UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

GERAÇÃO DE CONJUNTOS CONSISTENTES DE REGRAS PARA CLASSIFICAÇÃO MULTIRRÓTULO COM ALGORITMO EVOLUTIVO MULTIOBJETIVO

THIAGO ZAFALON MIRANDA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Inteligência Artificial.

Orientador: Prof. Dr. Ricardo Cerri

São Carlos – SP

junho, 2020

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Dissertação de Mestrado do candidato Thiago Zafalon Miranda, realizada em 25/06/2020.

Comissão Julgadora:

Prof. Dr. Ricardo Cerri (UFSCar)

Profa. Dra. Heloisa de Arruda Camargo (UFSCar)

Profa. Dra. Gisele Lobo Pappa (UFMG)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.
O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

Este trabalho é dedicado às minhas avós e aos meus avôs: Pérsio Miranda, Maria Luiza Martins Miranda, Neide de Lourdes Paulino e Didier Zafalon

AGRADECIMENTOS

Agradeço às minhas avós e aos meus avôs por todo suporte e carinho; ao meu orientador pelo comprometimento, dedicação e paciência; e aos desenvolvedores do software *NProfiler*, por fornecerem uma licença de estudante.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior-Brasil (CAPES) - Código de Financiamento 001.

RESUMO

A classificação multirrótulo é uma tarefa da área de aprendizado de máquina cujo objetivo é gerar modelos capazes de aprender relações entre características descritivas de objetos e os conjuntos de classes as quais tais objetos podem pertencer. Em certas aplicações, é importante que os modelos sejam interpretáveis para que seus usuários confiem nele ou para que suas predições possam ser explicadas. Nesta pesquisa, explorou-se a geração de modelos de classificação multirrótulo baseados em conjuntos consistentes de regras. Para tanto, propôs-se um algoritmo evolutivo e dois algoritmos auxiliares que orientam seu processo de geração de regras, garantindo que as regras criadas sejam consistentes entre si. Um conjunto de regras é consistente se sempre que mais de uma regra cobrir um objeto, tais regras associam a este objeto o mesmo conjunto de classes. O algoritmo evolutivo proposto, ao empregar técnicas de otimização multiobjetivo, gera coleções de modelos de classificação que oferecem diferentes compromissos entre interpretabilidade e poder preditivo. Foram conduzidos experimentos com o algoritmo proposto e algoritmos da literatura e, a partir de análise estatística, concluiu-se que os modelos gerados são, em termos de interpretabilidade, superiores aos da literatura e, em termos de poder preditivo, comparáveis à maioria.

Palavras-chave: classificação multirrótulo; interpretabilidade; regras consistentes; otimização multiobjetivo; algoritmos evolutivos.

ABSTRACT

Multi-label classification a machine learning task whose objective is to generate models capable of learning relationships between descriptive characteristics of objects and the sets of classes to which such objects belong. In certain applications, it is important for the models to be interpretable so that their users can trust it or so that its predictions can be explained. In this research, we investigated the generation of multi-label classification models based on consistent sets of rules. We proposed an evolutionary algorithm and two auxiliary algorithms that guide the rule generation process, ensuring that the rules created were consistent with each other. A set of rules is consistent if whenever multiple rules covers an object, such rules predict the same set of classes. The proposed evolutionary algorithm utilized multi-objective optimization techniques to generate collections of classification models that offer different compromises between interpretability and predictive power. Experiments were conducted with the proposed algorithms and with algorithms from the literature and, based on statistical analysis, we concluded that the generated models were, in terms of interpretability, superior to those generated by literature's algorithms and, in terms of predictive power, they were comparable to most.

Keywords: multi-label classification; interpretability; consistent rules; multi-objective optimization; evolutionary algorithms.

SUMÁRIO

CAPÍTULO 1–INTRODUÇÃO 8
1.1 Contextualização 8
1.2 Hipótese e Objetivo 9
1.3 Organização do Documento 9

CAPÍTULO 2–FUNDAMENTAÇÃO TEÓRICA 11
2.1 Classificação Multirrótulo 11
2.2 Interpretabilidade 16
2.3 Otimização Multiobjetivo 18
2.4 Algoritmos Genéticos 19

CAPÍTULO 3–DESENVOLVIMENTO TEÓRICO 23
3.1 Contextualização e definições 24
3.1.1 Consistência entre regras 24
3.1.2 Suposições e exemplos 24
3.2 Algoritmos de orientação de busca 26
3.2.1 *Constrained Feature Space Greedy Search* 26
3.2.2 *Constrained Feature Space Box-Enlargement* 30
3.2.3 Comparação entre CFSGS e CFSBE 32
3.3 MINOTAUR 35
3.3.1 Modelagem 35
3.3.2 Inicialização e criação de regras 36
3.3.3 Mutação e seleção 38
3.3.4 Pseudo-código e implementação de referência 41

CAPÍTULO 4–VALIDAÇÃO EXPERIMENTAL 43
4.1 Descrição e reprodutibilidade 43
4.2 Resultados e análise 45

CAPÍTULO 5–CONSIDERAÇÕES FINAIS 52
5.1 Síntese do trabalho 52
5.2 Trabalhos futuros 52

REFERÊNCIAS 54

Capítulo 1
INTRODUÇÃO

Neste Capítulo, são apresentados o contexto em que esta pesquisa foi desenvolvida, a hipótese e o objetivo da pesquisa e como o texto é organizado.

É importante destacar que na literatura há dois tipos de regras de classificação, *fuzzy* e *crisp* (FREITAS, 2013), mas essa pesquisa contempla apenas regras do tipo *crisp*. De fato, os algoritmos propostos no Capítulo 3 tem características indesejáveis para a geração de regras *fuzzy*. A escolha por uma delineação de escopo estrita, que não inclui regras *fuzzy*, não foi tomada em função de méritos ou desméritos de uma abordagem sobre a outra, mas sim considerando os limites de tempo da pesquisa e preferências estéticas e subjetivas do autor.

1.1 Contextualização

Na área de Aprendizado de Máquina (AM), uma das tarefas mais comuns é a classificação, descrita sucintamente por Otero e Freitas (2013) como a geração de um modelo que aprende relações entre características preditivas e os valores que uma variável categórica pode assumir[1]. Este aprendizado ocorre por meio do ajuste de parâmetros internos do modelo, que são guiados por uma coleção de fatos, geralmente em forma de um conjunto de dados. Se o modelo pode predizer o valor de múltiplas variáveis binárias simultaneamente, diz-se que a classificação é multirrótulo (TSOUMAKAS; KATAKIS, 2007). Exemplificando: predizer a espécie de uma planta seria uma tarefa de classificação tradicional (monorrótulo), pois cada planta pode estar associada a apenas uma espécie, enquanto predizer as funções que uma proteína pode desempenhar seria uma tarefa de classificação multirrótulo, pois uma mesma proteína pode desempenhar diversas funções.

Como os algoritmos de classificação são capazes de processar grandes quantidades de dados, eles podem identificar relações entre as variáveis de um problema mais rapidamente do que humanos. Se os modelos gerados forem, além de eficazes, interpretáveis, então as próprias relações aprendidas podem ser utilizadas no estudo do problema (CLARE; KING, 2001; OTERO

[1] Uma definição formal é apresentada no Capítulo 2, Seção 2.1.

et al., 2010; FREITAS, 2014).

Em alguns contextos, como aplicações médicas e financeiras, é importante que o modelo gerado seja interpretável para que seus usuários confiem nele (FREITAS, 2014). Gerar um modelo com alto poder preditivo e elevada interpretabilidade configura uma tarefa de otimização multiobjetivo, pois modificar o modelo pode alterar o valor de uma destas características sem necessariamente alterar o valor de outra.

Em problemas de otimização multiobjetivo, otimizar uma solução em relação a um objetivo específico pode fazer com que a mesma tenha uma qualidade inaceitável nos outros objetivos (KONAK et al., 2006); faz-se necessário, então, que a busca por soluções considere, de alguma forma, os múltiplos objetivos.

As três abordagens mais comuns para a otimização multiobjetivo são a composição de objetivos, a ordenação lexicográfica e a abordagem baseada em dominação de Pareto (FREITAS, 2004). Entre as três, a mais robusta é a baseada em dominação de Pareto, pois ela explora melhor o espaço de busca e retorna múltiplas soluções, permitindo que o usuário analise os compromissos oferecidos por diferentes soluções.

Konak et al. (2006) apresentam Algoritmos Genéticos (GAs) como ferramentas apropriadas para a solução de problemas de otimização multiobjetivo, visto que não requerem a definição de prioridades, pesos ou escalas para os objetivos, e são capazes de gerar aproximações do conjunto ótimo de Pareto a cada iteração. Além disso, como tais algoritmos exploram múltiplas partes do espaço de soluções simultaneamente, tais aproximações podem ser bastante diversas.

1.2 Hipótese e Objetivo

Esta pesquisa tem como hipótese que algoritmos genéticos multiobjetivo podem gerar, de forma eficiente, modelos de classificação multirrótulo baseados em conjuntos **consistentes** de regras, isto é, conjuntos que não contêm regras que cobrem uma mesma região do *feature space* e predizem classes diferentes. O objetivo da pesquisa, portanto, é mecanismos eficientes que garantam que os modelos gerados serão consistentes.

1.3 Organização do Documento

Neste Capítulo, o projeto de pesquisa foi contextualizado, a hipótese definida e o objetivo explicitado. No Capítulo 2, são discutidos os aspectos práticos e teóricos que justificam a pesquisa, como a questão da interpretabilidade de modelos de classificação, e que fundamentam o projeto, como a classificação multirrótulo, a otimização multiobjetivo e os algoritmos genéticos multiobjetivo. No Capítulo 3, são apresentados os principais algoritmos desenvolvidos nesta pesquisa, e no Capítulo 4, experimentos que comparam o desempenho de tais algoritmos com da

literatura. Finalmente, no Capítulo 5, são apresentadas as considerações finais sobre os algoritmos propostos e possíveis trabalhos futuros.

Capítulo 2
FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo, são discutidos os principais aspectos teóricos envolvidos na descrição e no tratamento do problema, isto é, a geração de modelos interpretáveis de classificação multirrótulo.

2.1 Classificação Multirrótulo

Uma das tarefas mais comuns na área de Aprendizado de Máquina é a classificação, descrita informalmente como a criação e ajuste de um modelo capaz de aprender relações entre características descritivas de objetos e classes às quais esses objetos pertencem, efetivamente viabilizando a associação automática de objetos a classes pré-definidas (TAN, 2018).

Formalmente, a classificação pode ser definida como: dado um conjunto de pares ordenados $(x_i, y_i) \in X \times Y$, sendo Y um conjunto finito, e assumindo que existe uma função desconhecida $f : X \to Y, y_i = f(x_i)$, pretende-se gerar uma função $g : X \to Y$ que aproxime f.

Como consequência de tal definição, na classificação tradicional, também conhecida como monorrótulo, as classes cujos objetos podem pertencer são mutuamente exclusivas, isto é, se um objeto pertence a uma classe (está associado ao rótulo dessa classe), ele não pode pertencer a nenhuma outra (BOUTELL et al., 2004).

Existem, porém, problemas cujos objetos de estudo podem pertencer a múltiplas classes simultaneamente, isto é, há uma sobreposição entre as classes. A tarefa que visa gerar modelos capazes de associar um objeto a um *conjunto* de classes é chamada classificação multirrótulo (TSOUMAKAS; KATAKIS, 2007).

De forma similar à classificação tradicional, pode-se definir formalmente a classificação multirrótulo como: dado um conjunto de pares ordenados $(x_i, y_i) \in X \times \mathcal{P}(Y)$, sendo Y um conjunto finito, $\mathcal{P}(Y)$ denotando o conjunto potência de Y, e assumindo que existe uma função desconhecida $f : X \to \mathcal{P}(Y), y_i = f(x_i)$, pretende-se gerar uma função $g : X \to \mathcal{P}(Y)$ que aproxime f.

Como exemplos de tais problemas, citam-se a classificação de músicas em sentimen-

tos (TROHIDIS et al., 2008), a classificação de textos em tópicos (KATAKIS et al., 2008), a classificação semântica de imagens (BOUTELL et al., 2004), a predição de funções de proteínas (CERRI et al., 2016) e a classificação de genes em classes funcionais (PEÑA-CASTILLO et al., 2008).

Tsoumakas e Katakis (2007) discutem duas abordagens para lidar com problemas de classificação multirrótulo. A primeira consiste em transformar o problema em uma coleção de problemas monorrótulo, possibilitando a utilização de algoritmos de classificação tradicionais. A segunda consiste em transformar os algoritmos monorrótulo, ou criar novos algoritmos, para lidarem com conjuntos de dados multirrótulo.

Das técnicas de adaptação de problema, uma das mais simples e mais utilizadas é a *Binary Relevance* (BR) (ZHANG et al., 2018). Essa técnica consiste em treinar múltiplos classificadores binários, um para cada classe do conjunto de dados multirrótulo, e agregar as predições de tais classificadores para gerar a predição multirrótulo. As duas maiores limitações dessa abordagem são o elevado custo computacional, decorrente do treinamento de múltiplos classificadores, e a desconsideração das relações que podem haver entre as classes.

Uma outra técnica de adaptação de problema comumente utilizada é a *Classifier Chains* (CC) (READ et al., 2011). De forma similar a *Binary Relevance*, essa técnica envolve o treinamento de múltiplos classificadores binários, um para cada classe do conjunto de dados. A principal diferença é que este método é capaz de aproveitar relações existentes entre classes do problema; para tanto, os classificadores são "encadeados", isto é, a saída de um classificador é concatenada com a descrição dos objetos que estão sendo classificados e fornecida para o próximo classificador da corrente. Duas limitações dessa técnica são: erros que acontecem no começo da corrente são propagados ao longo dela, e a qualidade das predições finais é dependente da ordem em que os classificadores foram organizados.

Uma terceira técnica de adaptação de problemas é a *Label Powerset* (LP) (TSOUMAKAS; VLAHAVAS, 2007; TSOUMAKAS; KATAKIS, 2007). Essa técnica gera um novo conjunto de dados monorrótulo a partir da combinação dos rótulos presentes no conjuntos de dados. Por exemplo, se um objeto possui os rótulos $\{A, B, F\}$ e outro objeto possui rótulos $\{A, B, C\}$, criam-se as classes $[ABF]$ e $[ABC]$. O LP é computacionalmente vantajoso comparado ao BR ou ao CC, pois não envolve o treinamento de múltiplos modelos; porém, o conjunto de dados gerado pode conter uma grande quantidade de classes com poucos objetos associados a elas.

Uma quarta técnica de adaptação de problemas é o Random k-Labelsets (Rakel) (TSOUMAKAS; VLAHAVAS, 2007). Nessa técnica são treinados m classificadores sobre subconjuntos do conjunto de dados com k classes selecionadas aleatoriamente, sendo m e k hiperparâmetros do algoritmo. O treinamento de cada modelo ocorre de forma idêntica ao LP, isto é, gerando novas classes que representam combinações de rótulos. Ao utilizar $k = 1$ e $m = |L|$, sendo $|L|$ o número de classes no conjunto multirrótulo, o método torna-se idêntico ao *Binary Relevance*.

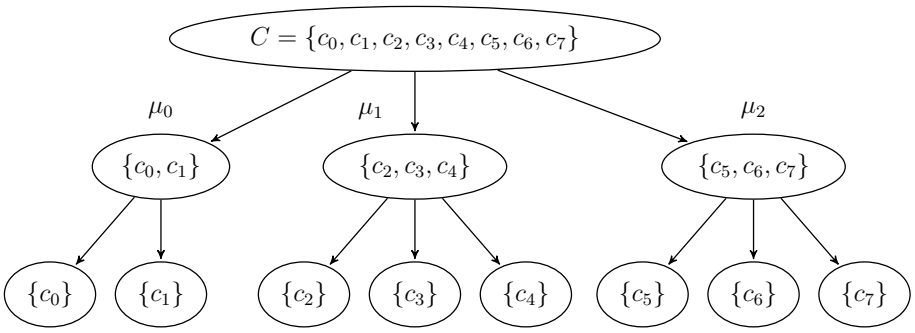


Figura 1 – Exemplo de particionamento recursivo do conjunto de classes. Adaptada de (TSOUMAKAS et al., 2008)

Analogamente, ao utilizar $k = |L|$ e $m = 1$, o método torna-se idêntico ao LP. Experimentos indicam que, com valores apropriados para k e m, este método pode gerar resultados superiores ao *Binary Relevance* e ao *Label Powerset* (TSOUMAKAS; VLAHAVAS, 2007).

Uma quinta técnica de adaptação de problemas é a Hierarchy Of Multilabel classifiERs (HOMER) (TSOUMAKAS et al., 2008), que utiliza uma coleção de classificadores multirrótulo[1] organizados em uma estrutura de árvore. O formato da árvore e, consequentemente, a quantidade de classificadores utilizados, depende de quais e quantas "meta-classes" o algoritmo gera.

Dado um conjunto de dados com $|C|$ classes, onde C é o conjunto de classes, o algoritmo HOMER emprega um processo recursivo para particionar[2] C até que os subconjuntos gerados possuam apenas uma classe. Cada subconjunto[3] gerado em uma iteração do processo de particionamento forma uma meta-classe. Na Figura 1 é apresentado o resultado de tal processo aplicado a um conjunto com oito classes, onde μ_i denota uma meta-classe e c_i denota uma classe.

