ANDRÉ BRITTO DE CARVALHO

OTIMIZAÇÃO POR NUVEM DE PARTÍCULAS MULTIOBJETIVO NO APRENDIZADO INDUTIVO DE REGRAS: EXTENSÕES E APLICAÇÕES

CURITIBA

2009

ANDRÉ BRITTO DE CARVALHO

OTIMIZAÇÃO POR NUVEM DE PARTÍCULAS MULTIOBJETIVO NO APRENDIZADO INDUTIVO DE REGRAS: EXTENSÕES E APLICAÇÕES

CURITIBA

2009

AGRADECIMENTOS

Inicialmente gostaria de agradecer à minha orientadora Aurora Pozo, pela ótima orientação durante o desenvolvimento deste trabalho e pela amizade. Agradeço também a chance que me proporcionou durante este mestrado, de trabalhar em uma área que gosto, em um ambiente excelente.

À CAPES, pelo financiamento deste mestrado.

Aos alunos do Grupo EsCel, pela amizade, receptividade, e disposição em ajudar sempre que me foi necessário.

À Beatriz, quem eu amo muito, e que sempre esteve ao meu lado e me apoiou durante todo o mestrado.

Aos meus pais, por tudo que fizeram pela minha formação, e pelo apoio e exemplo que têm me dado durante toda a minha vida.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

AM	Aprendizado de Máquina
AR	Aprendizado de Regras
AUC	*Area Under the ROC Curve*
BN	*Bayes Net*, Redes Bayesianas
FN	Falso Negativo (*False Negative*)
FP	Falso Positivo (*False Positive*)
MD	Mineração de Dados
$MHMO$	Metaheurísticas Multiobjetivo
$MOPSO$	*Multiobjective Particle Swarm Optimization*
NB	*Naive Bayes*, Redes Bayesianas
PSO	*Particle Swarm Optimization*
POO	Projeto Orientado a Objeto
RN	Redes Neurais
ROC	*Receiver Operating Characteristics*
SVM	*Support Vector Machines*
TN	Verdadeiro Negativo (*True Negative*)
TP	Verdadeiro Positivo (*True Positive*)

RESUMO

A área da Mineração de Dados (MD), ou *Data Mining*, consiste em analisar uma grande quantidade de dados buscando-se identificar associações e relações entre os dados que não são conhecidas a priori. Nesta área há uma necessidade por novas ferramentas e técnicas com a habilidade de ajudar software de forma inteligente na análise de grandes massas de dados. Neste sentido, algoritmos de Aprendizado de Máquina são as técnicas mais indicadas para estas aplicações. Para a representação do conhecimento extraído, regras são as formas mais utilizadas atualmente, devido a seu caráter intuitivo e simplicidade. O Aprendizado de Regras é uma técnica de Aprendizado de Máquina que têm o objetivo de produzir um conjunto de regras a partir de um conjunto de dados de entrada, que representam o conhecimento extraído. Além disso, o modelo gerado pode ser usado como um classificador. Este trabalho tem como objetivo o desenvolvimento de um algoritmo para o problema do Aprendizado de Regras no contexto da Mineração de Dados. Para isto, a técnica escolhida é a metaheurística Otimização por Nuvem de Partículas Multiobjetivo. Esta metaheurística é pouco explorada no Aprendizado de Regras e possui alguns problemas ainda sem solução. Assim, além do desenvolvimento do algoritmo, são propostas algumas soluções de problemas que surgem na aplicação da técnica MOPSO no Aprendizado de Regra. Neste trabalho, busca-se também produzir um bom classificador em termo da área abaixo da curva ROC, AUC. Para a validação do algoritmo e suas extensões é proposto um conjunto de experimentos, que comparam a técnica MOPSO com alguns algoritmos conhecidos da literatura. Por fim, o algoritmo é aplicado num estudo de caso do contexto da predição de defeitos em softwares.

ABSTRACT

Data mining is the overall process of extracting knowledge from data. In this context there is a significant need for tools and techniques with the ability to intelligently assist humans in analyzing very large collections of data in search of useful knowledge. In this sense, Machine Learning Algorithms play an important role and they are the most indicated techniques for these applications. In the study of how to represent knowledge in data mining context, rules are one of the most used representation form. Therefore, Rule Learning is a Machine Learning technique which has the goal to produce a rule set from the original dataset. Besides, the generated model can be used as a unordered classifier. This work has the goal to develop an algorithm to the Rule Learning problem in the Data Mining Context. For this, the chosen technique is the metaheuristic of Multiobjective Particle Swarm Optimization, that has some interest topics not studied yet and some problems without solution. So, along the algorithm implementation, this work proposes some extensions that solve problems that arise from the application of MOPSO technique in the rule induction. Besides, this work has the goal to produce a good classifier in terms of the area under the ROC curve, AUC. The algorithm and all the proposed extensions are evaluated through a set of experiments that compares the MOPSO technique with other well-known algorithms from the literature. Finally, is presente a study-case that applies the MOPSO algoritmo in the fault-proneness prediction context.

CAPÍTULO 1

INTRODUÇÃO

A área da Mineração de Dados (MD), ou *Data Mining*, consiste em analisar uma grande quantidade de dados buscando-se identificar associações e relações entre os dados que não são conhecidas a priori. Esta tarefa visa construir ferramentas inteligentes que buscam um conjunto de padrões que não são percebidos de forma simples como uma consulta ao Banco de Dados, mas sim informações interessantes que possam introduzir um novo conhecimento no contexto da sua aplicação. A MD é aplicada em diversas áreas do conhecimento como a bioinformática, sistemas financeiros, diagnósticos médicos, entre outros [31].

No contexto da Mineração de Dados há realização de diversas pesquisas. Uma das áreas de interesse é o estudo de como obter e representar o conhecimento extraído. Neste sentido, regras são uma das formas mais utilizadas para a representação do conhecimento extraído. Isto é devido à sua simplicidade, aspecto intuitivo e pelo fato de poder-se obter estas regras diretamente dos dados minerados [15].

Neste sentido, o Aprendizado de Regras tem sido definido como um componente fundamental de sistemas de Mineração de Dados. O Aprendizado de Regras (AR), ou Indução de Regras, é uma técnica de Aprendizado de Máquina que têm o objetivo de produzir um conjunto de regras a partir de um conjunto de dados rotulados, ou seja, um conjunto de dados que o valor das classes, para todos os exemplos pertencentes à base. Vale ressaltar que o AR foi a primeira técnica de Aprendizado de Máquina a se tornar parte de aplicações comerciais de sucesso.

No AR, um conjunto de regras é induzido a partir de um conjunto de dados históricos. Este conjunto de regras forma um modelo de classificação. A classificação é um dos principais métodos para a derivação de conhecimento novo e a predição de eventos futuros [24]. Através deste modelo, chamado de classificador, é possível a predição da classe de no-

vos exemplos ainda não rotulados. Porém, na tarefa do AR através de bases de dados rotuladas, vários problemas podem surgir como: ruídos nos dados históricos, desbalanceamentos das classes e o fato dos dados não representam corretamente a realidade, entre outros [24, 37].

Para a solução do problema do AR várias abordagens foram propostas. Os sistemas tradicionais utilizam uma abordagem em que um procedimento de busca é executado iterativamente. Nestes sistemas, normalmente a cada iteração a melhor regra é obtida e em seguida todos os exemplos cobertos por esta regra são retirados da base. O sistema de busca utilizado normalmente é um algoritmo guloso [38]. Este processo é executado até que todos os exemplos sejam cobertos por alguma regra. Esta abordagem possui grandes problemas, pois a remoção de exemplos da base pode causar a super-especialização das regras e além disto o classificador produzido é um classificador ordenado, logo a interpretação de uma regra depende de regras pré-descendentes [37].

Porém, existe um conjunto de novas técnicas que introduzem novas abordagens para a indução de regras. As Metaheurísticas Multiobjetivo (MHMO) apresentam uma abordagem onde as propriedades das regras podem ser avaliadas através de diferentes objetivos, como por exemplo, construir as regras mais simples e mais precisas [25] [45]. Além disso, estas técnicas encontram as regras em uma única execução, sem a necessidade da remoção de regras da base. Elas permitem a criação de classificadores que explorem conceitos da Fronteira de Pareto e assim gerem um conjunto com as melhores regras para os objetivos escolhidos. Por fim, as regras podem ser utilizadas como um classificador não-ordenado, mais intuitivo e de mais fácil interpretação. Porém, a construção de modelos que trabalhem com mais de um objetivo não é uma tarefa trivial [24]. Normalmente os objetivos selecionados estão em conflito, assim quando há a maximização de um valor o outro é minimizado.

