoritmos de detecção ruído foram encontrados nos trabalhos.

seria o custo computacional, pois é necessária a execução prévia do mesmo algoritmo diversas vezes para formar a meta-base, algoritmos mais custosos não são utilizados muito frequentemente, o que pode influenciar na escolha do conjunto de algoritmos a serem recomendados.

Por fim, verificou-se quais foram as métricas de avaliação mais utilizadas. Notou-se que as pesquisas trabalhavam com acurácia, precisão, *recall*, *f1-score*, área debaixo da curva ROC ou alguma métrica própria. Normalmente os métodos próprios utilizam alguma proporção das medidas mencionadas anteriormente em conjunto com algum outro dado relevante ao tema como, por exemplo, o tempo de execução do algoritmo. Estudando as métricas, não foi encontrada nenhuma que era notoriamente mais utilizada que as outras, mas como a escolha da métrica depende fortemente do que se pretende otimizar nos modelos, era de se esperar esse comportamento.

Todo o processo de avaliação dos artigos por meio das perguntas e das anotações permitiu avaliar qualitativa e quantitativamente os trabalhos sobre recomendação de algoritmos de pré-processamento. Aparentemente, existe um esforço maior da comunidade em aplicar MtL em técnicas de FS, sendo o tema mais abordado nos trabalhos encontrados. Percebe-se também que alguns algoritmos de pré-processamento são mais utilizados do que outros, mas não existe um conjunto que é sempre escolhido. Algo semelhante é observado ao verificar as MFe, nesse caso, quando utilizam características das classes simples, estatísticas ou da teoria da informação, são encontradas algumas MFes mais frequentes, mas o mesmo não ocorre nas outras classes, onde não foram encontradas características que apareçam tanto. Notou-se também uma maior constância nos modelos de AM no nível base, mas, ao trabalhar no nível meta, nenhum se destacou tão fortemente.

Capítulo 3

Revisão Teórica

Este capítulo apresenta uma revisão teórica dos conceitos que serão utilizados neste trabalho com o objetivo de introduzir ao leitor as definições e técnicas aplicadas em AM, pré-processamento de dados, MtL e recomendação de algoritmos. Este capítulo é dividido da seguinte forma: Na Seção 3.1 são introduzidos os conceitos de AM; na Seção 3.2 são apresentadas definições de pré-processamentos e como funcionam os algoritmos que serão utilizados; em seguida, na Seção 3.3 são explicados os conceitos relacionados ao MtL; por fim, na Seção 3.4 o capítulo é concluído com alguns trabalhos relacionados ao tema.

3.1 Aprendizado de Máquina

Uma das formas de definir AM é como a habilidade de um sistema se adaptar a novas situações, detectar e extrapolar padrões [1]. Porém, esta definição, mesmo correta, é muito abrangente. Assim, neste capítulo, utilizaremos uma outra definição baseada em tarefas, desempenho e experiência apresentada por Mitchell et al. (1997) [75] como: "Um programa de computador aprende a partir de uma experiência E em uma tarefa T e uma métrica de desempenho Y , se o desempenho na tarefa T medida por Y melhorar a partir da experiência E."

Nessa definição, o aprendizado, realizado a partir de uma tarefa, torna-se o objetivo a ser atingido com com a implementação do AM. Normalmente, a tarefa define como o sistema deve reagir a partir de um exemplo, sendo o exemplo um conjunto de atributos representado por um vetor $x \in \mathbb{R}^n$. Entre as tarefas mais comuns estão a classificação e a regressão. A classificação procura encontrar a função $f : \mathbb{R}^n \rightarrow \{1, ..., k\}$ sendo k a categoria que o vetor x pertence. Já no caso da regressão, o objetivo é encontrar a função $f : \mathbb{R}^n \rightarrow \mathbb{R}$ [76]. Assim, enquanto a classificação tenta modelar um problema com resultados categóricos, a regressão trabalha com predições de valores reais.

A experiência é adquirida por meio do treinamento. O tipo de treinamento que será aplicado depende do algoritmo de aprendizagem que será utilizado e do conjunto de dados que se tem disponível. Entre os principais tipos de treinamentos estão o supervisionado e o não supervisionado. Quando a aprendizagem é supervisionada, o conjunto de dados utilizado no treino são previamente rotulados, permitindo que os algoritmos melhorem seu desempenho comparando o resultado com o que está presente nos rótulos e a métrica de desempenho é escolhida de forma a aproveitar os rótulos. Na aprendizagem não supervisionada, esses rótulos não existem, assim, o algoritmo precisa aprender sem ter um atributo com objetivo para auxiliar a métrica [77]. Normalmente o treinamento supervisionado é utilizado para tarefas de regressão e classificação, enquanto o não supervisionado é comum em agrupamento [78].

As métricas de desempenho são fortemente conectadas à tarefa que se deseja solucionar, afinal, pela própria definição de AM, ela será otimizada no treinamento, sendo a forma como se mede o aprendizado realizado pelo algoritmo. Infelizmente, não existe uma métrica única que satisfaça todos os problemas de AM e a escolha da métrica influencia diretamente o resultado obtido no treinamento [79].

Um algoritmo de AM sempre procura otimizar sua métrica para resolver a tarefa da melhor forma possível. Porém, ao aplicar um algoritmo para melhorar seu desempenho ou até mesmo tornar sua execução viável, recomenda-se que se anteriormente a sua execução seja implementada um etapa de pré-processamento.

3.2 Pré-processamento de dados

Um conjunto de dados, ao ser extraído, pode estar incompleto, inconsistente, apresentar muitos ou poucos dados ou possuir ruídos requirindo, as vezes, um processamento prévio para tornar possível extrair informações relevantes dos dados [80]. Um dos motivos para essa necessidade é que um algoritmo de AM normalmente é elaborado para conjuntos distribuídos, sem dados incorretos ou faltantes e onde seus atributos são importantes [80]. Dessa forma, a etapa de pré-processamento de dados se torna fundamental ao aplicar técnicas de AM, possibilitando que um algoritmo consiga ser executado e obtenha melhores resultados.

Famili et al. (1997) [81] define os processos realizados antes da aplicação de algoritmos de AM, chamados de pré-processamento de dados, como uma transformação T que modifica um conjunto de vetores de dados brutos X' para um novo conjunto de dados X tal que $X = T(X')$ onde X mantém informações úteis e válidas para a tarefa e que possibilite a melhoria ou implementação de um algoritmo. Este processo é recomendado para: corrigir falhas que possam impedir um algoritmo de executar; compreender melhor

os dados e melhorar o desempenho do algoritmo; ou melhorar a extração de conhecimento dos dados.

É importante ressaltar que existem diversos tipos de pré-processamentos, podendo ser divididos entre preparação dos dados e redução de dados [13]. A etapa de preparação inclui a limpeza, normalização, imputação de dados faltantes e detecção de ruído [13]. Já na redução, encontramos métodos como seleção de atributos, seleção de instâncias, discretização e extração de atributos [13]. Porém, este trabalho tem como objeto de estudo somente a detecção de ruídos.

3.2.1 Detecção de ruído

Em um conjunto de dados do mundo real frequentemente encontram-se erros chamados de ruídos [82]. Os ruídos podem ocorrer nos atributos e nas classes, porém, quando presentes em classes, normalmente geram uma maior alteração no resultado dos algoritmos de AM pois um conjunto normalmente possui uma quantidade menor de classes e indicam que a instância não está corretamente rotulada. Quando o ruído ocorre nos atributos, também é gerada uma redução da qualidade dos resultados, mas, para isso, normalmente é necessária que ele apareça em mais instâncias [83, 82].

Freney e Verleysen (2013) [82] indica que os ruídos de classe aparecem de três formas diferentes:

- **Ruídos Completamente Aleatórios (RCA):** São casos em que o erro ocorre de forma independente de outras variáveis e sua distribuição ocorre de forma uniforme entre as classes, ou seja, todas as classes possuem a mesma porcentagem de ruídos. Nesse caso, temos que a probabilidade do erro $Prob_e$ é dada por $Prob_e = Prob(E = 1) = Prob(Y \neq Y')$ onde Y é a classe verdadeira da instância, Y' a classe atual da instância e E uma variável binária indicando se $(Y \neq Y')$, e, quando $Prob_e \geq 1/2$ a classe não carrega nenhuma informação útil

- **Ruídos Aleatórios (RA):** Neste caso o erro ocorre de maneira independente das outras variáveis, porém, sua distribuição não é uniforme, ou seja, a classe pode influenciar a probabilidade que o ruído aconteça. Assim, a probabilidade do erro $Prob_e$ é dada por

$$Prob_e = Prob(E = 1) = \sum_{y \in \bar{Y}} Prob(Y = y) Prob(E = 1 | Y = y)$$

  sendo $\bar{Y}$ o conjunto de classes. No RA, ainda se considera necessário $Prob_e \leq 1/2$ para que exista alguma informação relevante. Note também que o RCA é um caso

especial do RA onde a probabilidade de erros ocorre de forma uniforme entre as classes.

- **Ruídos Não Aleatórios (RNA):** Ocorre quando um ruído pode depender tanto da classe quanto do conjunto de atributos X. Assim, como atributos podem influenciar, um erro pode acontecer com mais frequência em determinada região do espaço de X.

Encontrar ruídos em conjuntos de dados não é algo trivial e muitas vezes envolve a utilização de um ou mais algoritmos de AM [84]. Entre os tipos de algoritmos para filtrar ou alterar o ruídos estão [82]: (i) *filtros de classificação*, onde é utilizado um algoritmo de classificação e as instâncias que são classificadas de forma errada acabam removidas;(ii) *filtros de votação*, onde um número de algoritmos é executado e cada um vota para eliminar ou manter as instâncias; (iii) *métodos baseados em KNN*, onde os classificadores de KNN, que são sensíveis ao ruído, são utilizados para identificá-los; (iv) métodos baseados em *ensemble* ou *boosting*, em que são utilizadas características de algoritmos específicos de *boosting* ou *ensembling*, como tendência ao *overfitting*, para encontrar e remover o ruído.

Uma lista de filtros de ruídos é apresentada por Morales et al. (2016) [85], entre eles estão:

High Agreement Random Forest (HARF) [86]: HARF utiliza o classificador Random Forest (RF) para agir como um filtro de ruído. Nesse caso, ao invés de classificar as instâncias a partir de uma maioria mínima, ou seja, mais de 50% dos votos, o algoritmo exige uma porcentagem $X > 50\%$ de concordância entre a predição e a classe majoritária. Caso essa porcentagem não seja atingida, a instância é considerada como ruído e será removida do conjunto de dados.

Generalized Edition (GE) [87]: Trata de uma variação do algoritmo ENN [88], porém, este algoritmo implementa a possibilidade de alterar a classe da instância ao invés de somente eliminar quando considerar que é um ruído. Para realizar o algoritmo, considere os $k-1$ vizinhos mais próximos da instância e um limiar k', caso a instância e seus $k-1$ vizinhos formarem uma maioria de ao menos k' a classe é removida, caso contrário altera-se a sua classe para a classe majoritária.

Dynamic Classification Filter (DCF) [89]: Neste algoritmo é aplicado uma junção de classificadores onde cada um possui um voto, os classificadores são KNN (com k = {3, 5, 7}), SVM, CART, C4.5, RF, Naive Bayes e MLP. Entre esses, são selecionados os m classificadores com maior similaridade nos resultados da predição. As instâncias que obtiverem a maioria, podendo ser absoluta ou majoritária, de classificações diferentes do real será considerada como ruído.

Outlier Removal Boosting Filter (ORB) [90]: Este filtro é uma variação do algoritmo AdaBoost [91], utilizando a característica do AdaBoost de aumentar o peso de

importância de valores atípicos para realizar uma filtragem a partir de um valor limite d. A cada iteração do algoritmo, os pesos das instâncias são alterados. Caso este peso seja maior que o limiar d, a instância é considerada como ruído e é eliminada.

Edge Boosting Filter (EDB) [92]: Assim como o ORB, este filtro utiliza o algoritmo AdaBoost para classificar uma instância como ruído. Após m interações do algoritmos, o *edge value* de cada instância é calculado, em que caso esse valor seja acima de um limiar t essa instancia é classificada como ruído. Em seguida, as instancias classificadas como ruído com maiores *edge values* são removidas até atingirem uma porcentagem limite remoções ou não existirem mais instâncias que superem o limiar t.

Hybrid Repair-Remove Filter (HRF) [93]: O HRF utiliza um conjunto de classificadores para identificar se uma instância é ou não um ruído. São utilizados quatro classificadores, sendo estes o SVM, MLP, CART, e KNN (combinação de K = 3, 5 e 7). Cada classificador vota se uma instância deve ser considerada como ruído ou não de acordo com a sua predição, caso uma instância obtenha a maioria dos votos, podendo ser absoluta ou majoritária, ela é considerada como ruído. Neste caso, o algoritmo pode alterar o rótulo da instância, removê-la ou decidir entre uma das duas de acordo com o voto do classificador KNN.

All-k Edited Nearest Neighbors (AENN) [94]: Este filtro aplica algoritmo Edited Nearest Neighbours (ENN) [88] para valores de 1 a k, onde, para cada instância são verificados os K vizinhos mais próximos e suas classes, caso a maioria das classes dos K vizinhos não possua o mesmo rótulo que a da instância atual ela será considerada como ruído. Ao final das k execuções, qualquer instância considerada como ruído no processo é removida.

PReprocessing Instances that Should be Misclassified (PRISM) [95]: Neste algoritmo são calculados 5 heurísticas: uma baseada nas classes dos mais próximas das instâncias; duas baseadas na *likelihood* das instâncias; e duas extraídas de características das folhas geradas com o algoritmo C4.5. Caso as heurísticas calculadas atinjam os pré-requisito definidos[1] ela será classificada como ruído e será removida.

3.3 Meta-Aprendizado

MtL, comumente abordado como *aprender como aprender*, é uma forma de utilizar experiências prévias para realizar tarefas semelhantes [29]. Quando aplicado em recomendações de algoritmos, o processo de MtL está fortemente ligado com a escolha do melhor algo-

[1]Uma formula é apresentada definir se o a instância está classificada incorretamente, caso necessário mais detalhes, consulte a referência do algoritmo

ritmo. Assim, para compreender melhor esse processo, é importante explicar o problema de seleção de algoritmos de Rice (1976) [27].

O problema de seleção de algoritmos de Rice (1976) [27] tem como objetivo principal selecionar o melhor algoritmos entre os disponíveis para algum problema específico. A Figura 1.1 apresenta o modelo proposto para resolver este problema com base em características, em que pode-se encontrar quatro espaços:

- **espaço do problema**: P é o conjunto de problemas envolvido de alta dimensão, incluindo as características independentes do problema que são importantes para a seleção do algoritmo e para o desempenho;
- **espaço das características**: F é um subespaço de P representando uma versão mais simples e de menor dimensão que P;
- **espaço de algoritmos**: A é o conjunto de algoritmos disponíveis para solucionar determinado problema;
- **espaço das métricas de desempenho**: Y trata-se do critério utilizado para medir o desempenho de um algoritmo em determinado problema.

Dessa forma, tem-se o seguinte processo: a partir de P podemos extrair um subconjunto $f(x) \in F = \mathbb{R}^m$ diminuindo a complexidade de P, de F é possível mapear o algoritmo $a \in A$ por meio de uma função $S(f(x))$. Idealmente, $S(f(x))$ determina o algoritmo que possui o melhor desempenho y podendo ser encontrada a com $p(A, x)$. Desse fluxo, Rice afirma que o problema de seleção de algoritmo é determinar $S(f(x))$ por um critério de seleção que pode ser, por exemplo, o problema da *melhor escolha* que procura selecionar o mapeamento $B(x)$ que gera o melhor desempenho para cada problema tal qual:

$$||p(B(x), x)|| \geq ||p(A, x)|| \forall a \in A$$

Rice (1976) [27] também apresenta outros critérios de melhor seleção, incluindo: melhor seleção para uma subclasse de problemas, melhor seleção a partir de uma subclasse de mapeamentos e melhor seleção a partir de uma subclasse de mapeamentos e problemas. Além disso, Rice enfatiza a importância de se extrair as melhores características de P, gerando assim os problemas de escolha das melhores características para um algoritmo, uma classe de algoritmos ou uma subclasse do mapeamento da seleção $S(f(x))$. Assim, o objetivo é encontrar o melhor conjunto de k características que são melhores para predizer o desempenho de um conjunto de algoritmos ou um algoritmos específico.

O problema de seleção de algoritmos foi estendido por Smith-Miles em [28], a Figura 3.1 apresenta um diagrama do *framework* proposto para suportar o problema de MtL. O

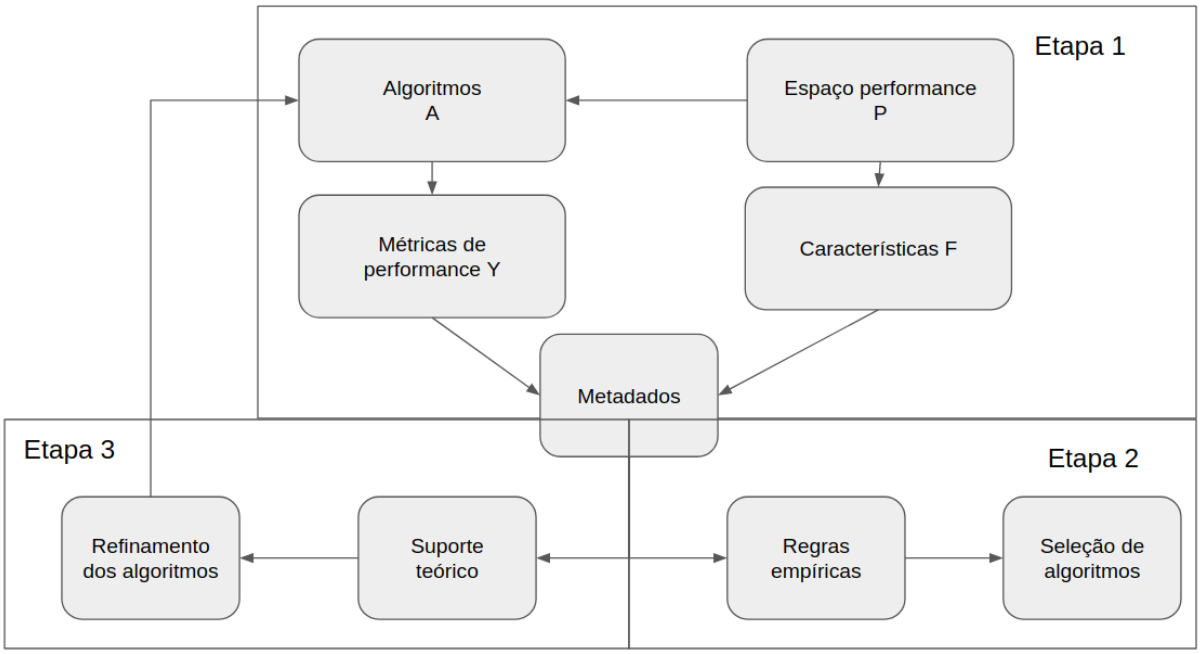

Figura 3.1: *Framework* apresentado por Smith-Miles estendendo a solução de Rice.

fluxo é dividido em três fases. Na primeira, a partir dos conjuntos P, A, Y, F é gerado um conjunto de metadados formado pelos resultados Y obtidos pelos conjuntos de algoritmos A e as características F extraídas de P. Os metadados são utilizados na fase dois como a fonte de dados para encontrar relações entre F e Y. Eles possibilitam a criação de regras empíricas ou algoritmos de ranqueamentos que tem como objetivo indicar a seleção de algoritmos. A partir do método utilizado na fase anterior, é iniciada a fase três, em que as regras ou algoritmos são analisados e refinados verificando assim seu funcionamento e ocasionalmente melhorando seus resultados.

Ao gerar o conjunto de metadados, separa-se o ambiente em dois níveis: o nível meta e o nível base. No nível base estão presentes os conjuntos P, A, Y, ou seja, são estudados o problema, os algoritmos disponíveis e os resultados obtidos por esses algoritmos. O nível meta é o estudo dos metadados, incluindo, não somente, a análise do conjunto, como os algoritmos de inferência, ranqueamento e classificação que são utilizados para extrair informações deles. Para compreender melhor esses níveis precisa-se entender o que forma cada um desses conjuntos, ou seja, o que seriam os quatro espaços definidos por Rice (1976) [27] e Smith-Miles (2009) [28] em uma solução de MtL.

3.3.1 O espaço do problema P

O espaço P representa o repositório de problemas $X \in P$ que está sendo estudado [96]. No caso de técnicas de MtL, este espaço engloba os problemas relacionados ao AM, como por exemplo a classificação, onde X representa um conjunto de dados contendo as informações do problema. Generalizando, P é um conjunto de conjuntos de dados que possui todos os problemas X.

Na prática, em soluções de MtL, para formar P juntam-se vários conjuntos de dados que serão utilizados para a aplicação de algoritmos pertencentes a A ou para se extrair as características $f(x) \in F$. Existem diversos repositórios onde é possível encontrar essas bases para formar P como, por exemplo, OpenML[2], Kaggle[3], KEEL[4] e UCI[5].

3.3.2 O conjunto de algoritmos A

Também chamado de algoritmos-base, o conjunto A é formado por todo $a(X, \lambda) \in A$, onde a é um algoritmo que procura solucionar o problema X tendo como entradas $X \in P$ e λ o conjunto de hiperparâmetros que configuram a. Em outras palavras, A é composto por todos algoritmos que abordam os problemas pertencentes a P [28]. Esses algoritmos são utilizados inicialmente para formar o conjunto de metadados a partir da métrica Y e posteriormente o espaço de busca das regras empíricas utilizadas para a seleção de algoritmos.