Para cada meta-classe é criado um classificador multirrótulo que prediz a quais meta-classes, ou classes (dependendo da posição do classificador na árvore), o objeto que está sendo classificado deve ser associado, como pode ser visto Figura 2, onde h_i denota um classificador multirrótulo. Classificadores nos níveis internos da árvore predizem meta-classes, enquanto os classificadores das folhas predizem classes.

Para classificar um objeto, este é fornecido para o classificador na raiz da árvore; para cada meta-classe que o classificador associar ao objeto, uma cópia desse objeto é fornecida para os classificadores associados a tais meta-classes; o processo é repetido até que cópias sejam fornecidas aos classificadores das folhas da árvore ou até que nenhuma meta-classe seja predita. A predição final do HOMER é dada pela união das classes preditas pelos classificadores das folhas da árvore.

[1] É possível utilizar classificadores monorrótulo suplementados por técnicas de adaptação de problema.

[2] Tsoumakas et al. (2008) discutem diferentes métodos para realizar o particionamento, por exemplo, distribuindo aleatoriamente as classes em uma quantidade aleatória de subconjuntos.

[3] Mais especificamente, cada subconjunto com mais do que um elemento é considerado uma meta-classe, posto que conjuntos com apenas um elemento descrevem uma classe.

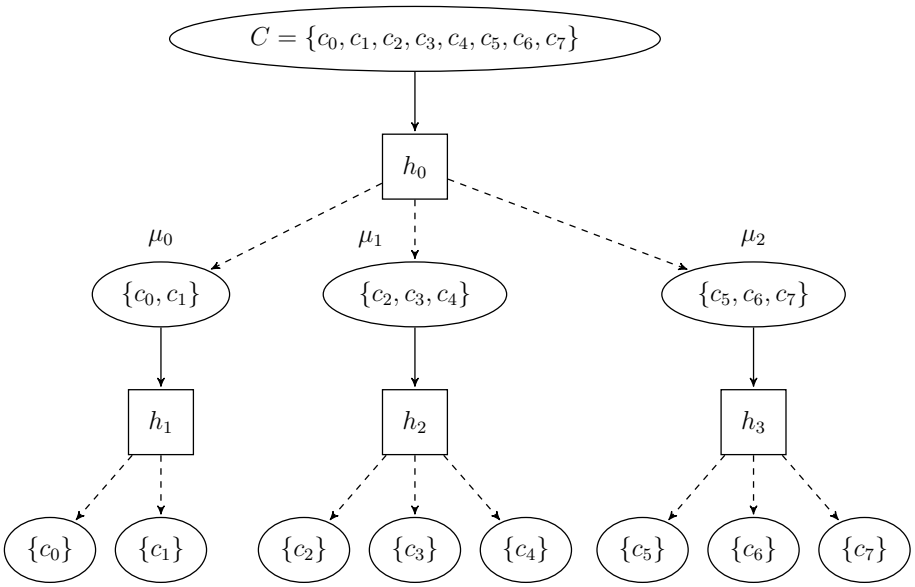


Figura 2 – Meta-classes e classificadores do método HOMER.
Adaptada de (TSOUMAKAS et al., 2008)

A abordagem baseada em transformação de problema, embora bastante utilizada, não é ideal, pois todo método de transformação incorre em uma penalidade, como elevado custo computacional ou perdas de informações que podem diminuir o poder preditivo do classificador, como correlações entre classes (TSOUMAKAS; KATAKIS, 2007).

Os trabalhos apresentados a seguir empregam a segunda abordagem para lidar com problemas de classificação multirrótulo, baseada na criação ou adaptação de algoritmos. É importante observar que alguns dos trabalhos discutidos tratam da classificação hierárquica multirrótulo, isto é, as classes às quais os objetos podem pertencer estão organizadas em uma hierarquia (classificação hierárquica), e os objetos podem pertencer a múltiplas classes simultaneamente (classificação multirrótulo). Em alguns dos trabalhos, como em Clare e King (2001), os autores empregam adaptação de algoritmo para lidar com o aspecto multirrótulo e adaptação de problema para lidar com o aspecto hierárquico; realizando, em última instância, classificação hierárquica multirrótulo.

Clare e King (2001) adaptaram Árvores de Decisão (QUINLAN, 1986) para realizar classificação hierárquica multirrótulo de genes em classes funcionais. Para lidar com a hierarquia de classes, os autores utilizaram a abordagem *one local classifier per level* (SILLA; FREITAS, 2011). Para realizar a classificação multirrótulo, os autores elaboraram uma fórmula de entropia que considera as múltiplas classes às quais um objeto pode pertencer.

Zhang e Zhou (2005) apresentam o algoritmo ML-*k*NN, uma adaptação do algoritmo k-Nearest Neighbors, para realizar classificação multirrótulo. Segundo os autores, o ML-*k*NN foi o primeiro classificador multirrótulo *lazy*. Para determinar as classes de um objeto, os autores calculam a frequência de classes na vizinhança do objeto e utilizam a regra de Bayes para calcular

a quais classes o objeto pertence.

Parpinelli et al. (2002) apresentam o primeiro algoritmo de classificação baseado em colônias de formigas, chamado Ant-Miner. O algoritmo gera listas de regras de classificação e, em sua versão original, tais regras operam apenas sobre dados categóricos. Esse algoritmo é particularmente importante pois serviu de base para diversos outros, como: Multi-Label Ant-Miner (MuLAM) (CHAN; FREITAS, 2006), que gera regras de classificação multirrótulo plana (não hierárquica); cAnt-Miner (OTERO et al., 2008), que removeu a limitação de operar apenas sobre atributos categóricos; h-Ant-Miner (OTERO et al., 2009), que gera regras de classificação hierárquica monorrótulo; e hm-Ant-Miner (OTERO et al., 2010), que gera regras de classificação hierárquica multirrótulo.

Otero et al. (2013) apresentam uma nova estratégia de cobertura sequencial para o cAnt-Miner, utilizada em um novo algoritmo chamado cAnt-Miner$_{pb}$. Esse novo algoritmo, por sua vez, serviu de base para outro algoritmo, chamado Unordered cAnt-Miner$_{pb}$, que gera conjuntos (coleções não ordenadas) de regras, em vez de listas de regras (OTERO; FREITAS, 2013). Os autores da nova versão sugerem que conjuntos de regras são mais interpretáveis do que listas de regras e propõem uma nova métrica de interpretabilidade, chamada *Prediction-Explanation Size*, que penaliza a interdependência de regras, presente em listas e ausente em conjuntos.

Cerri et al. (2019) apresentam o algoritmo HMC-GA, uma extensão do algoritmo proposto em Cerri et al. (2012), que, de acordo com os autores, foi o primeiro algoritmo genético criado para classificação hierárquica multirrótulo[4]. O algoritmo emprega uma estratégia de cobertura sequencial, isto é, a cada iteração ele remove do conjunto de dados os objetos cobertos pelas regras geradas. O consequente de cada regra é gerado deterministicamente, em função do antecedente da regra, que, por sua vez, é evoluído pelo algoritmo. A função de fitness é baseada na quantidade de objetos cobertos pela regra e na quantidade de testes ativos na regra, favorecendo regras menores e que cobrem mais exemplos.

Cerri et al. (2016) apresentam o algoritmo HMC-LMLP para realizar classificação hierárquica multirrótulo. Os autores observam que a utilização de redes neurais para classificação multirrótulo é conveniente; basta que a última camada da rede possua um neurônio para cada classe do problema. Para lidar com o aspecto hierárquico, os autores associam uma rede neural a cada nível da hierarquia; as saídas da rede neural em uma camada são concatenadas com as informações do objeto sendo classificado, e o vetor aumentado gerado é utilizado pela rede neural da camada seguinte.

A literatura sobre classificação multirrótulo, bem como sobre as possíveis aplicações, é vasta; a maioria dos trabalhos descritos nesta seção compõem um subconjunto da literatura que segue a abordagem evolucionária, como algoritmos genéticos ou colônias de formigas, ou cujos modelos gerados podem ser considerados interpretáveis. A questão da interpretabilidade

[4] De acordo com Gonçalves et al. (2018), este ainda é o único Algoritmo Genético que gera modelos completos de classificação hierárquica multirrótulo.

de modelos de classificação será discutida na Seção 2.2.

2.2 Interpretabilidade

Freitas (2014) destaca que em diversos contextos, como aqueles de aplicações médicas ou de bioinformática, o modelo de classificação deve ser interpretável para que seus usuários confiem nele; alta eficiência em conjuntos de dados de testes não é suficiente.

É importante observar que interpretabilidade, no contexto de modelos de classificação, não é um conceito formalmente definido (LIPTON, 2016); diferentes autores utilizam esse termo para se referir a diferentes propriedades que um modelo pode ter. Não obstante, mesmo sem uma definição formal, é razoável dizer que alguns tipos de modelos são mais interpretáveis do que outros. Modelos de classificação baseados em regras, por exemplo, são intuitivamente mais interpretáveis do que modelos baseados em redes neurais.

Lipton (2016) discute a questão da interpretabilidade de modelos de classificação e argumenta que ela pode se dar de duas formas: transparência e explicações *post-hoc*. Afirmar que um modelo é transparente, em vez de opaco, significa que é relativamente fácil entender os mecanismos que o regem. Lipton (2016) considera que a transparência pode ocorrer em três níveis: no nível do modelo (*simulatability*), no nível dos componentes do modelo (*decomposability*) e no nível do algoritmo de treinamento do modelo (*algorithmic transparency*).

Simulatability refere-se ao quão fácil é entender o modelo como um todo; se é possível para um humano, a partir dos parâmetros do modelo, realizar a classificação de um objeto de forma semelhante ao modelo. Por exemplo, modelos mais concisos ou mais simples possuem maior *simulatability* do que modelos grandes ou complexos.

Decomposability refere-se ao quão fácil é explicar cada parte do modelo, como parâmetros e cálculos realizados. Por exemplo, é mais fácil explicar as diferentes partes do modelo gerado pelo C4.5, (estrutura da árvore ou testes nos nós) do que aquelas de uma rede neural (topologia da rede ou funções de ativação dos neurônios).

Algorithmic transparency refere-se ao quão bem entendido é o algoritmo que gera o modelo. Pode-se provar, por exemplo, que modelos lineares convergem para uma solução única, mesmo para conjuntos de dados desconhecidos, enquanto métodos baseados em *Deep Learning* não possuem essa garantia; logo, modelos lineares possuem maior *algorithmic transparency*.

Lipton (2016) chama de *explicações post-hoc* as técnicas que geram informações e explicações para o usuário sem necessariamente explicitarem como o modelo funciona. Por exemplo, técnicas que projetam o conjunto de dados em duas dimensões e geram uma imagem a partir dessa projeção, como no trabalho de Maaten e Hinton (2008), podem demonstrar para o usuário que certas classes ou objetos são similares.

Outro exemplo de *explicação post-hoc* é o método proposto por Caruana et al. (1999), em

que o sistema, ao classificar um objeto, retorna também qual objeto do conjunto de treinamento mais se assemelha a ele. Os objetos são comparados em função dos "padrões de ativação" das partes internas do modelo; se o modelo for uma rede neural, tais padrões são as saídas dos neurônios nas camadas escondidas, codificadas como um vetor. Ao comparar objetos em função de tais "padrões de ativação", em vez de utilizar os valores das características preditivas dos objetos, torna-se possível gerar explicações como "o modelo classificou tal objeto de tal forma pois o considerou parecido com tal outro objeto".

Em aplicações em que a interpretabilidade é uma das prioridades, como em aplicações médicas, modelos baseados em regras podem ser preferíveis, por serem mais transparentes. Como Freitas (2014) observa, tais modelos são, de forma geral, considerados os mais compreensíveis. As regras utilizadas nesses sistemas frequentemente são da forma SE *condições* ENTÃO *classes*, sendo que *condições* é uma conjunção de predicados sobre as características dos objetos que estão sendo classificados, por exemplo: SE *comprimento_sepala* ≥ 2 *E comprimento_petala* ≤ 1 ENTÃO *classe=iris-setosa*. É comum que regras de classificação sejam descritas em função de suas partes: antecedente (testes realizados) e consequente (classes preditas).

Em relação às métricas utilizadas para quantificar o quão interpretável é um modelo baseado em regras, Freitas (2014) argumenta que a medida *model size*, frequentemente utilizada, não é ideal, visto que não contempla aspectos como dependências entre regras, presentes em modelos que empregam listas de regras e ausentes em modelos que utilizam conjuntos de regras.

É importante observar que o termo *model size* é utilizado de forma genérica para se referir à complexidade do modelo; em árvores de decisão, pode mensurar o número de nós da árvore ou o número de caminhos da raiz até as folhas; em redes neurais, o número de neurônios ou conexões entre neurônios; em classificadores baseados em regras, o número de regras ou predicados presentes nas regras; em redes Bayesianas, o número de arcos.

Os trabalhos apresentados a seguir discutem outras abordagens ou métricas relacionadas à interpretabilidade.

Clare e King (2001), após treinarem o modelo, optaram por "quebrar" regras de classificação multirrótulo em regras de classificação monorrótulo; cada regra que predizia mais de uma classe era transformada em várias regras com mesmo antecedente (mesmas condições), mas prevendo apenas uma classe.

Smaldon e Freitas (2007) argumentam que, no contexto de classificação em conjuntos de dados esparsos de bioinformática, a utilização de predicados que testam a presença (em vez da ausência) de certos atributos tornam as regras mais interpretáveis e diminuem a quantidade de regras geradas, mas ao custo de poder preditivo.

Otero e Freitas (2013) argumentam que conjuntos (coleções não ordenadas) de regras são mais facilmente interpretados do que listas (coleções ordenadas). De fato, a ordenação, presente em listas, cria uma interdependência entre as regras. Por exemplo, para analisar a n-ésima regra

de uma lista, é necessário considerar as $n - 1$ regras anteriores. Um conjunto de regras, por sua vez, pode se tornar inconsistente caso contenha regras que cubram uma mesma região do *feature space* mas associam diferentes classes a tal região. No Capítulo 3, discute-se com mais detalhes essa questão da consistência de conjuntos de regras.

Gerar um modelo que seja simultaneamente interpretável e eficaz configura uma tarefa de otimização multiobjetivo (FREITAS, 2014), pois é possível aumentar o tamanho de um modelo, o que diminui sua interpretabilidade, sem melhorar seu poder preditivo. Analogamente, é possível melhorar o poder preditivo de um modelo sem aumentar seu tamanho, por exemplo, ao selecionar valores mais apropriados para os testes dos nós de uma árvore de decisão.

2.3 Otimização Multiobjetivo

Diversos problemas do mundo real envolvem a otimização simultânea de múltiplos objetivos não comensuráveis[5] (ZITZLER, 1999; MARLER; ARORA, 2004). Nestes problemas, otimizar uma solução em relação a um objetivo pode torná-la inaceitável para outros objetivos. Um exemplo na área de mineração de dados é a seleção de atributos, em que se deseja minimizar a quantidade de atributos selecionados e a quantidade de informações relevantes perdidas (FREITAS, 2004).

Freitas (2004) discute as três principais abordagens utilizadas para realizar otimização multiobjetivo na área de mineração de dados: a composição de objetivos, a ordenação lexicográfica e a abordagem de Pareto. A composição de objetivos, amplamente utilizada, transforma o problema de otimização multiobjetivo em um problema de otimização com um único objetivo. Tal conversão pode se dar, por exemplo, pela soma ponderada dos objetivos; mas isso não é trivial, pois é necessário escolher pesos apropriados para os objetivos, frequentemente descritos em unidades / escalas diferentes, e tais pesos incorrem em diferentes vieses na busca por soluções.

Na ordenação lexicográfica, associa-se a cada objetivo uma prioridade. Dadas duas soluções, para identificar qual é melhor, comparam-se as qualidades das soluções em relação ao objetivo com maior prioridade; caso elas sejam equivalentes, então são comparadas em relação à qualidade do objetivo com segunda maior prioridade, e assim por diante. Essa abordagem resolve diversos problemas que afetam a composição de objetivos, por exemplo: ela não mistura objetivos medidos em diferentes unidades e não requer a determinação de pesos para os objetivos.

A abordagem de Pareto é baseada no conceito de dominação de Pareto. Diz-se que uma solução domina outra se, e somente se, ela for superior em relação a pelo menos um objetivo e não for inferior em nenhum outro. Para exemplificar esse conceito, a Tabela 1 apresenta um conjunto de soluções para um problema em que se deseja maximizar os valores O_1 e O_2.

Na abordagem de Pareto, as melhores soluções são as não dominadas; no exemplo da

[5] Duas gradezas são incomensuráveis se não podem ser medidas com a mesma unidade, como idade (segundos/anos) e altura (metros).