1.1 Objetivos e Contribuições

Este trabalho tem como objetivo o desenvolvimento de um algoritmo para a solução do problema do Aprendizado de Regras no contexto da Mineração de Dados. Este algoritmo

utiliza as vantagens da Otimização por Nuvem de Partículas e da Otimização Multiobjetivo, chamado de MOPSO (do inglês, *Multiobjective Particle Swarm Optimization*), que ainda são pouco exploradas na MD. A técnica MOPSO é uma Metaheurísticas Multiobjetivo (MHMO) que permite a concepção de uma nova abordagem para a indução de classificadores. Nesta abordagem, as propriedades das regras podem ser expressas em diferentes objetivos. Além disso, outra propriedade interessante desta abordagem é que as regras podem ser usadas como um classificador não ordenado, facilitando a interpretação. O trabalho apresentado em [46] desenvolve um estudo no Aprendizado de Regras com a metaheurística da Nuvem de Partículas Multiobjetivo. Neste trabalho um algoritmo é desenvolvido e um conjunto de experimentos que analisam a influência dos parâmetros nos resultados da busca e confrontam os resultados com diferentes algoritmos da literatura é executado. O algoritmo aqui proposto utilizou alguns conceitos apresentados em [46].

As vantagens do MOPSO são aplicadas neste contexto do AR visando obter um classificador formado por regras SE-ENTÃO, que representem bem o domínio. O objetivo principal deste trabalho é a busca por produzir soluções para alguns problemas que surgem na aplicação da técnica MOPSO no contexto da MD. A principal contribuição deste trabalho é produzir soluções para os seguintes problemas:

- **Regras com pouca expressividade:** O conjunto de regras extraído deve representar algum conhecimento útil para o problema em questão. Assim, estas regras devem representar este conhecimento da melhor forma possível. Para evitar que as regras geradas contenham pouca informação e que contenham alguma informação útil, foram desenvolvidos uma extensão do MOPSO para trabalhar com dados numéricos e um procedimento que aumenta a expressividade dos atributos nominais. Além disso, foi implementado um procedimento que reduz o número total de regras do modelo, facilitando a interpretação.

- **Resultados de classificação ruins:** Como o modelo de regras pode ser usado como um classificador é interessante que está classificação produza bons resultados, comparados a métodos conhecidos da literatura. Neste sentido, o algoritmo

que trabalha com dados numéricos também tem o objetivo de produzir um classificador mais preciso. Foram desenvolvidos alguns métodos utilizados na etapa de classificação das regras para a melhora da classificação do algoritmo.

- **Execução do algoritmo em base de dados grandes:** Para ser aplicado na MD é interessante que o algoritmo produza os resultados em um tempo hábil de execução. Assim, é apresentada uma proposta de paralelização do algoritmo MOPSO. Esta proposta tem como intuito aplicar a técnica estudada neste trabalho em bases de dados de grande porte.

Além de desenvolver um algoritmo metaheurístico e buscar produzir soluções para problemas que surgem na aplicação do algoritmo no contexto do AR, este trabalho tem o desafio de produzir um classificador com um bom desempenho em termos da área abaixo da curva ROC (Receiver Operating Characteristics) [15]. A área da curva ROC, chamada de AUC, é considerada um critério relevante quando se trabalha com dados não balanceados, custos em erros na classificação ou dados com ruído.

Uma curva ROC é um gráfico que relaciona a taxa de falsos positivos (FP) (eixo-x) e a taxa de verdadeiros positivos (TP) (eixo-y) de um classificador [15]. Uma curva ROC pode ser obtida de uma única regra, um classificador parcial ou um classificador completo. Enfocando este desafio, dois objetivos para a execução do algoritmo foram escolhidos: sensitividade e especificidade [29]. Estes objetivos estão diretamente relacionados com a curva ROC. Sensitividade é uma medida de quanto um classificador prediz corretamente os exemplos da classe positiva (os verdadeiros positivos sobre o total de positivos). Especificidade é uma medida similar, mas em relação aos exemplos negativos (os verdadeiros negativos sobre o total de instâncias negativas). A sensitividade é equivalente ao eixo Y do gráfico ROC e a especificidade é o complemento do eixo X. Além disso, o trabalho apresentado em [24] mostra um amplo estudo sobre a qualidade de classificadores bi-objetivos através da maximização da sensitividade e da especificidade. Para a obtenção da AUC é necessário que as bases de dados analisadas possuam somente duas classes, assim é importante ressaltar que modelo desenvolvido neste estudo contempla bases de dados com somente duas classes.

Neste trabalho é feito um estudo sobre os principais tópicos relacionados ao algoritmo desenvolvido. Os conceitos de aprendizado de máquina e aprendizado de regras são discutidos, bem como são discutidas questões sobre como as regras são utilizadas como um classificador. É feito também um estudo mais aprofundado na metaheurística da Nuvem de Partículas Multiobjetivo. Além disso, são discutidos os experimentos de validação do algoritmo proposto. Nestes experimentos cada extensão proposta é avaliada e é feita uma comparação do algoritmo desenvolvido com alguns algoritmos de literatura, em termos de AUC. O desenvolvimento do algoritmo, suas extensões e validações foram apresentados nos trabalhos [6, 7, 8, 9].

Além do desenvolvimento e da validação do algoritmo MOPSO através de experimentos, este trabalho apresenta um estudo de caso. Neste estudo de caso, o algoritmo MOPSO é aplicado para a predição de defeitos em módulos de software na etapa de testes. O algoritmo é utilizado como um preditor de defeitos, através do uso de métricas de qualidade de métodos e classes. Este preditor é construído através de bases de dados com informações de projetos passados, que contêm os valores das métricas para os módulos já desenvolvidos. Através da construção do modelo de classificação é possível identificar quais módulos possuem uma maior probabilidade de ter defeito e assim reduzir o custo associado a tarefa de testes. Os resultados do estudo de caso foram apresentados nos trabalhos [10, 8].

1.2 Organização do Trabalho

Este trabalho está organizado da seguinte maneira. Inicialmente será apresentada a revisão bibliográfica sobre os trabalhos mais relevantes para o desenvolvimento do algoritmo proposto. Em seguida o problema do Aprendizado de Regras é discutido no Capítulo 3. O Capítulo 4 apresenta a metaheurística da Otimização por Nuvem de Partículas Multiobjetivo utilizada para a solução do problema de AR. Em seguida, o Capítulo 5 define os detalhes do algoritmo proposto e todas as soluções para os problemas apresentados. No Capítulo 6 são discutidos os resultados da validação do algoritmo proposto. Por fim, o estudo de caso que aplica o algoritmo MOPSO no contexto da predição de defeitos em

software é apresentado no Capítulo 7 e o Capítulo 8 apresenta as conclusões e desafios futuros deste trabalho.

CAPÍTULO 2

REVISÃO BIBLIOGRÁFICA

Neste capítulo são discutidos os principais trabalhos utilizados para o desenvolvimento do algoritmo MOPSO. São destacados os principais pontos de cada trabalho. Como apresentado na introdução, esta dissertação tem como objetivo o desenvolvimento de um algoritmo de Aprendizado de Regras baseado na metaheurística da Otimização por Nuvem de Partículas Multiobjetivo no contexto da Mineração de Dados. Este problema consiste na obtenção de regras de classificação, através de uma base de exemplos, que serão utilizadas posteriormente para a predição da classe de novos exemplos. Os trabalhos discutidos a seguir abrangem desde da área de Aprendizado de Máquina até tópicos mais específicos do projeto.