Para entender melhor A, considere um exemplo onde procura-se encontrar o melhor algoritmo para problemas de classificação, nesse caso, idealmente, A seria formado por todos os algoritmos de classificação. Porém, como isso é inviável, na prática, em uma aplicação de MtL, A é formado por um conjunto finito de algoritmos combinados com seus hiperparâmetros. Suponha um conjunto $A = \{a_1, a_2, a_3\}$, cada a_i possuindo seus respectivos conjunto λ_i, neste caso, para formar o conjunto de metadados, será necessário aplicar previamente todos os três algoritmos nos $|P|$ problemas gerando futuramente um espaço de busca de tamanho três. Este exemplo mostra que aplicações de MtL podem ser computacionalmente custosas para se formar um conjunto de metadados.

3.3.3 As métricas de desempenho Y

As métricas de desempenho estão diretamente relacionadas com o conjunto de algoritmos. Pode-se definir o conjunto de métricas como funções $y(a(X, \lambda)) \in Y$, isso é, a partir

[2] `https://www.openml.org/`
[3] `https://www.kaggle.com/`
[4] `http://www.keel.es/`
[5] `https://archive.ics.uci.edu/ml/datasets.php`

do resultado de algoritmos $a(X, \lambda) \in A$, calcula-se uma métrica y que permite avaliar ou comparar o desempenho de a. O resultado obtido por y é utilizado diretamente na construção do conjunto de metadados.

Continuando no exemplo de encontrar o melhor classificador para o conjunto P, para cada algoritmo a, será calculada alguma métrica $y(a(X))$, podendo ser, por exemplo, a acurácia de um algoritmo, onde em etapas futuras, busca-se encontrar o algoritmo com maior acurácia. Neste caso, o algoritmo utilizado para se extrair a métrica de desempenho é chamado de *base-learner*. Assim como em técnicas de AM, a escolha da métrica de desempenho é influenciada pela informação que deseja-se obter ao final do processo e o que se pretende optimizar na escolha dos algoritmos.

3.3.4 O conjunto de características F

Em técnicas de MtL F é chamado de MFe [96], assim como as métricas de desempenho, o conjunto F é utilizado na construção do conjunto de metadados, porém, ao invés de ser obtido a partir de A, ele é extraído de P. Rivolli et al. (2022) [31] apresenta MFe $f \in F$ como uma função $f : X \to \mathbb{R}^k$ que, ao aplicada no conjunto X, retorna um conjunto de k valores que caracterizam o conjunto e auxiliam na predição do desempenho y podendo ser representado por $f(X) = \sigma(m(h_m), h_s)$ onde , $m : X \to \mathbb{R}^{k'}$ é a medida de caracterização, $\sigma : \mathbb{R}^{k'} \to \mathbb{R}^k$ a função de compactação e h_m e h_s são seus respectivos hiperparâmetros. Ao formar o conjunto F procura-se juntar informações capazes de distinguir os algoritmos bases um dos outros ao mesmo tempo que em seu processo de extração não deve ser mais complexo que a execução dos algoritmos A e, para evitar o processo de *overffiting*, sua dimensionalidade não deve ser muito maior que a de P [30].

Normalmente F é subdividido em grupos que possuem características semelhantes, onde as classificações de grupos mais comuns são [31, 30]: (i) os simples, formado pelas características que são extraídas de forma simples, sem a necessidade de um grande custo computacional, entre eles estão valores como o número de classes, de atributos ; (ii) estatísticos, contendo valores estatísticos como correlações, médias e desvios padrões; (iii) de teoria da informação, que capturam a informação do conjunto de dados, em sua maioria são baseados em entropia (iv) baseados em modelos, onde são criadas características a partir das propriedades de modelos, como a quantidade de folhas em uma árvore de decisão, e; (v) *landmarking*, que são características a partir de estimativas de desempenho de algoritmos.

Rivolli at. al. (2022) [31] também acrescenta um grupo que engloba as outras medidas não presentes nesses cinco grupos, como as relacionadas ao tempo, baseadas em agrupamentos e distâncias, entre outras. Considerando os cinco grupos principais, as medidas mais utilizadas são as três primeiras [30], isso também pode ser percebido na análise das

características mais utilizadas presente na seção 2.3. Uma seleção de MFe comumente utilizados em MtL pode ser encontrada em [29].

Ainda no exemplo anterior do melhor classificador, um possível conjunto F pode ser formado pelo número de classes, número de atributos, entropia, média dos atributos e correlações entre classe e atributos. Esse conjunto de MFe, junto com Y é utilizado para formar o conjunto de metadados, que, na próxima etapa, servirá para formar o mapeamento $S(f(X))$.

Também é interessante notar que, assim como existem ferramentas para facilitar implementação de algoritmos de AM, por conta da grande quantidade de características que podem ser calculadas, algumas bibliotecas como `pymfe` e `mfe` [97] foram propostas para realizar a extração das características, além disso, base de dados como OpenML permitem baixar MFe de seus conjuntos de dados já calculadas previamente em seu sistema.

3.3.5 Regras empíricas

Após a formação do conjunto de metadados, são geradas regras empíricas para mapear as características extraídas e os algoritmos, produzindo assim uma recomendação. Essas regras podem ser construídas de diversas formas, sendo bem comum a utilização de técnicas de AM para gerá-las, porém, isso depende do tipo de recomendação que se deseja produzir. Brazdil et al. (2008) [30] apresentam quatro tipos de recomendações diferentes: melhor algoritmo, melhor subconjunto de algoritmos, ranqueamento de algoritmos, e estimar o resultado dos algoritmos.

No caso da escolha do algoritmo mais adequado, a recomendação torna-se um problema de classificação. Qualquer algoritmo de classificação pode ser utilizado nesse processo, onde, entre os algoritmos A, apenas um algoritmo a_i será recomendado. Neste caso, como somente um algoritmo é retornado, o usuário do sistema não terá nenhuma informação extra, como quais algoritmos tentar utilizar caso o recomendado falhe ou não atinja o resultado satisfatório.

Uma solução para isso seria apresentar o melhor subconjunto de algoritmos. Nesse caso, será retornado ao usuário um conjunto $\{a_1, a_2, ..., a_i, \}$ de algoritmos que pode solucionar o problema. Uma das formas de formar esse conjunto é estabelecer uma margem de desempenho onde todos os algoritmos que atingirem esse valor serão adicionados. Nessa abordagem, ao apresentar o subconjunto ao usuário, não existe nenhuma indicação de qual deve ser testado primeiro.

Já no caso do ranqueamento de algoritmos, esse subconjunto é apresentado com uma ordem específica, assim, apresentando mais informações ao usuário indicando não somente o melhor algoritmo como quais deveriam ser os outros melhores. Nesse caso, adaptações

de algoritmos regressão podem ser usados gerando como resultado o subconjunto de algoritmos já ordenado.

Por fim, ao estimar o resultado dos algoritmos, ao invés de apresentar uma classificação, são informadas quais os resultados que se espera para esse problema para cada algoritmo presente em A, nesse caso, a recomendação pode ser feita por várias regressões, cada uma estimando o valor de um algoritmo.

3.4 Trabalhos Relacionados

Leyva et al. (2013) [46] propõe um *framework* para a recomendação de algoritmos de IS, tentando prever, de acordo com a preferência do usuário, tanto a acurácia obtida após a aplicação do algoritmo KNN, quanto à quantidade de instâncias que serão removidas. No total, são utilizados 40 conjuntos de dados adquiridos a partir da plataforma KEEL, oito MFe, seis algoritmos de IS para serem recomendados e seis algoritmos, servindo como meta-regressores, para avaliar qual obtém o melhor desempenho. Os resultados mostraram que a abordagem apresenta um bom resultado considerando o custo computacional para a execução da recomendação.

Garcia et al. (2016) [55] realizam a seleção de algoritmos para recomendar algoritmos que tratem ruídos. Para formar a meta-base, foram utilizados 53 conjuntos de dados retirados do UCI e KEEL, nos quais foram gerados ruídos em uma quantidade entre 5% a 20% do total de instâncias. Para tratar os ruídos, foram separados seis filtros e extraídas 70 MFes de cada conjunto de dados para serem utilizados nos meta-regressores. No total, foram utilizados três algoritmos de regressão, o KNN, RF e SVM, para prever os resultados do *f1-score* obtidos pelos algoritmos de AM após a aplicação dos filtros. Ao analisar o *Mean Square Error* dos regressores, comparando-os com o melhor algoritmo de filtro geral e escolhas aleatórias dos algoritmos, Garcia conclui que o melhor rendimento foi obtido pelo RF.

Garcia et al. (2016) [56] apresenta um sistema de MtL que procura recomendar o algoritmo de detecção de ruído que obtém a melhor precisão na remoção. O estudo utiliza cinco algoritmos de redução de ruído diferentes utilizados para formar 26 combinações de *ensembles* que são avaliadas e utilizadas na recomendação. Para formar os metadados são utilizados 90 conjuntos de dados que são modificados gerando uma quantidade de 5, 10, 20 e 40 por cento de ruído de forma aleatória. Um total de 70 MFe são utilizados para realizar a classificação do melhor *ensemble* de algoritmos, e para realizar a meta-classificação são comparadas cinco técnicas de AM. Ao final, o melhor classificador foi a DT que obteve uma acurácia na classificação próxima de 75%.

Morais et al. (2016) [54] utiliza MtL para a seleção de algoritmos de subamostragem para dados desbalanceados. No processo são utilizados 29 conjuntos de dados, um total de 10 características e como meta-classificador foi utilizado o KNN. Sete algoritmos de subamostragem diferentes foram escolhidos, porém, também foram consideradas suas variações de hiperparâmetros na escolha, ou seja, cada conjunto (a_i, λ_i) é considerado como um algoritmo pelo meta-classificador, aumentando o espaço de busca e o tamanho de A. Foram utilizadas cinco métricas para avaliar os resultados após a recomendação: Acurácia; curva ROC; *f1-score*; especificidade; e NPV. Em seguidas, elas também foram comparadas com o resultado obtido por escolhas aleatórias. A recomendação obtida pelo sistema de MtL obteve resultados melhores que a escolha aleatória. O trabalho também indica que considerar os hiperparâmetros pode melhorar consideravelmente os resultados.

Em uma série de publicações, Bilalli apresenta um sistema de recomendação envolvendo vários tipos de pré-processamento, incluindo NorDis, MVI e PCA como forma de redução de dimensionalidade [60, 39, 63]. Para gerar seu conjunto de metadados foram utilizados mais de 500 conjuntos de dados extraídos da base OpenML, onde, para cada um, foram aplicados os tipos de pré-processamentos e 5 algoritmos de AM, DT, Naive Bayes [98], PART [99], regressão e IBK [100] para se extrair as métricas de desempenho. Foram analisados quais efeitos das transformações no resultado dos algoritmos mostrando que o processo pode tanto melhorar, piorar ou não representar nenhuma mudança no desempenho.

Para realizar o a recomendação por meio de MtL, nestes trabalhos, Bilalli utiliza 61 MFe, o algoritmo RF foi utilizado para a regressão, possibilitando prever os resultados, que são ordenados e apresentados ao usuário em forma de um ranque. Os resultados obtidos foram comparados com seleções aleatórias das transformações e com usuários reais, obtendo uma acurácia melhor que as escolhas aleatórias e que os usuários leigos na maioria dos casos.

Parmezan et al. (2017) [58] mostra a escolha do melhor algoritmo de seleção de características. Para isso, foram utilizados 150 conjuntos de dados, dos quais foram extraídos 21 MFe. O autor escolheu trabalhar somente com problemas de classificação e separou quatro métodos de seleção de características diferentes: Consistency-based filter (CBF) [101], Correlation-based feature (CFS) [102], InfoGain [103] e ReliefF [104]. Os dois primeiros pertencentes à classe de seleção de subconjunto de atributos e os outros dois de ranqueamento. Para escolher o melhor algoritmo, criou-se três conjuntos de metadados. O primeiro é utilizado para classificar qual tipo de classe de algoritmo deve ser utilizada. Os outros dois para indicar qual algoritmo deve ser selecionado. Ou seja, caso seja escolhido um algoritmo de *ranking* é utilizada uma meta-base, caso contrário a outra. Assim, durante todo o processo trabalha-se com classificações binárias. Como meta-classificadores

foram utilizados DT, obtendo uma média de acurácia nos conjuntos de dados testados de até 90

Parmezan et al. (2021) [33] expandiu sua pesquisa anterior [58], adicionando mais um algoritmo de FS, o wrapper subset evaluation (WSE) [105], e implementando a recomendação de hiperparâmetros para os algoritmos selecionados. Na sua abordagem, é proposta a realização de várias classificações em sequência: primeiramente selecionando o tipo de algoritmo de FS, em seguida escolhendo o algoritmo de acordo com o tipo escolhido na etapa anterior e, por fim, realizando a recomendação de seus hiperparâmetros . Para isso, foram criados nove conjuntos de metadados: um para verificar se será utilizado um filtro ou um *wrapper*, outro para definir o tipo do filtro, dois para escolher o algoritmo de filtragem e outros cinco para definir os hiperparâmetros. O projeto utilizou 213 conjuntos de dados e 161 MFe, incluindo um grupo de características baseadas em imagens e outro em grafos. Novamente, foi possível obter uma acurácia de recomendação de até 90%.

Pio et al. (2022) [106] realizam a recomendação de algoritmos de detecção de ruído tentando prever e identificar o algoritmo que gera mais ganho na métrica *f1-score*. O sistema utiliza 73 MFe extraídos de 323 conjuntos de dados diferentes que foram modificados para conter 5, 10, 20 ou 40 porcento de ruído. Foram utilizados três filtros de ruído: o HARF, ORB, e o GE, três *base-learners* para calcular a métrica de desempenho: DT, RF, e o KNN, e três regressores utilizados para formar o ranque dos algoritmos recomendados: o RF, PCT [107], e o KNN. Os resultados mostraram que foi possível gerar um ganho no desempenho escolhendo sempre o algoritmo presente na primeira posição do ranque, tendo o algoritmo de RF como o meta-regressor que obteve os melhores resultados.

Todos estes trabalhos mostram diferentes aplicações e abordagens para o problema de recomendação de algoritmos. Bilalli et al. (2019) [39] é o único que implementa um teste comparando o resultado do seu sistema com pessoas e, além disso, aborda mais de uma classe de pré-processamento. Porém, seu conjunto de recomendações possui algumas técnicas que, como enfatizado em seu trabalho, dependendo do algoritmo base, não possuem nenhum efeito no resultado final. Leyva et al. [46] mostram uma forma interessante de balancear o desempenho com a quantidade de instâncias a serem removidas, porém utilizam um conjunto de MFe pequeno que pode ser insuficiente para extrair os conhecimentos necessários para a recomendação. Somente dois trabalhos implementam a recomendação de hiperparâmetros, Morais et al. (2016) [54] e Parmezan et al. (2021) [33], porém nenhum destes trabalhos aborda a detecção de ruído.

Entre as pesquisas cujo objetivo é detecção de ruído, duas possuem uma abordagem diferente como objetivo de recomendação. Garcia et al. (2016)[55] e [56] utilizam como métrica de previsão o quão correto é um algoritmo em identificar os ruídos de um conjunto de dados. Neste trabalho, a métrica de desempenho é extraída após a execução de um

base-learner especificando anteriormente, prevendo assim os efeitos que o algoritmo irá causar no desempenho de técnicas de AM.

Este trabalho implementa duas abordagens para a recomendação de algoritmos. A primeira é uma expansão da proposta por Pio et al. (2022) [106], onde são acrescentados mais algoritmos de remoção de ruído e analisados os ganhos resultantes da inclusão de sugestões de hiperparâmetros para algumas das técnicas. A segunda abordagem é uma adaptação da proposta por Parmezan et al. (2021) [33], onde a recomendação é realizada por meio de uma sequência de meta-classificadores. Neste caso, foram alterados o conjunto de MFe e o conjunto de algoritmos, sendo adaptados para técnicas de detecção de ruído.

Capítulo 4

Metodologia

Até aqui foi apresentada a base teórica sobre o tema de recomendação de filtros de ruído de dados para AM utilizando técnicas de MtL. Neste capítulo será mostrada a metodologia do trabalho para este problema, que, por abordar MtL é dividida em duas etapas: nível base, onde realiza-se as coletas de dados, transformações e é gerada a meta-base; e nível meta, onde realiza-se a recomendação por meio de MtL e as avaliações. Neste trabalho foram utilizadas duas abordagens diferentes de MtL: a primeira aplicando técnicas de ranqueamento e a segunda executando múltiplas classificações, de forma semelhante à apresentada por Parmezan et al. [33]. O restante deste capítulo é dividido em três seções: na Seção 4.1 explica o processo realizado no nível base para a formação das meta-bases; na Seção 4.2 apresenta o processo de recomendação do filtro no nível meta bem como as suas etapas de avaliação.

4.1 Nível Base

No nível base, é realizado todo o processo necessário para a formação da meta-base, incluindo a extração e geração dos dados, aplicação dos filtros de ruído, extração das MFes e extração das métricas de desempenho. É importante enfatizar que este processo é fundamental para se obter um bom desempenho no nível meta (Seção 4.2), pois a meta-base é a fonte de informações para que os algoritmos de recomendação possam extrair conhecimento prévio [30].

Neste trabalho, são comparadas duas abordagens diferentes de recomendação. Uma utiliza um processo de ranqueamento de algoritmos como MtL, enquanto a outra é baseada na abordagem proposta por Parmezan et al. [33], onde são utilizadas múltiplas classificações em sequência para se chegar a uma sugestão. Para fins de simplicidade, estas abordagens serão referenciadas como MtL-Rank e MtL-Multi, respectivamente. Além disso, cada uma dessas abordagens possui duas variações: uma contendo a recomendação

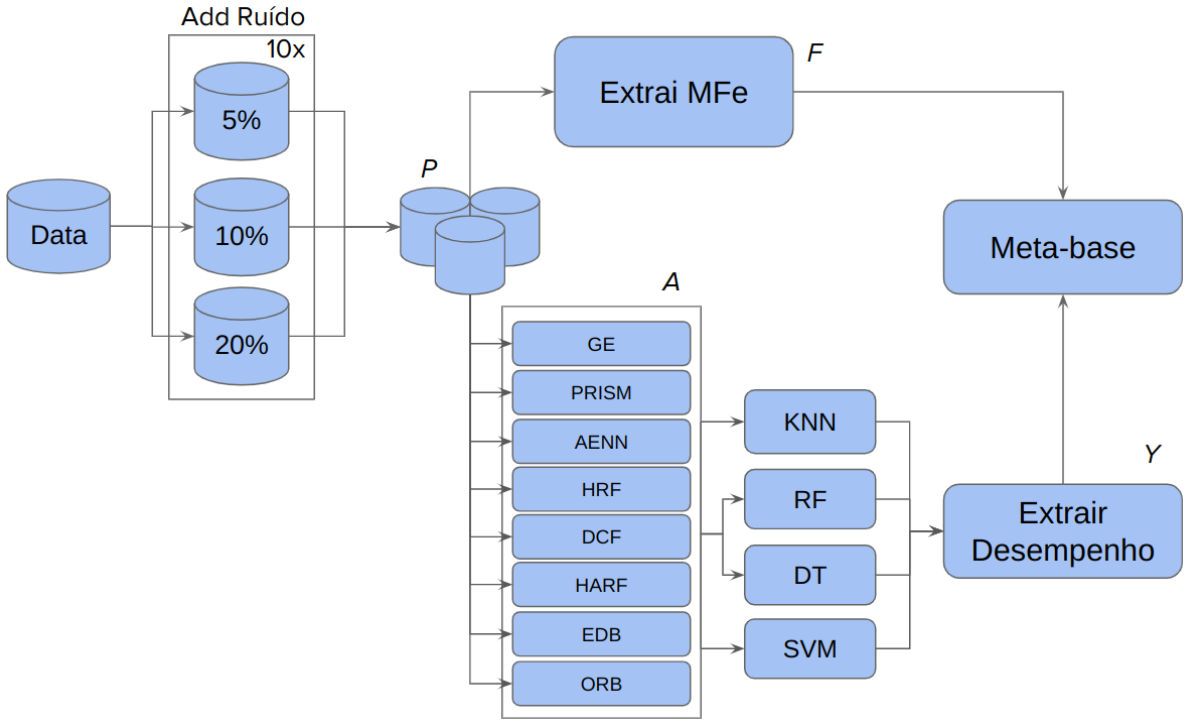

Figura 4.1: Etapas para criação da meta-base para MtL-Rank.

somente dos algoritmos e outra expandida contendo recomendações de hiperparâmetros considerados importantes para as técnicas de pre-processamento. Nessa seção, serão explicados os processos de criação da meta-base para o MtL-Rank e MtL-Multi.

4.1.1 Meta-base para MtL-Rank

O processo de criação da meta-base pode ser ilustrado pelo diagrama presente na Figura 4.1. Neste trabalho, definiu-se que todos os dados utilizados seriam tabulares, o que limita o escopo de técnicas de pré-processamentos a serem utilizadas. Além disso, com as informações extraídas a partir da revisão sistemática apresentada no Capítulo 2, notou-se que entre os tipos de pré-processamento encontrados, técnicas de redução de ruído eram as menos abordadas. Assim, foi definido a recomendação de algoritmos de detecção de ruído como o objetivo de estudo.