Tabela 1 – Soluções para problema de otimização multiobjetivo

Solução	O_1	O_2	É dominada por	Domina
A	2	3	C, D, F	
B	3	2	C, D, F	
C	4	6		A, B
D	6	5		A, B, F
E	8	0		
F	6	4	D	A, B

Tabela 1, seriam as soluções C, D e E. Essa é uma diferença notável entre a abordagem de Pareto e a ordenação lexicográfica; algoritmos baseados na abordagem de Pareto retornam múltiplas soluções em vez de apenas uma, permitindo que o usuário analise soluções com diferentes compromissos nas qualidades dos objetivos.

Se uma solução não é dominada por nenhuma outra, ela é chamada de *Pareto optimal*; o conjunto de todas as soluções *Pareto optimal* é chamado conjunto ótimo de Pareto. Zitzler (1999) observa que a tarefa de gerar uma aproximação do conjunto ótimo de Pareto é uma tarefa de otimização multiobjetivo, pois deseja-se, simultaneamente, minimizar a distância da aproximação até o conjunto real e fazer com que as soluções da aproximação sejam tão diversas quanto possível.

Algoritmos Genéticos são bastante utilizados na resolução de problemas de otimização multiobjetivo (KONAK et al., 2006), pois podem gerar aproximações do conjunto ótimo de Pareto a cada iteração, e com a utilização de funções de fitness e algoritmos de seleção apropriados, tais soluções podem ser bastante diversas (ZITZLER, 1999).

2.4 Algoritmos Genéticos

Algoritmos Genéticos (GAs) são inspirados na teoria da evolução de Darwin (EIBEN et al., 2003), que explica como uma coleção de indivíduos, vivendo em um ambiente com uma quantidade finita de recursos, modifica-se ao longo do tempo, frequentemente gerando indivíduos mais bem adaptados a esse ambiente.

Nessa concepção, indivíduos mais bem adaptados tendem a sobreviver por mais tempo e, consequentemente, têm mais oportunidades para se reproduzirem e passar suas características para seus descendentes. Dessa forma, há uma tendência de que a cada geração, as características que tornam indivíduos mais bem adaptados tornem-se mais comuns na população.

Quando a Teoria da Evolução foi proposta, o mecanismo que viabilizava a transmissão de características entre gerações era desconhecido. Hoje, porém, a genética explica que tais características são, em parte, consequência da expressão do genoma do indivíduo e que, durante a reprodução, o genoma dos descendentes é criado a partir de uma combinação dos genomas de seus genitores.

GAs codificam soluções de problemas como cromossomos (parcelas do genoma) e, ao simular mecanismos da Teoria da Evolução, "evoluem" uma coleção de soluções. A tendência é que, a cada iteração do algoritmo (cada geração da população), as soluções presentes (indivíduos da população) resolvam o problema de forma mais eficiente (os indivíduos sejam mais bem adaptados). Os mecanismos simulados da Teoria da Evolução são a reprodução, a mutação e a seleção, discutidos a seguir.

Simular a reprodução permite a exploração do espaço de soluções ao combinar parâmetros de soluções já existentes. Por exemplo, se um indivíduo I_1 apresenta bom desempenho em um aspecto a_1, e um indivíduo I_2 apresenta um bom desempenho em um aspecto a_2, é possível que um indivíduo descrito por uma combinação dos genomas de I_1 e I_2 apresente um bom desempenho simultaneamente em a_1 e a_2.

A mutação, de forma similar à reprodução, também permite a exploração do espaço de soluções, mas, em vez de gerar novas soluções a partir da combinação de soluções conhecidas, novas soluções são geradas a partir de pequenas modificações em soluções individuais. Ao utilizar valores desconhecidos para os parâmetros das soluções (em vez de valores já utilizados em outras soluções), novas regiões podem ser exploradas.

A seleção, ao simular a morte de indivíduos menos aptos, tem dois efeitos importantes na busca por soluções melhores: primeiro, ela regula o tamanho da população e, consequentemente, o custo computacional para executar a próxima iteração do algoritmo; segundo, ela direciona a busca no espaço de soluções, posto que, ao eliminar os indivíduos menos aptos, a busca é focada nos valores de parâmetros mais promissores das soluções.

Para simular a seleção, deve haver alguma função que quantifique o quão adequada é uma solução (o quão bem adaptado é um indivíduo); essa função é chamada função de fitness. Se os indivíduos que o GA está evoluindo representam classificadores, a função de fitness poderia retornar, por exemplo, a precisão ou o F-Score dos classificadores.

Dos vários algoritmos genéticos descritos na literatura, os interessantes para esta pesquisa são os *Multiobjective Optimization Genetic Algorithms* (MOGAs), isto é, aqueles elaborados especificamente para a tarefa de otimização multiobjetivo. E destes, os realmente interessantes são os elitistas, como os discutidos a seguir. Um algoritmo genético é elitista se ele implementa algum mecanismo que garante a sobrevivência dos indivíduos mais bem adaptados. No trabalho de Villalobos-Arias et al. (2006), prova-se que essa propriedade é necessária para a convergência de certos algoritmos genéticos multiobjetivo.

O algoritmo NSGA-II, proposto por Deb et al. (2002), seleciona quais indivíduos comporão a população da próxima geração através da organização dos indivíduos em frontes, de forma que os indivíduos pertencentes a um dado fronte são dominados apenas por aqueles pertencentes aos frontes anteriores; os indivíduos do primeiro fronte, portanto, são não dominados. A nova população é formada pela união de frontes até que a união com um novo fronte torne a população

maior que o limite populacional. Como apenas uma parcela dos indivíduos desse último fronte pode ser adicionada à população, o algoritmo emprega um mecanismo de seleção baseado na distância de cada indivíduo até o indivíduo mais próximo. Ao remover os indivíduos com as menores distâncias, melhora-se a diversidade da população.

O algoritmo SPEA, proposto por Zitzler e Thiele (1999), incorpora ideias discutidas em trabalhos publicados anteriormente, como a utilização de uma população externa e a poda da população externa utilizando agrupamento. Os autores também apresentam o conceito de *força*, que se aplica apenas a indivíduos da população externa. A força de um indivíduo é dada pela proporção entre o número de indivíduos da população interna que ele domina e o tamanho da população interna.

O fitness de cada indivíduo da população interna é dado pela soma das forças dos indivíduos da população externa que o dominam, acrescida de 1. O fitness de cada indivíduo da população externa é igual à sua força. Somar 1 aos fitnesses da população interna garante que eles serão maiores que os da população externa; nesse algoritmo o fitness deve ser minimizado.

Ao utilizar essa função de fitness, indivíduos que são dominados por muitos indivíduos muito fortes (i.e., que dominam muitos indivíduos), são desfavorecidos. Esse mecanismo melhora a diversidade das soluções, pois tais indivíduos dominados apresentam compromissos similares nos objetivos sendo otimizados.

Quando o tamanho da população externa ultrapassa o tamanho limite, o algoritmo emprega o método de agrupamento *average linkage* (MORSE, 1980) para n *clusters*, sendo n o tamanho máximo da população. Apenas os medoides dos *clusters* são mantidos na população externa.

O algoritmo SPEA2, proposto por Zitzler et al. (2001), modifica dois aspectos do algoritmo original. Primeiro, os fitnesses de todos os indivíduos passam a ser dados em função dos indivíduos que eles dominam e daqueles que os dominam, remediando o problema da função de fitness original que atribuía valores iguais a indivíduos dominados pelos mesmos membros da população externa, mesmo que um deles dominasse o outro. A segunda mudança refere-se ao mecanismo de poda da população externa, que passa a utilizar as distâncias dos indivíduos até seus vizinhos. Ao priorizar indivíduos cujos vizinhos mais próximos estão distantes, a remoção de soluções nos extremos do espaço de soluções é evitada, melhorando a diversidade.

O algoritmo PESA, proposto por Corne et al. (2000), como o SPEA e o SPEA2, também utiliza uma população externa para armazenar os melhores indivíduos encontrados. O mecanismo utilizado para selecionar quais indivíduos participarão da etapa da reprodução é o mesmo usado para selecionar quais indivíduos permanecerão na população externa após a poda; tal mecanismo é baseado em um *squeeze factor*. Para computar o *squeeze factor* dos indivíduos, o espaço-objetivo é dividido em N^M hipercubos, sendo que N é um parâmetro do algoritmo e M é o número de objetivos sendo otimizados, ou seja, o número de dimensões do espaço-objetivo. O

squeeze factor de um indivíduo é a quantidade de indivíduos presentes no mesmo hipercubo; escolher indivíduos com menor *squeeze factor*, portanto, aumenta a diversidade da população.

É interessante mencionar que a utilização de GAs especificamente para a geração de regras de classificação também é encontrada na literatura como *Learning Classifier Systems* (LCS) (VALLIM, 2009).

As duas abordagens mais frequentemente utilizadas na modelagem de LCSs são a abordagem de Michigan e a abordagem de Pittsburg. Na abordagem de Michigan, cada indivíduo da população representa uma regra de classificação, e a população como um todo representa um classificador.

A abordagem de Pittsburg, adotada neste trabalho, define que cada indivíduo representa um classificador completo, isto é, uma coleção de regras de classificação. Ao utilizar uma função de fitness que retorne um vetor (contendo, por exemplo, a acurácia e *model size*), torna-se possível empregar os algoritmos discutidos anteriormente para otimizar simultaneamente o poder preditivo e a interpretabilidade da população, como é feito nos algoritmos apresentados no Capítulo seguinte.

Capítulo 3
DESENVOLVIMENTO TEÓRICO

Neste Capítulo, são apresentados a definição de consistência, no contexto de regras e conjuntos de regras, e os três algoritmos desenvolvidos nesta pesquisa: *Constrained Feature Space Greedy Search* (CFSGS), *Constrained Feature Space Box-Enlargement* (CFSBE), e ***m**ulti-objec**t**ive evolutio**n**ary alg**o**rithm for consis**t**ent and interpre**t**able m**u**lti-label **r**ules* (MINOTAUR).

O CFSGS encontra regiões do *feature space*[1] não cobertas por nenhuma regra e que, portanto, poderiam ser utilizadas na criação do antecedente de uma nova regra consistente[2] com aquelas já existentes. O CFSBE tem o mesmo objetivo do CFSGS, mas com complexidade assintótica reduzida e desempenho teoricamente inferior.

O MINOTAUR é um algoritmo evolutivo que gera modelos de classificação multirrótulo baseados em conjuntos de regras consistentes, utilizando o CFSGS ou CFSBE para orientar o processo de geração de suas regras. Quando o MINOTAUR precisa criar uma regra para um dado modelo, o CFSGS ou o CFSBE é invocado para identificar regiões do *feature space* não cobertas pelo modelo. O MINOTAUR, então, cria o antecedente da nova regra de forma que ele cubra apenas um volume interno à região encontrada.

É interessante observar que o MINOTAUR foi originalmente desenvolvido como um algoritmo genético, porém, experimentos preliminares indicaram que a etapa de reprodução (*crossover*) não alterava significativamente os modelos gerados, e removê-la poderia simplificar o algoritmo, especialmente quanto à sua implementação. Com a remoção dessa etapa, comumente considerada fundamental aos algoritmos genéticos, julgou-se mais apropriado recategorizar o algoritmo como evolutivo, em vez de genético.

[1] Ao longo texto, e especialmente neste capítulo, são utilizados vários termos e expressões em inglês que possuem análogos em português, por exemplo *feature* em vez de *atributo*. Optou-se pela adoção de termos em inglês para que houvesse consistência entre os termos utilizados no texto, nos artigos e, principalmente, na implementação dos algoritmos descritos.

[2] A definição de regras consistentes é apresentada na Seção 3.1.

3.1 Contextualização e definições

3.1.1 Consistência entre regras

De acordo com Otero e Freitas (2013), modelos de classificação baseados em conjuntos de regras podem ser mais interpretáveis que modelos baseados em listas de regras. A premissa é que a ordem de aplicação das regras, presente em listas e ausente em conjuntos, dificulta a análise de uma regra isoladamente. Por exemplo, para identificar a região do *feature space* coberta pela n-ésima regra de uma lista, é necessário "subtrair" as regiões cobertas por regras anteriores.

Sem uma ordem de aplicação, uma coleção de regras torna-se inconsistente ao conter regras que cubram uma mesma região do *feature space* e predizem classes diferentes. Neste trabalho, duas regras são consideradas consistentes se seus consequentes forem iguais ou se a intersecção das regiões descritas por seus antecedentes for vazia. Analogamente, um conjunto de regras é considerado consistente se todas as suas regras forem consistentes entre si[3].

É comum a utilização de técnicas de resolução de conflito (OTERO; FREITAS, 2013) para garantir o funcionamento de modelos baseados em conjuntos de regras que podem ser inconsistentes. Por exemplo, para classificar um objeto coberto por múltiplas regras escolhe-se o consequente da regra com maior poder preditivo no conjunto de treinamento. Os algoritmos apresentados a seguir foram desenvolvidos com outra perspectiva, a de evitar a geração de regras inconsistentes e, portanto, tornar técnicas de resolução de conflito desnecessárias, potencialmente simplificando o modelo.

Em uma primeira implementação do MINOTAUR, o mecanismo utilizado para garantir que os modelos fossem consistentes era baseado em tentativa e erro: operações que geravam regras, como operadores de mutação, o faziam de forma aleatória e verificavam se adicionar a nova regra ao modelo o tornaria inconsistente; em caso positivo, a regra era descartada e iniciava-se um novo ciclo de mutação; em caso negativo, a regra era adicionada ao modelo. Experimentos preliminares sugeriram que essa abordagem era proibitivamente ineficiente.

A segunda implementação do MINOTAUR (a apresentada neste texto), utiliza outra abordagem, baseada na premissa de que o processo de geração de regras pode ser orientado. Tal orientação é realizada pelos algoritmos CFSGS e CFSBE, que identificam regiões do *feature space* que não são cobertas por nenhuma regra existente. É importante destacar que o processo de criação de uma regra, dada uma região do *feature space*, está fora do escopo dos algoritmos CFSGS e CFSBE; neste trabalho, o MINOTAUR realiza esse processo.

3.1.2 Suposições e exemplos

Os algoritmos apresentados neste Capítulo assumem que as regras possuem exatamente um *feature test* para cada *feature* do conjunto de dados, e que o antecedente de uma regra é a

[3] Um conjunto de regras vazio, portanto, é consistente.

conjunção de tais *feature tests*, como na Equação 3.1. Um *feature test* é uma função que verifica se um valor pertence a um intervalo, como na Equação 3.2. Em ambas as equações f denota o conjunto de *features*.

$$rule := test_1 \wedge test_2 \wedge \cdots \wedge test_{|f|} \tag{3.1}$$

$$test := test.lower \leq f_i < test.upper \tag{3.2}$$

É importante observar que o limite inferior de um *feature test* é inclusivo, enquanto o limite superior é exclusivo. Isso permite que *feature tests* de diferentes regras possam se "tocar" sem gerar inconsistências. Formalmente, utilizar o limite inferior como inclusivo e o superior como exclusivo permite que o *feature test* relativo à *feature* i, de uma regra r_1, possa ser igual ao limite inferior do *feature test* relativo a mesma *feature* i, de uma regra r_2, sem gerar uma inconsistência. Por exemplo, as regras da Figura 3 seriam inconsistentes se a comparação do limite superior fosse inclusiva, pois ambas cobririam uma instância do conjunto de dados com $age = 18$, porém prediriam classes diferentes.

$rule_a :=$ **IF** 0 $\leq$ age $<$ 18 **THEN** class=minor
$rule_b :=$ **IF** 18 $\leq$ age $<$ 99 **THEN** class=adult

Figura 3 – Regras com *feature tests* que se "tocam"

A última suposição é que o conjunto de dados possui apenas *features* contínuas, o que permite a visualização e o tratamento do antecedente das regras como um hiper-retângulo em um espaço com $|f|$ dimensões, sendo f o conjunto de *features*. Modificações dos algoritmos, para que eles operem sobre conjuntos de dados com *features* categóricos, são discutidas no Capítulo 5, e suas implementações delegadas a trabalhos futuros.

Finalmente, visando auxiliar a explicação dos algoritmos, a Figura 4 foi sintetizada para representar um problema fictício de classificação com duas *features* contínuas, em que sete instâncias do conjunto de dados são representadas por pontos pretos e três regras de classificação são representadas por retângulos coloridos. Os antecedentes das regras representadas na Figura 4 são formalmente descritos pelas Equações 3.3, 3.4 e 3.5.

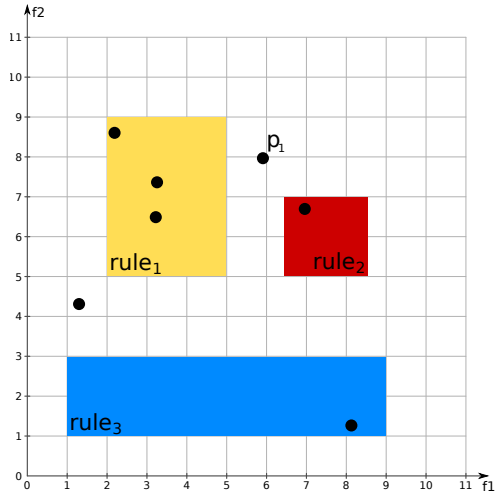


Figura 4 – Regras e instâncias de exemplo

$$rule_1 := \begin{cases} test_1 := 2 \leq f_1 < 5 \\ test_2 := 5 \leq f_2 < 9 \end{cases} \tag{3.3}$$

$$rule_2 := \begin{cases} test_1 := 6.5 \leq f_1 < 8.5 \\ test_2 := 5 \leq f_2 < 7 \end{cases} \tag{3.4}$$

$$rule_3 := \begin{cases} test_1 := 1 \leq f_1 < 9 \\ test_2 := 1 \leq f_2 < 3 \end{cases} \tag{3.5}$$

3.2 Algoritmos de orientação de busca

3.2.1 *Constrained Feature Space Greedy Search*

A ideia central do *Constrained Feature Space Greedy Search* (CFSGS) é que regiões do *feature space* que não são cobertas por nenhuma regra existente podem ser utilizadas na criação de antecedentes de novas regras que seriam, por definição, consistentes com as regras já existentes.