Em [37] é apresentado um estudo sobre técnicas de Aprendizado de Máquina com ênfase nos problemas que surgem quando estas técnicas são aplicadas no contexto Mineração de Dados. Este problemas normalmente derivam-se do fato de que nem sempre os dados obtidos para a mineração estão bem definidos e não há somente a necessidade de se encontrar um modelo que produza uma classificação precisa, mas também extrair informações sobre ela. Assim, através da imposição de novas restrições, a Mineração de Dados propicia novas direções para o Aprendizado de Máquina. Nesta tese são discutidos os conceitos do Aprendizado de Máquina e de Aprendizado de Regras e o estudo se concentra na solução de três problemas: a obtenção de novas abordagens para a geração de regras, a investigação de vários aspectos da proporção de exemplos entre classes e a construção de um método para a combinação de rankings. Para a solução destes problemas o trabalho propõe um novo algoritmo chamado ROCCER, que é baseado em conceitos da curva ROC. O algoritmo visa construir um bom classificador ordenado, através da idéia de maximizar a AUC gerada por um conjunto inicial de regras. O método proposto apresenta um bom resultado em relação à técnicas da literatura porém a interpretação

das regras é mais difícil devido à ordenação das regras e há a necessidade da execução de um algoritmo para a geração das regras iniciais.

Para o estudo e a implementação do módulo do cálculo da curva ROC o artigo [16] foi analisado. Este artigo apresenta um estudo sobre vários aspectos da curva ROC, e tem o intuito de ser usado como guia prático para pesquisas. Inicialmente o artigo define os conceitos sobre a curva ROC. Este conceitos são detalhados no Capítulo 3 deste trabalho. A curva ROC é largamente utilizado para a medição da qualidade de classificadores através da área abaixo da curva, AUC. Esta medida é utiliza para a comparação dos resultados apresentados neste trabalho. Além de discutir os conceitos, este trabalho também discute os detalhes de implementação da geração e cálculo da área da curva.

No artigo [42], é apresentado uma revisão de vários métodos da Otimização por Nuvem de Partículas Multiobjetivo, MOPSO. Este artigo foi utilizado para a definição das características do algoritmo proposto e serviu de base para os conceitos apresentados no Capitulo 4. Neste artigo, inicialmente são definidos os conceitos básicos para o entendimento da metaheurística MOPSO. Além dos conceitos gerais o artigo apresenta também a discussão de diversas soluções encontradas na literatura para o problema.

Em [46], a metaheurística MOPSO é aplicada ao problema do Aprendizado de Regras. Este trabalho apresenta o desenvolvimento de um algoritmo, bem como uma análise dos resultados da classificação. O algoritmo desenvolvido trabalha com bases de dados com somente atributos nominais. Também são discutidos alguns aspectos importantes para a aplicação das metaheurística no contexto do AR. Neste trabalho as regras são representadas através de um vetor de números reais, onde cada célula representa uma restrição de um atributo. As definições do uso de mais de um objetivo é efetuada através de conceitos da dominância de Pareto. Além destas definições o trabalho detalha questões sobre a implementação das equações básicas do técnica da Nuvem de Partículas. Nos experimentos executados, o classificador construído é baseado em dois conjuntos de objetivos: sensitividade e especificidade, e a confiança negativa e positiva. Nestes experimentos o algoritmo desenvolvido é confrontado com diferentes algoritmos da literatura, como o C4.5 [41] e o ROCCER [38]. Além disso, uma análise da influência dos parâmetros nos resultados da

busca é efetuada. Este trabalho mostrou que a metaheurística MOPSO é competitiva com outras técnicas já validadas. Devido aos bons resultados e pela simplicidade da técnica, que possibilita a extensão para trabalhar com os dados numéricos este trabalho serviu de base para o desenvolvimento do nosso projeto.

Os próximos capítulos irão apresentar detalhes sobre o problema do Aprendizado de Regras, sobre a metaheurística MOPSO e sobre o algoritmo desenvolvido neste projeto. Nestes capítulos, os conceitos básicos para o entendimento do algoritmo serão discutidos. Este conceitos derivam do estudo dos trabalhos obtidos nesta pesquisa bibliográfica.

CAPÍTULO 3

APRENDIZADO DE REGRAS

Na Mineração de Dados há uma demanda por ferramentas inteligentes para auxiliar a descoberta de conhecimento através de grandes bases de dados. Nesta tarefa, algoritmos de Aprendizado de Máquina (AM) têm sido largamente utilizados [31]. O Aprendizado de Máquina é uma sub-área da inteligência artificial em que se procura construir máquinas capazes de melhorar seus resultados através de sua própria experiência. O AM inclui qualquer programa de aprendizagem que melhora o seu desempenho em alguma tarefa utilizando alguma experiência[43]. No AM, o Aprendizado de Regras (AR) é uma tarefa que tem como objetivo a construção de regras a partir de um conjunto de dados rotulados, ou seja, que possuem informação da classe, visando a extração de conhecimento até então não conhecido. O AR é uma das aplicações de Aprendizado de Máquina (AM) mais importantes. A importância do Aprendizado de Regras deriva do fato do uso de regras ser uma das formas mais úteis para a representação de conhecimento no contexto da Mineração de Dados [15]. O uso de regras na Mineração de Dados é útil por causa de sua simplicidade, modularidade, de seu aspecto intuitivo e da possibilidade de obtenção das regras diretamente do banco de dados analisado. Além disso, o AR foi uma das primeiras técnicas de Aprendizado de Máquina a fazer parte, com sucesso, de aplicações comerciais de Mineração de Dados

Neste capítulo, inicialmente serão apresentadas as principais características do Aprendizado de Máquina, que são base para o entendimento do Aprendizado de Regras. Inicialmente serão discutidos os modelos de aprendizagem. Em seguida serão apresentados os métodos de inferência, os quais são usados pelas máquinas para manipular o conhecimento prévio e assim descobrir um novo conhecimento. Em seguida serão apresentados os conceitos básicos da aprendizagem de regras, tais como as linguagens de representação utilizadas, métodos para a avaliação das regras e a utilização de um conjunto não-ordenado

de regras como um classificador.

3.1 Modelos de Aprendizagem

Na aprendizagem de máquina o tipo de realimentação, ou seja, a forma com que o agente verifica se suas ações estão de acordo com as medidas esperadas, é importante para o processo de aprendizagem. Basicamente, os sistemas de AM são classificados em três tipos, de acordo com sua realimentação: aprendizagem supervisionada, aprendizagem não-supervisionada e aprendizagem por reforço [43].

Aprendizagem supervisionada: Envolve a aprendizagem de uma função a partir de exemplos das entradas e saídas do agente. Neste tipo de aprendizagem o agente observa o efeito de suas ações e verifica se eles estão de acordo com o esperado. Em um programa que aprende através de exemplos, os exemplos são rotulados, ou seja, a classe ou atributo meta de cada exemplo é conhecida. No aprendizado supervisionado, o objetivo é classificar os novos exemplos da entrada dentre as classes conhecidas.

Aprendizagem não-supervisionada: Envolve a aprendizagem de padrões de acordo com a entrada. Nesta aprendizagem o agente não verifica se suas ações estão de acordo com uma saída esperada, mas busca determinar novos padrões entre as entradas obtidas até então, e assim executar novas ações. Neste aprendizado, as entradas não são rotuladas, e o objetivo é descobrir similaridades entre elas.

Aprendizagem por reforço: A aprendizagem por reforço é a mais geral e envolve a idéia de recompensa ou penalização do agente a cada ação executada. Neste tipo de aprendizagem o agente deve aprender como o ambiente em que ele está inserido funciona. O objetivo é maximizar a quantidade de recompensa recebida em todo o processo.

O trabalho desenvolvido neste projeto envolve o aprendizado supervisionado. As técnicas de Mineração de Dados visam identificar novos exemplos de acordo com um conjunto já conhecido. A idéia é identificar a qual classe esta nova entrada pertence, buscando identificar quais atributos são importantes para o novo exemplo ser classificado.

3.2 Métodos de Inferência

O processo de aprendizagem está relacionado com a maneira em que o conhecimento é manipulado [37]. Os métodos de inferência estão relacionados com o modo em que o agente descobre novos conhecimentos de acordo com suas percepções. Assim, os métodos de inferência são a forma em que o agente consegue aprender de acordo com suas ações e percepções. Esses métodos estão classificados em três tipos: dedução, abdução e indução [43].