Após selecionar a categoria dos dados e definir detecção de ruído como o tipo de pré-processamento a ser recomendado, inicializa-se a etapa de extrair os conjuntos de dados e aplicar as transformações necessárias para cada um deles. Nesse processo, é possível trabalhar com dois tipos de dados: os reais e os artificiais. Idealmente, seria interessante trabalhar com conjuntos de dados reais ou que façam parte de *benchmarks* frequentemente utilizados em pesquisas semelhantes. No entanto, ao realizar a revisão sistemática, não foi

encontrado nenhum conjunto padrão. Além disso, técnicas de MtL normalmente precisam de um conjunto de dados variado para serem treinadas, o que dificulta a utilização de dados completamente reais.

Desta forma, escolheu-se gerar o ruído de forma sintética tanto em conjuntos reais quanto artificiais, facilitando assim o controle dos testes. Para extrair os conjuntos de dados, utilizou-se a plataforma OpenML, que permite filtrar os dados de acordo com as limitações desejadas, como quantidade de classes, quantidade de instâncias, se possui dados faltantes, entre outros. Escolheu-se extrair somente os conjuntos com quantidade de instâncias entre 100 e 20000 e com número de atributos variando entre 3 e 100, limitando o tamanho e a dimensionalidade dos conjuntos de dados. Além disso, para simplificar a avaliação, só foram considerados conjuntos com duas classes e sem dados faltantes.

Os conjuntos de dados obtidos passam por uma etapa de pré-processamento simples onde os atributos categóricos são convertidos em numéricos, criando uma variável *dummy* para cada um de seus valores. Ao término da conversão, os conjuntos que tiverem mais de 200 atributos são descartados devido sua alta dimensionalidade. Por fim, as classes com valores categóricos são transformadas em valores binários. Esse processo é necessário para assegurar que todos estejam aptos para a extração das MFes, a serem executados pelos filtros de ruído e posteriormente pelos algoritmos de AM (*base-learners*) utilizados para extrair a métrica de desempenho.

Em seguida, inicia-se a etapa de adição de ruído. O objetivo dessa etapa é garantir que existam ruídos para serem removidos pelos filtros de ruído. Assim, são alterados os valores das classes de forma completamente aleatória. Esta alteração é realizada em 5%, 10% ou 20% das instâncias da mesma forma como encontrado no trabalho [56]. Como este processo é realizado de forma aleatória, é possível que a alteração da classe ocorra nos limites das classes, o que pode alterar os resultados dos filtros de ruído. Desta forma, para cada porcentagem de ruído, são gerados 10 conjuntos de dados diferentes. Como trabalha-se com três valores diferentes, para cada conjunto de dados são gerados 30 conjuntos sintéticos com ruídos. Ao aplicar este processo em todos os conjuntos extraídos, gera-se o espaço do problema P, que representa os conjuntos de dados com ruído dos quais serão aplicados os algoritmos e extraídas as MFes.

Com o conjunto P construído, inicia-se a escolha dos filtros que formam o conjunto A de algoritmos a serem recomendados. No processo de escolha procurou-se selecionar abordagens diferentes de detecção de ruído referentes a filtragem das amostras. Como apresentado na Figura 4.2, foram separados oito filtros, sendo três pertencentes ao grupo de votação: HARF [86]; o HRF [93]; DCF [89], três baseados em distância: o GE [87]; o AENN [94] e o PRISM [95], e dois baseados em *ensemble* o EDB [92] e o ORB [90]. A classificação dos filtros foi realizada de acordo com as classes propostas por Freney e Verleysen

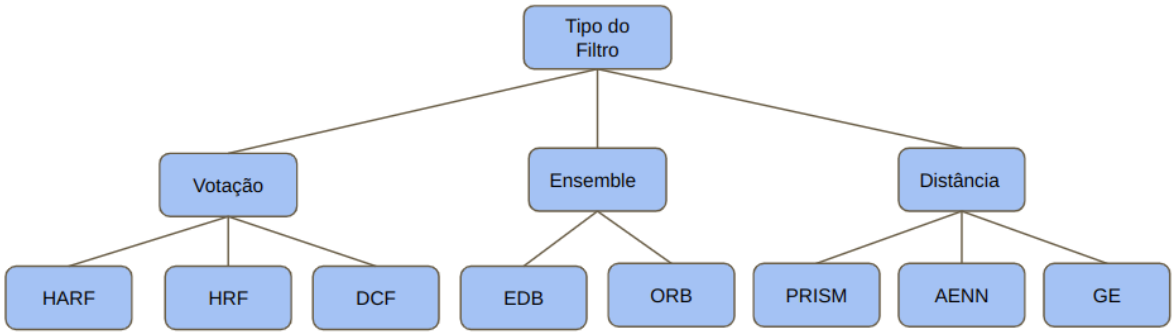

Figura 4.2: Tipos de algoritmos de ruído.

(2013) [82] e, para executar os filtros, foi utilizada a biblioteca `NoiseFiltersR`[85].

O conjunto de algoritmos A também suporta variações de hiperparâmetros, ou seja, $(a_i, \lambda_i) \in A$, dessa forma, cada variação de hiperparâmetro em um filtro é considerado com um outro elemento deste conjunto. A Tabela 4.1 apresenta quais foram as configurações utilizadas e, em negrito, estão os valores padrões da biblioteca `NoiseFiltersR` que são considerados quando não se realiza a sugestão dos hiperparâmetros. Por limitações computacionais, no máximo um hiperparâmetro é variado por filtro, além disso, no caso dos algoritmos DCF e HRF nenhuma variação de parâmetros foi aplicada por serem algoritmos mais custosos computacionalmente. Como notação, ao referenciar qualquer variação de hiperparâmetro de um filtro de ruído, ele sera seguido pelo valor do parâmetro que foi utilizado, por exemplo, AENN9 indica que o algoritmo AENN foi executado com seu parâmetro $k = 9$, caso o filtro seja executado com seus valores padrões, ele poderá ser referenciado como somente AENN.

Após a execução de cada um destes filtros, é gerada uma nova base de dados da qual é possível extrair as métricas de desempenho e verificar quantitativamente o ganho adquirido, formando a parte Y da meta-base. Neste trabalho, para medir o desempenho obtido pelos algoritmos, foi utilizado o *f1-score* calculado a partir da execução dos quatro *base-learners*: KNN [108], RF [109], DT [110] e SVM [111]. Esses algoritmos foram executados seguindo as configurações padrões presentes na biblioteca `Scikit-Learn`.

Extraídas as métricas de desempenho, para completar a meta-base, realiza-se a extração das MFes. As características são extraídas utilizando a biblioteca `pymfe` [97], o que possibilita calcular MFes simples, estatísticas, de informação, baseadas em modelos e *landmarking* [31]. Ao agrupar o conjunto F com o Y adquire-se na meta-base para o processo MtL-Rank. É importante ressaltar que, para cada *base-learner* utilizado para se extrair um conjunto de métricas de desempenho, é gerada uma meta-base diferente.

Algoritmo	Hiperparâmetros
HARF	AgreementLevel = **70**; 75; 80; 85; 90 por cento Número de árvores = 500 nfolds = 10
HRF	Votação consensual Remoção hibrida (remove ou repara a instância)
DCF	nfolds = 10 m = 3 Votação consensual
EDB	Quantidade de iterações = 15 percentual de remoção = 0.05 limiar = 5; 10; **15**; 20
ORB	Quantidade de iterações = 20 limiar d = 3; 7; **11**; 15; 19
PRISM	Não se aplica
AENN	k = 3, **5**, 7, 9, 11
GE	k = 3, **5**, 7, 9, 11 kk = k/2

Tabela 4.1: Tabela contendo os algoritmos e seus seus respectivos hiperparâmetros utilizados, os valores em negrito representam o que é aplicado quando a abordagem não realiza sugestão de hiperparâmetros.

4.1.2 Meta-base para MtL-Multi

O processo de criação da meta-base para a abordagem MtL-Multi possui suas semelhanças com a meta-base para MtL-Rank. De fato, a formação dos conjuntos P e F são idênticas nas duas abordagens. Entretanto, a MtL-Multi utiliza uma série de classificações em sequência, em que a cada etapa se define qual deverá ser a próxima recomendação. A Figura 4.3 apresenta as etapas de recomendação implementadas nesta abordagem. Na primeira etapa, seleciona-se qual o tipo de filtro é mais adequado para o conjunto. O resultado desta etapa define qual classificador deve ser executado na segunda etapa, onde se seleciona o filtro de ruído. Por fim, na última etapa, escolhe-se os hiperparâmetros do filtro. É importante ressaltar que esta última etapa pode simplesmente não ser implementada, resultando em um sistema que não realiza a escolha de hiperparâmetros.

Considerando todas as três etapas, são utilizados até nove meta-classificadores (representados por MtL-Multi-n na Figura 4.3), e cada meta-classificador necessita de sua própria meta-base. A Figura 4.4 representa um diagrama de como é formada cada uma das três meta-bases utilizadas na segunda etapa da MtL-Multi. A partir do conjunto P, são executados os algoritmos A_D, A_V e A_E, representando os conjuntos de filtros baseados em distância, votação e *ensemble*, respectivamente. Em seguida, são executados os algoritmos de AM do qual é extraída a métrica de desempenho utilizada para a definir

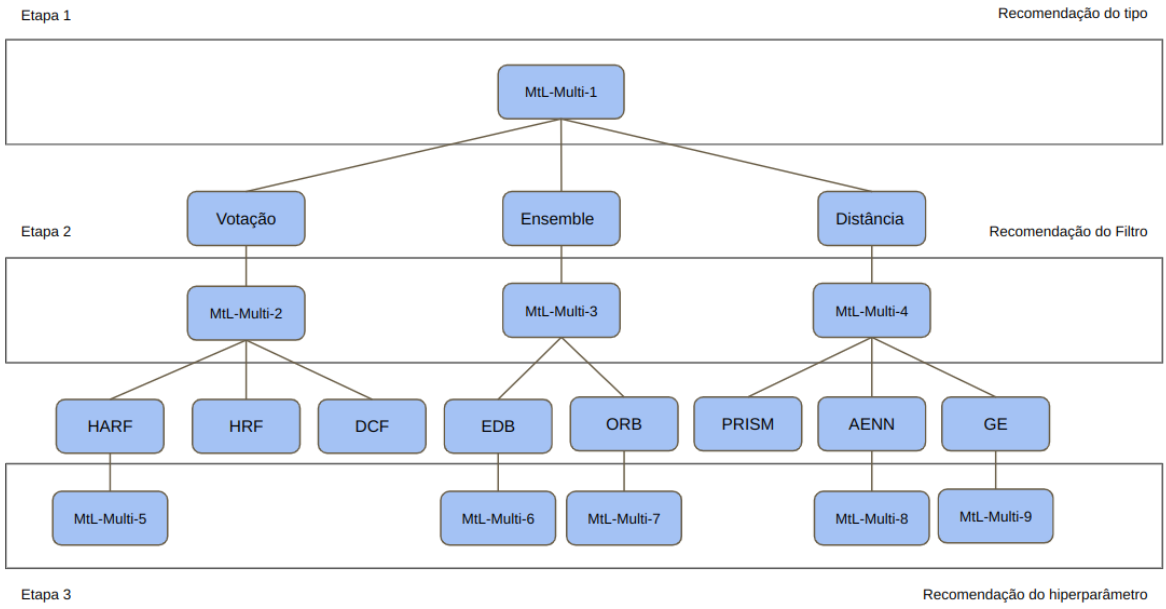

Figura 4.3: Esquema indicando quais MtL são utilizados durante a MtL-Multi.

qual é o melhor filtro. Tanto a métrica de avaliação quanto os conjuntos de algoritmos de AM utilizados nesta etapa são os mesmos dos apresentados na abordagem MtL-Rank. Por fim, as métricas são comparadas entre seus respectivos tipos e é definido o filtro com melhor desempenho. Como a métrica é o *f1-score*, seleciona-se o filtro que gerou o maior valor após a execução de seu *base-learner*. A lista dos filtros selecionados em conjunto com o conjunto F formam cada meta-base.

O processo para a formação das meta-bases nas outras duas etapas é semelhante. Na Etapa 1, onde é recomendado o tipo de filtro a ser utilizado, todos os filtros de ruído disponíveis são executados, seus resultados são comparados e o tipo do filtro que possui o melhor desempenho é selecionado e agrupado com o conjunto F. Já na Etapa 3, as variações dos hiperparâmetros são executadas de acordo com a Tabela 4.1 e para cada um, o hiperparâmetro que gerar o melhor desempenho é agrupado com o conjunto F.

4.2 Nível Meta

Da mesma forma como as abordagens se diferem na criação da meta-base, no nível meta cada uma possui um modo de ser implementada e avaliada. A maior diferença entre ambas é a forma como realizam a recomendação, pois, enquanto MtL-Rank apresenta um ranqueamento do resultado para representar as melhores escolhas de algoritmos, MtL-Multi retorna somente um algoritmo selecionado. Assim, MtL-Rank implementa um

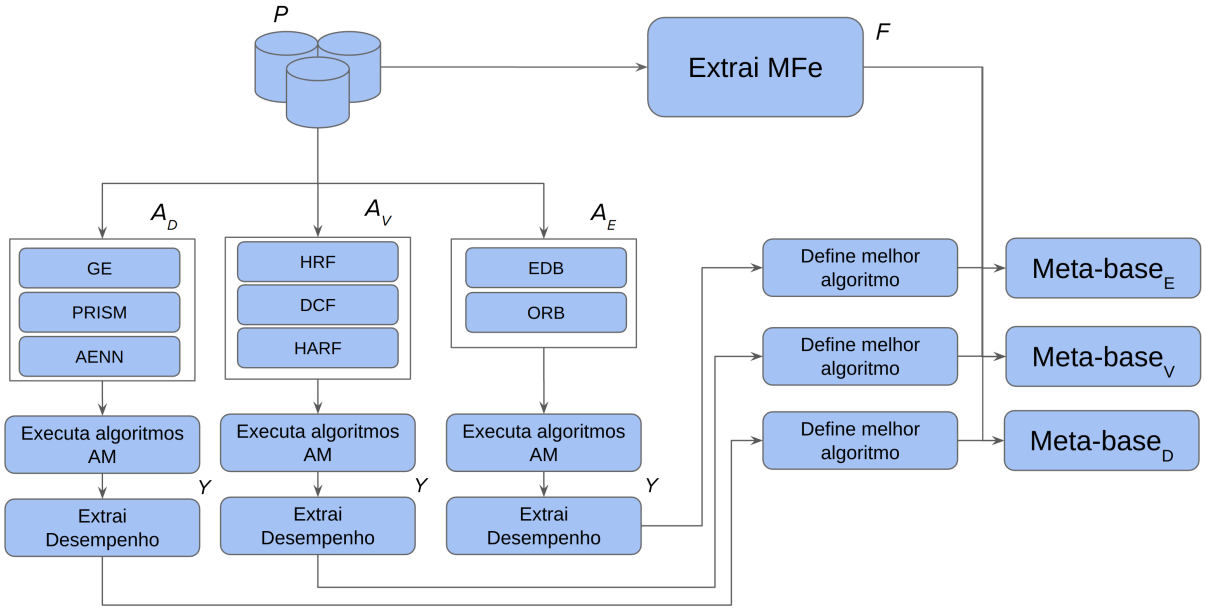

Figura 4.4: Diagrama para criação das meta-bases da segunda etapa da MtL-Multi.

meta-ranqueador para realizar a sugestão, enquanto que a MtL-Multi utiliza vários meta-classificadores no processo.

4.2.1 Nível meta para MtL-Rank

A Figura 4.5 representa um fluxograma simplificado de como a recomendação é realizada e avaliada para a MtL-Rank. A partir da meta-base, aplicam-se algoritmos de regressão para se realizar a recomendação. O resultado da regressão passa por um processo para ser transformado em um ranqueamento, que é submetido a duas formas de avaliação: a do nível base e do nível meta.

O processo de recomendação pode ser realizado por diferentes algoritmos. Neste trabalho, são escolhidos três que serão comparados posteriormente. Com o intuito de utilizar tipos de algoritmos de tipos variados, selecionou-se um algoritmo de *boosting*, outro baseado em distância e um de *bagging*: o GB [112], KNN [108] e RF [109], respectivamente. É importante ressaltar que, como alguns conjuntos de dados são sintetizados a partir do mesmo conjunto, para evitar interferência no treinamento dos meta-ranqueadores, durante a validação dos dados utiliza-se uma variação do método *Leave-One-Out* [113], onde todos os dados da meta-base que são derivados do mesmo conjunto de dados originais são separados e validados conjuntamente de forma a não comprometer o treinamento.

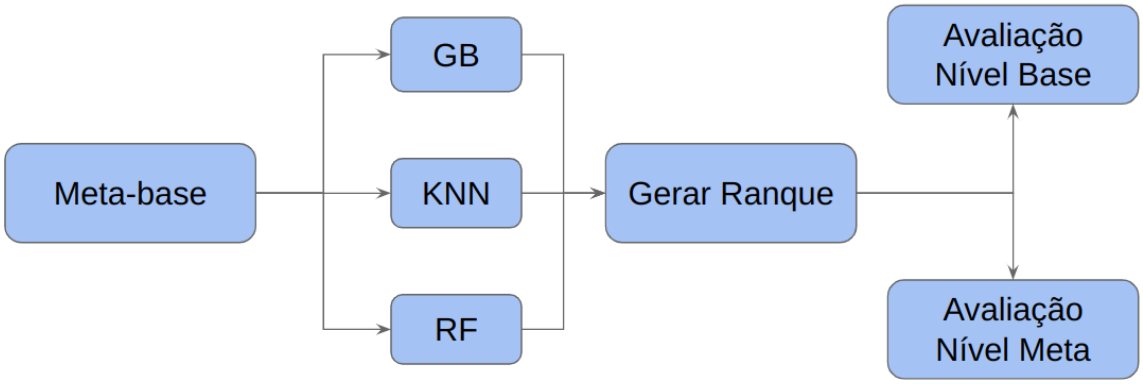

Figura 4.5: Etapas do nível meta para MtL-Rank.

Para transformar o resultado de regressão em um ranqueamento, segue-se o mesmo processo em todas as três abordagens. Os valores obtidos são ordenados de forma crescente pois a métrica predita representa o *f1-score* e procura-se a técnica a maior métrica. Em seguida, o filtro que obtiver o maior resultado entre os listados é rotulado como o 1º, o segundo melhor como 2º e assim por diante até todos serem rotulados. Em caso de empate, ambos são classificados com o menor rótulo.

Em seguida é iniciada a etapa de avaliação, que é realizada no nível base e meta. No nível base, os resultados obtidos por todas as abordagens de meta-ranqueadores são comparados com a escolha de um único filtro como *baseline*, possibilitando verificar se a métrica de desempenho melhora com a utilização dos algoritmos de pré-processamento quando segue-se a recomendação primeira do sistema.

No nível meta, verifica-se a acurácia de cada um dos filtros de ruído utilizados como meta-ranqueadores. Como o resultado obtido no processo é um ranqueamento, para medir sua acurácia é necessário determinar quando a recomendação está correta ou não. Para isso, utilizou-se a definição de que, caso o melhor filtro estivesse entre os K melhores colocados no ranqueamento, a recomendação estaria correta. Dessa forma, os resultados com os valores de $K = 1$ (Top-1) e $K = 3$ (Top-3) são analisados e novamente comparados com a acurácia obtida quando se escolhe os filtros de *baseline* que são os com melhor desempenho na métrica *f1-score* ou na acurácia.

Por fim, para avaliar a qualidade do ranque retornado pelo meta-ranqueador, é utilizado o coeficiente de correlação de Spearman [114]. Esta correlação é calculada entre o ranque ótimo dos pré-processamentos e as obtidas pelos meta-ranqueadores, verificando não somente as melhores posições, mas todas as posições como um conjunto.

4.2.2 Nível meta para MtL-Multi

A abordagem MtL-Multi realiza a recomendação por meio de uma sequência de classificadores, podendo utilizar até três classificadores para resultar na sugestão, como apresentado na Figura 4.3. Assim como ocorre na MtL-Rank, vários algoritmos podem ser utilizados para realizar essa tarefa e, para fins de comparação, foram selecionados os mesmos três algoritmos de AM selecionados na etapa de regressão da MtL-Rank, porém em suas respectivas versões de classificação (novamente com seus parâmetros padrões definidos pela biblioteca `Scikit-Learn`).

Os resultados desta abordagem também são submetidos às avaliações de nível base e nível meta. Assim como feito durante a MtL-Rank, os resultados são comparados com *baselines* para confirmar que existe uma vantagem ao realizar a recomendação. No nível base, verifica-se se os filtros recomendados geram um ganho na métrica de desempenho em comparação com o *baseline*. No nível meta, compara-se as acurácias obtidas pelas recomendações, porém, como é utilizado um meta-classificador, o cálculo da acurácia é feito conferindo o filtro obtido pela recomendação com o esperado. Caso os filtros de ruídos sejam os mesmos, a recomendação será considerada correta.

Capítulo 5

Resultados

Neste capítulo, são apresentados os resultados obtidos a partir da implementação da metodologia proposta. Com o objetivo de averiguar a viabilidade das técnicas de MtL para a recomendação de filtros de ruído, inicialmente, são verificados os efeitos de suas aplicações após executar os algoritmos de AM. Em seguida, implementam-se as duas abordagens propostas de MtL - MtL-Rank e MtL-Multi - cada uma apresentada em duas variações: uma que recomenda somente algoritmos e outra que também recomenda os hiperparâmetros do filtro de acordo com a Tabela 4.1. Essas variações permitem verificar se o aumento da complexidade do problema com a adição dos hiperparâmetros gera algum ganho para a solução proposta. Por fim, as abordagens são comparadas verificando qual delas obtém o melhor resultado. O restante deste capítulo é dividido em duas partes: na Seção 5.1, são apresentados os experimentos realizados juntamente com os seus resultados; em seguida, na Seção 5.2, compara-se os resultados das duas metodologias apresentadas.