Como a região do *feature space* coberta por uma regra é dada pela conjunção de seus *feature tests*, a região não coberta por uma regra é dada pela negação dessa conjunção, conforme Equação 3.6. Analogamente, a região não coberta por um *feature test* é descrita pela sua negação, conforme Equação 3.7.

$$
\begin{aligned}
rule &:= test_1 \wedge \cdots \wedge test_n \\
\neg rule &:= \neg(test_1 \wedge \cdots \wedge test_n) \\
\neg rule &:= (\neg test_1) \vee \cdots \vee (\neg test_n)
\end{aligned}
\tag{3.6}
$$

$$
\begin{aligned}
test_i &:= lower_i \leq f_i < upper_i \\
test_i &:= (lower_i \leq f_i) \wedge (f_i < upper_i) \\
\neg test_i &:= \neg((lower_i \leq f_i) \wedge (f_i < upper_i)) \\
\neg test_i &:= \neg(lower_i \leq f_i) \vee \neg(f_i < upper_i) \\
\neg test_i &:= (lower_i > f_i) \vee (f_i \geq upper_i) \\
\neg test_i &:= (f_i < lower_i) \vee (f_i \geq upper_i)
\end{aligned}
\tag{3.7}
$$

Por exemplo, ao negar os *feature tests* da regra r_1, descrita pela Equação 3.3 e representada pelo retângulo amarelo na Figura 4, são obtidas quatro desigualdades (ou *constraints*), cada uma descrevendo uma região do *feature space* não coberta pela regra r_1:

- $f_1 < 2$, a região à esquerda do retângulo amarelo;
- $f_1 \geq 5$, a região à direita do retângulo amarelo;
- $f_2 < 5$, a região abaixo do retângulo amarelo;
- $f_2 \geq 9$, a região acima do retângulo amarelo;

Aplicando este processo de geração de desigualdades a um conjunto de regras, obtém-se uma coleção de *constraints* C, em que $C_{i,j}$ denota a j-ésima *constraint* obtida a partir da i-ésima regra. Como exemplo, a Tabela 2 contém as *constraints* geradas a partir das regras $\{r_1, r_2, r_3\}$, ilustradas na Figura 4.

Tabela 2 – Exemplo de conjunto de *constraints*

Índice	*Constraint*
$C_{1,1}$	$f_1 < 2$
$C_{1,2}$	$f_1 \geq 5$
$C_{1,3}$	$f_2 < 5$
$C_{1,4}$	$f_2 \geq 9$
$C_{2,1}$	$f_1 < 6.5$
$C_{2,2}$	$f_1 \geq 8.5$
$C_{2,3}$	$f_2 < 5$
$C_{2,4}$	$f_2 \geq 7$
$C_{3,1}$	$f_1 < 1$
$C_{3,2}$	$f_1 \geq 9$
$C_{3,3}$	$f_2 < 1$
$C_{3,4}$	$f_2 \geq 3$

Ao utilizar o Algoritmo 1 é possível organizar as *constraints* de C em um grafo dirigido acíclico (*Directed Acyclic Graph - DAG*), em que *caminhos consistentes* que vão da raiz até um vértice folha representam regiões não vazias do *feature space* que não são cobertas por nenhuma regra existente. Um caminho é considerado consistente se as *constraints* representadas por seus vértices podem ser todas satisfeitas simultaneamente, como as descritas na Equação 3.8.

$$\{C_{1,1}, C_{2,1}, C_{3,1}\} = (f_1 < 2) \wedge (f_1 < 6.5) \wedge (f_1 < 1) \tag{3.8}$$

O Algoritmo 1 é inicializado com a construção de um DAG contendo apenas um vértice, chamado de $root$. Então, varre-se o conjunto de *constraints* C, em ordem de *constraints* pertencentes a uma mesma regra (linha 4) e para cada *constraint*, adiciona-se ao DAG um novo vértice que a representa (linha 6). Caso a *constraint* pertença à primeira regra processada (linha 7), cria-se uma aresta partindo do $root$ até o novo vértice (linha 8); caso contrário, criam-se arestas partindo dos vértices relativos à última regra processada até o novo vértice, ou seja, dos vértices do nível anterior do grafo até o novo vértice (linhas 9 ... 13).

Como exemplo, aplicar o Algoritmo 1 às regras presentes na Figura 4 resulta no grafo da Figura 5. Como foi assumido que os *feature tests* têm o formato da Equação 3.2, o parâmetro n_f é igual a duas vezes o número de *features* no conjunto de dados.

Algorithm 1 CFSGS - DAG construction

Require:
C: collection of constraints
n_r: number of existing rules
n_f: number of constraints generated from a rule

function $build\_dag(C, n_r, n_f)$
  $G \leftarrow$ new directed graph
  $G.V \leftarrow \{root\}$
  for $i \in \{1, 2, \ldots, n_r\}$ **do**
    for $j \in \{1, 2, \ldots, n_f\}$ **do**
      $G.V \leftarrow G.V \cup \{C_{i,j}\}$
      if $i = 1$ **then**
        $G.E \leftarrow G.E \cup \{(root, C_{i,j})\}$
      else
        for $k \in \{1, 2, \ldots, n_f\}$ **do**
          $G.E \leftarrow G.E \cup \{(C_{i-1,k}, C_{i,j})\}$
        end for
      end if
    end for
  end for
  return G
end function

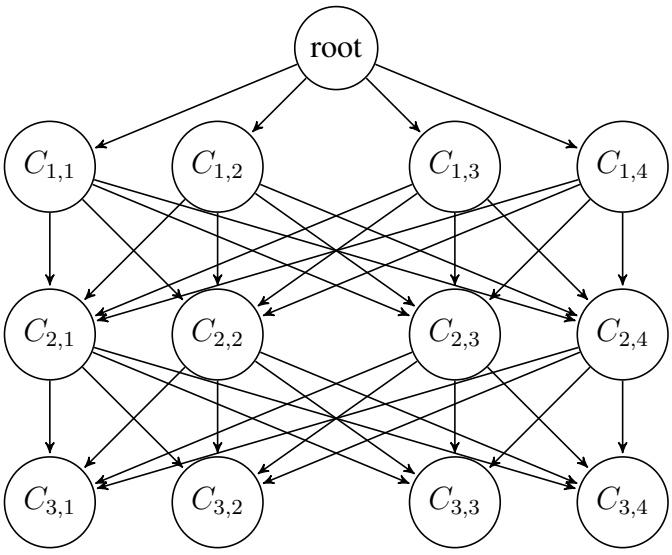


Figura 5 – *Constraints* organizadas em um DAG

Para encontrar caminhos consistentes no DAG executa-se uma busca em profundidade partindo da raiz e tentando atingir um vértice folha. Se adicionar um vértice ao caminho fizer com que ele se torne inconsistente, aplica-se um mecanismo de *backtrack*.

O DAG poderia ser explorado de forma exaustiva para que fossem encontradas todas as regiões do *feature space* não cobertas pelas regras existentes, mas, como encontrar uma região

é suficiente para a criação de uma nova regra, o processo pode ser interrompido assim que a primeira região for encontrada.

É importante observar que a ordem em que os vértices são explorados determina a ordem em que os caminhos são encontrados. Se, por exemplo, os vértices forem explorados seguindo uma ordem lexicográfica, isto é, se $C_{1,1}$ for explorado antes de $C_{1,2}$, então os primeiros caminhos encontrados terão um viés para as regiões inferiores das primeiras *features*. No caso representado pela Figura 4, as primeiras regiões encontradas seriam aquelas do canto inferior esquerdo. Um modo de reduzir tal viés seria explorar aleatoriamente os vértices de cada nível do grafo.

O CFSGS pode encontrar todas as regiões do *feature space* não cobertas pelas regras existentes, porém seu custo computacional é proibitivo. Por exemplo, o número de caminhos que devem ser explorados durante a busca por caminhos consistentes pode ser exponencial em função do número de regras. No pior cenário, as regras de entrada cobrem todo o *feature space*; consequentemente, não existe um caminho consistente e todos os caminhos são explorados. Como o grafo possui n_r níveis, cada um com n_f nós, existem $(n_f)^{n_r}$ caminhos.

3.2.2 *Constrained Feature Space Box-Enlargement*

O algoritmo CFSBE partilha a interpretação geométrica do CFSGS, porém, utiliza mais informações disponíveis (como instâncias do conjunto de dados) e opera com um espaço de busca reduzido, o que diminui seu custo computacional e, teoricamente, sua eficiência. Na Seção 3.2.3, são discutidas as diferenças entre o CFSGS e o CFSBE, explicitando quais regiões do *feature space* o CFSBE não é capaz de encontrar e o CFSGS sim.

A ideia central do CFSBE é: se, ao criar uma regra, forem anotadas as instâncias do conjunto de dados que ela cobre, por exemplo, utilizando um *associative array*, então, torna-se possível identificar, de forma eficiente, quais instâncias não são cobertas por nenhuma regra. Tais instâncias poderiam ser utilizadas para direcionar a busca por regiões do *feature space* não cobertas por nenhuma regra.

Dada uma instância não coberta por nenhuma regra, chamada *seed*, o CFSBE cria um hiper-retângulo em volta dessa instância e o "aumenta" ao longo de cada dimensão até que qualquer expansão adicional resulte em uma intersecção do hiper-retângulo e das regiões descritas pelas regras existentes. Uma regra criada dentro desse hiper-retângulo, isto é, uma regra cujo antecedente cubra apenas um volume interno ao hiper-retângulo, seria, por definição, consistente com as regras já existentes.

Destaca-se que a ordem em que as dimensões são expandidas altera o hiper-retângulo resultante. Por exemplo, ao utilizar o ponto p_1 da Figura 4 como $seed$ e (f_1, f_2) como ordem de expansão, o resultado é o retângulo verde ilustrado na Figura 6; por outro lado, se a expansão seguisse (f_2, f_1), o resultado seria o retângulo verde ilustrado na Figura 7.

Para identificar os limites da expansão do hiper-retângulo em uma dimensão é necessário

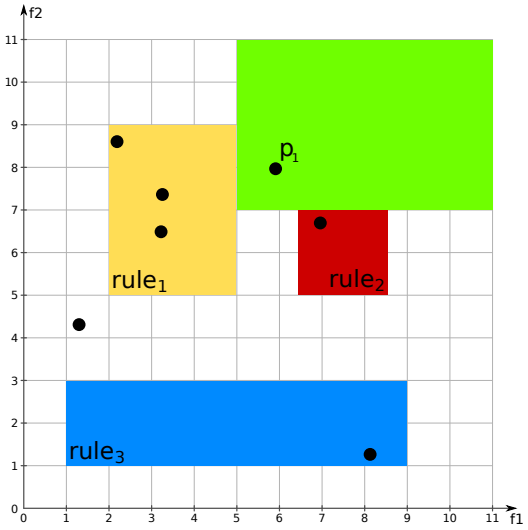


Figura 6 – Resultado da expansão seguindo (f_1, f_2)

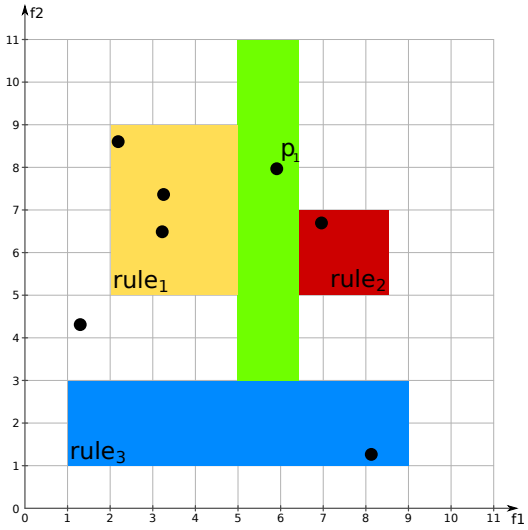


Figura 7 – Resultado da expansão seguindo (f_2, f_1)

verificar com quais regras o hiper-retângulo "intersecta" nas *outras* dimensões. Por exemplo, embora o ponto p_1 da Figura 4 não seja coberto por nenhuma regra, ele passa no *feature test* da feature f_2 da regra r_1, isto é, ele "intersecta" a regra r_1 na dimensão f_2. Se fosse criado um hiper-retângulo em volta de p_1 e a expansão começasse por f_1, eventualmente surgiria uma intersecção do hiper-retângulo com r_1. O Algoritmo 2 descreve como o CFSBE realiza a criação e a expansão de um hiper-retângulo evitando que este sobreponha as áreas descritas pelos antecedentes de regras já existentes.

A complexidade assintótica do CFSBE $\in \mathcal{O}(|order|^2 \cdot |rules|)$. Para estimá-la, observa-se no Algoritmo 2 que: I) o corpo do algoritmo consiste em dois *loops* aninhados, o externo com $|order|$ iterações e o interno com $|rules|$ iterações; II) o *loop* interno invoca a função $intersect$, que, por sua vez, consiste em um *loop* com $|order| - 1$ iterações; e III) o custo de rastreamento das instâncias não cobertas por nenhuma regra tem custo amortizado $\in \mathcal{O}(1)$.

Algorithm 2 Constrained Feature-Space Box-Enlargement

Require:
$rules$: a set of consistent rules
$seed$: a dataset instance not covered by any rule in $rules$
$order$: a permutation of $\{1, 2, .., |f|\})$, where $|f|$ is the number of features of the dataset

function $cfsbe(seed, rules, order)$
  $h \leftarrow$ hyper-rectangle with $|order|$ dimensions, zero volume and centered around $seed$
  for $d \in order$ **do**
    $h_d.lower \leftarrow -\infty$
    $h_d.upper \leftarrow +\infty$
    for $r \in rules$ **do**
      if $intersects(h, r, d)$ **then**
        if $seed_d \geq r_d.upper$ **then**
          $h_d.lower \leftarrow max(h_d.lower, r_d.upper)$
        else ▷ Meaning $x_d < r_d.lower$, otherwise $seed$ would be covered by r
          $h_d.upper \leftarrow min(h_d.upper, r_d.lower)$
        end if
      end if
    end for
  end for
  return h
end function

function intersects(h: hyperrectangle, r: rule, s: dimension to skip)
  for $i \in \{1, 2, \ldots, |r|\} - \{s\}$ **do**
    $u \leftarrow h_i.upper > r_i.lower$
    $l \leftarrow h_i.lower < r_i.upper$
    if $\neg(u \wedge l)$ **then**
      return false
    end if
  end for
  return true
end function

3.2.3 Comparação entre CFSGS e CFSBE

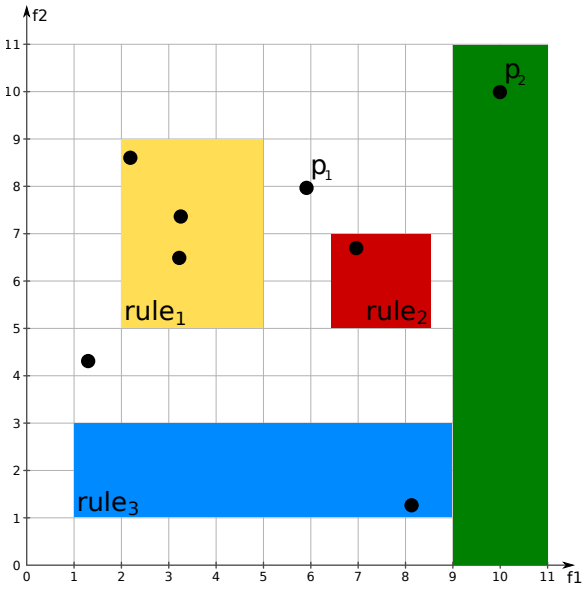
O algoritmo CFSGS, como mencionado anteriormente, tem um custo computacional proibitivamente alto, porém, é teoricamente capaz de encontrar todas regiões do *feature space* não cobertas pelas regras existentes. O CFSBE, embora não seja capaz de encontrar todas as regiões, tem um custo custo computacional suficientemente baixo para que possa ser incorporado em um algoritmo evolutivo.

As regiões que o CFSBE não é capaz de encontrar e o CFSGS sim, são aquelas que não contêm instâncias do conjunto de dados, isto é, $seeds$ para o CFSBE. Se uma região do *feature space* não contém instâncias do conjunto de dados, então as regras que forem criadas em tais regiões terão cobertura zero e, consequentemente, não será possível mensurar a qualidade de tais regras.