Dedução: Na dedução a inferência ocorre através de um conjunto de premissas verdadeiras que produzem uma conclusão que preserva a verdade. Nesta forma de inferência o resultado obtido é uma forma de transformação do conhecimento conhecido, gerando sempre um resultado verdadeiro. Este tipo de inferência só é aplicado quando se quer obter informações que estão presentes nos dados, mas de forma implícita. Este modelo é comumente definido através da regra de inferência de Modus Pones [43].

Abdução: Nesta forma de inferência a conclusão é obtida não somente do conjunto de premissas conhecidas, mas também através de extensões do conhecimento do problema. Assim, a abdução trabalha com um conhecimento incompleto, utilizando as informações conhecidas de indivíduos para completar o conhecimento. Assim, uma conclusão da inferência abdutiva só é verdade se a extensão utilizada também for verdade.

Indução: Nesta inferência busca-se extrapolar as medidas e inferir um conhecimento genérico, a partir de exemplos particulares. A inferência indutiva parte de um conhecimento específico e o generaliza, nem sempre havendo a preservação da verdade na conclusão. Vale ressaltar que este método de inferência é usado no aprendizado de regra, pois como será apresentado, este tipo de aprendizado parte de um conjunto inicial de exemplos e visa construir um conjunto genérico de regras que represente o conhecimento aprendido.

3.3 Conceitos de Aprendizagem de Regras

No Aprendizado de Regras em estudo neste projeto, têm-se como objetivo construir um modelo de aprendizagem indutiva, supervisionado e simbólico. O aprendizado simbólico significa que será construído um conjunto de regras que são interpretáveis pelos seres

humanos. A idéia é construir um conjunto de regras intuitivas que auxiliem inteligentemente a análise de grandes quantidades de dados. Como apresentado na seção anterior, a inferência indutiva obtém conclusões genéricas a partir de um conjunto de premissas. Assim, a aprendizagem indutiva visa obter novo conhecimento a partir de um conjunto de exemplos. A idéia é utilizar o aprendizado supervisionado e a partir de exemplos rotulados induzir uma hipótese e produzir um novo conceito para os dados em análise. Uma definição formal para este aprendizado, definida por *Bratko*, apresentada em [37], é a seguinte:

> Seja U o conjunto universal dos objetos, isto é, todos os objetos que o aprendiz pode encontrar. Não existem limites, a princípio, para o número de exemplos de U. Um conceito C pode ser formalizado como sendo um subconjunto de objetos em U, ou seja, $C \subset U$. Aprender o conceito C significa aprender a reconhecer objetos em C. Ou seja, uma vez que o conceito C é aprendido, para qualquer objeto $x \in U$, o sistema deve ser capaz de reconhecer $x \in C$.

No contexto da Mineração de Dados, o universo dos objetos é a grande massa de dados disponível para análise. O objetivo é a construção de um conjunto de regras, que irão representar um determinado conceito. Este conjunto de regras terá informações que estão disponíveis nos dados de entrada e irão representar algum novo conhecimento sobre eles. Como exemplo, dado um banco de dados médico, que possui um conjunto de informações de pacientes. Os atributos que descrevem os dados são informações sobre sintomas, e o atributo meta é a doença do paciente. Através do aprendizado indutivo de regras pode-se induzir um conceito que associa um determinado conjunto de sintomas a uma doença conhecida. Assim, através das regras encontradas é possível analisar a importância dos sintomas para a presença da doença. Além disso, com a construção do modelo, caso haja informação de um novo paciente, pode-se verificar se o conceito induzido reconhece esse novo paciente. Ou seja, pode-se verificar se o novo paciente tem a doença reconhecida pelas regras encontradas.

Formalmente o problema pode ser definido da seguinte forma: Seja Q um conjunto finito de atributos, os quais na prática correspondem a campos em uma base de dados.

Cada atributo $q \in Q$ tem um domínio associado, $Dom(q)$. Um teste de atributo, b, consiste de um atributo, $at(b) \in Q$, e um conjunto de valores $Val(b) \in Dom(at(b))$ e pode ser escrito como $at(b) \in Val(b)$. Um registro satisfaz este teste se o seu valor para o atributo $at(b)$ pertence ao conjunto de valores $Val(b)$. Um algoritmo de Aprendizado de Regras permite somente alguns tipos de restrições nos valores dos atributos. Alguns tipos de restrições aos atributos categóricos ou nominais são descritos a seguir:

- Valor: $Val(b) = \{v(b)\}$, onde $v(b) \in Dom(at(b))$. Isto pode ser escrito $at(b) = v(b)$.
- Desigualdade: $Val(b) = \{x \in Dom(at(b)) : x \neq v(b)\}$, onde $v(b) \in Dom(at(b))$. Pode ser escrito como $at(b) \neq v(b)$.
- Subconjunto: Val(b) é irrestrito, ou seja, qualquer subconjunto em $Dom(at(b))$.

Tipos de testes para atributos numéricos são descritos a seguir:

- Partição Binária: $Val(b) = \{x \in Dom(at(b)) : x \leq v(b)\}$ or $Val(b) = \{x \in Dom(at(b)) : x \geq v(b)\}$, onde $v(b) \in Dom(at(b))$. Neste caso, o teste de atributo pode ser escrito como $at(b) \leq v(b)$ or $at(b) \geq v(b)$, respectivamente.
- Intervalo de Valores: $Val(b) = \{x \in Dom(at(b)) : l(b) \leq x \leq u(b)\}$, onde $l(b), u(b) \in Dom(at(b))$. O teste de atributo é escrito como $l(b) \leq at(b) \leq u(b)$.

Nesta seção serão apresentados os principais conceitos sobre o aprendizado indutivo de regras. No Capítulo 4 são discutidos os detalhes de como um programa realmente aprende, que é através do mapeamento do problema para um problema de busca. Inicialmente, será apresentado como os exemplos utilizados na aprendizado são descritos e como as regras induzidas são representadas. A seguir, serão discutidos métodos de avaliação da qualidade das regras criadas. Por fim será apresentada a utilização de um conjunto de regras como um classificador não-ordenado bem como métodos de medição da qualidade do classificador.

3.3.1 Descrição dos exemplos e representação de regras

O programa induz um conjunto de regras a partir de exemplos rotulados. Nesta seção, será apresentada a forma de descrição de exemplos utilizada, bem como será apresentado como as regras são representadas.

3.3.1.1 Linguagem de representação dos exemplos

Para qualquer tarefa de aprendizado é necessária uma forma de descrição dos exemplos da entrada. Para isso existem diversas linguagens utilizadas pelos algoritmos de aprendizado [37]. Dentre essas linguagens pode-se citar a lógica proposicional, a lógica de primeira ordem e a lógica de atributos. Cada linguagem diferencia de acordo com sua complexidade e poder de expressão. A lógica de atributos é amplamente utilizada pelos algoritmos de aprendizado [36] e será utilizada no desenvolvimento do algoritmo proposto. A lógica de atributos é equivalente à lógica proposicional, mas possui uma notação mais poderosa. Nesta linguagem os exemplos são representados através de um conjunto de atributos que podem possuir diferentes valores. Essa notação é conhecida como `atributo-valor`. Essa notação é largamente utilizada em aplicações comerciais. Um exemplo da notação é:

$$previsao = sol \wedge temperatura = 30 \wedge vento = nao \rightarrow classe = bom\_para\_esportes$$

No Aprendizado de Regras, um programa recebe como entrada um conjunto de dados, que são os exemplos rotulados. O conjunto de dados possui uma lista de atributos que descreve os exemplos e pode ser definida como $A_i \mid i \in \{1, ..., n_{atrib}\}$. Esta lista é fixa e cada atributo pode assumir valores discretos (qualitativo), ou numéricos (quantitativo). Além da lista, cada exemplo possui um atributo meta, que é a classe, que possui um conjunto finito de valores discretos. Um exemplo, $e_j = (v_{1,j}, ..., v_{n\_atrib,j}, c_{i,j})$, é definido por uma lista com valores para cada atributo, $v_{i,j}$, mais o valor da classe, $c_{i,j}$. O conjunto de dados é definido por uma tabela contendo n exemplos com possíveis valores para os atributos e a classe. A Tabela 3.1 mostra um exemplo de um conjunto de dados.