5.1 Experimento

O primeiro passo para elaborar o experimento é a formação do conjunto P, pois este é a base para extração de F e Y, além de ser o mesmo tanto para MtL-Rank quanto para MtL-Multi. Os conjuntos de dados utilizados para formar P são todos tabulares, sem dados faltantes, contendo duas classes, entre 100 e 20000 instâncias, entre 3 e 100 atributos e foram extraídos da plataforma OpenML. Em seguida, os conjuntos que possuíam os atributos categóricos passaram por uma etapa de pré-processamento onde esses atributos foram transformados em variáveis *dummy*. Como esse processo normalmente aumenta a dimensionalidade do conjunto, aplicou-se um segundo filtro onde os conjuntos que tiverem mais de 200 atributos seriam descartados, resultando em um total de 358 conjuntos de dados. A lista dos conjuntos utilizados está disponível no Anexo II.

Para garantir que existam ruídos nos dados utilizados, em cada conjunto de dados foi adicionado ruído de classe artificialmente e de forma completamente aleatória. O processo gerou um total de 30 novos conjuntos contendo 5%, 10% ou 20% de ruído, 10 conjuntos para cada porcentagem listada, totalizando 10740 conjuntos de dados com ruídos de classes.

Em seguida, inicia-se a etapa de extração das MFe e da execução dos filtros de ruído. O conjunto de MFe foi calculado utilizando a biblioteca `pymfe`, onde, para cada um dos 10740 conjuntos de dados, foram computadas 97 características. A lista das MFe utilizadas está presente no Anexo III.

Os filtros que foram executados estão presentes na Tabela 4.1, foram selecionados oito algoritmos diferentes, 27 se consideradas suas variações de hiperparâmetros. Para executa-los se utilizou a biblioteca `NoiseFiltersR`. Cada filtro de ruído retorna um conjunto de dados no qual são aplicadas as técnicas de AM utilizadas para extrair o *f1-score*, métrica escolhida para se avaliar o desempenho. Com Y e F calculados, é possível formar as meta-bases, no entanto, antes, avaliou-se os efeitos dos filtros do ruído ao aplicar os *base-learners*.

5.1.1 Análise dos filtros de ruído

Neste experimento, foram utilizadas quatro técnicas de AM diferentes: DT[110], RF[109], KNN[108] e SVM[111]. Como cada *base-learner* resulta em uma métrica de desempenho, todas foram analisadas separadamente. O objetivo principal desta análise é saber quais são os melhores filtros de ruído, verificando se existe diferença de desempenho entre eles, quantificando essa diferença e verificando a sua distribuição em uma forma de ranque.

Nem sempre os filtros de ruído geram ganhos nas métricas de desempenho. É comum que, em alguns casos, sejam removidas instâncias importantes para a classificação, resultando em uma redução do desempenho. A Figura 5.1 apresenta a quantidade de vezes que cada filtro resultou em um efeito positivo ou negativo para a métrica de desempenho quando comparado com a execução do algoritmo de classificação sem o filtro. Os resultados estão ordenados de forma decrescente, onde o primeiro algoritmo é o que possui maior frequência de resultados positivos. Na mesma figura, também está presente a quantidade de vezes que o resultado foi negativo. No caso, o resultado também é considerado negativo caso o *base-learner* falhe durante a execução.

Analisando a Figura 5.1, notam-se as semelhanças e divergências ao aplicar diferentes *base-learners*. O algoritmo com maior quantidade de resultados positivos é o HARF70, enquanto o com menor quantidade é o ORB19. Todos os filtros possuem uma quantidade significativamente maior de resultados positivos do que negativos, o que indica sua eficiência para essa métrica de desempenho. Geralmente, os filtros de *ensemble* utilizados

Figura 5.1: Gráfico contendo a quantidade de filtros que geraram um resultado positivo ou negativo nos *base-learners*.

apresentam a menor quantidade de resultados positivos, o que pode ser devido a eles realizarem mais remoções do que os outros [82]. Os algoritmos HARF, AENN e DCF estão sempre entre os com maior número de resultados positivos, ou seja, quando aplicados, eles geram ganhos na etapa de classificação.

Ao avaliar os filtros de ruído, também é importante quantificar o tamanho do ganho que eles geram. A Figura 5.2 mostra a soma dos ganhos resultante da aplicação dos filtros na métrica *f1-score* para cada *base-learner*, sendo possível perceber quais geram mais ganho na métrica. Neste caso, o melhor é o DCF, superando os outros como o HARF e AENN que obtiveram uma maior quantidade de resultados positivos, como apresentado na Figura 5.1. Isso indica que, além de ter uma boa taxa de positividade, este filtro consegue ganhos maiores.

Também é observado que a técnica de AM afeta o resultado final do ganho. Alguns

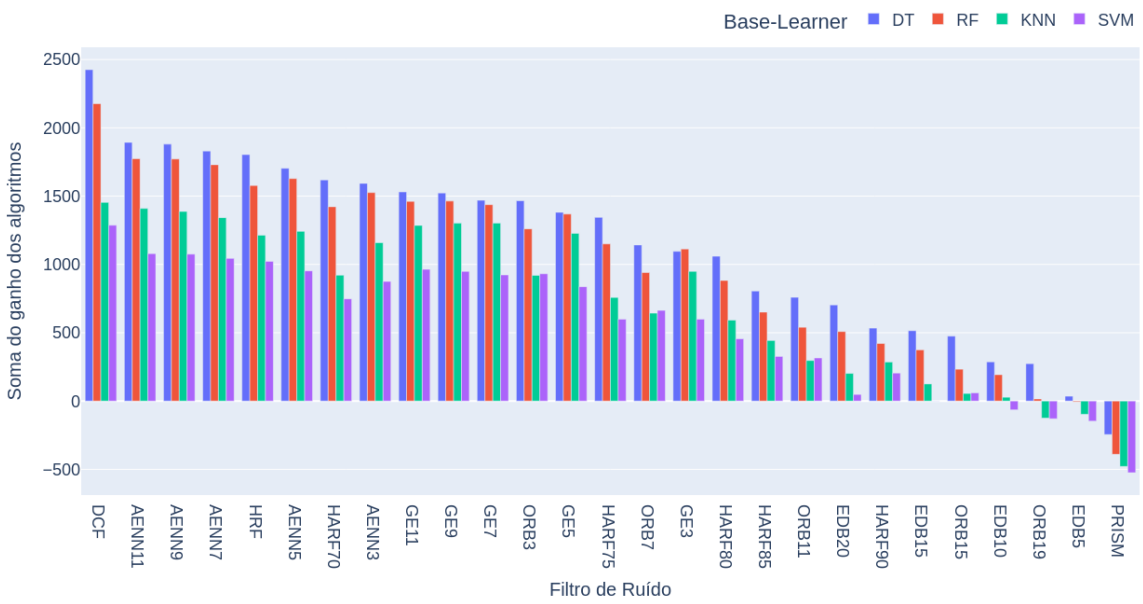

Figura 5.2: Gráfico contendo a soma do ganho obtido na métrica *f1-score* em todos os conjuntos de dados após a aplicação dos filtros de ruído.

filtros apresentam melhor desempenho em determinado *base-learner*, como o HARF70, que é melhor do que o GE5 quando aplicado com o DT ou o RF, enquanto, obtém resultados piores com os outros dois. Em geral, os filtros resultam em um desempenho melhor com o DT e pior com o SVM. Percebe-se também que o pior sempre foi o PRISM.

Para complementar a análise de magnitude de ganho, é importante saber o número de vezes onde filtro é o melhor ou pior para cada *base-learner*. Para isso, foi elaborado um ranque contendo os resultados obtidos por cada conjunto de dados e contou-se quantas vezes cada filtro apareceu em cada posição. Caso dois ou mais filtros gerem o mesmo ganho, ambos são classificados com a mesma posição no ranque, o que faz com que nem sempre existam 27 posições em todos os conjuntos de dados. As Figuras 5.3, 5.4, 5.5 e 5.6 apresentam um mapa de calor contendo a quantidade de vezes que cada filtro aparece em cada posição do ranque para os *base-learners* DT, RF, KNN e SVM, respectivamente.

Os resultados obtidos variam de acordo com a técnica de AM utilizada. É possível perceber que a os filtros formam grupos semelhantes quando comparados DT com RF ou KNN com SVM. Considerando os filtros mais frequentes nas menores posições, a grande diferença está no DCF, sendo o segundo mais frequente na primeira posição do ranque para DT e RF e o sexto para KNN e SVM. Além disso, é evidente que o ORB e suas variações de hiperparâmetros são os que aparecem com mais frequência como os geradores de mais ganho, porém, também aparecem com uma frequência elevada nas últimas 10 posições.

Mapa de calor contendo da posição de cada filtro no ranque com o base-learner DT

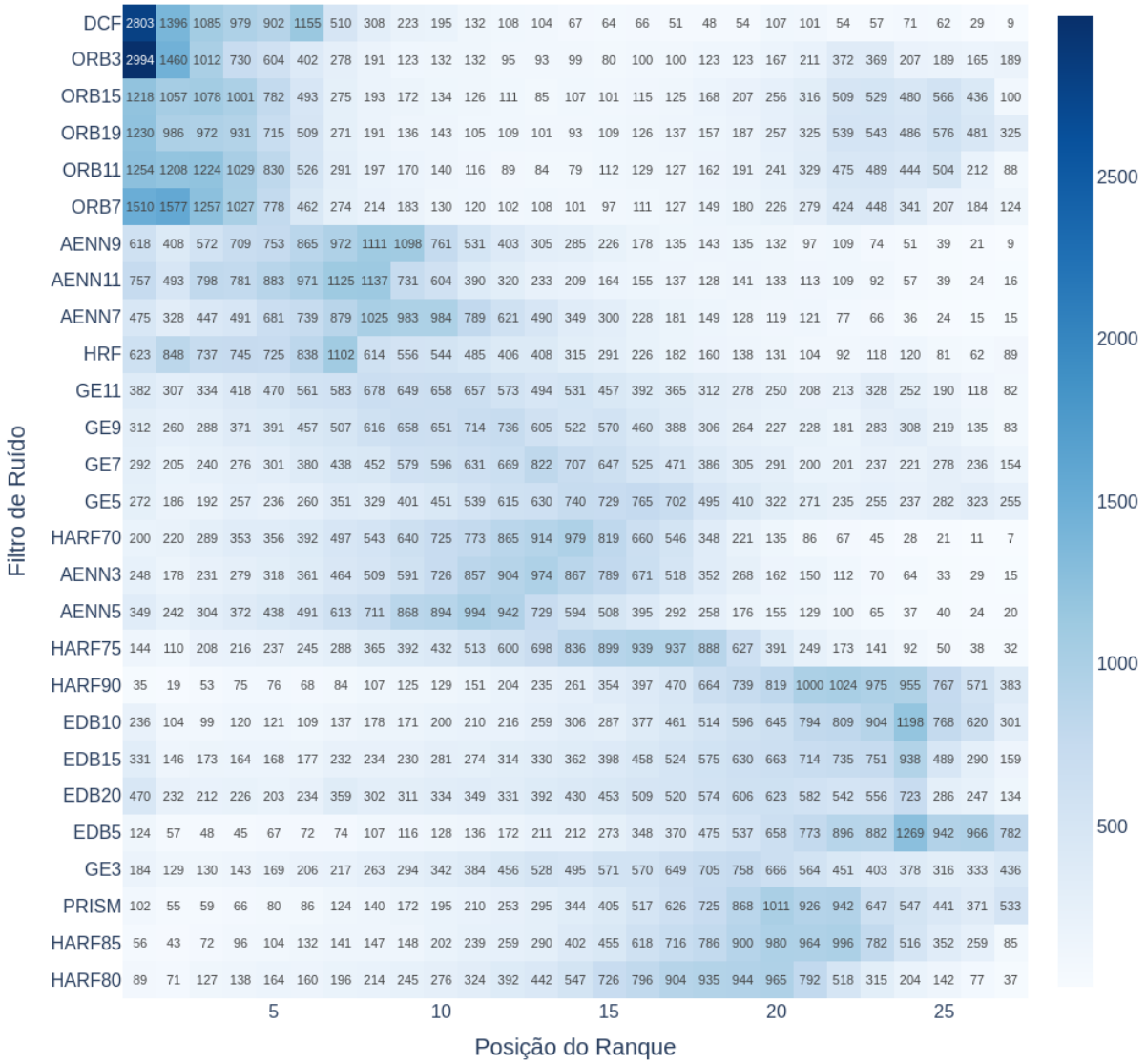

Figura 5.3: Mapa de calor contendo a quantidade de vezes que cada algoritmo com seu respectivo hiperparâmetro foi classificado em cada posição quando ordenados em um ranque a partir da métrica de desempenho *f1-score* extraída com o *base-learner* DT.

Mapa de calor contendo da posição de cada filtro no ranque com o base-learner RF

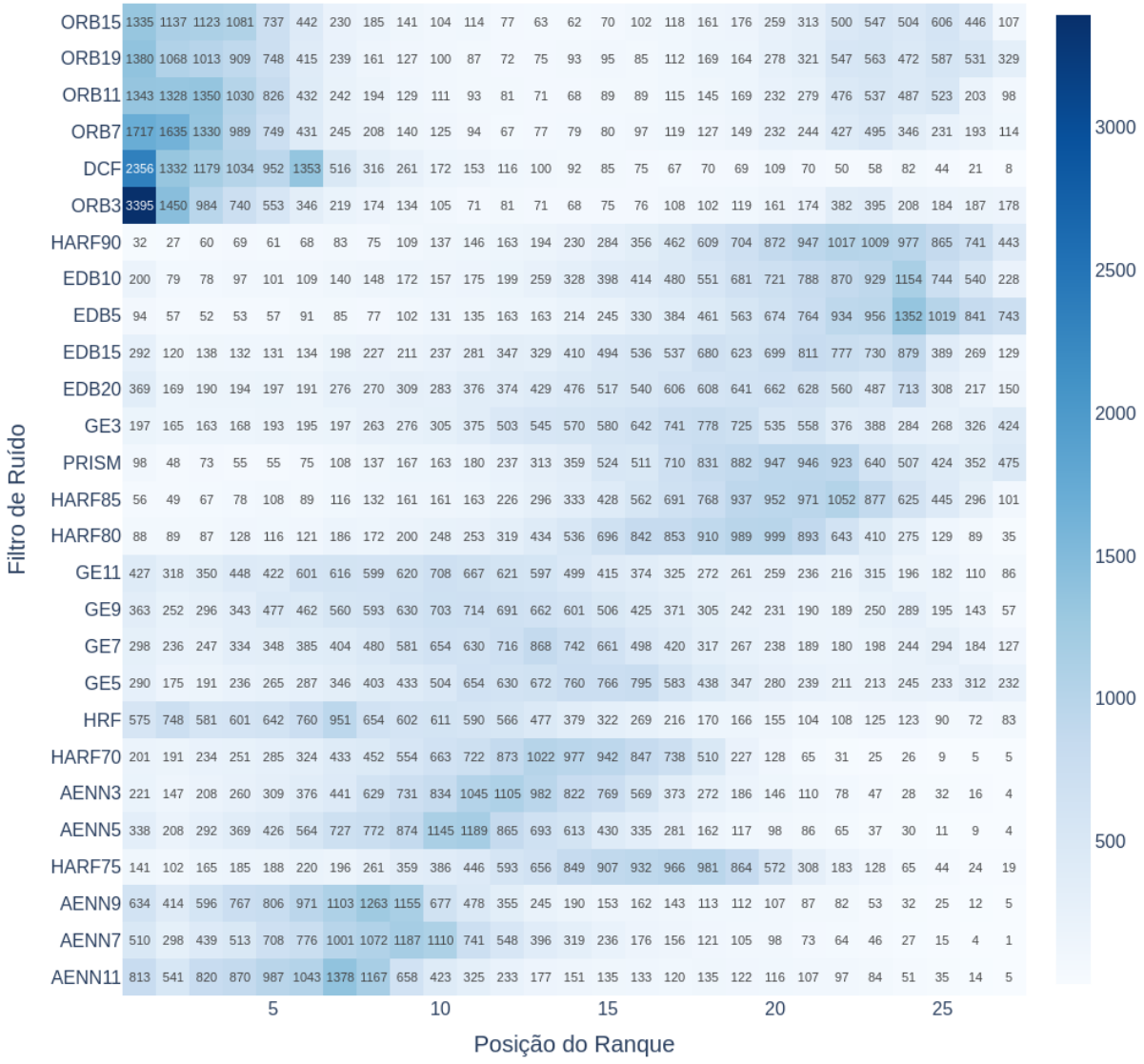

Figura 5.4: Mapa de calor contendo a quantidade de vezes que cada filtro com seu respectivo hiperparâmetro foi classificado em cada posição quando ordenados em um ranque a partir da métrica de desempenho *f1-score* extraída com o *base-learner* RF.

Mapa de calor contendo da posição de cada filtro no ranque com base-learner KNN

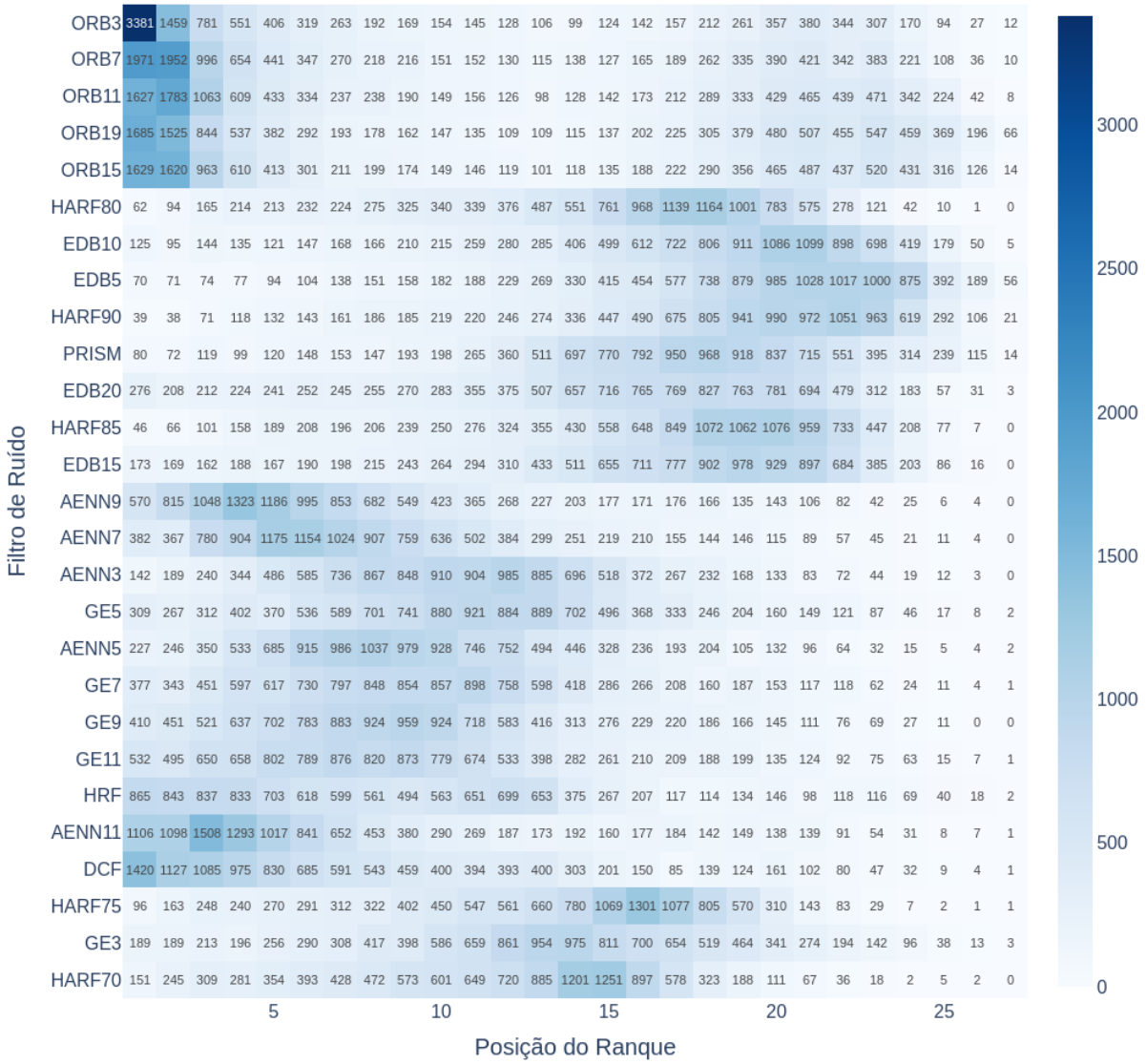

Figura 5.5: Mapa de calor contendo a quantidade de vezes que cada filtro com seu respectivo hiperparâmetro foi classificado em cada posição quando ordenados em um ranque a partir da métrica de desempenho *f1-score* extraída com o *base-learner* KNN.