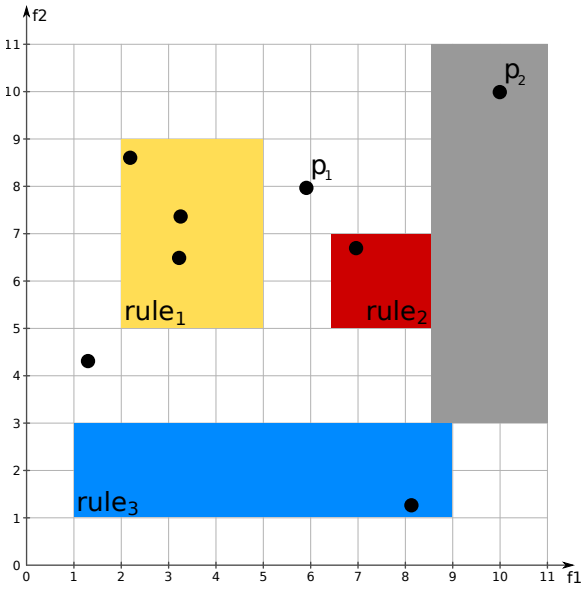
Para exemplificar esse fenômeno, considere a região abaixo do retângulo azul da Figura 4. O CFSGS é capaz de encontrar tal região e o CFSBE não, porém, ela não contém instâncias do conjunto de dados. Argumenta-se, portanto, que essa limitação não seja particularmente relevante.

Um efeito similar ocorre se houver apenas uma instância em uma dada região. Considere o conjunto de dados fictício da Figura 4 com um novo ponto p_2, cujas coordenadas são $(10, 10)$. É possível criar quatro hiper-retângulos que cobrem p_2, não intersectam outros retângulos e têm área máxima, ilustrados na Figura 8. Os retângulos que o CFSBE não é capaz de encontrar (a partir de uma expansão de um retângulo que inicialmente cobre apenas p_2) são coloridos de cinza, e os retângulos que o CFSBE é capaz de encontrar são coloridos de verde. Os retângulos cinza não podem ser encontrados pois a expansão realizada pelo CFSBE é *greedy*, isto é, a expansão ao longo de uma dimensão só começa depois que a expansão na dimensão anterior não seja mais possível.

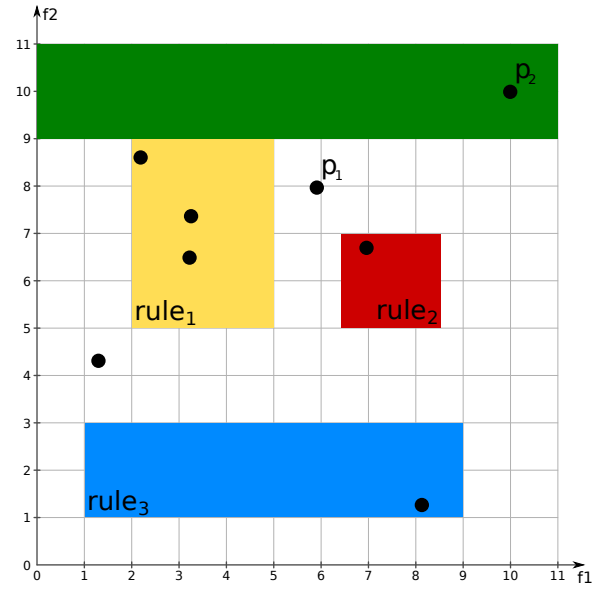
De forma análoga ao caso anterior, as regiões cinza não são particularmente interessantes, visto que: contêm apenas uma instância, já coberta pelas regiões verdes, como nas Figuras 8 (a), (b) e (c); ou contêm instâncias que poderiam ser cobertas invocando o CFSBE com outra $seed$, como na Figura 8 (d). O CFSBE, portanto, oferece um compromisso aceitável entre regiões que pode identificar e custo computacional.



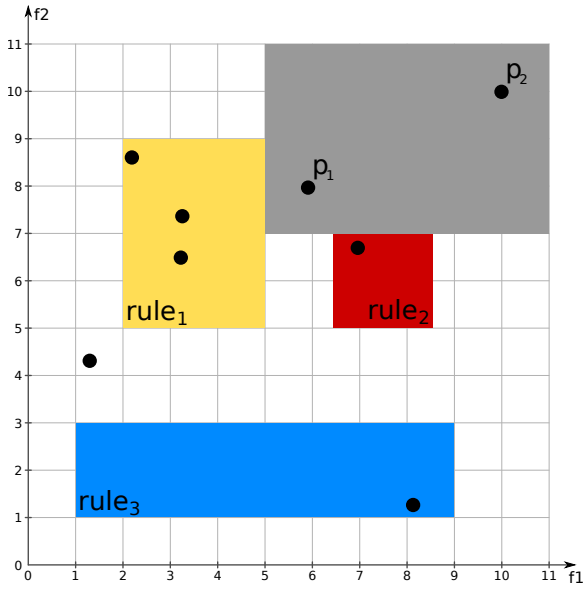
(a) Região encontrada através de expansão *greedy*, começando por f_2, e que pode ser encontrada pelo CFSBE



(b) Região encontrada através de expansão *não-greedy*, começando por f_2, e que não pode ser encontrada pelo CFSBE



(c) Região encontrada através de expansão *greedy*, começando por f_1, e que pode ser encontrada pelo CFSBE



(d) Região encontrada através de expansão *não-greedy*, começando por f_1, e que não pode ser encontrada pelo CFSBE

Figura 8 – Regiões que podem ou não ser encontradas pelo CFSBE, em função da estratégia de expansão *greedy*

3.3 MINOTAUR

O principal algoritmo desenvolvido nesta pesquisa, chamado ***m**ulti-objec**t**ive evolutio**n**ary alg**o**rithm for consis**t**ent and interpret**a**ble m**u**lti-label **r**ules* (MINOTAUR), gera, para um dado conjunto de dados, uma coleção de modelos de classificação multirrótulo baseados em conjuntos consistentes de regras. A utilização de técnicas de otimização multiobjetivo possibilita a geração de modelos com diferentes compromissos entre poder preditivo e interpretabilidade.

3.3.1 Modelagem

Cada indivíduo da população é um conjunto consistente de regras com no mínimo uma regra e sem número máximo; diferentes modelos podem conter diferentes quantidades de regras. Cada regra de classificação possui duas partes, o antecedente e o consequente.

O antecedente de uma regra segue o formato descrito na Seção 3.1, e os *feature tests* do antecedente seguem o formato descrito na Seção 3.1. Os valores utilizados como limites de um *feature test* podem ser $-\infty$, ∞ ou qualquer valor presente no conjunto de dados para a *feature* sendo testada.

Como os algoritmos requerem que todas as regras possuam o mesmo número de *feature tests*, os valores $-\infty$ e ∞ são utilizados para indicar que o *feature test*, ou pelo menos parte dele, não é importante para a regra e, portanto, tal *feature test* poderia ser removido ou simplificado em uma etapa de pós-processamento. Por exemplo, um *feature test* com valores infinitos para ambos os limites, como $ft(age) := -\infty \leq age < \infty$, poderia ser removido da regra, pois é tautológico.

O consequente de uma regra é um vetor com $|f|$ dimensões, sendo f o conjunto de classes do problema. Tal vetor é uma representação codificada de um sub-conjunto de f, em que o número 1 na posição i do vetor indica que o sub-conjunto contém a i-ésima classe do problema, e o número 0 indica que o esse sub-conjunto não contém tal classe.

Para um modelo classificar uma instância do conjunto de dados, verifica-se com qual regra do modelo tal instância casa; o modelo prediz o conjunto de classes codificado no consequente da regra. Como os modelos são consistentes, no máximo uma regra casará com a instância; porém, as regras de um dado modelo não necessariamente cobrem todo o *feature space*.

Caso nenhuma regra do modelo cubra a instância que está sendo classificada, emprega-se um mecanismo comumente[4] utilizado na literatura: prediz-se o "conjunto médio de classes" do conjunto de regras. Dado que o conjunto de classe às quais uma instância pertence é representado por um vetor, o "conjunto médio de classes" é dado pela média dos vetores de classes das instâncias do conjunto de dados.

[4] Os algoritmos HMC-GA (CERRI et al., 2019) e Clus-HMC (VENS et al., 2008), por exemplo, utilizam esse mesmo mecanismo.

O vetor resultante do cálculo da média frequentemente conterá valores entre 0 e 1, portanto é necessário "arredondar" os valores. Neste trabalho, valores maiores ou iguais a 0.5 foram arredondados para 1, enquanto valores menores foram arrendondados para 0.

Finalmente, a modelagem adotada pode ser resumida, de cima para baixo, como:

- A população do MINOTAUR é uma coleção de modelos de classificação multirrótulo.
- Um indivíduo da população, isto é, um modelo de classificação, é um conjunto não vazio de regras de classificação.
- Cada regra de classificação possui duas partes: o antecedente e o consequente.
- O antecedente de uma regra é um conjunto com $|f|$ *feature tests*.
  - Um *feature test* é uma função que verifica se um valor pertence a um intervalo.
- O consequente de uma regra é o conjunto de classes que ela prediz.

3.3.2 Inicialização e criação de regras

Uma das partes mais importantes do MINOTAUR é o mecanismo de criação de regras, invocado durante a etapa de inicialização e durante a etapa de mutação. Tal mecanismo faz uso do CFSBE que, como mencionado anteriormente, retorna uma região do *feature space*, mais especificamente, uma estrutura de dados que representa um hiper-retângulo com $|f|$ dimensões, sendo f o conjunto de *features*.

Existem diversos modos para criar uma regra a partir de um hiper-retângulo; nesta pesquisa foram explorados dois métodos. Para simplificar a explicação dos métodos, será assumido que o hiper-retângulo é representado como uma lista de pares, em que os valores do n-ésimo par da lista representam os limite inferior e superior do hiper-retângulo na n-ésima dimensão. Por exemplo, o hiper-retângulo relativo à região do *feature space* coberta pela $rule_1$ (Equação 3.3), seria representado pela lista $[(2, 5), (5, 9)]$.

O primeiro método, chamado método A, consiste em: para cada *feature* do conjunto de dados (par da lista do hiper-retângulo), escolhem-se aleatoriamente, entre valores presentes no conjunto de dados relativos a tal *feature*, dois valores que estão dentro do intervalo descrito pelo par do hiper-retângulo. Tais valores são utilizados como os limites do *feature test* relativo a essa *feature*. A coleção de *feature tests* gerados forma o antecedente de uma nova regra, enquanto o consequente é gerado aleatoriamente. Experimentos preliminares indicaram que modelos gerados com esse método sub-ajustam o conjunto de dados.

O segundo método, chamado método B, consiste em: dados um hiper-retângulo h e um número de instâncias alvo t, escolhe-se uma instância do conjunto de dados contida em h, chamada $seed_2$ e cria-se um novo hiper-retângulo h_2 em volta da $seed_2$. O novo hiper-retângulo é expandido até que cubra as t instâncias mais próximas da $seed_2$ ou até que h_2 alcance alguma superfície de h.

Após a expansão de h_2, os pares da lista que representam os limites de h_2 são utilizados como limites de *feature tests*, isto é, a região do *feature space* que será coberta pela regra é um volume interno a h, idêntico a h_2. O consequente da regra é criado a partir da média das classes das instâncias cobertas por h_2. Experimentos preliminares com esse método apresentaram resultados promissores e, portanto, foi o método adotado na implementação do MINOTAUR. No Algoritmo 3, é apresentado o pseudo-código relativo a este método.

Algorithm 3 Create Rule

Require:
$rules$: a set of consistent rules
Access **global** constant t, which defines the target number of instances the new rule should cover
Read-only access to **global** training dataset $dataset$

1: **function** $create\_rule(rules)$
2: $uncovered \leftarrow get\_instances\_not\_covered(dataset, rules)$
3: $seed_1 \leftarrow select\_random(uncovered)$
4: $f \leftarrow dataset.features$
5: $o_1 \leftarrow shuffle(range(1, |f|))$
6: $h_1 \leftarrow cfsbe(seed_1, rules, o_1)$
7: $covered \leftarrow get\_instances\_covered(dataset, h_1)$
8: $seed_2 \leftarrow select\_random(covered)$
9: $h_2 \leftarrow$ Hyper-rectangle with $|f|$ dimensions, zero volume and centered around $seed_2$
10: $sorted \leftarrow covered$ sorted by ascending distance to $seed_2$
11: $closest \leftarrow take(sorted, t)$
12: **for** $instance \in closest$ **do**
13: Enlarge h_2 along each dimension necessary so that it contains $instance$ or until one of its bounds reach h_1
14: **end for**
15: $antecedent \leftarrow \{new\_feature\_test(b.lower, b.upper) \mid b \in h_2\}$
16: $labels \leftarrow \{get\_vector\_of\_classes(i) \mid i \in covered\}$
17: $\vec{consequent} \leftarrow \frac{1}{1+|covered|} \cdot \sum labels$
18: **return** $new\_rule(antecedent, \vec{consequent})$
19: **end function**

Sempre que se fizer necessário criar uma regra para um indivíduo ind, executa-se o Algoritmo 3 com $rules = ind.rules$.

A etapa de inicialização consiste em criar os indivíduos da primeira geração; tais indivíduos conterão, em um primeiro momento, apenas uma regra. Para criar a primeira regra destes indivíduos, executa-se o Algoritmo 3 com $rules = \emptyset$. Como $rules = \emptyset$, o hiper-retângulo

retornado pelo CFSBE cobre todo o *feature space*; a regra resultante, portanto, depende apenas das variáveis geradas aleatoriamente, como $seed_1$, o_1 e $seed_2$.

3.3.3 Mutação e seleção

Após a criação da população inicial, o MINOTAUR inicia um ciclo mutação-seleção até que um dos dois critérios de parada seja atingido. O primeiro critério de parada é o número máximo de iterações que o ciclo pode ser executado; o segundo critério será explicado mais adiante.

A etapa de mutação consiste em gerar m novos indivíduos a partir da aplicação de operadores de mutação. Para gerar um novo indivíduo selecionam-se aleatoriamente um operador de mutação e um indivíduo da população da geração anterior e verifica-se se é possível aplicar o operador ao indivíduo. Caso seja possível, cria-se uma cópia do indivíduo e modifica-se a cópia utilizando o operador. Caso não seja possível, incrementa-se um contador de mutações falhas e seleciona-se outro par $(operador, individuo)$. No Algoritmo 4, é apresentado o pseudo-código relativo à etapa de mutação.

Algorithm 4 Generate mutants

Require:
pop: a set of individuals
$op\_weights$: weights associated with each mutation operator
max_fails: maximum number of mutation attempts that can fail in a single generation
$target$: number of mutants to generate
t: **global** constant, hyper-parameter for the create_rule function
Hard-coded **global** enum that represents mutation operators op_codes

1: **function** $generate\_mutants$(*pop*, $weights$, max_fails, $target$)
2: $fails \leftarrow 0$
3: $mutants \leftarrow \emptyset$
4: **while** $|mutants| \neq target$ **do**
5: $candidate \leftarrow select\_random(pop)$
6: $operator \leftarrow select\_random\_weighted(op\_codes, op\_weights)$
7: **if** $can\_apply(operator, candidate) = false$ **then**
8: $fails \leftarrow fails + 1$
9: **if** $fails = max\_fails$ **then**
10: **return** $\emptyset$
11: **end if**
12: Goto next iteration of the loop
13: **else**
14: $mutant \leftarrow apply(operator, candidate, t)$
15: $mutants \leftarrow mutants \cup \{mutant\}$
16: **end if**
17: **end while**
18: **return** $mutants$
19: **end function**

Foram implementados três operadores de mutação: adição, remoção e substituição de regras. Os pesos associados a cada operador, isto é, a probabilidade que cada operador tem para ser selecionado, são hiper-parâmetros do MINOTAUR.

O operador de adição utiliza o Algoritmo 3 para criar uma nova regra e a adiciona ao conjunto de regras do indivíduo. Este operador não pode ser aplicado caso o conjunto de regras do indivíduo selecionado cubra todas as instâncias do conjunto de dados, pois nesse caso não há uma $seed$ para o CFSBE.

O operador de remoção seleciona uma regra aleatória do conjunto de regras do indivíduo e a remove. Este operador não pode ser aplicado caso o conjunto de regras do indivíduo selecionado contenha apenas uma regra, pois, de acordo com a modelagem descrita na Seção 3.3.1, todo modelo deve conter pelo menos uma regra.

O operador de substituição remove uma regra aleatória do conjunto de regras do indivíduo, mesmo que essa seja a última regra do indivíduo. Aplica-se, então, o operador de adição de regras a esse indivíduo. O operador de substituição pode ser aplicado a qualquer indivíduo.

No Algoritmo 5, é apresentado o pseudo-código relativo à verificação da possibilidade de aplicação de operador de mutação sobre um indivíduo.

Algorithm 5 Verify if mutation operator can be applied

Require:
op: a mutation operator
$candy$: an individual that is going to be mutated

function $can\_apply(op, candy)$
  if $op = substitute\_rule$ **then**
    return true
  end if
  if $op = remove\_rule$ **then**
    return $|candy.rules| > 1$
  end if
  return $dataset\_coverage(candy.rules) < 100\%$
end function

Finalmente, no Algoritmo 6, é apresentado o pseudo-código relativo ao processo de aplicação de um operador de mutação a um indivíduo específico.