A Tabela 3.2 apresenta um exemplo de uma base de dados com informações sobre o tempo para a prática de esportes. Os exemplos são descritos por quatro atributos

Tabela 3.1: Conjunto de exemplos no formato atributo-valor.

	A_1	A_2	$\cdots$	$A_{n_{atrib}}$	**Classe**
e_1	$v_{1,1}$	$v_{1,2}$	$\cdots$	$v_{1,n_{atrib}}$	c_1
e_2	$v_{2,1}$	$v_{2,2}$	$\cdots$	$v_{2,n_{atrib}}$	c_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
e_n	$v_{n,1}$	$v_{n,2}$	$\cdots$	$v_{n,n_{atrib}}$	c_n

Tabela 3.2: Conjunto de informações sobre o tempo para a prática de esportes.

Previsão	**Temperatura**	**Umidade**	**Vento**	**Praticar esportes**
sol	85	85	não	não
sol	80	90	sim	não
nublado	83	86	não	sim
chuvoso	70	90	não	sim
chuvoso	68	80	não	sim
chuvoso	65	70	sim	não
nublado	64	65	sim	sim
sol	72	95	não	não
sol	69	70	não	sim
chuvoso	75	80	não	sim
sol	75	70	sim	sim
nublado	72	90	sim	sim
nublado	81	75	não	sim
chuvoso	71	91	sim	não

que representam as condições do tempo: previsão do tempo (sol, chuvoso ou nublado), temperatura (valor numérico medido em fahrenheit), umidade (valor numérico que mede a umidade relativa do ar) e vento (sim ou não). Cada exemplo é uma combinação de valores desses atributos, e para cada um há a classe que representa se o tempo é bom ou não para a prática de esportes.

O conjunto de dados pode ser usado em diferentes etapas do aprendizado. As duas principais formas de uso dos dados no aprendizado são apresentadas a seguir:

Conjunto de treinamento: É o conjunto de entrada do aprendizado. A partir desses dados o programa irá aprender e induzir as regras. Este conjunto deve ser representativo de toda distribuição da população [36].

Conjunto de testes: Este conjunto é usado para avaliar o modelo após o aprendizado. É através destes dados que serão calculadas as medidas do classificador.

3.3.1.2 Linguagem de representação das regras

Segundo a definição do aprendizado indutivo de regras, o programa busca aprender um conceito a partir de um conjunto universo de exemplos. Este conceito deve ser um subconjunto deste universo. Assim, as regras induzidas, que representam o conceito aprendido, são uma combinação de valores dos atributos da base, mais o valor da classe. No aprendizado supervisionado uma regra é do tipo `SE-ENTÃO`, ou seja:

SE *antecedente* ENTÃO *conseqüente*

O antecedente contém uma ou mais restrições de valores dos atributos e o conseqüente a classe. As restrições são formadas por um atributo, um valor e uma condição que pode ser $<$, $\leq$, $=$, $>$, ou $\geq$. O antecedente não precisa ser formado por restrições para todos os atributos, deve ter no mínimo um e no máximo o número total de atributos. Para cada atributo deve existir somente uma restrição. O conseqüente assume um dos possíveis valores da classe presente na base em análise. Um exemplo de uma regra para Tabela 3.2 pode ser:

SE previsao=sol e temperatura=30 e vento=nao ENTÃO classe=sim

A notação SE-ENTÃO pode ser simplificada no par ordenado (consequente, antecedente) ou (`corpo`, `cabeça`), respectivamente. Este par ordenado pode ser representado da seguinte forma:

corpo $\rightarrow$ cabeça ou B $\rightarrow$ H (*Body, Head*)

Dado um exemplo, e_i, do conjunto de dados, uma regra R cobre o exemplo se todos os valores do exemplo satisfazem as restrições da regra. R cobre corretamente o exemplo se ambos são da mesma classe. Assim, se R cobre e_i, B é verdadeiro. Se cobre corretamente B e H são verdadeiros. Caso o antecedente não seja satisfeito, diz-se que B não é verdadeiro, e caso as classes sejam diferentes diz-se que H não é verdadeiro. Estas definições são aspectos básicos para a avaliação das regras, próximo tópico desta seção.

3.3.2 Avaliação de Regras

A avaliação das regras induzidas é um fator importante para medir se o conhecimento aprendido esta de acordo com os dados da entrada. Para isto existem várias formas de avaliação. Cada regra pode ser analisada individualmente ou ainda pode-se obter medidas de todo o classificador. Nesta seção são apresentados os conceitos para a análise individual de cada regra. Na Seção 3.3.3 será apresentado como utilizar um conjunto de regras não ordenadas como um classificador.

3.3.2.1 Matriz de contingência

Para uma análise mais precisa do conjunto de regras gerado é necessária a avaliação de cada regra individualmente. Com está análise é possível a descoberta de regras que representem melhor o domínio e que introduzam um novo conhecimento ao problema. Além disso, é possível o uso destas medidas de avaliação como objetivos no processo de busca, que será apresentado no Capítulo 4

Para a medição da qualidade das regras é utilizada a `matriz de contingência` [29]. Uma matriz de contingência para uma regra R no formato $B \rightarrow H$ é apresentada na Tabela 3.3. Na Tabela 3.3, B denota o conjunto de exemplos as quais o corpo de R é verdadero e $\overline{B}$, o seu complemento, o conjunto de instâncias que o corpo é false; o mesmo é verdade para H, o conjunto de exemplos o qual a cabeça da regra é verdadeira, e $\overline{H}$, conjunto de exemplos que a cabeça da regra e falsa. HB denota os exemplos que são cobertos corretamente pela regra. $\overline{H}B$ denota os exemplos os quais o corpo é verdadeiro mas a cabeça é falsa e assim por diante. A cardinalidade de cada conjunto também é representada na tabela. Assim, $b = B$, representa o número de exemplos que possuem o corpo como verdadeiro para R. O mesmo é verdade para os outros conjuntos, h, $\overline{h}$ e $\overline{b}$. Há também o número total de exemplos, n_{ex},que é a soma $h + \overline{h}$ ou $b + \overline{b}$.

Tabela 3.3: Matriz de Contingência.

	H	$\overline{H}$	
B	hb	$\overline{h}b$	b
$\overline{B}$	$h\overline{b}$	$\overline{h}\overline{b}$	$\overline{b}$
	h	$\overline{h}$	n_{ex}

3.3.2.2 Medidas de avaliação

Através da matriz de contingência é possível o cálculo de diversas medidas para regras [29]. Estas medidas são definidas através da probabilidade condicional dos conjuntos especificados na seção anterior. Nesta seção serão apresentadas as medidas mais utilizadas no Aprendizado de Regras.

Precisão (Acc): Mede o quanto a regra é específica para o problema. Quanto maior o valor da precisão, mais a regra está associada à classe em questão. Também conhecida como confiança.

$$Acc(R) = P(H|B) = \frac{P(HB)}{P(B)} = \frac{hb}{b} \tag{3.1}$$

Erro (Err): É o complemento da precisão, $1 - Acc$. Quanto maior o valor do erro menos a regra está associada à classe.

$$Err(R) = P(\overline{H}|B) = \frac{\overline{h}b}{b} \tag{3.2}$$

Confiança negativa ($NegRel$): É equivalente a precisão, mas agora para os exemplos não cobertos pela regra.

$$NegRel(R) = P(\overline{H}|\overline{B}) = \frac{\overline{hb}}{\overline{b}} \tag{3.3}$$

Precisão de Laplace (Acc): Equivale a medida da precisão, mas penaliza as regras que cobrem poucos exemplos. N_{cl} representa o número de classes do domínio.

$$Acc(R) = P(H|B) = \frac{P(HB)}{P(B)} = \frac{hb+1}{b+N_{cl}} \tag{3.4}$$

Sensitividade ($Sens$): É uma medida relativa dos exemplos da classe positiva que são cobertos pela regra. Quanto maior é o valor da sensitividade, maior é o número de

exemplos da classe positiva que são cobertos pela regra. Também chamada de completeza.

$$Sens(R) = P(B|H) = \frac{hb}{h} \tag{3.5}$$

Especificidade (*Spec*): É equivalente a sensitividade, mas para os exemplos que não são cobertos pela regra. Quanto maior o valor da especificidade, menos exemplos da classe negativa são cobertos de forma errônea pela regra.

$$Spec(R) = P(\overline{B}|\overline{H}) = \frac{\overline{hb}}{\overline{h}} \tag{3.6}$$

Cobertura (*Cov*): É a medida relativa no número de exemplos cobertos pela regra, exemplos em que o corpo é verdadeiro, sem levar em consideração a cabeça. Quando maior o valor da cobertura, maior é o número de exemplos cobertos pela regra.

$$Cov(R) = P(B) = \frac{b}{n_{ex}} \tag{3.7}$$

Suporte (*Sup*): É a medida relativa do número de exemplos cobertos corretamente pela regra.

$$Sup(R) = P(HB) = \frac{hb}{n_{ex}} \tag{3.8}$$

3.3.3 Classificador Não-Ordenado

Além de representar o conhecimento extraído, o conjunto de regras pode ser agregado para a construção de um classificador. A tarefa de classificar novos exemplos que ainda não contenham a informação sobre o atributo meta é chamada de classificação [43]. O modelo gerado, que é utilizado na classificação, é chamado de classificador.