Mapa de calor contendo da posição de cada filtro no ranque com o base-learner SVM

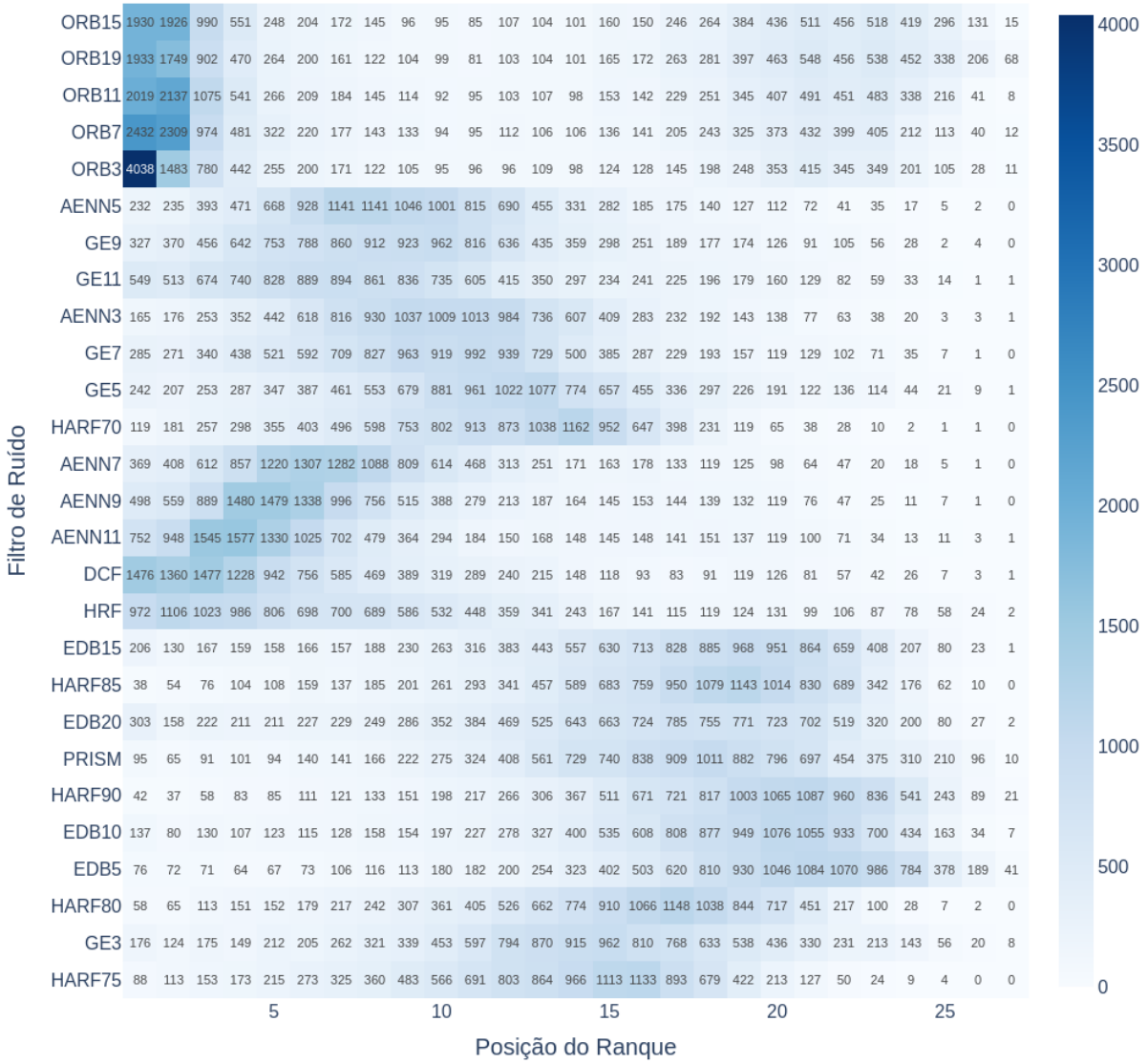

Figura 5.6: Mapa de calor contendo a quantidade de vezes que cada filtro com seu respectivo hiperparâmetro foi classificado em cada posição quando ordenados em um ranque a partir da métrica de desempenho *f1-score* extraída com o *base-learner* SVM.

A inconstância encontrada no algoritmo ORB explica o fato de que, mesmo obtendo bons resultados em uma quantidade considerável de execuções, ao verificar o ganho total dos filtros na Figura 5.2, nenhum deles se destaca. Isso mostra que o alto índice de ocorrências negativas percebido na Figura 5.1 afeta seu desempenho geral, o que indica que um sistema de recomendação do melhor filtro seria útil para o caso do ORB.

Figura 5.7: Mapas de calor contendo a quantidade de vezes que cada filtro foi classificado em cada posição quando ordenados em um ranque a partir da métrica de desempenho *f1-score* extraída com os *base-learners* DT, RF, KNN e SVM quando não são consideradas as variações de hiperparâmetros.

Até agora, toda a análise foi realizada mostrando a distribuição dos filtros de ruído, mas neste trabalho também é verificado se a recomendação de hiperparâmetros gera algum ganho quando comparada com a recomendação apenas dos filtros. Assim, se faz necessária uma análise das distribuições destes utilizando somente suas configurações padrões. A Figura 5.7 mostra os mapas de calor para os quatro *base-learners* contendo a quantidade de vezes que cada filtro aparece em uma classificação considerando somente os oito algoritmos com suas configurações padrões, ou seja, sem considerar os hiperparâmetros.

Diferentemente do que acontece considerando os hiperparâmetros, as distribuições são mais semelhantes nas técnicas de AM. Nesse caso, o DCF se destaca, frequentemente sendo

o melhor ou o segundo melhor algoritmo. Além disso, é fácil perceber que o PRISM e EDB são os piores, enquanto que o HARF, AENN e GE aparecem mais frequentemente em regiões centrais do mapa. Note que padrões semelhantes também estão presentes no mapas de calor onde se considera os hiperâmetros, porém, de uma forma menos concentrada.

Considerando todas as avaliações de desempenho, nota-se uma vantagem do DCF em relação aos outros, sendo o que em média resulta em maior ganho de desempenho e tendo uma taxa de resultados positivos mais elevada, além de aparecer com mais frequência nas primeiras posições da classificação. As avaliações mostram que os filtros não estão distribuídos igualmente quanto ao desempenho, havendo grupos que raramente são classificados entre os melhores. A grande variação de desempenho em algoritmos como o ORB indica que uma ferramenta de recomendação pode ser útil, evitando a utilização de filtros que possuem desempenho negativo.

5.1.2 Abordagem MtL-Rank

Após verificar o impacto dos filtros de ruído no conjunto P, iniciou-se o processo de formação da meta-base para a MtL-Rank. Com ambos os conjuntos F e Y já calculados, implementou-se a metodologia proposta para criar a meta-base juntando estes dois conjuntos. Foi criada uma meta-base para cada *base-learner*, resultando em quatro meta-bases diferentes. O mesmo processo foi realizado para o conjunto de algoritmos sem considerar variações de hiperparâmetros.

A abordagem MtL-Rank retorna o resultado da recomendação em um formato de ranque. Para isso, utiliza-se um regressor para prever a métrica de desempenho que seria obtida após a execução dos filtros. Para fins de comparação, foram utilizados três regressores diferentes: RF, KNN e GB. Em seguida, é gerado um ranque dos resultados obtidos por cada regressor.

Para avaliar os resultados da recomendação, escolheu-se verificar o ganho total na métrica de desempenho obtido pela abordagem e a acurácia das recomendações. Como o resultado está em formato de ranque, é necessário decidir qual filtro escolher entre as opções dadas. Para verificar o ganho, resolveu-se sempre selecionar o primeiro posicionado no ranque.

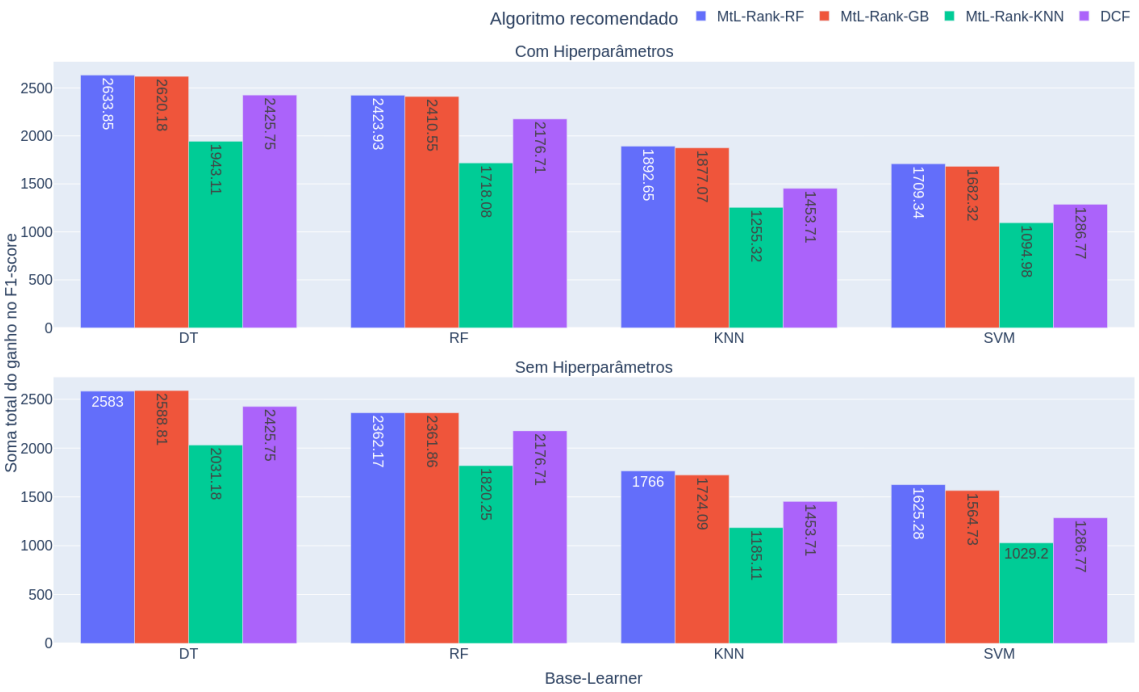

Figura 5.8: Gráfico contendo a soma do ganho obtido após aplicar as abordagens MtL-Rank com e sem a recomendação de hiperparâmetros comparada com o *baseline*.

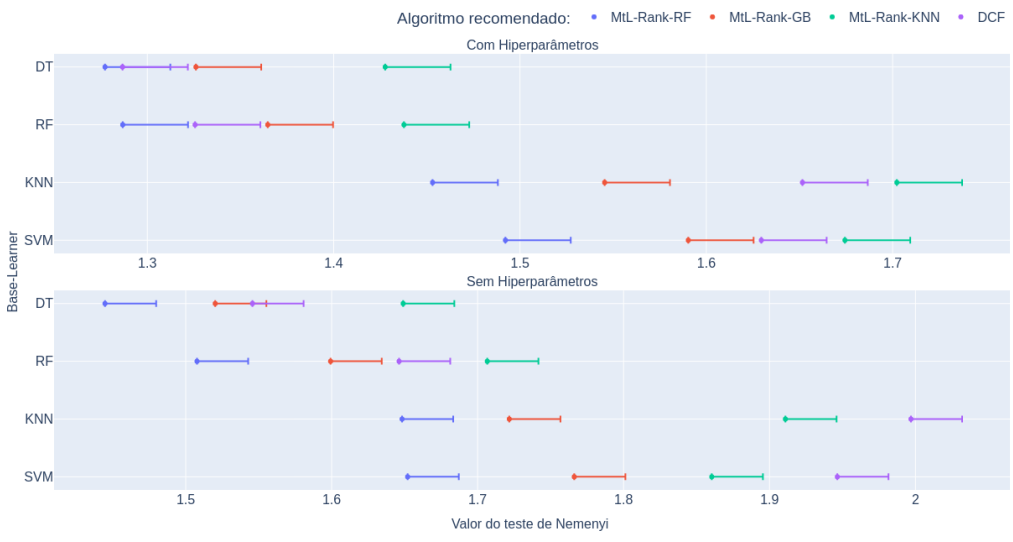

Figura 5.9: Gráfico dos resultados do teste de Nemenyi nos ganhos da abordagem MtL-Rank. Os menores valores são melhores e são considerados significativamente diferentes quando não há sobreposição de linhas.

A Figura 5.8 apresenta o resultado da soma total do ganho obtido ao usar o filtro recomendado pela abordagem MtL-Rank, comparado ao *baseline* que sempre é o filtro DCF, escolhido por ter o melhor desempenho de acordo com a Figura 5.2. No gráfico, é mostrado o resultado com ou sem a recomendação de hiperparâmetros. Observa-se que a recomendação com RF e GB sempre é superior ao *baseline*. Algo que não ocorreu com o KNN que sempre obteve resultados inferiores as outras abordagens. Comparando os resultados com e sem hiperparâmetros, verifica-se que o ganho sempre é um pouco maior quando elas são considerados no processo de escolha. Além disso, nos dois casos, mostrou-se que a recomendação é quase superior a opção de sempre escolher o *baseline*. Para validar se os resultados são significativamente diferentes, utilizou-se o teste de Friedman e Nemenyi [115]. A Figura 5.9 apresenta os resultados obtidos pelo teste, onde cada ponto representa o valor obtido no teste de cada recomendador e suas respectivas linhas a distância crítica da média. Nota-se que na maioria dos resultados são significativamente diferentes, sendo as únicas exceções o MtL-Rank-RF e o DCF quando utilizado o algoritmo DT como *base-learner* e são considerados os hiperparâmetros dos filtros, ou entre o MtL-Rank-GB junto com o DCF quando os hiperparâmetros não são utilizados.

Acurácia do Top-1 dos meta-ranqueadores comparados com os *baselines*

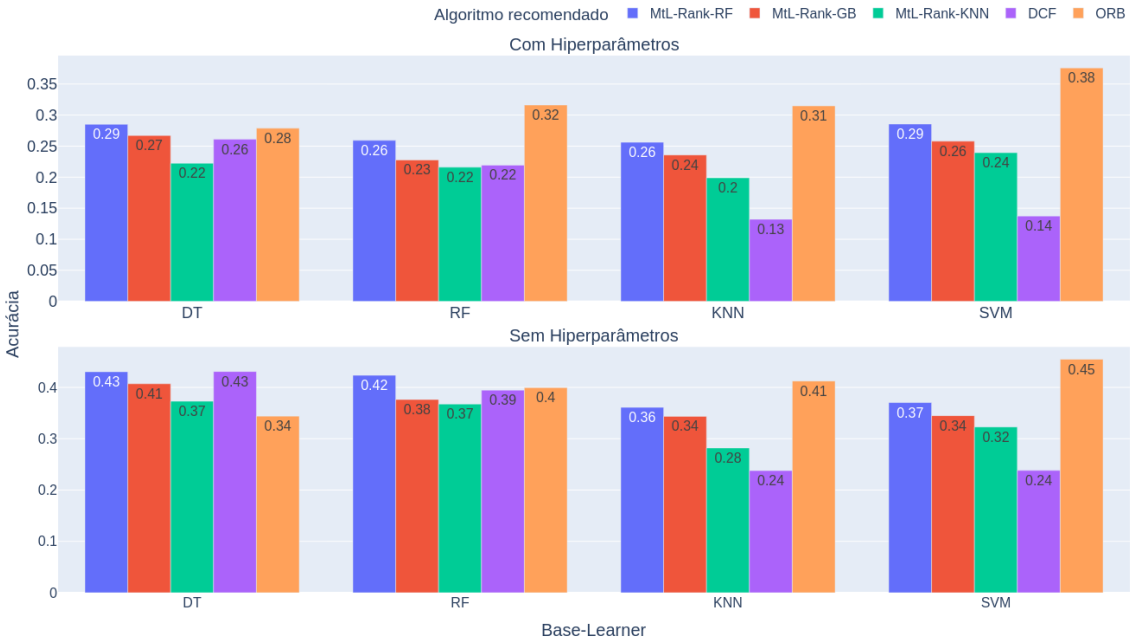

Figura 5.10: Gráfico contendo a acurácia do Top-1 de cada meta-ranqueador comparados com dois *baselines* quando considerados a recomendação com e sem hiperparâmetros.

Para avaliar a acurácia da recomendação, foi necessário escolher quando considerar a recomendação como correta. Desta forma, foram definidos dois critérios: *Top-1* e *Top-*

3 (explicados na Seção 4.2.1). As Figuras 5.10 e 5.11 apresentam a acurácia com base nesses critérios. Nesta avaliação, foram usados dois algoritmos como *baselines*, o DCF e o ORB, sendo que, para a abordagem com hiperparâmetros, foi escolhido o ORB3 por ter a melhor acurácia entre suas variantes.

Acurácia do Top-3 dos meta-ranqueadores comparados com os *baselines*

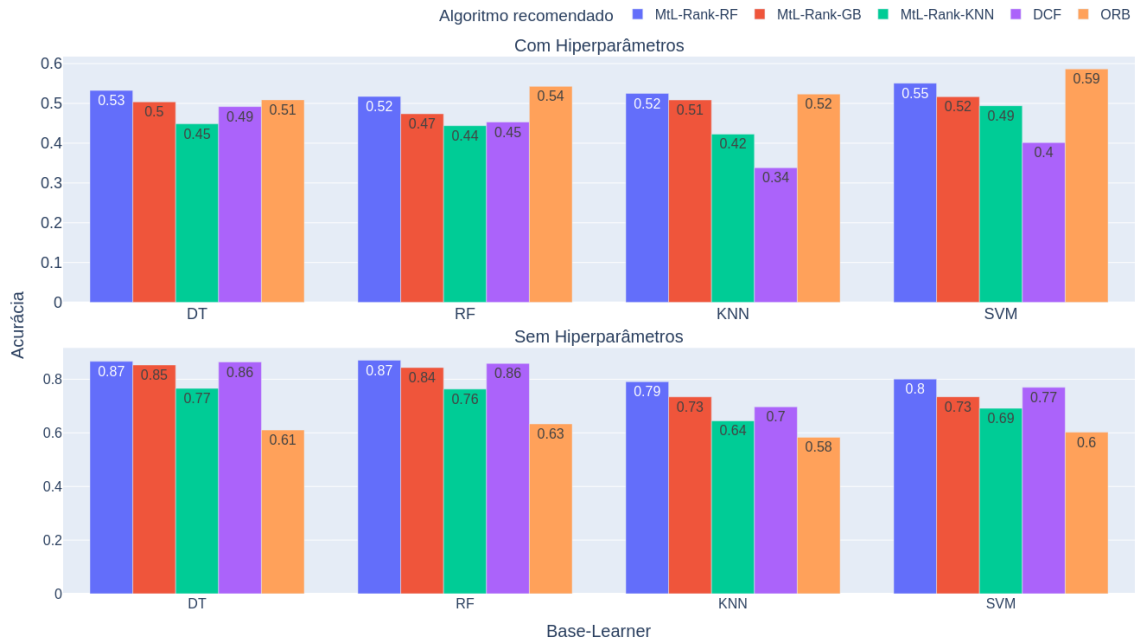

Figura 5.11: Gráfico contendo a acurácia do Top-3 de cada meta-ranqueador comparados com dois *baselines* quando considerados a recomendação com e sem hiperparâmetros.

Percebe-se que, ao comparar a acurácia, a recomendação da MtL-Rank-RF é sempre melhor ou igual à obtida pelo DCF em todos os casos verificados. Entretanto, ao comparar com a acurácia do ORB, essa tendência não se repete. Quando considera-se a abordagem MtL-Rank-RF e o resultado do Top-1, a recomendação só apresenta resultados melhores para os *base-learners* DT e RF e não são considerados os hiperparâmetros. Já no caso do Top-3, a sugestão consegue superar os *baselines* na maioria dos casos, sendo sempre a melhor opção quando considerando a implementação sem recomendação de hipeparâmetros. Vale notar que, em contraste com o observado no ganho, ao analisar a acurácia, as recomendações conseguem desempenhos melhores sem os hiperparâmetros.

Por fim, calculou-se as correlações de Spearman entre os ranques pela MtL-Rank e a ordem ótima, verificando o grau de semelhança entre o ambos. Os resultados das correlações estão presentes nas tabelas 5.2 e 5.1, mostrando a média das 10740 correlações considerando a recomendação com e sem hiperparâmetros, respectivamente. Em todos os

casos, os resultados mostram uma correlação maior nos resultados da MtL-Rank-RF e a MtL-Rank-KNN sempre apresenta os piores valores.

	DT	KNN	RF	SVM
MtL-Rank-RF	0.56 ± 0.31	0.54 ± 0.33	0.57 ± 0.31	0.51 ± 0.35
MtL-Rank-GB	0.55 ± 0.29	0.52 ± 0.33	0.55 ± 0.30	0.47 ± 0.35
MtL-Rank-KNN	0.39 ± 0.33	0.39 ± 0.36	0.40 ± 0.33	0.39 ± 0.36

Tabela 5.1: Tabela contendo a correlação de Spearman entre a classificação obtida pelo sistema de MtL e a classificação ideal considerando a recomendação de hiperparâmetros.

	DT	KNN	RF	SVM
MtL-Rank-RF	0.57 ± 0.33	0.55 ± 0.35	0.58 ± 0.33	0.52 ± 0.37
MtL-Rank-GB	0.56 ± 0.32	0.51 ± 0.36	0.56 ± 0.33	0.47 ± 0.37
MtL-Rank-KNN	0.43 ± 0.37	0.40 ± 0.40	0.44 ± 0.37	0.40 ± 0.40

Tabela 5.2: Tabela contendo a correlação de Spearman entre a classificação obtida pelo sistema de MtL e a classificação ideal sem considerar a recomendação de hiperparâmetros.