Algorithm 6 Apply mutation operator

Require:
op: a mutation operator
$candy$: an individual that is going to be mutated
Access to **global** constant t, which is used by Algorithm 3
Hard-coded **global** enum that represents mutation operators op_codes

1: **function** $apply(op, candy, t)$
2: **if** $op = remove\_rule$ **then**
3: $r \leftarrow select\_random(candy.rules)$
4: $mutant\_rules \leftarrow candy.rules - \{r\}$
5: **return** $new\_individual(mutant\_rules)$
6: **end if**
7: **if** $op = add\_rule$ **then**
8: $r \leftarrow create\_rule(candy.rules, t)$
9: $mutant\_rules \leftarrow candy.rules \cup \{r\}$
10: **return** $new\_individual(mutant\_rules)$
11: **end if**
12: $r \leftarrow select\_random(candy.rules)$
13: $mutant\_rules \leftarrow candy.rules - \{r\}$
14: $r_2 \leftarrow create\_rule(mutant\_rules, t)$
15: $mutant\_rules_2 \leftarrow mutant\_rules \cup \{r_2\}$
16: **return** $new\_individual(mutant\_rules_2)$
17: **end function**

Optou-se por implementar apenas mutações que modificam o conjunto de regras, em vez de as próprias regras, para que houvesse uma garantia de preservação da consistência dos modelos. As mutações que removem regras de um conjunto diminuem sua cobertura e, portanto, nunca o tornam menos consistente, e as mutações que aumentam ou alteram a cobertura do conjunto de regras invocam o CFSBE que, como explicado anteriormente, garante que a preservação da consistência.

Caso um número suficientemente grande (um hiperparâmetro do MINOTAUR) de mutações falhe em uma única geração (linhas 9 e 10 do Algoritmo 4), o ciclo de mutação-seleção é interrompido; este é o segundo critério de parada. Caso seja possível gerar m novos indivíduos antes que esse critério de parada seja atendido, o contador de mutações falhas é zerado e os novos indivíduos adicionados à população que, então, é submetida ao processo de seleção.

É interessante observar que o segundo critério de parada foi implementado para evitar possíveis *loops* infinitos, mas, durante os experimentos, não foi a causa de parada de nenhuma execução.

O fitness de cada indivíduo da população é uma tupla que contém o Micro Averaged F-Score (TSOUMAKAS et al., 2009), representando o poder preditivo do modelo; e o inverso do número de regras do modelo, representando sua interpretabilidade. Depois que o fitness de

cada indivíduo é calculado, utiliza-se o algoritmo NSGA-II (DEB et al., 2002) para selecionar os indivíduos da próxima geração.

Optou-se por utilizar o inverso do número de regras do modelo para que ambos os objetivos, interpretabilidade e poder preditivo, pudessem ser maximizados, o que simplifica a implementação do algoritmo e a visualização dos fitnesses durante a análise dos resultados. O Micro Averaged F-Score de um modelo é dado pela Equação 3.9, onde A_i denota o conjunto de rótulos associados à instância i do conjunto de dados; P_i denota o conjunto de rótulos preditos pelo modelo para a instância i; e n denota o número de instâncias presentes no conjunto de dados.

$$F\text{-}Score = \frac{1}{n}\sum_{i=1}^{n}\frac{2|A_i \cap P_i|}{|A_i| + |P_i|} \tag{3.9}$$

3.3.4 Pseudo-código e implementação de referência

No Algoritmo 7 é apresentado o pseudo-código do MINOTAUR, e no GitHub[5], há uma implementação de referência. Destaca-se que, frequentemente, a versão mais recentemente disponibilizada no GitHub não é estável. No Capítulo 4, há *links* para repositórios relativos aos experimentos conduzidos, e o repositório relativo à execução do MINOTAUR contém um arquivo de texto com uma descrição da versão do MINOTAUR que foi utilizada e um *link* para download do código-fonte de tal versão.

A implementação de referência do MINOTAUR foi escrita na linguagem C#. A escolha por essa linguagem se deu pelos seguintes motivos: I) excelência da coleção de ferramentas de desenvolvimento (*tooling*); o *debugger* do Visual Studio, por exemplo, permite uma depuração fácil de aplicações com múltiplas *threads*; II) possibilidade de geração de executáveis para múltiplos sistemas operacionais; e III) a linguagem e o *framework* oferecem um bom compromisso entre desempenho e facilidade de codificação.

A implementação disponibilizada é relativamente bem otimizada[6]. Por exemplo, como os indivíduos são imutáveis (lembrando que a mutação opera sobre uma cópia do indivíduo), seus fitnesses também são. Isso permitiu o armazenamento (*caching*) dos fitnesses. Logo, caso um indivíduo sobreviva múltiplas gerações não será necessário submetê-lo mais que uma vez ao processo de cálculo de fitness.

Além disso, os processos de geração de mutantes e cálculo de fitnesses foram paralelizados. Em um cenário ideal, cada núcleo do processador pode gerar um mutante e/ou calcular o fitness de um indivíduo.

[5] <https://github.com/Mirandatz/minotaur>

[6] Parte do processo de otimização foi realizado com auxílio da ferramenta NProfiler que permitiu a identificação de gargalos nos processos de sincronização. Os desenvolvedores da ferramenta gentilmente forneceram uma licença de estudante para a realização da pesquisa.

Algorithm 7 MINOTAUR

Require:

max_gen: maximum number of generations, evolutionary stopping criteria
pop_size: population size
$target\_mut$ number of mutants to create each generation
max_fails: maximum number of mutation attempts that can fail in a single generation, evolutionary stopping criteria
t: **global** constant, hyper-parameter for the create_rule function
$weights$: weights associated with each mutation operator

function $minotaur(max\_gen, pop\_size, t, target\_mut, max\_fails, weights)$
  $pop \leftarrow \emptyset$
  for $counter \in range(1, pop\_size)$ **do**
    $rule \leftarrow create\_rule(\emptyset)$
    $individual \leftarrow new\_individual(\{rule\})$
    $pop \leftarrow pop \cup \{individual\}$
  end for

  for $gen\_nr \in range(1, max\_gen)$ **do**
    $mutants \leftarrow generate\_mutants(pop, max\_fails, weights, target\_mut)$
    if mutants = $\emptyset$ **then** ▷ Maximum failed mutation attempts per generation reached
      return pop
    else
      $pop \leftarrow pop \cup mutants$
    end if
    $fitnesses \leftarrow compute\_fitnesses(pop)$
    $pop \leftarrow nsga\_ii(pop, fitnesses)$
  end for
  return pop
end function

A partir da conjunção desses fatores foi possível conduzir[7] a bateria de experimentos do MINOTAUR (descrita no Capítulo 4) em 2 horas e 31 minutos, incluindo o tempo necessário para ler, *parse* e verificar a integridade dos conjuntos de dados, e escrever os resultados.

Nessa bateria, foram gerados 40 mutantes a cada geração, e cada execução dura 200 gerações. Como o MINOTAUR foi executado 30 vezes em cada um dos 10 *folds* de cada um dos 8 conjuntos de dados, foram gerados aproximadamente 1.9×10^7 modelos.

[7] Os experimentos foram conduzidos em um computador com dois processadores Intel Xeon E5-2640V4 (2.4 GHz) e 96 GBs de memória RAM (2400 MHz).

Capítulo 4

VALIDAÇÃO EXPERIMENTAL

Neste capítulo são descritos os experimentos conduzidos com o MINOTAUR (utilizando o CFSBE como mecanismo de orientação de busca) e com algoritmos da literatura. Também é apresentada a análise dos resultados de tais experimentos bem como os recursos utilizados para conduzi-los.

4.1 Descrição e reprodutibilidade

Para mensurar a qualidade dos modelos gerados pelo MINOTAUR foram conduzidos experimentos com oito conjuntos de dados (*datasets*). Quatro dos conjuntos são de problemas reais e estão disponíveis publicamente[1]: *CAL500* (TURNBULL et al., 2008), *emotions* (TROHIDIS et al., 2008), *scene* (BOUTELL et al., 2004) e *yeast* (UEDA; SAITO, 2003). Os outros quatro conjuntos foram gerados com os geradores multirrótulo discutidos por Read et al. (2012): *synthetic0* (*hyper-plane*), *synthetic1* (*radial basis function*), *synthetic2* (*random tree*) e *synthetic3* (*wave-form*). A Tabela 3 contém o número de *features*, classes e instâncias de cada conjunto de dados.

Tabela 3 – Características dos datasets

Nome	#Features	#Classes	#Instâncias
CAL500	68	174	502
emotions	72	6	593
scene	294	6	2407
synthetic0	10	5	10000
synthetic1	80	22	10000
synthetic2	30	8	10000
synthetic3	21	7	10000
yeast	103	14	2417

[1] <http://mulan.sourceforge.net/datasets-mlc.html>

Os conjuntos de dados foram particionados utilizando a estratégia *10-fold cross-validation*. As partições geradas e os *scripts* utilizados para particionar e pré-processar os conjuntos de dados[2] estão disponíveis no GitHub[3].

É importante lembrar que o MINOTAUR gera uma coleção de modelos. Para comparar o MINOTAUR com algoritmos da literatura, que geram apenas um modelo, escolheu-se o modelo com maior F-Score (no conjunto de treino) da população como "representante" do MINOTAUR, tanto em termos de poder preditivo quanto de interpretabilidade. Os scripts utilizados para executar o MINOTAUR e analisar seus resultados também estão hospedados no GitHub[4].

Os algoritmos da literatura utilizados para comparação foram: J48, Support Vector Machines (SVM), Naive Bayes (NB) e k-Nearest Neighbors (kNN). Para utilizar tais algoritmos em um contexto multirrótulo, foram empregadas as técnicas de transformação de problema: Classifier Chains (CC), Binary Relevance (BR) e Label Powerset (LP). Também foram utilizados algoritmos do estado da arte em geração de modelos baseados em regras para classificação multirrótulo: HMC-GA (CERRI et al., 2019) e Clus-HMC (VENS et al., 2008).

Os algoritmos J48, SVM, NB e kNN foram executados com as bibliotecas Weka (HALL et al., 2009), Mulan (TSOUMAKAS et al., 2009) e ExecuteMulan (MOYANO et al., 2018), enquanto os algoritmos HMC-GA e Clus-HMC foram executados através de suas implementações de referência. Como o MINOTAUR e o HMC-GA são indeterminísticos, eles foram executados 30 vezes por *fold*, e os desvios-padrões de seus F-Scores foram registrados para verificar se os algoritmos são estáveis.

Todos os algoritmos foram executados com os valores padrões para seus hiperparâmetros, exceto o MINOTAUR e o HMC-GA. O primeiro, porque não possui valores padrões, e o segundo por possuir apenas valores recomendados.

Os valores utilizados para os hiperparâmetros do HMC-GA, como recomendados por Cerri et al. (2019), foram: tamanho da população inicial 50, número de elitismo 1, taxa de mutação 40%, taxa de reprodução (*crossover*) 90%, tamanho do torneio 2, número máximo de gerações 50, número máximo de instâncias cobertas por regra 300, número mínimo de instâncias cobertas por regra 5, e número máximo de instâncias não cobertas por nenhuma regra 10. O HMC-GA também possui um hiperparâmetro p que determina a probabilidade de utilizar um teste em uma regra; o valor utilizado foi $p = 0.3$. Durante a fase de inicialização o tamanho médio das regras, n, é dado em função do valor de p e do número de *features* no conjunto de dados $|f|$: $n = p \times |f|$. Os valores de n são apresentados na Tabela 4.

Os valores utilizados para os hiperparâmetros do MINOTAUR foram escolhidos arbitrariamente: tamanho da população 80, número máximo de gerações 200, número máximo de mutações falhas por geração 2000, peso do operador de adição de regra 1, peso do ope-

[2] Basicamente conversões de formato, como *.arff* para *.csv*.
[3] <https://github.com/Mirandatz/arxiv.minotaur.datasets>
[4] <https://github.com/Mirandatz/arxiv.minotaur.experiments>

rador de substituição de regra 4, peso do operador de remoção de regra 2, e número de mutantes gerados por geração 40. O hiperparâmetro t, relativo ao método B de criação de regras, foi ajustado para cada conjunto de dados. Experimentos foram conduzidos com $t \in \{2, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096\}$, e os valores que resultaram em indivíduos com maior F-Score (no conjunto de treino) foram utilizados; tais valores são apresentados na Tabela 4.

Tabela 4 – Valores dos t e n

Dataset	MINOTAUR (t)	HMC-GA (n)
CAL500	64	20.4
emotions	128	21.6
scene	512	88.2
synthetic0	4096	3
synthetic1	32	24
synthetic2	2048	9
synthetic3	1024	6.3
yeast	512	30.9

4.2 Resultados e análise

Os F-Scores dos modelos gerados são apresentados, por uma questão de formatação, em duas tabelas: os algoritmos baseados em adaptação de algoritmo, na Tabela 5, e os algoritmos baseados em transformação de problema, na Tabela 6. Os melhores resultados, considerando ambas as tabelas, são destacados em negrito.

Tabela 5 – F-Scores dos algoritmos baseados em adaptação de algoritmo

Dataset	MINOTAUR	HMC-GA	Clus-HMC
CAL500	0.31 ± 0.00	0.34 ± 0.01	0.31
emotions	0.36 ± 0.02	0.49 ± 0.03	0.59
scene	0.31 ± 0.01	0.45 ± 0.03	0.60
synthetic0	0.59 ± 0.00	0.48 ± 0.01	0.52
synthetic1	0.02 ± 0.00	0.24 ± 0.01	0.12
synthetic2	0.47 ± 0.00	0.38 ± 0.01	0.34
synthetic3	0.64 ± 0.00	0.61 ± 0.01	0.66
yeast	0.55 ± 0.00	0.56 ± 0.01	0.56

Tabela 6 – F-Scores dos algoritmos baseados em adaptação de problema

Dataset	Classifier Chains				Binary Relevance				Label Powerset			
	J48	SVM	NB	kNN	J48	SVM	NB	kNN	J48	SVM	NB	kNN
CAL500	**0.36**	0.34	0.29	0.34	0.35	0.33	0.33	0.34	0.33	0.34	0.34	0.34
emotions	0.60	0.68	0.66	0.62	0.60	0.65	0.66	0.62	0.59	**0.70**	0.62	0.62
scene	0.63	0.70	0.56	0.69	0.62	0.68	0.56	0.69	0.59	**0.74**	0.64	0.69
synthetic0	0.58	0.60	0.60	0.49	0.56	0.56	0.58	0.49	0.49	**0.61**	**0.61**	0.49
synthetic1	0.39	0.15	0.08	**0.65**	0.47	0.03	0.03	**0.65**	0.26	0.24	0.13	**0.65**
synthetic2	0.50	0.48	**0.51**	0.41	0.37	0.41	0.39	0.41	0.44	0.50	0.50	0.41
synthetic3	0.67	0.67	0.67	0.62	0.68	0.65	0.65	0.62	0.65	**0.71**	0.69	0.62
yeast	0.55	**0.64**	0.54	0.60	0.58	0.63	0.55	0.60	0.54	**0.64**	0.60	0.60

Os desvios-padrões próximos de zero sugerem que o MINOTAUR e o HMC-GA são estáveis e, portanto, podem ser comparados com os algoritmos da literatura. Para comparar múltiplos algoritmos testados em múltiplos conjuntos de dados seguiu-se o procedimento sugerido por Demšar (2006): aplicar o *Friedman test*, com a estatística derivada proposta por Iman e Davenport (1980), para identificar se há uma diferença estatisticamente significante entre os algoritmos e, em caso positivo, proceder com *post-hoc Nemenyi test* para identificar quais algoritmos são superiores a quais outros. As etapas seguidas foram:

1. Definição da hipótese nula (H_0) e hipótese alternativa (H_1):
   - H_0 := não há diferença estatisticamente significante entre os algoritmos.
   - H_1 := existe pelo menos um algoritmo que é significativamente superior ou inferior aos demais.
2. Definição do nível de significância, $\alpha = 0.05$.
3. Cálculo das médias dos *ranks* dos algoritmos, apresentadas na Tabela 7.
4. Cálculo da estatística de *Friedman*, dada pela Equação 4.1, em que r_i denota a média dos ranks do i-ésimo algoritmo, k denota o número de algoritmos e N denota o número de conjuntos de dados. Obteve-se $F = 39.3549$.
5. Cálculo da estatística derivada, dada pela Equação 4.2, em que F denota a estatística de *Friedman* e N denota o número de conjuntos de dados. Obteve-se $F_2 = 3.7921$.
6. Comparação do valor de F_2 com valor crítico da distribuição da estatística derivada (distribuição F), com $df_1 = k - 1 = 14$ e $df_2 = (k-1) \cdot (N-1) = 98$. O valor crítico, computado com o biblioteca SciPy (VIRTANEN et al., 2020), foi $F_c = 1.7939$.