Regras podem ser utilizadas como um classificador através de duas abordagens. Na primeira, as regras representam uma lista ordenada onde a interpretação de cada regra depende das regras precedentes [38]. Na segunda abordagem, as regras são obtidas sem a criação de uma lista e podem ser utilizadas como um classificador não-ordenado. A técnica em estudo neste trabalho produz um classificador não-ordenado. Esta seção irá

apresentar o funcionamento de um classificador de regras não-ordenado.

A tarefa de classificação de novas instâncias através de um conjunto de não-ordenado de regras é efetuada através de um processo de votação. Este processo é apresentado na Figura 3.1. Para a classificação de uma nova instância de entrada e, inicialmente é verificado quais regras que pertencem ao classificador cobrem o exemplo. Caso nenhuma regra cubra esse exemplo é tomada uma decisão padrão, por exemplo, votar o exemplo como sendo pertencente à classe majoritária da base de dados. Após a identificação das regras que cobrem e, cada regra é separada em subconjuntos de acordo com a sua classe. Após a separação dos subconjuntos de regras, cada subconjunto é ordenado de acordo com algum critério. Normalmente é utilizado como critério para ordenação alguma das medidas apresentadas na Seção 3.3.2.2. Após a ordenação das regras é feita uma seleção das k melhores regras de cada subconjunto de acordo com o critério da seleção. Assim, neste processo de seleção somente as melhores regras de cada subconjunto votam. O número de regras selecionado pode ser desde o número de classes do domínio até o total de regras no conjunto. Por fim, após a seleção das regras que irão votar cada regra executa uma votação ponderada. É utilizado como peso uma das medidas da Seção 3.3.2.2. É utilizado um peso na votação para produzir uma classificação mais precisa. Assim regras que possuam um valor ruim no peso selecionado possuem uma menor influência na votação. De acordo com o critério de ordenação, o número k de regras que votam e com a escolha do peso é possível obter um melhor resultado na classificação. Em [48] é apresentado um estudo com diversas formas de ordenação e seleção do número de regras. Ao final do processo, a classe que obtiver a maior soma de votos é escolhida como a classe do exemplo e. O número de votos do exemplo também é utilizado para a definição do ranking utilizado no cálculo da AUC (Seção 3.3.4).

Para a validação do classificador existem diversas medidas. As próximas seções irão apresentar as principais medidas utilizadas para a comparação de classificadores. Estas medidas serão utilizadas no Capítulo 6 que apresenta a validação do estudo proposto neste trabalho.

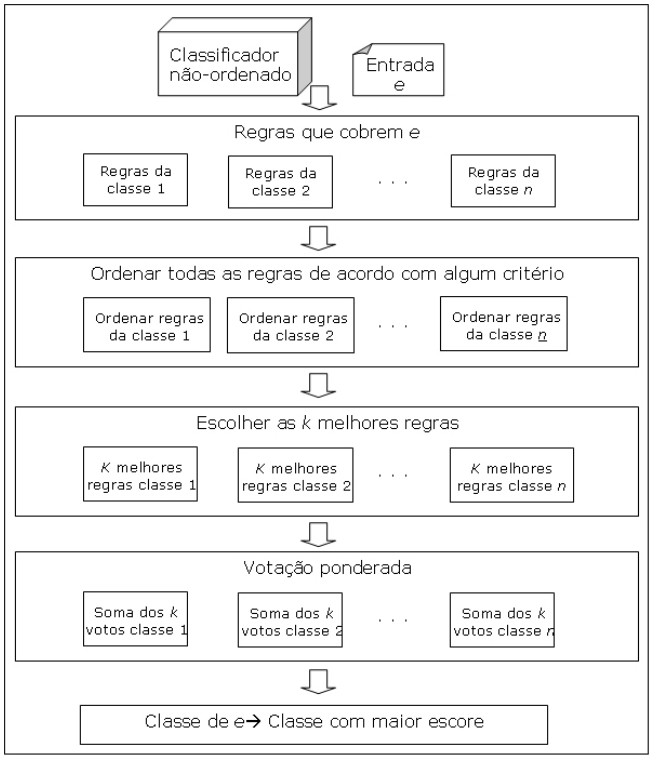

Figura 3.1: Processo de votação de um classificador não-ordenado de regras.

3.3.3.1 Matriz de confusão

A matriz de confusão é utilizada para medir a taxa de erros na classificação através de exemplos ainda não conhecidos. Esta medição é chamada de `precisão`, e é obtida através dos erros e acertos do classificador. A matriz é construída após a indução do classificador. Após a etapa de treinamento, um novo conjunto de dados é submetido ao classificador, o conjunto de teste. Para cada exemplo deste conjunto, o modelo induzido prevê qual classe esse exemplo pertence. Baseado nos erros e acertos do modelo a matriz de confusão é criada.

Nesta seção será apresentada uma matriz de confusão binária, somente com duas classes, pois o estudo apresentado neste trabalho somente contempla bases de dados binárias. Logo, para a construção da matriz os exemplos devem estar classificados de forma binária. Uma classe é considerada a classe `positiva` e a outra a `negativa`. Baseado nas duas possíveis classes, a construção da matriz produz quatro possíveis resultados:

Verdadeiro Positivo ou *True Positive*(TP): Ocorre quando o exemplo de teste submetido ao modelo pertence à classe positiva e é predito também como positivo.

Tabela 3.4: Matriz de Confusão.

		Predição	
		(+)	(−)
Real	(+)	TP	FN
	(−)	FP	TN

Falso Positivo ou *False Positive*(FP): Ocorre quando o exemplo de teste pertence à classe negativa, mas é predito como positivo.

Verdadeiro Negativo ou *True Negative*(TN): Ocorre quando o exemplo de teste submetido ao modelo pertence à classe negativa e é predito também como negativo.

Falso Negativo ou *False Negative*(FN): Ocorre quando o exemplo de teste pertence à classe positiva, mas é predito como negativo.

Dados os possíveis resultados a matriz é preenchida com os valores da predição para todos os exemplos do conjunto de teste. A Tabela 3.4 apresenta um exemplo da matriz de confusão. Como apresentado anteriormente, através dos acertos e erros do classificador é possível calcular a precisão. Essa precisão pode ser calculada para as duas classes, a precisão da classe positiva e da classe negativa. Há também o cálculo da precisão total do modelo. A seguir será apresenta uma descrição de algumas medidas que são obtidas através da matriz de confusão.

A precisão total do classificador, ou acurácia, relaciona o número total de instâncias corretamente classificadas (TP e TN) com o número total de exemplos. Quanto maior o valor da acurácia, maior são os acertos do classificador, ou seja, um classificador com alto valor de acurácia possui boas predições tanto para a classe positiva quanto para a negativa.

$$Acuracia = \frac{TP + TN}{N} \tag{3.9}$$

A precisão também pode ser calculada para cada classe em análise. Ela relaciona o número total de instâncias corretamente classificadas para uma classe, por exemplo TP para a classe positiva, com o número total de exemplos preditos para esta classe. Quanto maior o valor da precisão da classe, menor é o número de erros na classificação.