5.1.3 Abordagem MtL-Multi

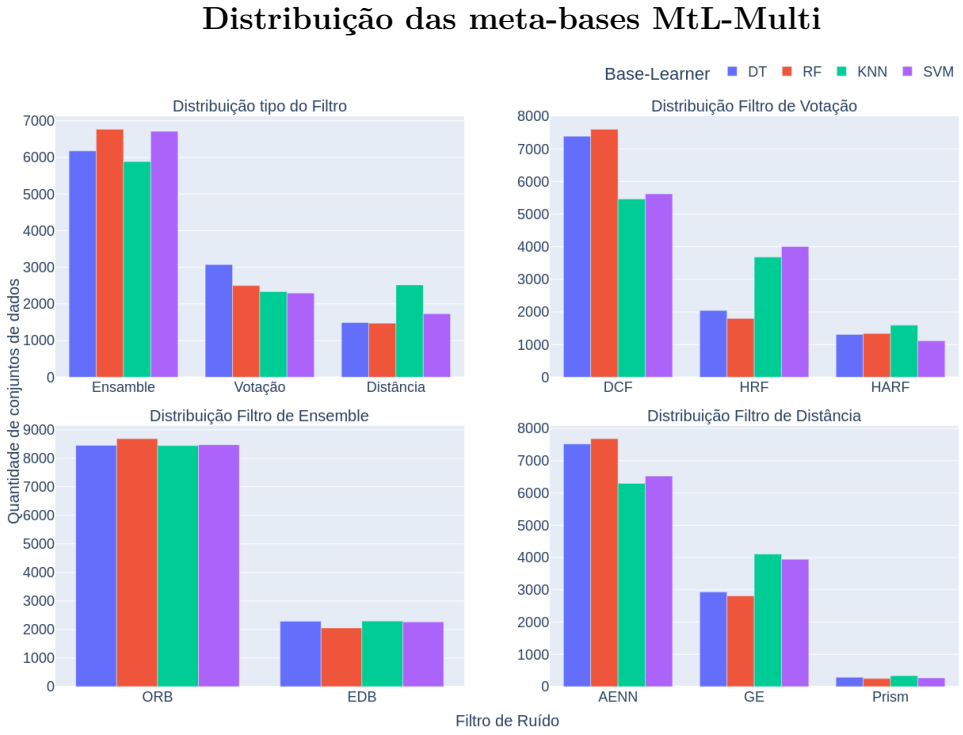

Figura 5.12: Gráficos contendo as distribuições das classes nas meta-base referentes ao tipo de algoritmo, aos algoritmos de votação, de *ensemble* e de distância quando os hiperparâmetros são considerados.

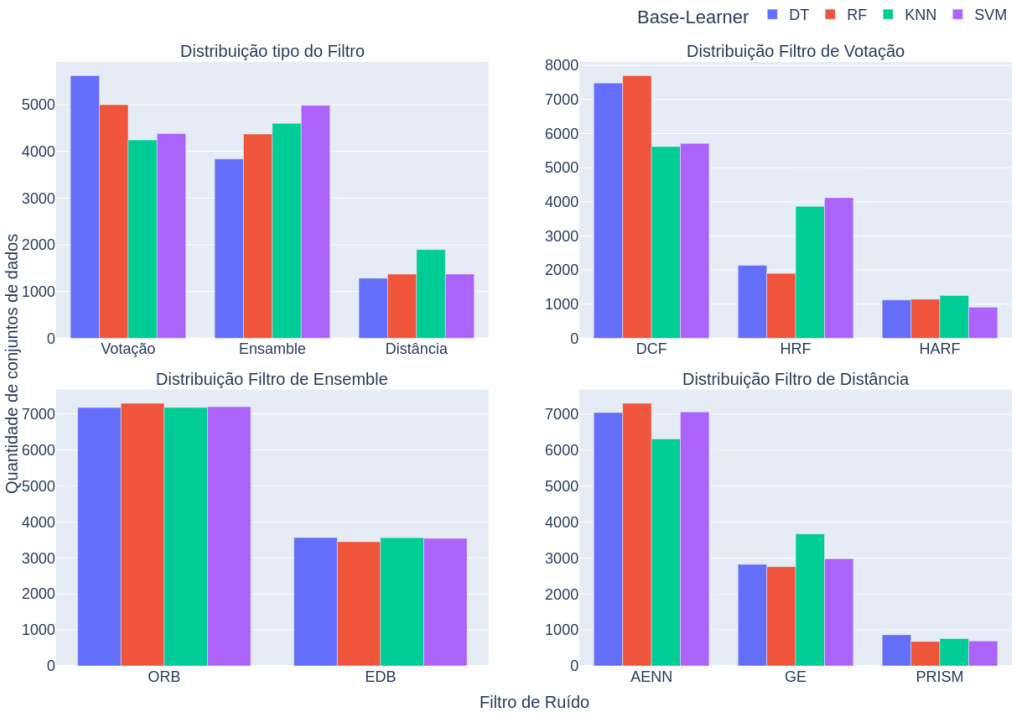

Figura 5.13: Gráficos contendo as distribuições das classes nas meta-base referentes ao tipo de algoritmo, aos algoritmos de votação, de *ensemble* e de distância sem considerar os hiperparâmetros.

Diferentemente da abordagem anterior, a MtL-Multi usa várias classificações sequenciais para suas recomendações. Dessa forma, são necessárias várias meta-bases para realizar a sugestão: uma para recomendar tipo de filtro, outras três para escolher o filtro de ruído e mais cinco para sugerir os hiperparâmetros. Essas meta-bases são construídas a partir de F, Y e A previamente calculados.

Antes de implementar as recomendações, verificou-se a distribuição de classes das meta-bases. As figuras 5.12 e 5.13 mostram quantas vezes cada classe aparece nas bases de tipos de filtro, filtros de votação, *ensemble* e distância, considerando conjuntos com e sem hiperparâmetros, respectivamente. Nenhuma das meta-bases está balanceada, sempre tendo um filtro contendo a maioria das classes de seu grupo. Também há mudança na distribuição do tipo de filtro considerado como melhor com ou sem o uso de recomendação de hiperparâmetros, onde, quando não utilizados, o tipo de votação é o mais predominante, caso contrário, os de *ensemble* são os mais comuns, atingindo mais de 50% das ocorrências.

Em seguida, a análise dos resultados foi iniciada, visando novamente encontrar a acurácia e o ganho total após a aplicação da recomendação. A Figura 5.14 apresenta o ganho total obtido pela abordagem utilizando o meta-classificador RF, GB ou KNN, comparado com o DCF como *baseline*. Nota-se que a sugestão raramente consegue obter

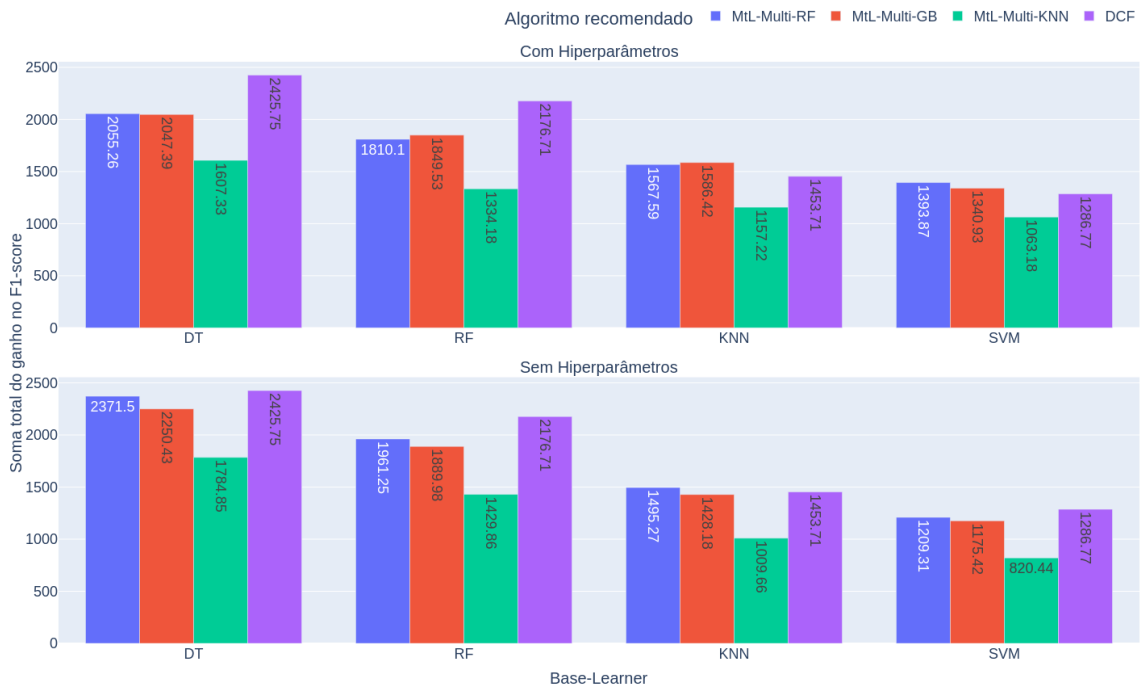

Figura 5.14: Gráfico contendo a soma do ganho obtido após aplicar as abordagens MtL-Multi com e sem a recomendação de hiperparâmetros comparadas o *baseline*.

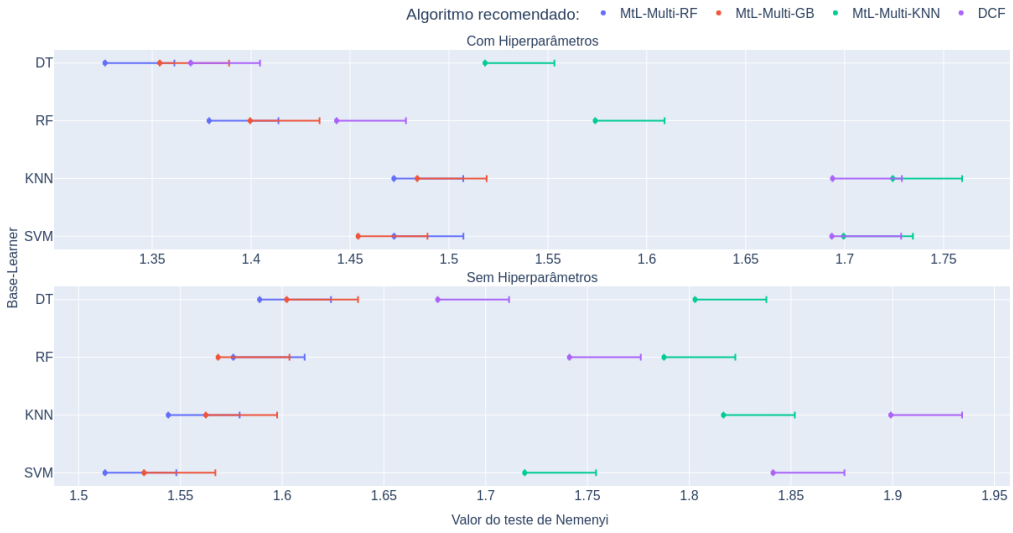

Figura 5.15: Gráfico dos resultados do teste de Nemenyi nos ganhos da abordagem MtL-Multi. Os menores valores são melhores e são considerados significativamente diferentes quando não há sobreposição de linhas.

um ganho maior do que o *baseline*, além disso, a diferença entre as abordagens e o *baseline* é menor na abordagem sem hiperparâmetros. Novamente foi realizado o teste de Friedman e Nemenyi para verificar se a os resultados eram significativamente diferentes. A Figura 5.15 Apresenta o resultado do teste de Nemenyi, onde em todos os casos não foi possível perceber diferença entre o resultado da abordagem MtL-Multi-RF e MtL-Multi-GB.

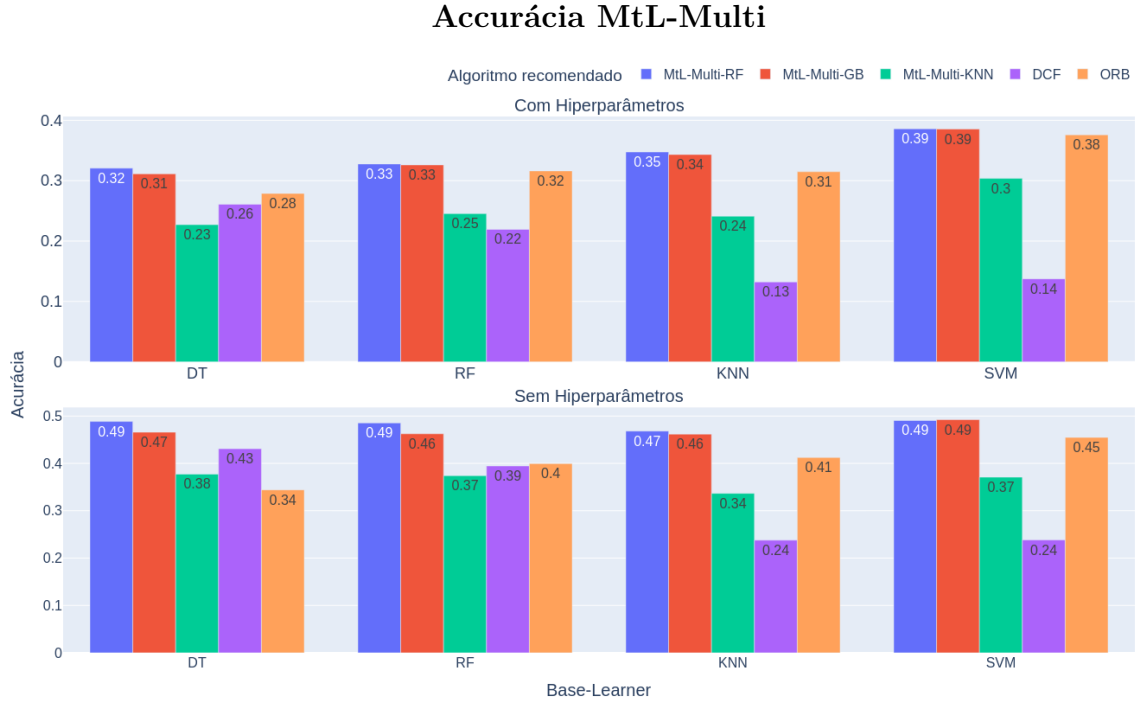

Figura 5.16: Gráfico contendo as acurácias das abordagens MtL-Multi comparadas com dois *baselines* quando considerados a recomendação com e sem hiperparâmetros.

Em relação à avaliação de acurácia, a MtL-Multi sempre obtém resultados superiores aos seus *baselines*. A Figura 5.16 mostra a acurácia obtida por cada um dos meta-classificadores, considerando uma abordagem com e sem a sugestão de hiperparâmetros, juntamente com os *baselines* DCF e ORB. No caso da otimização de hiperparâmetros, o ORB3 foi o algoritmo selecionado. Nota-se que o melhor meta-classificador para esse problema é o RF, sempre obtendo as maiores acurácias, seguido pelo GB, que também consegue superar as *baselines* em todos os casos.

5.2 Discussões

As duas abordagens implementadas neste experimento apresentam formas diferentes de recomendar os algoritmos. Enquanto a MtL-Rank fornece um resultado final no formato de ranque, permitindo maior flexibilidade na escolha do filtro pelo usuário final e baseia-se

em modelos de regressão, a MtL-Multi realiza a sugestão em múltiplas classificações em sequencia. Essas diferenças resultaram em resultados opostos: enquanto a MtL-Multi obteve maior acurácia, a MtL-Rank resultou em um ganho geral maior na métrica de desempenho.

Uma das razões desses resultados divergentes é o objetivo de otimização de cada abordagem. Utilizando regressores e buscando sempre o filtro com o maior *f1-score*, ao treinar o algoritmo de AM para recomendação, há uma tendência de gerar um ranqueamento onde os filtros com ganho na métrica de desempenho são priorizados. Portanto, durante o treinamento, em vez de otimizar a acurácia, a MtL-Rank é otimizada para encontrar o maior *f1-score*.

O treinamento da MtL-Multi ocorre de uma forma diferente. Neste caso, a meta-base é criada substituindo valores numéricos da métrica de desempenho por valores categóricos que representam um filtro ou tipo de filtro. O filtro recomendado continua sendo o que tem a maior métrica de desempenho, porém, durante o treinamento a técnica de AM é otimizada para obter uma maior acurácia na classificação.No entanto, buscar a maior acurácia nem sempre é o mais desejável. Isso ocorre pois um filtro frequentemente recomendado pode gerar mais resultados negativos ou falhas, como no caso do ORB. Além disso, escolher o melhor filtro ou o segundo melhor pode resultar em diferenças pequenas na métrica de desempenho. Desta forma, torna-se mais interessante uma abordagem com ganho médio maior do que uma que frequentemente recomenda o melhor resultado, mas também recomenda resultados negativos mais vezes. Portanto, considera-se que a abordagem MtL-Rank obteve resultados mais promissores do que a MtL-Multi.

Vale ressaltar que um dos grandes motivos da diferença de desempenho entre as abordagens também é decorrente do conjunto de algoritmos A e dos conjuntos de dados P utilizados. Neste experimento temos o caso de que o filtro classificado mais vezes como melhor frequentemente resulta em um desempenho negativo. Dessa forma, sugerir incorretamente este algoritmo pode ser muito custoso para a média do ganho da recomendação. Além disso, a meta-base formada é desbalanceada, especialmente quando considera-se a recomendação de hiperparâmetros, resultando em recomendações frequentes do mesmo algoritmo. De fato, é exatamente isso que acontece na abordagem MtL-Multi, onde o filtro ORB3 chegou a ser recomendado mais de dois terços das vezes.

Um dos objetivos deste experimento também é verificar se existem benefícios em incluir a recomendação de hiperparâmetros no processo. Em todos os casos, incluir esses parâmetros resultou em uma redução da acurácia, o que era esperado devido ao aumento do número de algoritmos a serem considerados (de 8 para 27). No entanto, ao considerar o ganho de desempenho, a abordagem MtL-Rank obteve os melhores resultados ao incluir a sugestão de hiperparâmetros. Desta forma, a recomendação de hiperparâmetros dos

filtros pode gerar ganhos para a aplicação.

Porém nem sempre a adição desta camada de complexidade é positiva. Na abordagem MtL-Multi em alguns casos a recomendação de hiperparâmetros piorou tanto a acurácia quanto o ganho total da aplicação. Além disso, ao construir a meta-base para essas aplicações exite um custo bem maior devido ao grande número de filtros a serem executados.

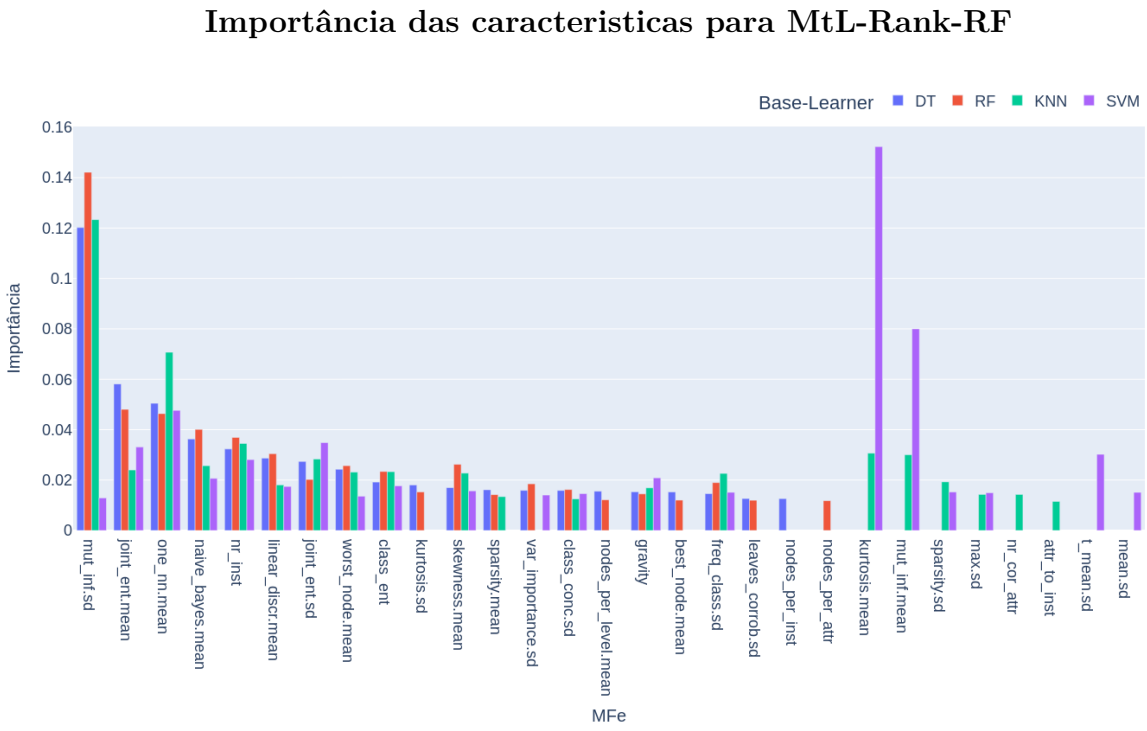

Figura 5.17: Grafico contendo as características que foram mais importantes em cada base-learner quando considerado a MtL-Rank-RF.

Por fim, convém compreender quais são as MFe mais uteis para esse problema. A abordagem MtL-Rank-RF foi a escolhida para se realizar a análise das MFe mais importantes pois obteve os maiores ganhos na métrica de desempenho. A Figura 5.17 apresenta a média da importância das 20 MFe mais relevantes quando utilizado o regressor RF para cada *base-learner*.