Tabela 7 – Médias dos *ranks* (F-Score)

Algoritmo	Rank
LP SVM	**2.81**
CC SVM	4.25
LP NB	5.31
CC J48	6.38
BR J48	7.38
LP kNN	7.50
CC kNN	7.50
BR kNN	7.50
BR SVM	8.19
CC NB	8.38
BR NB	10.00
LP J48	10.44
Clus-HMC	11.06
MINOTAUR	11.56
HMC-GA	11.75

$$F = \frac{12N}{k(k+1)} \left[\sum_i r_i^2 - \frac{k(k+1)^2}{4} \right] \tag{4.1}$$

$$F_2 = \frac{(N-1)F}{N(k-1) - F} \tag{4.2}$$

Como o valor obtido $F_2 = 3.7921$ foi maior que o valor crítico $F_c = 1.7939$, rejeitou-se a hipótese nula[5]. Para proceder com o *post hoc Nemenyi*, calculou-se a diferença crítica CD, dada pela Equação 4.3, em que k denota o número de algoritmos, N denota o número de conjuntos de dados e q_α é um valor tabelado em função de $\alpha = 0.05$ e $k = 15$. Obteve-se o valor $CD = 7.582$.

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{4.3}$$

O *post-hoc Nemenyi* compara os *ranks* dos algoritmos, par a par, e verifica se a diferença entre eles é maior que a diferença crítica CD. Caso positivo, a diferença entre os desempenhos dos algoritmos é estatisticamente significante. Na Figura 9, é apresentado o diagrama crítico comparando os algoritmos em função de seus F-Scores; algoritmos conectados com uma linha não apresentam diferenças estatisticamente significantes.

Como não há uma linha conectando o LP-SVM ao MINOTAUR, ao HMC-GA e ao Clus-HMC, concluiu-se que o o LP-SVM é superior. Não houve outras diferenças estatisticamente significantes. Esse resultado é interessante, tendo em vista que o LP-SVM é focado

[5] O *p-value* do teste de Friedman foi $p = 0.0003289$.

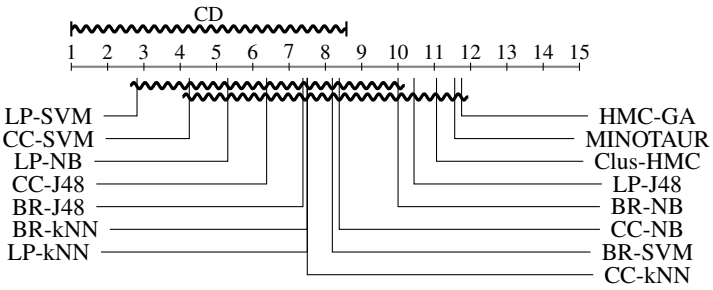


Figura 9 – Diagrama crítico (F-Score)

exclusivamente em poder preditivo, enquanto o MINOTAUR gerou modelos interpretáveis e com poder preditivo comparável aos demais algoritmos.

O MINOTAUR também foi comparado em termos de interpretabilidade, mensurada em função do número de regras, com os outros algoritmos de classificação multirrótulo baseados em regras: o HMC-GA e o Clus-HMC. Os resultados, apresentados na Tabela 8, são as médias computadas nos *folds*. Os melhores resultados são destacados em negrito.

Tabela 8 – Médias dos números de regras dos modelos

Dataset	MINOTAUR	HMC-GA	Clus-HMC
CAL500	27.94 $\pm$ 1.43	9.65 $\pm$ 1.18	**0.00**
emotions	11.03 $\pm$ 1.63	**8.86** $\pm$ 0.96	21.30
scene	**10.41** $\pm$ 0.98	17.11 $\pm$ 1.33	67.90
synthetic0	**9.26** $\pm$ 1.17	37.31 $\pm$ 4.19	13.70
synthetic1	**12.36** $\pm$ 0.75	399.42 $\pm$ 8.22	103.50
synthetic2	**11.25** $\pm$ 1.96	145.76 $\pm$ 3.54	40.90
synthetic3	**13.38** $\pm$ 1.54	66.54 $\pm$ 2.82	32.90
yeast	**15.93** $\pm$ 2.07	26.74 $\pm$ 2.32	97.90
rank	**1.38**	2.38	2.25

Como somente três modelos são comparados em termos de interpretabilidade, utilizou-se o teste par a par de Wilcoxon (WILCOXON, 1945), com $\alpha = 0.1$, para verificar se as diferenças são estatisticamente significantes. Os resultados dos valores de p são apresentados na Tabela 9, em que os valores sublinhados indicam a rejeição da hipótese nula, isto é, a superioridade do algoritmo da linha em relação ao algoritmo da coluna. Conclui-se, portanto, que o MINOTAUR é, em termos de interpretabilidade dos modelos gerados, superior ao HMC-GA e ao Clus-HMC.

Tabela 9 – Valores p da rejeição da hipótese nula

	MINOTAUR	HMC-GA	Clus-HMC
MINOTAUR	–	<u>0.0781</u>	<u>0.0547</u>
HMC-GA	0.0781	–	0.5469
Clus-HMC	0.0547	0.5469	–

Destaca-se que o modelo gerado pelo Clus-HMC para o conjunto de dados *CAL500* não possui regras[6], isto é, o modelo gerado sempre prediz o vetor médio de classes. O resultado chama a atenção pois o modelo, mesmo vazio, possui um F-Score comparável aos F-Scores dos outros algoritmos.

Esse fenômeno naturalmente leva à questão: quanto do poder preditivo dos modelos gerados deve-se à não-cobertura de uma região do *feature space* por nenhuma regra? Ou, em outras palavras, com que frequência instâncias não são cobertas por nenhuma regra e, portanto, são associadas ao vetor médio de classes? Como o MINOTAUR foi implementado pelo próprio autor, foi possível modificá-lo, sem dificuldades particulares, de forma que tais frequências fossem armazenadas. Os resultados, apresentados na Tabela 10, são, novamente, relativos ao "modelo representante" do MINOTAUR, ou seja, aquele com maior F-Score no conjunto de treino.

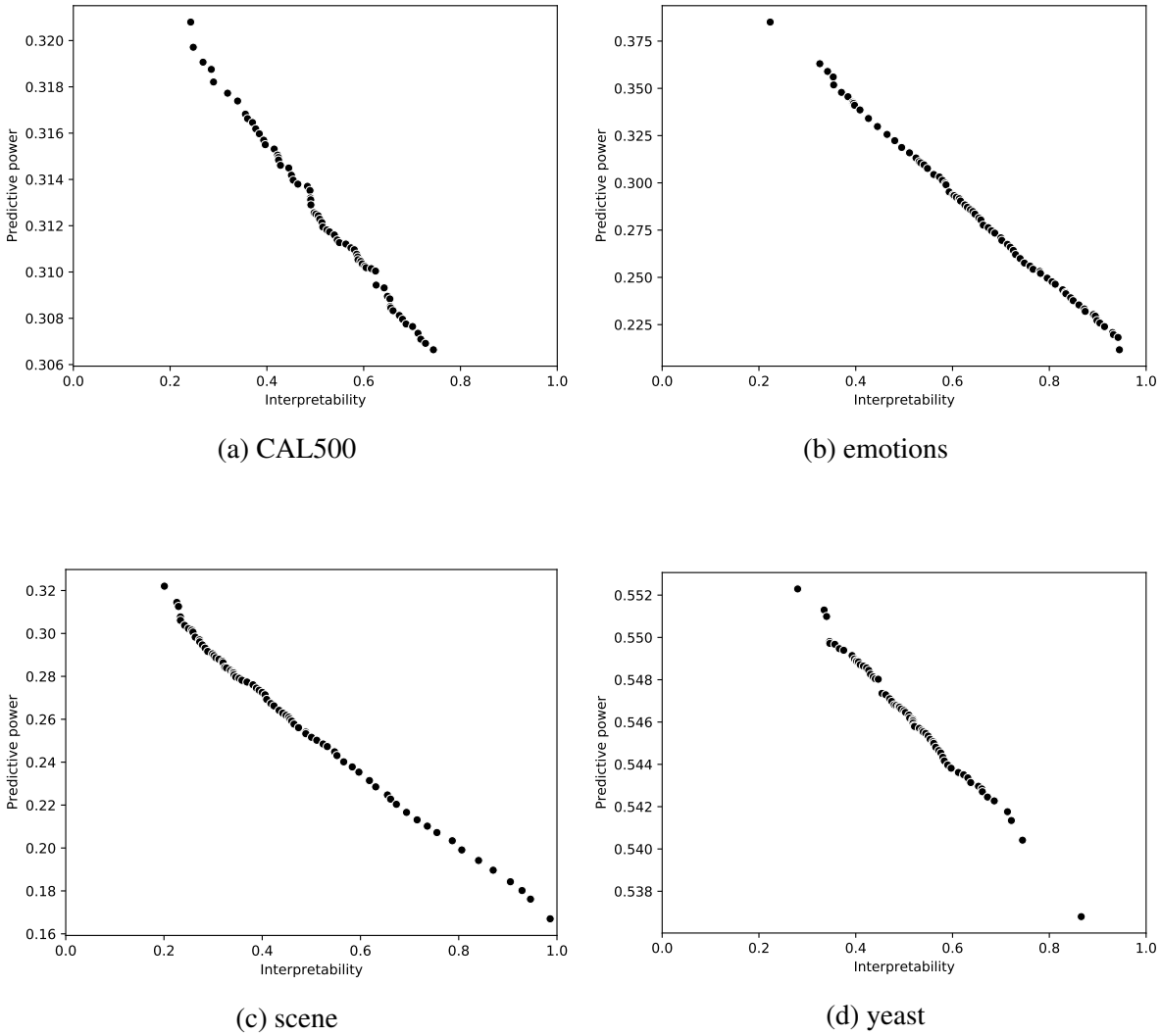
Tabela 10 – Fração das instâncias que não são cobertas por nenhuma regra, sendo que o valor 0 indica nenhuma instância e 1 indica todas as instâncias

Dataset	Partições de treino	Partições de teste
CAL500	0.56 ± 0.08	0.85 ± 0.06
emotions	0.25 ± 0.12	0.61 ± 0.11
scene	0.36 ± 0.06	0.57 ± 0.06
synthetic0	0.13 ± 0.09	0.13 ± 0.09
synthetic1	0.96 ± 0.01	0.97 ± 0.01
synthetic2	0.03 ± 0.00	0.03 ± 0.01
synthetic3	0.18 ± 0.04	0.20 ± 0.04
yeast	0.35 ± 0.04	0.48 ± 0.05

Por limitações de tempo não foram conduzidas análises para identificar possíveis correlações entre os F-Scores, tamanhos dos modelos e cobertura das instâncias dos conjuntos de treino e teste. Porém, é interessante observar que a discrepância entre as taxas de cobertura das instâncias de treino e de teste foram consistentemente menores nos conjuntos de dados sintéticos. Além disso, as coberturas nas partições de teste foram frequentemente inferiores às coberturas nas partições de treino, um sintoma de *overfitting*. Novamente, sem uma análise estatística, não foi confirmado se tais discrepâncias são significantes.

Também é interessante observar que no conjunto de dados *synthetic1*, em que o MINOTAUR teve um desempenho particularmente ruim, quase nenhuma instância do conjunto de dados é coberta pelas regras evoluídas. Em trabalhos futuros, talvez seja interessante investigar o que ocorre nesse conjunto de dados que dificulta a criação de regras com coberturas apropriadas. O desempenho bom do kNN indica que uma abordagem geométrica, como atualizada atualmente, não seria inapropriada.

[6] Isso ocorre em função da estratégia de poda empregada pelo algoritmo.



(a) CAL500

(b) emotions

(c) scene

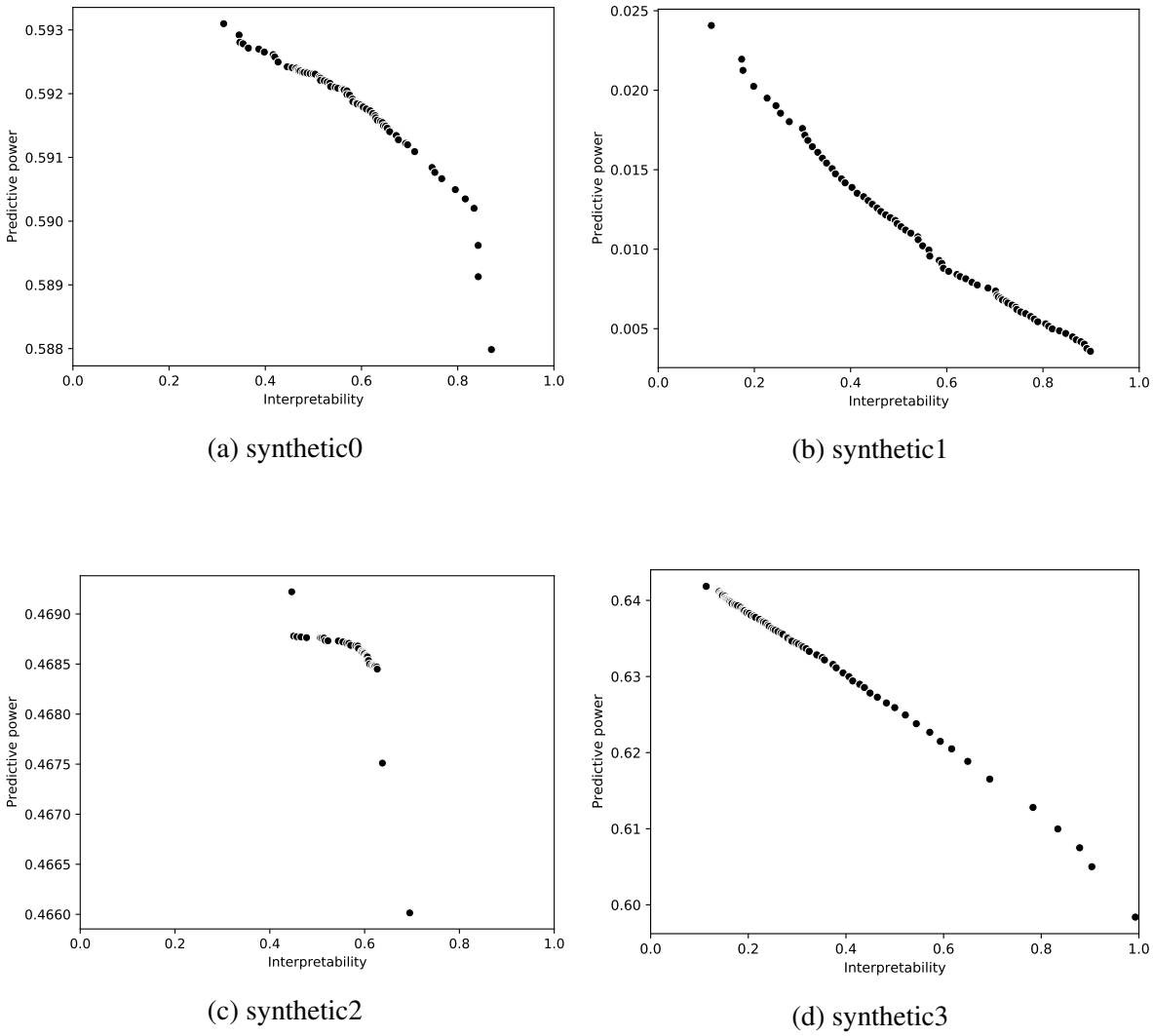
(d) yeast

Figura 10 – Compromissos entre poder preditivo e interpretabilidade dos modelos em conjuntos de dados reais

A cobertura das regras foi, em quase todos conjuntos de dados, relativamente baixa. Esse problema, identificado apenas durante a análise dos experimentos, destaca-se, juntamente com a impossibilidade de processar conjuntos de dados com *features* categóricas, como uma das principais limitações do algoritmo.

Finalmente, para visualizar os compromissos entre poder preditivo e interpretabilidade dos modelos de uma população gerada pelo MINOTAUR, foram sintetizados gráficos de dispersão. Na Figura 10, são apresentados os gráficos relativos aos conjuntos de dados de problemas reais, enquanto, na Figura 11, são apresentados os gráficos relativos aos conjuntos de dados sintéticos. O eixo vertical de cada gráfico representa o F-Score e o eixo horizontal, o inverso do número de regras do modelo.

Para sintetizar as Figuras 10 e 11, realizou-se o seguinte procedimento: para cada um dos 8 conjunto de dados, para cada um dos 10 *folds*, para cada uma das 30 execuções, ordenou-se



(a) synthetic0

(b) synthetic1

(c) synthetic2

(d) synthetic3

Figura 11 – Compromissos entre poder preditivo e interpretabilidade dos modelos em conjuntos de dados sintéticos

lexicograficamente os fitnesses[7] dos indivíduos e armazenaram-se tais fitnesses em uma matriz. Como a população tinha 80 indivíduos e o fitness era um par, a matriz gerada possuía 80 linhas e 2 colunas. Foram geradas, portanto, para cada conjunto de dados, 300 matrizes (30 execuções para cada um dos 10 folds). Para cada conjunto de dados calculou-se a soma de suas 300 matrizes, e dividiu-se o resultado pelo número de matrizes, isto é, calculou-se a "matriz média". Cada linha dessa "matriz média" foi desenhada como um ponto no gráfico de dispersão.