Classificadores com altos valores de precisão para a classe positiva não predizem exemplos positivos como negativos, o mesmo é verdade para a classe negativa. A Equação 3.10 apresenta o cálculo da precisão para a classe positiva.

$$Precisao = \frac{TP}{TP + FP} \tag{3.10}$$

O *recall* é uma medida também obtida para cada classe do domínio. Ele relaciona o número de instâncias corretamente classificadas com o total de exemplos pertencentes a esta classe. Quanto maior o *recall*, maior são os acertos do classificador para uma determinada classe. Um classificador com alto valor de *recall* para uma determinada classe, identifica um maior número de exemplos para esta classe. A Equação 3.11 apresenta o cálculo do *recall* para a classe positiva.

$$Recall = \frac{TP}{TP + FN} \tag{3.11}$$

A medida *F-measure* relaciona os valores da precisão e do recall de um classificador. É uma medida útil, pois mede o *trade-off* entre a precisão e o recall. Por exemplo, classificadores com um alto valor de precisão podem possuir um valor baixo de *recall* (o classificador não faz muitas predições erradas, porém não prediz um grande número de instâncias). Assim, a análise do *F-measure* provê uma melhora avaliação do que analisar a precisão e o recall separadamente.

$$F - measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.12}$$

Porém, apesar das medidas apresentadas serem medidas dos erros e acerto do classificador, nem sempre representam a qualidade do modelo. Neste sentido será a próxima seção irá apresentar a Análise do Gráfico ROC que é uma medida gráfica para a comparação de classificadores largamente usada no contexto da Mineração de Dados.

3.3.4 Análise do Gráfico ROC

A análise do Gráfico ROC (*Receiver Operating Characteristics*) é um método de análise da qualidade de sistemas de predição. Este método foi inicialmente utilizado na teoria de sinais para a descrição dos acertos e erros na recepção [15]. O seu uso foi estendido para áreas como a medicina, economia e previsão do tempo [37]. A análise ROC tem sido largamente usada no Aprendizado de Máquina para a avaliação e comparação de classificadores. Esta técnica é um método gráfico genérico de medição de performance que possui propriedades que são úteis para a análise de classes desbalanceadas. Nesta seção serão apresentados alguns conceitos sobre a análise ROC e como ela é utilizada como medida de qualidade de classificadores. A discussão será restrita para problemas com somente duas classes.

Como apresentado na seção anterior, é possível obter medidas de um classificador através da matriz de confusão. Porém, o uso de uma medida única pode não ser adequado para a a escolha do melhor classificador. Uma medida que se mostra ideal para a análise de um determinado domínio pode não ser útil em outros problemas. Por exemplo, o uso da precisão, que é amplamente usada no Aprendizado de Máquina, nem sempre é bom. Esta medida mede a taxa de acertos e erros de um classificador. Porém, ela só leva em consideração exemplos de uma classe, e pode introduzir uma falsa idéia de boa performance. Caso a base de dados seja desbalanceada, ou seja, possui muito mais exemplos de uma determinada classe, o uso da precisão pode não ser apropriado. Nesta situação, como há muitos exemplos de uma só classe a taxa de erros é muito pequena, logo o classificador terá uma alta precisão. Porém, haverá dificuldade na classificação de exemplo da classe minoritária e o classificador irá errar muito nesses casos.

Para evitar avaliações erradas de classificadores existem outros métodos para esta análise. Uma alternativa é uma análise gráfica dos resultados produzidos pelo classificador. É neste contexto que se insere o gráfico ROC.

O gráfico ROC é obtido através de medidas derivadas da matriz de confusão. Este gráfico descreve as relações de benefícios e custos do classificador [15]. Um gráfico ROC é um gráfico bi-dimensional, onde o eixo Y corresponde a taxa de verdadeiros positivos,

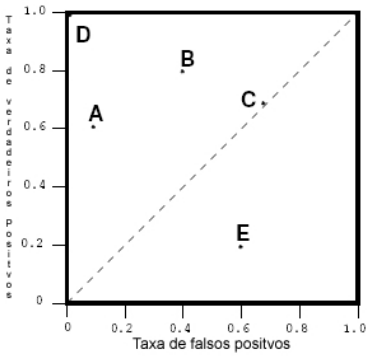

Figura 3.2: Gráfico ROC com cinco classificadores

true positive rate (tpr) e o eixo X corresponde a taxa de falsos positivos, *false positive rate* (fpr). As equações de tpr e fpr são:

$$tpr = \frac{TP}{P}, \text{ onde } P \text{ é o número dos exemplos positivos;}$$
$$fpr = \frac{FP}{N}, \text{ onde } N \text{ é o número dos exemplos negativos.}$$

Cada classificador em análise produz um ponto no gráfico ROC, (fpr, tpr). A Figura 3.2 mostra um gráfico ROC com cinco classificadores. A posição em que cada ponto ocupa no gráfico é importante para a avaliação dos classificadores. O ponto $(0, 0)$ representa um classificador que não tem nenhum falso positivo, mas também não possui nenhum verdadeiro positivo. O ponto $(1, 0)$ representa o classificador perfeito, pois a taxa de verdadeiro positivo é máxima e não há nenhum falso positivo. O ponto D no gráfico representa este classificador. Um classificador possui um valor melhor que o outro se ele estiver a cima e a esquerda do outro.

A linha tracejada, $y = x$ representa um classificador aleatório. Nestes casos o classificador tem a mesma probabilidade de produzir um falso positivo e um verdadeiro positivo. Logo ele sempre terá o valor de tpr semelhante ao de fpr. Classificadores aleatórios terão seus valores sempre próximos a diagonal do sistema de coordenadas. O ponto C é um classificador aleatório. Assim, para obter uma boa performance, o classificador deve ficar na parte superior da diagonal. Caso o ponto do classificador fique abaixo ou muito próximo da diagonal seu desempenho é semelhante a uma classificador aleatório, logo ele não apresenta bons resultados. O ponto E representa um classificador com baixa performance.

Porém nem sempre é possível determinar qual classificador é melhor através do gráfico ROC. Por exemplo, não é possível determinar qual dos pontos A e B da Figura 3.2 é melhor. Há ainda situações em que dois pontos estão muito próximos e não se tem noção o quanto um é melhor do que o outro. Para obter uma medida mais exata da qualidade dos classificadores através do gráfico ROC existe a construção de curvas ROC. A construção destas curvas é possível através da ordenação dos exemplos, ao invés de somente calcular as taxas tpr e fpr do classificador. Esta ordenação pode ser efetuada de diversas maneiras, dependendo do classificador usado. Por exemplo, nas Redes Neurais pode-se usar o valor da saída para ordenar as entradas. Através da ordenação dos exemplos é possível a definição de um limiar que divide os exemplos em dois grupos, um grupo pertencente a uma classe, e outro pertencente a segunda classe. Com esta definição de grupos é possível o cálculo das taxas de acerto. Um exemplo de uma ordenação é apresentado a seguir:

Seja C um classificador genérico. É dado como entrada para C o seguinte conjunto de testes:

$$A_1,\ A_2,\ A_3,\ A_4,\ A_5 \text{ - pertencentes à classe } A$$
$$B_1,\ B_2,\ B_3,\ B_4,\ B_5 \text{ - pertencentes à classe } B$$

Dada esta entrada, C produz a ordenação presente na Tabela 3.5. Se for definido como limiar o índice 5 e considerado tudo que estiver atrás do limiar, incluindo o limiar, como pertencente ao grupo A, classe positiva, e os que estiverem a frente como pertences ao grupo B, classe negativa, será feita uma classificação perfeita. Neste caso ter-se-ia a taxa de verdadeiros positivos sendo 100% e a de falsos positivos de 0%. Porém a escolha deste limiar é feita de forma arbitrária e nem sempre é possível a escolha do melhor valor do limiar. No exemplo, caso o limiar escolhido fosse o índice 1, ter-se-ia a taxa de verdadeiros positivos igual a 20% e a de falsos positivos de 0%.