No total 29 MFe foram encontradas, detre essas 12 foram comuns em todos os quatro *base-learners*. Entre as MFe selecionadas, 10 pertencem a classe de estatística, 6 de teoria da informação, 5 de baseada em modelos, 5 *Landmarking* e 3 da classe simples. Considerando semente as 12 presentes em todos os *base-learners*, 6 pertencem a classe de teoria da informação, 4 de *Landmarking*, 2 de estatísticas e 1 simples. Na maioria dos casos, o desvio padrão da *mut_inf* é a característica mais relevante. Além disso, comparalas com as MFe encontradas na etapa de revisão sintemática (Figura 2.6), nota-se que a

maioria das MFe encontradas ao menos 9 vezes durante a revisão também estão presentes entre as 29 aqui analisadas.

Capítulo 6

Conclusão

Este trabalho teve como objetivo principal verificar a viabilidade da utilização de técnicas de MtL para recomendação de filtros de ruído. Para atingir esse objetivo, primeiramente realizou-se um estudo com intuito de encontrar o estado da arte referente a utilização de MtL e recomendação de técnicas de pré-processamento. Este processo foi feito por meio de uma revisão sistemática da literatura. Em seguida, foram propostas e comparadas duas abordagens diferentes para implementar a recomendação de algoritmos de detecção de ruído.

Durante a etapa de revisão sistemática da literatura, apresentada no Capítulo 2, estudou-se uma série de artigos referentes à sugestão de algoritmos de pré-processamento utilizando técnicas de MtL. Como resultado, a revisão possibilitou ter uma visão mais ampla sobre quais tipos de pré-processamento, algoritmos e métricas de desempenho e MFe são mais utilizadas na literatura, indicando o estado da arte sobre o tema. Notou-se que as técnicas mais recomendadas de pré-processamento na literatura eram referentes a seleção de características, sendo encontrada em aproximadamente 50% dos trabalhos, enquanto detecção de ruído era o foco de apenas 8% dos trabalhos. Foi possível detectar que as MFe mais utilizadas pertencem aos grupos de características simples, estatísticas ou de teoria da informação, que normalmente possuem um custo computacional menos elevado. Além disso, percebeu-se uma tendência de utilização de técnicas de AM menos custosas para formar a meta-base.

Justamente por ser a técnica menos frequente nas pesquisas, a detecção de ruído foi escolhida como o foco das recomendações deste trabalho. No Capítulo 4 foram introduzidas duas abordagens para realizar as sugestões dos filtros, a primeira, MtL-Rank, realiza a recomendação em um formato de ranque, onde são aplicadas regressões tentando prever o desempenho de algoritmos de AM após aplicar os filtros de ruído. Ao final, o resultado das regressões são ordenados formando um ranque das recomendações. Na segunda abordagem, MtL-Multi, são utilizadas diversas classificações em sequência tentando predizer

o filtro de ruído que resultará em maior ganho à métrica de desempenho, inicialmente tenta-se prever o tipo de filtro de ruído, e de acordo com o tipo, é recomendado qual filtro é mais adequado. Em ambos os casos utilizou-se quatro algoritmos como *base-learners* para se calcular a métrica de desempenho *f1-score*: RF, DT, KNN e SVM. Como meta-regressores ou meta-classificadores foram utilizados três algoritmos: RF, GB, KNN. Além disso, nas duas abordagens se utilizou o mesmo conjunto contendo 97 MFe e os mesmos oito algoritmos de redução de ruído.

Os resultados dos experimentos foram apresentados no Capítulo 5 e mostraram que a abordagem MtL-Rank foi capaz de gerar um ganho médio na métrica de desempenho maior após suas recomendações quando comparada com a MtL-Multi e com o *baseline* em todos os testes. Ao verificar a acurácia das abordagens, ou seja, quantas vezes o filtro que resulta no melhor desempenho foi recomendado, a abordagem MtL-Multi obteve os melhores resultados, com um acurácia de até 49%.

Também analisou-se a possibilidade de realizar a recomendação de hiperparâmetros em conjunto com a sugestão do filtro. Os resultados mostraram que esta recomendação pode gerar ganhos para o sistema como no caso da MtL-Rank, onde o ganho na métrica de desempenho atingiu os melhores resultados, porém, a acurácia do modelo foi a mais baixa. Já no caso da MtL-Multi tanto a acurácia quanto o ganho de desempenho foram inferiores na implementação sem a recomendação de hiperparâmetros.

6.1 Trabalhos futuros

O estudo realizado neste trabalho mostrou a viabilidade da implementação da recomendações de algoritmos de detecção de ruídos por meio de técnicas de MtL. Porém, mesmo obtendo resultados positivos, acredita-se que os resultados possam ser ainda melhores com um conjunto de características otimizado para a recomendação. Desta forma, recomenda-se uma análise mais profunda sobre o conjunto de MFe que poderiam ser utilizados para a recomendação, verificando se existe alguma classe de MFe que é mais importante para a recomendação.

A recomendação de hiperparâmetros dos filtros também pode ser aprimorada, recomendando mais de um parâmetro por algoritmo. A depender da quantidade de hiperparâmetros a serem sugeridos por algoritmos, é possível que torne-se computacionalmente inviável a realização de técnicas de MtL para a recomendação, assim outras abordagens podem ser utilizadas neste processo como Otimização Bayesiana ou Programação Genética, talvez resultando em resultados superiores a uma abordagem que utiliza somente MtL.

Por fim, assim como esta abordagem obtém resultados positivos para filtros de ruído, este trabalho pode ser expandido para outros tipos de algoritmos de pré-processamento, sugere-se assim, a realização de um experimento contendo a indicação de outras técnicas como imputação de dados faltantes ou balanceamento de dados em conjunto com as técnicas de detecção de ruído, resultando em uma recomendação mais completa e automatizada com relação à etapa de preprocessamento de dados.

Referências

[1] Russell, Stuart J. e Peter Norvig: *Artificial Intelligence: a modern approach*. Pearson, 2009. 1, 21

[2] Wu, Carole Jean, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia *et al.*: *Machine learning at facebook: Understanding inference at the edge*. Em *2019 IEEE International Symposium on High Performance Computer Architecture*, páginas 331–344, 2019. 1

[3] Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard *et al.*: *Tensorflow: A system for large-scale machine learning*. Em *12th Symposium on Operating Systems Design and Implementation*, páginas 265–283, 2016. 1

[4] Labrinidis, Alexandros e Hosagrahar V Jagadish: *Challenges and opportunities with big data*. VLDB Endowment, 5(12):2032–2033, 2012. 1

[5] Witten, Ian H, Eibe Frank, Mark A Hall, CJ Pal e MINING DATA: *Practical machine learning tools and techniques*. Em *Data Mining*, volume 2, página 4, 2005. 1

[6] Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot e E. Duchesnay: *Scikit-learn: Machine learning in Python*. Journal of Machine Learning Research, 12:2825–2830, 2011. 1

[7] H2O.ai: *H2O: Scalable Machine Learning Platform*, 2022. `https://github.com/h2oai/h2o-3`. 1

[8] Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu e Xiaoqiang Zheng: *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. `https://www.tensorflow.org/`, Software available from tensorflow.org. 1

[9] Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai e Soumith Chintala: *Pytorch: An imperative style, high-performance deep learning library.* Em *Advances in Neural Information Processing Systems 32*, páginas 8024–8035. Curran Associates, Inc., 2019. `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`. 1

[10] Shalf, John: *The future of computing beyond moore's law.* Philosophical Transactions of the Royal Society, 378(2166):20190061, 2020. 1

[11] Wirth, Rüdiger e Jochen Hipp: *Crisp-dm: Towards a standard process model for data mining.* Em *4th International Conference on the Practical Application of Knowledge Discovery and Data Mining*, páginas 29–39, 2000. 1

[12] Fayyad, Usama M, David Haussler e Paul E Stolorz: *Kdd for science data analysis: Issues and examples.* Em *KDD*, páginas 50–56, 1996. 1

[13] García, Salvador, Julián Luengo e Francisco Herrera: *Data preprocessing in data mining*, volume 72. Springer, 2015. 2, 23

[14] Sáez, José A, Julián Luengo e Francisco Herrera: *Predicting noise filtering efficacy with data complexity measures for nearest neighbor classification.* Pattern Recognition, 46(1):355–364, 2013. 2

[15] Liu, Huan e Hiroshi Motoda: *Feature selection for knowledge discovery and data mining*, volume 454. Springer Science & Business Media, 2012. 2

[16] Liu, Huan e Hiroshi Motoda: *Instance selection and construction for data mining*, volume 608. Springer Science & Business Media, 2013. 2

[17] Luengo, Julián, Salvador García e Francisco Herrera: *On the choice of the best imputation methods for missing values considering three groups of classification methods.* Knowledge and Information Systems, 32(1):77–108, 2012. 2

[18] López, Victoria, Alberto Fernández, Salvador García, Vasile Palade e Francisco Herrera: *An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics.* Information Sciences, 250:113–141, 2013. 2

[19] Huang, Jianglin, Yan Fu Li e Min Xie: *An empirical analysis of data preprocessing for machine learning-based software cost estimation.* Information and Software Technology, 67:108–127, 2015. 2

[20] Wolpert, David H e William G Macready: *No free lunch theorems for optimization.* IEEE Transactions on Evolutionary Computation, 1(1):67–82, 1997. 2

[21] Truong, Anh, Austin Walters, Jeremy Goodsitt, Keegan Hines, C Bayan Bruss e Reza Farivar: *Towards automated machine learning: Evaluation and comparison of automl approaches and tools*. Em *31st International Conference on Tools with Artificial Intelligence*, páginas 1471–1479, 2019. 2, 3, 5

[22] Hutter, Frank, Lars Kotthoff e Joaquin Vanschoren: *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019. 2

[23] Thornton, Chris, Frank Hutter, Holger H Hoos e Kevin Leyton-Brown: *Auto-weka: Combined selection and hyperparameter optimization of classification algorithms*. Em *19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, páginas 847–855, 2013. 3

[24] Feurer, Matthias, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum e Frank Hutter: *Efficient and robust automated machine learning*. Em *Advances in Neural Information Processing Systems*, páginas 2962–2970, 2015. 3

[25] Olson, Randal S e Jason H Moore: *Tpot: A tree-based pipeline optimization tool for automating machine learning*. Em *Workshop on Automatic Machine Learning*, páginas 66–74, 2016. 3

[26] Nagarajah, Thiloshon e Guhanathan Poravi: *A review on automated machine learning (automl) systems*. Em *IEEE 5th International Conference for Convergence in Technology*, páginas 1–6. IEEE, 2019. 3

[27] Rice, John R: *The algorithm selection problem*. Em *Advances in Computers*, volume 15, páginas 65–118. Elsevier, 1976. 3, 26, 27

[28] Smith-Miles, Kate A: *Cross-disciplinary perspectives on meta-learning for algorithm selection*. ACM Computing Surveys, 41(1):1–25, 2009. 3, 26, 27, 28

[29] Vanschoren, Joaquin: *Meta-learning*. Em *Automated Machine Learning*, páginas 35–61. Springer, Cham, 2019. 4, 6, 25, 30

[30] Brazdil, Pavel, Christophe Giraud-Carrier, Carlos Soares e Ricardo Vilalta: *Metalearning - Applications to Data Mining*. Cognitive Technologies. Springer, 1ª edição, 2009. 4, 29, 30, 35

[31] Rivolli, Adriano, Luís PF Garcia, Carlos Soares, Joaquin Vanschoren e André CPLF de Carvalho: *Meta-features for meta-learning*. Knowledge-Based Systems, página 108101, 2022. 4, 15, 29, 38

[32] Munson, M Arthur: *A study on the importance of and time spent on different modeling steps*. ACM SIGKDD Explorations Newsletter, 13(2):65–71, 2012. 5

[33] Parmezan, Antonio Rafael Sabino, Huei Diana Lee, Newton Spolaôr e Feng Chung Wu: *Automatic recommendation of feature selection algorithms based on dataset characteristics*. Expert Systems with Applications, 185:115589, 2021. 5, 14, 33, 34, 35

[34] Denyer, David e David Tranfield: *Producing a systematic review.* The Sage Handbook of Organizational Research Methods, páginas 671–689, 2009. 8

[35] Brereton, Pearl, Barbara A Kitchenham, David Budgen, Mark Turner e Mohamed Khalil: *Lessons from applying the systematic literature review process within the software engineering domain.* Journal of Dystems and Software, 80(4):571–583, 2007. 8

[36] Pisani, Paulo Henrique e Ana Carolina Lorena: *A systematic review on keystroke dynamics.* Journal of the Brazilian Computer Society, 19(4):573–587, 2013. 8

[37] García, Salvador, Julián Luengo e Francisco Herrera: *Tutorial on practical tips of the most influential data preprocessing algorithms in data mining.* Knowledge-Based Systems, 98:1–29, 2016. 14

[38] Alexandropoulos, Stamatios Aggelos N, Sotiris B Kotsiantis e Michael N Vrahatis: *Data preprocessing in predictive data mining.* The Knowledge Engineering Review, 34:e1, 2019. 14

[39] Bilalli, Besim, Alberto Abelló, Tomàs Aluja-Banet e Robert Wrembel: *Presistant: Learning based assistant for data pre-processing.* Data & Knowledge Engineering, 123:101727, 2019. 14, 17, 32, 33

[40] Bernstein, Abraham, Foster Provost e Shawndra Hill: *Toward intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification.* IEEE Transactions on Knowledge and Data Engineering, 17(4):503–518, 2005. 14

[41] Zou, Ying, Aijun An e Xiangji Huang: *Evaluation and automatic selection of methods for handling missing data.* Em *IEEE International Conference on Granular Computing*, páginas 728–733, 2005. 14

[42] Smith-Miles, Kate e Rafiqul Islam: *Meta-learning for data summarization based on instance selection method.* Em *IEEE Congress on Evolutionary Computation*, páginas 1–8, 2010. 14

[43] Luebke, Karsten e Claus Weihs: *Linear dimension reduction in classification: adaptive procedure for optimum results.* Advances in Data Analysis and Classification, 5(3):201–213, 2011. 14

[44] Smith-Miles, Kate A e Rafiqul Islam: *Meta-learning of instance selection for data summarization.* Em *Meta-Learning in Computational Intelligence*, páginas 77–95. Springer, 2011. 14

[45] Wang, Guangtao, Qinbao Song, Heli Sun, Xueying Zhang, Baowen Xu e Yuming Zhou: *A feature subset selection algorithm automatic recommendation method.* Journal of Artificial Intelligence Research, 47:1–34, 2013. 14

[46] Leyva, Enrique, Antonio González e Raúl Pérez: *Knowledge-based instance selection: A compromise between efficiency and versatility.* Knowledge-Based Systems, 47:65–76, 2013. 14, 31, 33

[47] Shilbayeh, Samar e Sunil Vadera: *Feature selection in meta learning framework*. Em *2014 Science and Information Conference*, páginas 269–275. IEEE, 2014. 14

[48] Leyva, Enrique, Antonio González e Raul Perez: *A set of complexity measures designed for applying meta-learning to instance selection*. IEEE Transactions on Knowledge and Data Engineering, 27(2):354–367, 2014. 14

[49] Leyva, Enrique, Yoel Caises, Antonio González e Raúl Pérez: *On the use of meta-learning for instance selection: An architecture and an experimental study*. Information Sciences, 266:16–30, 2014. 14

[50] Oliveira Moura, Shayane de, Marcelo Bassani de Freitas, Halisson AC Cardoso e George DC Cavalcanti: *Choosing instance selection method using meta-learning*. Em *IEEE International Conference on Systems, Man, and Cybernetics*, páginas 2003–2007. IEEE, 2014. 14

[51] Filchenkov, Andrey e Arseniy Pendryak: *Datasets meta-feature description for recommending feature selection algorithm*. Em *Artificial Intelligence and Natural Language and Information Extraction, Social Media and Web Search FRUCT Conference*, páginas 11–18. IEEE, 2015. 14

[52] Bilalli, Besim, Alberto Abelló, Tomàs Aluja-Banet e Robert Wrembel: *Automated data pre-processing via meta-learning*. Em *International Conference on Model and Data Engineering*, páginas 194–208. Springer, 2016. 14

[53] Post, Martijn J, Peter Van Der Putten e Jan N Van Rijn: *Does feature selection improve classification? a large scale experiment in openml*. Em *International Symposium on Intelligent Data Analysis*, páginas 158–170. Springer, 2016. 14

[54] Morais, Romero FAB de, Péricles BC Miranda e Ricardo MA Silva: *A meta-learning method to select under-sampling algorithms for imbalanced data sets*. Em *5th Brazilian Conference on Intelligent Systems*, páginas 385–390. IEEE, 2016. 14, 32, 33

[55] Garcia, Luís PF, André CPLF de Carvalho e Ana C Lorena: *Noise detection in the meta-learning level*. Neurocomputing, 176:14–25, 2016. 14, 31, 33

[56] Garcia, Luís PF, Ana C Lorena, Stan Matwin e André CPLF de Carvalho: *Ensembles of label noise filters: a ranking approach*. Data Mining and Knowledge Discovery, 30(5):1192–1216, 2016. 14, 31, 33, 37

[57] Dôres, Silvia Nunes das, Carlos Soares e Duncan DA Ruiz: *Effect of metalearning on feature selection employment*. Em *AutoML@ PKDD/ECML*, páginas 84–90, 2017. 14

[58] Parmezan, Antonio Rafael Sabino, Huei Diana Lee e Feng Chung Wu: *Metalearning for choosing feature selection algorithms in data mining: Proposal of a new framework*. Expert Systems with Applications, 75:1–24, 2017. 14, 32, 33

[59] Morais, Romero FAB de, Péricles BC de Miranda e Ricardo MA Silva: *A multi-criteria meta-learning method to select under-sampling algorithms for imbalanced datasets*. Em *25th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2017. 14

[60] Bilalli, Besim, Alberto Abelló Gamazo e Tomàs Aluja Banet: *On the predictive power of meta-features in openml*. International Journal of Applied Mathematics and Computer Science, 27(4):697–712, 2017. 14, 17, 32

[61] Aduviri, Robert, Daniel Matos e Edwin Villanueva: *Feature selection algorithm recommendation for gene expression data through gradient boosting and neural network metamodels*. Em *IEEE International Conference on Bioinformatics and Biomedicine*, páginas 2726–2728, 2019. 14

[62] Bilalli, Besim, Alberto Abelló, Tomàs Aluja-Banet, Rana Faisal Munir e Robert Wrembel: *Presistant: data pre-processing assistant*. Em *International Conference on Advanced Information Systems Engineering*, páginas 57–65. Springer, 2018. 14

[63] Bilalli, Besim, Alberto Abelló, Tomàs Aluja-Banet e Robert Wrembel: *Intelligent assistance for data pre-processing*. Computer Standards & Interfaces, 57:101–109, 2018. 14, 17, 32

[64] Tanfilev, Igor, Andrey Filchenkov e Ivan Smetannikov: *Feature selection algorithm ensembling based on meta-learning*. Em *10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics*, páginas 1–6. IEEE, 2017. 14

[65] Smolyakov, Dmitry, Alexander Korotin, Pavel Erofeev, Artem Papanov e Evgeny Burnaev: *Meta-learning for resampling recommendation systems*. Em *Eleventh International Conference on Machine Vision (ICMV 2018)*, página 110411S, 2019. 14

[66] Kandanaarachchi, Sevvandi, Mario A Munoz e Kate Smith-Miles: *Instance space analysis for unsupervised outlier detection.* Em *EDML*, páginas 32–41, 2019. 14

[67] Quemy, Alexandre: *Two-stage optimization for machine learning workflow*. Information Systems, 92:101483, 2020. 14

[68] Shen, Zixiao, Xin Chen e Jonathan M Garibaldi: *A novel meta learning framework for feature selection using data synthesis and fuzzy similarity*. Em *IEEE International Conference on Fuzzy Systems*, páginas 1–8, 2020. 14

[69] Liu, Qian e Manfred Hauswirth: *A provenance meta learning framework for missing data handling methods selection*. Em *11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, páginas 0349–0358, 2020. 14

[70] Sahni, Deep, Satya Jayadev Pappu e Nirav Bhatt: *Aided selection of sampling methods for imbalanced data classification*. Em *8th ACM IKDD CODS and 26th COMAD*, páginas 198–202. 2021. 14

[71] Neutatz, Felix, Felix Biessmann e Ziawasch Abedjan: *Enforcing constraints for machine learning systems via declarative feature selection: An experimental study.* Em *International Conference on Management of Data*, páginas 1345–1358, 2021. 14

[72] Moniz, Nuno e Vitor Cerqueira: *Automated imbalanced classification via meta-learning.* Expert Systems with Applications, 178:115011, 2021. 14

[73] Zagatti, Fernando Rezende, Lucas Cardoso Silva, Lucas Nildaimon Dos Santos Silva, Bruno Silva Sette, Helena de Medeiros Caseli, Daniel Lucrédio e Diego Furtado Silva: *Metaprep: Data preparation pipelines recommendation via meta-learning.* Em *20th IEEE International Conference on Machine Learning and Applications*, páginas 1197–1202, 2021. 14

[74] Zhao, Yue, Ryan Rossi e Leman Akoglu: *Automatic unsupervised outlier model selection.* Advances in Neural Information Processing Systems, 34:4489–4502, 2021. 14

[75] Mitchell, Tom M e Tom M Mitchell: *Machine learning*, volume 1. McGraw-hill New York, 1997. 21

[76] Goodfellow, Ian, Yoshua Bengio e Aaron Courville: *Deep learning.* MIT Press, 2016. 21

[77] Alloghani, Mohamed, Dhiya Al-Jumeily, Jamila Mustafina, Abir Hussain e Ahmed J Aljaaf: *A systematic review on supervised and unsupervised machine learning algorithms for data science.* Supervised and Unsupervised Learning for Data Science, páginas 3–21, 2020. 22