Novamente, por limitações de tempo, não foram conduzidas análises estatísticas comparando os indivíduos de uma população, porém, as "curvas" apresentadas nas figuras, que lembram frontes de Pareto, sugerem que as populações geradas pelo MINOTAUR contêm indivíduos com diferentes compromissos entre poder preditivo e interpretabilidade.

[7] O fitness de um indivíduo era um par com o F-Score e o inverso do número de regras deste indivíduo.

Capítulo 5
CONSIDERAÇÕES FINAIS

5.1 Síntese do trabalho

Neste trabalho, foram estudados três tópicos: I) algoritmos de classificação multirrótulo; II) técnicas de otimização multiobjetivo, especificamente algoritmos genéticos; e III) questões de interpretabilidade de modelos de classificação.

O algoritmo evolutivo proposto, MINOTAUR, gera coleções de modelos de classificação multirrótulo baseados em conjuntos consistentes de regras. Para gerar tais coleções, a abordagem adotada foi baseada na restrição/orientação do processo de geração de regras. Para realizar tal orientação, o algoritmo CFSGS foi elaborado, porém, se mostrou proibitivamente custoso. Em seguida, o algoritmo CFSBE foi elaborado e implementado.

Os modelos gerados pelo MINOTAUR, utilizando o CFSBE como mecanismo de orientação do processo de criação de regras, foram, em termos de poder preditivo, comparáveis à maioria dos modelos gerados por algoritmos da literatura e, em termos de interpretabilidade, mensurada em função do número de regras no modelo, foram superiores. A estratégia de seleção de indivíduos mais bem adaptados, baseada em otimização multiobjetivo (NSGA-II), se mostrou eficaz; os modelos gerados oferecem diferentes compromissos entre poder preditivo e interpretabilidade.

5.2 Trabalhos futuros

Os algoritmos de orientação de busca, CFSGS e CFSBE, foram elaborados assumindo que o conjunto de dados não possui *features* categóricas. Porém, em problemas do mundo real, essa suposição frequentemente não é aplicável.

Modificar os algoritmos para que suportem tais *features* requer mudanças nos formatos dos *feature tests* e das regras. Uma possibilidade seria definir um *feature test* como uma união discriminada de um *feature test categórico* e um *feature test contínuo*. O *feature test contínuo* possuiria o mesmo formato do *feature test* atual, ou seja, seria uma função que verifica se um

valor pertence a um intervalo, com um limite inferior inclusivo e um limite superior exclusivo.

Se o *feature test categórico* for modelado como uma simples comparação de igualdade, por exemplo: $test_{cor} := f_i = yellow$, então sua negação, utilizada no algoritmo CFSGS, seria dada por uma desigualdade: $\neg test_{cor} := f_i \neq yellow$.

O antecedente das regras passaria a ser uma coleção de uniões discriminadas de *feature tests* e, portanto, o número de *feature tests* gerado por regra, utilizado pelo Algoritmo 1, seria igual ao número de *features* categóricas mais duas vezes o número de *features* contínuas.

Adaptar o CFSBE seria relativamente mais complexo. Inicialmente, seria necessário modificar a estrutura de dados utilizada para representar hiper-retângulos. A nova estrutura seria uma lista de uniões discriminadas de: I) pares de valores reais, utilizados para *features* contínuas; e II) conjuntos de valores categóricos, utilizados para *features* categóricas.

Depois de substituir a estrutura de dados, a função $intersects$, do Algoritmo 2, também precisaria ser modificada para lidar com *features* (dimensões) categóricas. Dentro do *loop* da linha 22, seria necessário identificar qual tipo de *feature* a dimensão i representa. Caso seja uma *feature* contínua, seria mantida a lógica atual (linhas 23 até 27). Caso seja uma *feature* categórica, seria necessário verificar se o valor do *feature test* i, da regra r, está contido no conjunto de valores relativos à dimensão i do hiper-retângulo B.

Ao considerar as limitações dos algoritmos propostos, a qualidade dos modelos gerados e as análises realizadas, sugerem-se como trabalhos futuros:

- modificar o MINOTAUR para que suporte conjuntos de dados com *features* categóricas;
- estudar as possíveis relações entre os valores dos hiper-parâmetros do MINOTAUR com os desempenhos e coberturas dos modelos gerados;
- elaborar mecanismos e/ou heurísticas para seleção de valores para o hiperparâmetro t do método de criação de regras B;
- elaborar novos mecanismos para criação de regras a partir de hiper-retângulos, além dos métodos A e B;
- implementar outros mecanismos de seleção além do NSGA-II;
- incorporar no próprio MINOTAUR, após o término do ciclo evolutivo, uma etapa de pós-processamento para simplificação das regras, por exemplo, removendo *feature tests* tautológicos.

REFERÊNCIAS

BOUTELL, M. R.; LUO, J.; SHEN, X.; BROWN, C. M. Learning multi-label scene classification. *Pattern recognition*, Elsevier, v. 37, n. 9, p. 1757–1771, 2004. Citado 3 vezes nas páginas 11, 12 e 43.

CARUANA, R.; KANGARLOO, H.; DIONISIO, J.; SINHA, U.; JOHNSON, D. Case-based explanation of non-case-based learning methods. In: AMERICAN MEDICAL INFORMATICS ASSOCIATION. *Proceedings of the AMIA Symposium*. [S.l.], 1999. p. 212. Citado na página 16.

CERRI, R.; BARROS, R. C.; CARVALHO, A. C. P. L. F. de. A genetic algorithm for hierarchical multi-label classification. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. New York, NY, USA: Association for Computing Machinery, 2012. (SAC '12), p. 250–255. ISBN 9781450308571. Citado na página 15.

CERRI, R.; BARROS, R. C.; CARVALHO, A. C. P. L. F. de; JIN, Y. Reduction strategies for hierarchical multi-label classification in protein function prediction. *BMC Bioinformatics*, v. 17, n. 1, p. 373, Sep 2016. ISSN 1471-2105. Disponível em: <https://doi.org/10.1186/s12859-016-1232-1>. Citado 2 vezes nas páginas 12 e 15.

CERRI, R.; BASGALUPP, M. P.; BARROS, R. C.; CARVALHO, A. C. de. Inducing hierarchical multi-label classification rules with genetic algorithms. *Applied Soft Computing*, Elsevier, v. 77, p. 584–604, 2019. Citado 3 vezes nas páginas 15, 35 e 44.

CHAN, A.; FREITAS, A. A. A new ant colony algorithm for multi-label classification with applications in bioinfomatics. In: ACM. *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. [S.l.], 2006. p. 27–34. Citado na página 15.

CLARE, A.; KING, R. D. Knowledge discovery in multi-label phenotype data. In: SPRINGER. *European Conference on Principles of Data Mining and Knowledge Discovery*. [S.l.], 2001. p. 42–53. Citado 4 vezes nas páginas 8, 9, 14 e 17.

CORNE, D. W.; KNOWLES, J. D.; OATES, M. J. The pareto envelope-based selection algorithm for multiobjective optimization. In: SPRINGER. *International conference on parallel problem solving from nature*. [S.l.], 2000. p. 839–848. Citado na página 21.

DEB, K.; PRATAP, A.; AGARWAL, S.; MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, IEEE, v. 6, n. 2, p. 182–197, 2002. Citado 2 vezes nas páginas 20 e 41.

DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, v. 7, n. Jan, p. 1–30, 2006. Citado na página 46.

EIBEN, A. E.; SMITH, J. E. et al. *Introduction to evolutionary computing*. [S.l.]: Springer, 2003. v. 53. Citado na página 19.

FREITAS, A. A. A critical review of multi-objective optimization in data mining: a position paper. *ACM SIGKDD Explorations Newsletter*, ACM, v. 6, n. 2, p. 77–86, 2004. Citado 2 vezes nas páginas 9 e 18.

FREITAS, A. A. *Data mining and knowledge discovery with evolutionary algorithms*. [S.l.]: Springer Science & Business Media, 2013. Citado na página 8.

FREITAS, A. A. Comprehensible classification models: a position paper. *ACM SIGKDD explorations newsletter*, ACM, v. 15, n. 1, p. 1–10, 2014. Citado 5 vezes nas páginas 8, 9, 16, 17 e 18.

GONÇALVES, E. C.; FREITAS, A. A.; PLASTINO, A. A survey of genetic algorithms for multi-label classification. In: IEEE. *2018 IEEE Congress on Evolutionary Computation (CEC)*. [S.l.], 2018. p. 1–8. Citado na página 15.

HALL, M.; FRANK, E.; HOLMES, G.; PFAHRINGER, B.; REUTEMANN, P.; WITTEN, I. H. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, ACM New York, NY, USA, v. 11, n. 1, p. 10–18, 2009. Citado na página 44.

IMAN, R. L.; DAVENPORT, J. M. Approximations of the critical region of the fbietkan statistic. *Communications in Statistics-Theory and Methods*, Taylor & Francis, v. 9, n. 6, p. 571–595, 1980. Citado na página 46.

KATAKIS, I.; TSOUMAKAS, G.; VLAHAVAS, I. Multilabel text classification for automated tag suggestion. In: *Proceedings of the ECML/PKDD*. [S.l.: s.n.], 2008. v. 18. Citado na página 12.

KONAK, A.; COIT, D. W.; SMITH, A. E. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, Elsevier, v. 91, n. 9, p. 992–1007, 2006. Citado 2 vezes nas páginas 9 e 19.

LIPTON, Z. C. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016. Citado na página 16.

MAATEN, L. v. d.; HINTON, G. Visualizing data using t-sne. *Journal of machine learning research*, v. 9, n. Nov, p. 2579–2605, 2008. Citado na página 16.

MARLER, R. T.; ARORA, J. S. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, Springer, v. 26, n. 6, p. 369–395, 2004. Citado na página 18.

MORSE, J. N. Reducing the size of the nondominated set: Pruning by clustering. Computers and Operations Research, 1980. Citado na página 21.

MOYANO, J. M.; GIBAJA, E. L.; CIOS, K. J.; VENTURA, S. Review of ensembles of multi-label classifiers: models, experimental study and prospects. *Information Fusion*, Elsevier, v. 44, p. 33–45, 2018. Citado na página 44.

OTERO, F. E.; FREITAS, A. A. Improving the interpretability of classification rules discovered by an ant colony algorithm. In: ACM. *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. [S.l.], 2013. p. 73–80. Citado 4 vezes nas páginas 8, 15, 17 e 24.

OTERO, F. E.; FREITAS, A. A.; JOHNSON, C. G. cant-miner: an ant colony classification algorithm to cope with continuous attributes. In: SPRINGER. *International Conference on Ant Colony Optimization and Swarm Intelligence*. [S.l.], 2008. p. 48–59. Citado na página 15.

OTERO, F. E.; FREITAS, A. A.; JOHNSON, C. G. A hierarchical classification ant colony algorithm for predicting gene ontology terms. In: SPRINGER. *European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*. [S.l.], 2009. p. 68–79. Citado na página 15.

OTERO, F. E.; FREITAS, A. A.; JOHNSON, C. G. A hierarchical multi-label classification ant colony algorithm for protein function prediction. *Memetic Computing*, Springer, v. 2, n. 3, p. 165–181, 2010. Citado 3 vezes nas páginas 8, 9 e 15.

OTERO, F. E.; FREITAS, A. A.; JOHNSON, C. G. A new sequential covering strategy for inducing classification rules with ant colony algorithms. *IEEE Trans. Evolutionary Computation*, v. 17, n. 1, p. 64–76, 2013. Citado na página 15.

PARPINELLI, R. S.; LOPES, H. S.; FREITAS, A. A. Data mining with an ant colony optimization algorithm. *IEEE transactions on evolutionary computation*, IEEE, v. 6, n. 4, p. 321–332, 2002. Citado na página 15.

PEÑA-CASTILLO, L.; TASAN, M.; MYERS, C. L.; LEE, H.; JOSHI, T.; ZHANG, C.; GUAN, Y.; LEONE, M.; PAGNANI, A.; KIM, W. K. et al. A critical assessment of mus musculus gene function prediction using integrated genomic evidence. *Genome biology*, BioMed Central, v. 9, n. 1, p. S2, 2008. Citado na página 12.

QUINLAN, J. R. Induction of decision trees. *Machine learning*, Springer, v. 1, n. 1, p. 81–106, 1986. Citado na página 14.

READ, J.; BIFET, A.; HOLMES, G.; PFAHRINGER, B. Scalable and efficient multi-label classification for evolving data streams. *Mach Learn*, Springer, v. 88, n. 1-2, p. 243–272, 2012. Citado na página 43.

READ, J.; PFAHRINGER, B.; HOLMES, G.; FRANK, E. Classifier chains for multi-label classification. *Machine learning*, Springer, v. 85, n. 3, p. 333, 2011. Citado na página 12.

SILLA, C. N.; FREITAS, A. A. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, Springer, v. 22, n. 1-2, p. 31–72, 2011. Citado na página 14.

SMALDON, J.; FREITAS, A. A. Improving the interpretability of classification rules in sparse bioinformatics datasets. In: *Research and Development in Intelligent Systems XXIII*. [S.l.]: Springer, 2007. p. 377–381. Citado na página 17.

TAN, P.-N. *Introduction to data mining*. [S.l.]: Pearson Education India, 2018. Citado na página 11.

TROHIDIS, K.; TSOUMAKAS, G.; KALLIRIS, G.; VLAHAVAS, I. P. Multi-label classification of music into emotions. In: *ISMIR*. [S.l.: s.n.], 2008. v. 8, p. 325–330. Citado 2 vezes nas páginas 12 e 43.

TSOUMAKAS, G.; KATAKIS, I. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, IGI Global, v. 3, n. 3, p. 1–13, 2007. Citado 4 vezes nas páginas 8, 11, 12 e 14.

TSOUMAKAS, G.; KATAKIS, I.; VLAHAVAS, I. Effective and efficient multilabel classification in domains with large number of labels. In: SN. *Proc. ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD'08)*. [S.l.], 2008. v. 21, p. 53–59. Citado 2 vezes nas páginas 13 e 14.

TSOUMAKAS, G.; KATAKIS, I.; VLAHAVAS, I. Mining multi-label data. In: *Data mining and knowledge discovery handbook*. [S.l.]: Springer, 2009. p. 667–685. Citado 2 vezes nas páginas 40 e 44.

TSOUMAKAS, G.; VLAHAVAS, I. Random k-labelsets: An ensemble method for multilabel classification. In: SPRINGER. *European conference on machine learning*. [S.l.], 2007. p. 406–417. Citado 2 vezes nas páginas 12 e 13.

TURNBULL, D.; BARRINGTON, L.; TORRES, D.; LANCKRIET, G. Semantic annotation and retrieval of music and sound effects. *IEEE Transactions on Audio, Speech, and Language Processing*, IEEE, v. 16, n. 2, p. 467–476, 2008. Citado na página 43.

UEDA, N.; SAITO, K. Parametric mixture models for multi-labeled text. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2003. p. 737–744. Citado na página 43.

VALLIM, R. M. M. *Sistemas classificadores evolutivos para problemas multirrótulo*. Tese (Doutorado) — Universidade de São Paulo, 2009. Citado na página 22.

VENS, C.; STRUYF, J.; SCHIETGAT, L.; DŽEROSKI, S.; BLOCKEEL, H. Decision trees for hierarchical multi-label classification. *Machine learning*, Springer, v. 73, n. 2, p. 185, 2008. Citado 2 vezes nas páginas 35 e 44.

VILLALOBOS-ARIAS, M.; COELLO, C. A. C.; HERNÁNDEZ-LERMA, O. Asymptotic convergence of metaheuristics for multiobjective optimization problems. *Soft Computing*, Springer, v. 10, n. 11, p. 1001–1005, 2006. Citado na página 20.

VIRTANEN, P.; GOMMERS, R.; OLIPHANT, T. E.; HABERLAND, M.; REDDY, T.; COURNAPEAU, D.; BUROVSKI, E.; PETERSON, P.; WECKESSER, W.; BRIGHT, J. et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, Nature Publishing Group, v. 17, n. 3, p. 261–272, 2020. Citado na página 46.

WILCOXON, F. Individual comparisons by ranking methods. *Biometrics bulletin*, JSTOR, v. 1, n. 6, p. 80–83, 1945. Citado na página 48.

ZHANG, M.-L.; LI, Y.-K.; LIU, X.-Y.; GENG, X. Binary relevance for multi-label learning: an overview. *Frontiers of Computer Science*, Springer, v. 12, n. 2, p. 191–202, 2018. Citado na página 12.

ZHANG, M.-L.; ZHOU, Z.-H. A k-nearest neighbor based algorithm for multi-label classification. In: IEEE. *Granular Computing, 2005 IEEE International Conference on*. [S.l.], 2005. v. 2, p. 718–721. Citado na página 14.

ZITZLER, E. *Evolutionary algorithms for multiobjective optimization: Methods and applications*. [S.l.]: Citeseer, 1999. v. 63. Citado 2 vezes nas páginas 18 e 19.

ZITZLER, E.; LAUMANNS, M.; THIELE, L. Spea2: Improving the strength pareto evolutionary algorithm. *TIK-report*, Eidgenössische Technische Hochschule Zürich (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK), v. 103, 2001. Citado na página 21.

ZITZLER, E.; THIELE, L. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, IEEE, v. 3, n. 4, p. 257–271, 1999. Citado na página 21.