[78] Qiu, Junfei, Qihui Wu, Guoru Ding, Yuhua Xu e Shuo Feng: *A survey of machine learning for big data processing.* EURASIP Journal on Advances in Signal Processing, 2016(1):1–16, 2016. 22

[79] Caruana, Rich e Alexandru Niculescu-Mizil: *Data mining in metric space: an empirical analysis of supervised learning performance criteria.* Em *10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, páginas 69–78, 2004. 22

[80] Zhang, Shichao, Chengqi Zhang e Qiang Yang: *Data preparation for data mining.* Applied Artificial Intelligence, 17(5-6):375–381, 2003. 22

[81] Famili, A, Wei Min Shen, Richard Weber e Evangelos Simoudis: *Data preprocessing and intelligent data analysis.* Intelligent Data Analysis, 1(1):3–23, 1997. 22

[82] Frénay, Benoît e Michel Verleysen: *Classification in the presence of label noise: a survey.* IEEE Transactions on Neural Networks and Learning Systems, 25(5):845–869, 2013. 23, 24, 38, 46

[83] Zhu, Xingquan e Xindong Wu: *Class noise vs. attribute noise: A quantitative study.* Artificial Intelligence Review, 22(3):177–210, 2004. 23

[84] Gupta, Shivani e Atul Gupta: *Dealing with noise problem in machine learning datasets: A systematic review.* Procedia Computer Science, 161:466–474, 2019. 24

[85] Morales, Pablo, Julián Luengo, Luís Paulo F Garcia, Ana Carolina Lorena, André CPLF de Carvalho, Francisco Herrera *et al.*: *The noisefiltersr package: Label noise preprocessing in r.* R J., 9(1):219, 2017. 24, 38

[86] Sluban, Borut, Dragan Gamberger e Nada Lavrač: *Ensemble-based noise detection: noise ranking and visual performance evaluation.* Data Mining and Knowledge Discovery, 28(2):265–303, 2014. 24, 37

[87] Koplowitz, Jack e Thomas A Brown: *On the relation of performance to editing in nearest neighbor rules.* Pattern Recognition, 13(3):251–255, 1981. 24, 37

[88] Wilson, Dennis L.: *Asymptotic properties of nearest neighbor rules using edited data.* IEEE Transactions on Systems, Man, and Cybernetics, SMC-2(3):408–421, 1972. 24, 25

[89] Garcia, Luis Paulo F, Ana Carolina Lorena e André CPLF Carvalho: *A study on class noise detection and elimination.* Em *Brazilian Symposium on Neural Networks*, páginas 13–18. IEEE, 2012. 24, 37

[90] Karmaker, Amitava e Stephen Kwek: *A boosting approach to remove class label noise.* International Journal of Hybrid Intelligent Systems, 3(3):169–177, 2006. 24, 37

[91] Freund, Yoav e Robert E Schapire: *A decision-theoretic generalization of on-line learning and an application to boosting.* Journal of Computer and System Sciences, 55(1):119–139, 1997. 24

[92] Wheway, Virginia: *Using boosting to detect noisy data.* Em *Pacific Rim International Conference on Artificial Intelligence*, páginas 123–130, 2001. 25, 37

[93] Miranda, André LB, Luís Paulo F Garcia, André CPLF Carvalho e Ana C Lorena: *Use of classification algorithms in noise detection and elimination.* Em *International Conference on Hybrid Artificial Intelligence Systems*, páginas 417–424, 2009. 25, 37

[94] Tomek, Ivan: *An experiment with the edited nearest-neighbor rule.* IEEE Transactions on Systems, Man, and Cybernetics, SMC-6(6):448–452, 1976. 25, 37

[95] Smith, Michael R e Tony Martinez: *Improving classification accuracy by identifying and removing instances that should be misclassified.* Em *International Joint Conference on Neural Networks*, páginas 2690–2697, 2011. 25, 37

[96] Cui, Can, Mengqi Hu, Jeffery D Weir e Teresa Wu: *A recommendation system for meta-modeling: A meta-learning based approach.* Expert Systems with Applications, 46:33–44, 2016. 28, 29

[97] Alcobaça, Edesio, Felipe Siqueira, Adriano Rivolli, Luís P. F. Garcia, Jefferson T. Oliva e André C. P. L. F. de Carvalho: *Mfe: Towards reproducible meta-feature extraction.* Journal of Machine Learning Research, 21(111):1–5, 2020. `http://jmlr.org/papers/v21/19-348.html`. 30, 38

[98] Lewis, David D: *Naive (bayes) at forty: The independence assumption in information retrieval*. Em *European Conference on Machine Learning*, páginas 4–15. Springer, 1998. 32

[99] Frank, Eibe e Ian H Witten: *Generating accurate rule sets without global optimization*. 1998. 32

[100] Aha, David W, Dennis Kibler e Marc K Albert: *Instance-based learning algorithms*. Machine Learning, 6(1):37–66, 1991. 32

[101] Dash, Manoranjan e Huan Liu: *Consistency-based search in feature selection*. Artificial Intelligence, 151(1-2):155–176, 2003. 32

[102] Hall, Mark Andrew *et al.*: *Correlation-based feature selection for machine learning*. 1999. 32

[103] Azhagusundari, B, Antony Selvadoss Thanamani *et al.*: *Feature selection based on information gain*. International Journal of Innovative Technology and Exploring Engineering, 2(2):18–21, 2013. 32

[104] Kira, Kenji, Larry A Rendell *et al.*: *The feature selection problem: Traditional methods and a new algorithm*. Em *Aaai*, volume 2, páginas 129–134, 1992. 32

[105] Kohavi, Ron e George H. John: *Wrappers for feature subset selection*. Artificial Intelligence, 97(1-2):273–324, 1997. 33

[106] Pio, Pedro Borges, Luís PF Garcia e Adriano Rivolli: *Meta-learning approach for noise filter algorithm recommendation*. Em *X Symposium on Knowledge Discovery, Mining and Learning*, volume InPress, 2022. 33, 34

[107] Todorovski, Ljupco, Hendrik Blockeel e Saso Dzeroski: *Ranking with predictive clustering trees*. Em *European Conference on Machine Learning*, páginas 444–455. Springer, 2002. 33

[108] Cover, Thomas e Peter Hart: *Nearest neighbor pattern classification*. IEEE Transactions on Information Theory, 13(1):21–27, 1967. 38, 41, 45

[109] Ho, Tin Kam: *Random decision forests*. Em *3rd International Conference on Document Analysis and Recognition*, páginas 278–282. IEEE, 1995. 38, 41, 45

[110] Breiman, Leo: *Classification and regression trees*. Routledge, 2017. 38, 45

[111] Cortes, Corinna e Vladimir Vapnik: *Support-vector networks*. Machine learning, 20:273–297, 1995. 38, 45

[112] Friedman, Jerome H: *Greedy function approximation: a gradient boosting machine*. Symposium on Knowledge Discovery, Mining and Learning, 29(2):1189–1232, 2001. 41

[113] Cawley, Gavin C e Nicola LC Talbot: *Efficient leave-one-out cross-validation of kernel fisher discriminant classifiers*. Pattern Recognition, 36(11):2585–2592, 2003. 41

[114] Zar, Jerrold H: *Spearman rank correlation.* Encyclopedia of Biostatistics, 7, 2005. 42

[115] Demšar, Janez: *Statistical comparisons of classifiers over multiple data sets.* The Journal of Machine learning research, 7:1–30, 2006. 55

Anexo I

Notas dos artigos avaliados

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Soma
A Provenance Meta Learning Framework for Missing Data Handling Methods Selection	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	8.0
Two-stage optimization for machine learning workflow	1.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	6.0
Aided Selection of Sampling Methods for Imbalanced Data Classification	1.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	6.0
A novel meta learning framework for feature selection using data synthesis and fuzzy similarity	1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	6.0
Feature selection algorithm recommendation for gene expression data through gradient boosting and neural network metamodels	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	5.0
PRESISTANT: Learning based assistant for data pre-processing	1.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	8.0
Meta-Learning for Resampling Recommendation Systems	0.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	6.0
PRESISTANT: Data pre-processing assistant	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	3.0
Intelligent assistance for data pre-processing	1.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0	7.0
Feature selection algorithm ensembling based on meta-learning	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
On the predictive power of meta-features in OpenML	0.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	5.0
A Meta-Learning Method to Select Under-Sampling Algorithms for Imbalanced Data Sets	1.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	7.0
Effect of metalearning on feature selection employment	1.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	7.0
A multi-criteria meta-learning method to select under-sampling algorithms for imbalanced datasets	1.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	7.0
Metalearning for choosing feature selection algorithms in data mining: Proposal of a new framework	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	11.0
Automated data pre-processing via meta-learning	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	3.0
Noise detection in the meta-learning level	1.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	9.0
Does feature selection improve classification? A large scale experiment in OpenML	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	4.0
Datasets meta-feature description for recommending feature selection algorithm	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	7.0
On the use of meta-learning for instance selection: An architecture and an experimental study	1.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0	7.0
Choosing instance selection method using meta-learning	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	8.0
Feature selection in meta learning framework	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	5.0
Evaluation and automatic selection of methods for handling missing data	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	9.0
Toward intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0	5.0
Meta-learning for data summarization based on instance selection method	0.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	6.0
Linear dimension reduction in classification: Adaptive procedure for optimum results	0.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	6.0
Meta-learning of instance selection for data summarization	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	8.0
A feature subset selection algorithm automatic recommendation method	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	10.0
Knowledge-based instance selection: A compromise between efficiency and versatility	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	11.0
A set of complexity measures designed for applying meta-learning to instance selection	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	10.0
Ensembles of label noise filters: a ranking approach	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	10.0
Instance space analysis for unsupervised outlier detection	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	5.0
Automated imbalanced classification via meta-learning	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	9.0
MetaPrep: Data preparation pipelines recommendation via meta-learning	0.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	5.0
Automatic Unsupervised Outlier Model Selection	0.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	5.0
Enforcing Constraints for Machine Learning Systems via Declarative Feature Selection: An Experimental Study	1.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	7.0
Automatic recommendation of feature selection algorithms based on dataset characteristics	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	10.0

Tabela I.1: Notas de todos artigos avaliados na revisão sistemática

Anexo II

Listas de conjuntos de dados

ID#	Nome do conjunto	ID#	Nome do conjunto
3	kr-vs-kp	908	chscase_census3
31	credit-g	909	chscase_census2
37	diabetes	910	fri_c1_1000_10
40	sonar	911	fri_c2_250_5
43	haberman	912	fri_c2_1000_5
44	spambase	913	fri_c2_1000_10
50	tic-tac-toe	915	plasma_retinol
53	heart-statlog	916	fri_c3_100_5
59	ionosphere	917	fri_c1_1000_25
310	mammography	918	fri_c4_250_50
311	oil_spill	920	fri_c2_500_50
333	monks-problems-1	921	analcatdata_seropositive
334	monks-problems-2	922	fri_c2_100_50
335	monks-problems-3	923	visualizing_soil
336	SPECT	925	visualizing_galaxy
337	SPECTF	926	fri_c0_500_25
346	aids	929	rabe_176
376	SyskillWebert-Sheep	931	disclosure_z
444	analcatdata_boxing2	932	fri_c4_100_50
446	prnn_crabs	933	fri_c4_250_25
448	analcatdata_boxing1	934	socmob
450	analcatdata_lawsuit	935	fri_c1_250_10
459	analcatdata_asbestos	936	fri_c3_500_10
461	analcatdata_creditscore	937	fri_c3_500_50

ID#	Nome do conjunto	ID#	Nome do conjunto
463	backache	941	lowbwt
464	prnn_synth	942	chscase_health
465	analcatdata_cyyoung8092	943	fri_c0_500_10
467	analcatdata_japansolvent	945	kidney
472	lupus	946	visualizing_ethanol
476	analcatdata_bankruptcy	947	arsenic-male-bladder
479	analcatdata_cyyoung9302	949	arsenic-female-bladder
682	sleuth_ex2016	950	arsenic-female-lung
683	sleuth_ex2015	951	arsenic-male-lung
713	vineyard	955	tae
714	fruitfly	958	segment
715	fri_c3_1000_25	959	nursery
716	fri_c3_100_50	962	mfeat-morphological
717	rmftsa_ladata	965	zoo
719	veteran	969	iris
720	abalone	970	analcatdata_authorship
721	pwLinear	971	mfeat-fourier
722	pol	973	wine
723	fri_c4_1000_25	974	hayes-roth
724	analcatdata_vineyard	976	JapaneseVowels
725	bank8FM	977	letter
726	fri_c2_100_5	979	waveform-5000
728	analcatdata_supreme	980	optdigits
730	fri_c1_250_5	983	cmc
731	baskball	987	collins
732	fri_c0_250_50	988	fl2000
733	machine_cpu	991	car
734	ailerons	994	vehicle
735	cpu_small	995	mfeat-zernike
736	visualizing_environmental	996	prnn_fglass
737	space_ga	997	balance-scale
740	fri_c3_1000_10	1004	synthetic_control
741	rmftsa_sleepdata	1005	glass
743	fri_c1_1000_5	1006	lymph
744	fri_c3_250_5	1009	white-clover

ID#	Nome do conjunto	ID#	Nome do conjunto
745	auto_price	1011	ecoli
746	fri_c1_250_25	1012	flags
747	servo	1013	analcatdata_challenger
748	analcatdata_wildcat	1014	analcatdata_dmft
749	fri_c3_500_5	1015	confidence
750	pm10	1016	vowel
751	fri_c4_1000_10	1019	pendigits
752	puma32H	1020	mfeat-karhunen
753	wisconsin	1021	page-blocks
754	fri_c0_100_5	1025	analcatdata_germangss
755	sleuth_ex1605	1026	grub-damage
756	autoPrice	1045	kc1-top5
758	analcatdata_election2000	1046	mozilla4
759	analcatdata_olympic2000	1048	jEdit_4
761	cpu_act	1049	pc4
762	fri_c2_100_10	1050	pc3
763	fri_c0_250_10	1054	mc2
764	analcatdata_apnea3	1055	cm1_req
765	analcatdata_apnea2	1056	mc1
766	fri_c1_500_50	1059	ar1
767	analcatdata_apnea1	1060	ar3
768	fri_c3_100_25	1061	ar4
769	fri_c1_250_50	1063	kc2
770	strikes	1064	ar6
771	analcatdata_michiganacc	1065	kc3
772	quake	1066	kc1-binary
773	fri_c0_250_25	1067	kc1
774	disclosure_x_bias	1068	pc1
775	fri_c2_100_25	1069	pc2
776	fri_c0_250_5	1071	mw1
778	bodyfat	1073	jEdit_4
779	fri_c1_500_25	1075	datatrieve
780	rabe_265	1120	MagicTelescope
782	rabe_266	1121	badges2
783	fri_c3_100_10	1167	pc1_req

ID#	Nome do conjunto	ID#	Nome do conjunto
784	newton_hema	1412	lungcancer_GSE31210
787	witmer_census_1980	1441	KungChi3
788	triazines	1442	MegaWatt1
789	fri_c1_100_10	1443	PizzaCutter1
790	elusage	1444	PizzaCutter3
792	fri_c2_500_5	1446	CostaMadre1
793	fri_c3_250_10	1447	CastMetal1
794	fri_c2_250_25	1448	KnuggetChase3
795	disclosure_x_tampered	1449	MeanWhile1
796	cpu	1450	MindCave2
797	fri_c4_1000_50	1451	PieChart1
799	fri_c0_1000_5	1452	PieChart2
800	pyrim	1453	PieChart3
801	chscase_funds	1455	acute-inflammations
803	delta_ailerons	1460	banana
804	hutsof99_logis	1462	banknote-authentication
805	fri_c4_500_50	1463	blogger
806	fri_c3_1000_50	1464	blood-transfusion-service -center
807	kin8nm	1467	climate-model-simulation -crashes
808	fri_c0_100_10	1471	eeg-eye-state
811	rmftsa_ctoarrivals	1473	fertility
812	fri_c1_100_25	1480	ilpd
813	fri_c3_1000_5	1487	ozone-level-8hr
814	chscase_vine2	1488	parkinsons
815	chscase_vine1	1489	phoneme
816	puma8NH	1490	planning-relax
818	diggle_table_a2	1494	qsar-biodeg
819	delta_elevators	1495	qualitative-bankruptcy
820	chatfield_4	1496	ringnorm
824	fri_c1_500_10	1498	sa-heart
825	boston_corrected	1504	steel-plates-fault
826	sensory	1506	thoracic-surgery
827	disclosure_x_noise	1507	twonorm

ID#	Nome do conjunto	ID#	Nome do conjunto
829	fri_c1_100_5	1510	wdbc
830	fri_c2_250_10	1511	wholesale-customers
832	fri_c3_250_25	1524	vertebra-column
833	bank32nh	1547	autoUniv-au1-1000
837	fri_c1_1000_50	1558	bank-marketing
838	fri_c4_500_25	4154	CreditCardSubset
841	stock	4329	thoracic_surgery
845	fri_c0_1000_10	4534	PhishingWebsites
846	elevators	23499	breast-cancer-dropped-missing-attributes-values
847	wind	40589	emotions
849	fri_c0_1000_25	40646	GAMETES_Epistasis_2-Way_20atts_0
850	fri_c0_100_50	40647	GAMETES_Epistasis_2-Way_20atts_0
853	boston	40648	GAMETES_Epistasis_3-Way_20atts_0
855	fri_c4_500_10	40649	GAMETES_Heterogeneity_20atts_1600_Het_0
859	analcatdata_gviolence	40650	GAMETES_Heterogeneity_20atts_1600_Het_0
860	vinnie	40669	corral
863	fri_c4_250_10	40680	mofn-3-7-10
865	analcatdata_neavote	40681	mux6
866	fri_c2_1000_50	40683	postoperative-patient-data
867	visualizing_livestock	40690	threeOf9
868	fri_c4_100_25	40693	xd6
869	fri_c2_500_10	40701	churn
870	fri_c1_500_5	40702	solar-flare
871	pollen	40704	Titanic
873	fri_c3_250_50	40705	tokyo1
874	rabe_131	40706	parity5_plus_5
875	analcatdata_chlamydia	40710	cleve
876	fri_c1_100_50	40713	dis
877	fri_c2_250_50	40900	Satellite

ID#	Nome do conjunto	ID#	Nome do conjunto
878	fri_c4_100_10	40981	Australian
879	fri_c2_500_25	40983	wilt
880	mu284	40999	jungle_chess_2pcs_ endgame_elephant_elephant
882	pollution	41005	jungle_chess_2pcs _endgame_rat_rat
884	fri_c0_500_5	41007	jungle_chess_2pcs _endgame_lion_lion
885	transplant	41156	ada
886	no2	41538	conference_attendance
887	mbagrade	41945	ilpd-numeric
888	fri_c0_500_50	41946	Sick_numeric
889	fri_c0_100_25	41976	TuningSVMs
890	cloud	42192	compas-two-years
891	sleuth_case1202	42665	ricci_vs_destefano
892	sleuth_case1201	43892	national-longitudinal -survey-binary
893	visualizing_hamster	43946	eye_movements
894	rabe_148	43947	KDDCup09_upselling
895	chscase_geyser1	43949	rl
896	fri_c3_500_25	43957	house_16H
900	chscase_census6	44089	credit
902	sleuth_case2002	44095	spambase_reproduced
903	fri_c2_1000_25	44097	credit-g_copy
904	fri_c0_1000_50	44098	credit-g_reproduced_1
906	chscase_census5	44149	Heart_disease_prediction_20
907	chscase_census4	44162	compass

Tabela II.1: Tabela contendo a lista dos conjuntos de dados que foram utilizados juntamente com seu número de ID da plataforma OpenML.

Anexo III

Listas de meta-características

attr_conc.mean	leaves_homo.mean	nr_norm
attr_conc.sd	leaves_homo.sd	nr_num
attr_ent.mean	leaves_per_class.mean	nr_outliers
attr_ent.sd	linear_discr.mean	ns_ratio
attr_to_inst	linear_discr.sd	one_nn.mean
best_node.mean	mad.mean	one_nn.sd
best_node.sd	mad.sd	p_trace
can_cor.mean	max.mean	random_node.mean
cat_to_num	max.sd	random_node.sd
class_conc.mean	mean.mean	range.mean
class_conc.sd	mean.sd	range.sd
class_ent	median.mean	sd.mean
cov.mean	median.sd	sd.sd
eigenvalues.mean	min.mean	skewness.mean
eigenvalues.sd	min.sd	skewness.sd
elite_nn.mean	mut_inf.mean	sparsity.mean
elite_nn.sd	mut_inf.sd	sparsity.sd
eq_num_attr	naive_bayes.mean	t_mean.mean
freq_class.mean	naive_bayes.sd	t_mean.sd
freq_class.sd	nodes	tree_depth.mean
gravity	nodes_per_attr	tree_depth.sd
inst_to_attr	nodes_per_inst	tree_imbalance.mean
iq_range.mean	nodes_per_level.mean	tree_shape.mean
iq_range.sd	nodes_per_level.sd	tree_shape.sd
joint_ent.mean	nodes_repeated.mean	var.mean
joint_ent.sd	nodes_repeated.sd	var.sd
kurtosis.mean	nr_attr	var_importance.mean
kurtosis.sd	nr_bin	var_importance.sd
leaves	nr_cat	w_lambda
leaves_branch.mean	nr_class	worst_node.mean
leaves_branch.sd	nr_cor_attr	worst_node.sd
leaves_corrob.mean	nr_disc	
leaves_corrob.sd	nr_inst	

Tabela III.1: Tabela contendo a lista das MFe utilizadas seguindo a nomenclatura apresentada pela biblioteca `pymfe`.