Como nem sempre é possível escolher o melhor limiar para a obtenção das taxas através da ordenação dos exemplos é efetuada uma simulação da escolha dos limiares [15]. Assim o limiar é variado de $-\infty$ até $+\infty$ e para cada limiar escolhido é obtido um ponto (fpr, tpr). Com o conjunto desses pontos é possível traçar uma curva no gráfico ROC e analisar a qualidade do classificador. A Figura 3.3 mostra um exemplo de 3 curvas ROC.

Tabela 3.5: Exemplo da ordenação do classificador C.

1	2	3	4	5	6	7	8	9	10
A_2	A_4	A_1	A_5	A_3	B_1	B_2	B_5	B_4	B_3

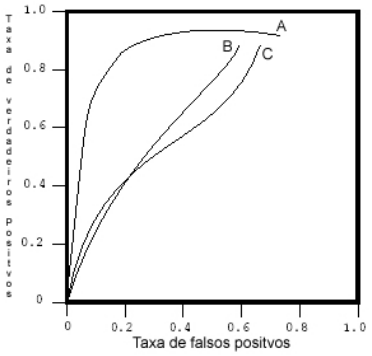

Figura 3.3: Exemplo de três curvas ROC.

Para comparar o resultado de duas ou mais curvas ROC deve-se analisar qual curva está mais próxima do ponto $(0, 1)$. Na figura 3.3, a curva A apresenta o melhor desempenho. Nas curvas B e C, pode-se dizer que C possui um melhor resultado até o ponto de intersecção. A partir deste ponto a curva B possui um melhor desempenho.

Para se obter uma medida mais exata da qualidade do classificador através da curva ROC é utilizada a AUC, *Area Under Curve*. Através desta área é possível obter uma medida da curva em relação ao quadrado do gráfico ROC. Logo, essa medida varia entre 0 e 1, que é a área total do quadrado. Devido ao fato de classificadores aleatórios produzirem seus pontos na diagonal do gráfico, um bom classificador deve ter sempre seu valor de AUC acima de $0,5$.

Um fator importante é que a AUC possui importantes propriedades estatísticas. A AUC é numericamente equivalente a probabilidade de uma nova instância ser classificada corretamente. Isto é equivalente ao teste estatístico de Wilcoxon [15]. A AUC é uma medida que vem sendo usada largamente para a comparação de métodos de Aprendizado de Máquina. Apesar de haver problemas em se utilizar um valor único para medir a qualidade de um classificador, a AUC se mostra a técnica com menos deficiências.

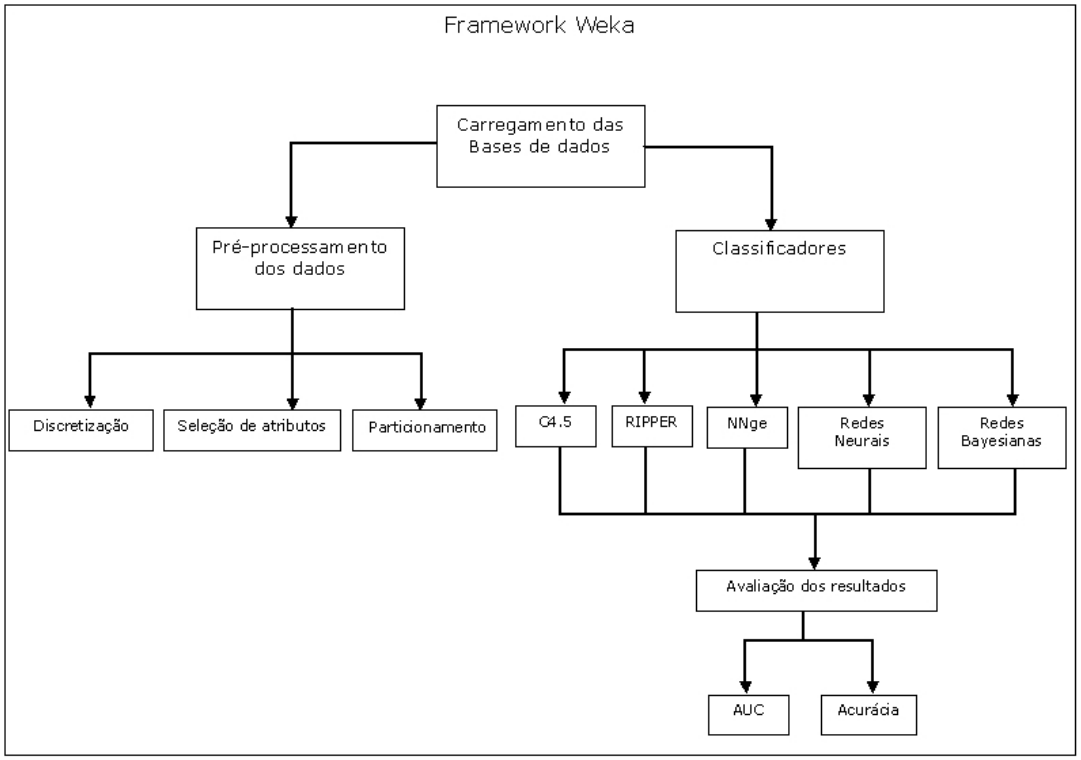

Figura 3.4: Principais funcionalidades do framework Weka utilizadas.

3.4 Framework de Aprendizado de Máquina

Esta seção apresenta detalhes do *framework* de Aprendizado de Máquina `Weka` [21]. Está ferramenta implementa diversos tópicos de Aprendizado de Máquina e de Aprendizado de Regras apresentados neste capítulo. Através dela é possível o acesso e alteração das bases de dados no formato de atributo-valor. Este *framework* disponibiliza um conjunto de classes que possibilitam o carregamento de bases de dado no formato `.ARFF`, padrão do Weka. Além disto, é existe um conjunto de métodos que possibilitam a utilização de filtros que podem modificar a base, tal como a discretização dos atributos ou o particionamento das bases de dados. O *framework* Weka também disponibiliza um grande quantidade de diferentes técnicas de Aprendizado de Máquina. É possível a execução destes métodos e obter várias medidas de avaliação de classificadores apresentadas nas Seções 3.3.3.1 e 3.3.4. A Figura 3.4 apresenta um diagrama com as principais funcionalidades do Weka utilizadas neste trabalho.

Na Figura 3.4 inicialmente é destacado o módulo de carregamento das bases de dados no formato atributo-valor. Com as bases de dados carregadas é possível a execução de

rotinas de pré-processamento dos dados como a discretização, a seleção de atributos e o particionamento das bases de dados. Além do pré-processamento dos dados é possível a execução de alguns métodos de AM como o algoritmo C4.5, as Redes Neurais, entre outros. Através dos classificadores obtidos é possível o cálculo de medidas, tais como a Acurácia e a AUC. Este *framework* é largamente utilizado neste trabalho na etapa de experimentos descrita no Capítulo 6.

CAPÍTULO 4

METAHEURÍSTICA NUVEM DE PARTÍCULAS MULTIOBJETIVO

O capítulo anterior apresentou os principais conceitos do Aprendizado de Máquina e da indução de regras. Porém, até então não foi apresentado como estes problemas são solucionados computacionalmente. Neste capítulo será apresentada uma solução para o problema da indução de regras.

Inicialmente será apresentado como a indução de regras é realizada computacionalmente através do mapeamento do problema para um problema de busca. Para o mapeamento do problema para um problema de busca inicialmente alguns aspectos devem ser definidos. Deve-se representar as soluções do problema através de estados. É necessária também a definição dos estados iniciais e finais do problema. Após a definição da representação do problema é necessária a definição dos operadores, os quais vão alterar as soluções de um estado para outro. Por fim é necessária a definição de uma Função Objetivo, que irá mensurar se o estado atual produz ou não um resultado esperado. Após o mapeamento para um problema de busca é possível a execução de um método de busca, o qual será o responsável pela busca da melhor solução [43]. A Seção 4.1 apresenta os principais aspectos dos problemas de busca e em seguida, nas Seções 4.2, 4.3 e 4.4, será apresentada a técnica utilizada para a solução construída neste projeto, a Otimização por Nuvem de Partículas Multiobjetivo [45].

4.1 Indução de regras como um Problema de Busca

Até aqui foram apresentadas as características básicas para o entendimento do Aprendizado de Regras. Foram discutidos os conceitos de aprendizagem de máquina, de inferência, porém ainda não foi descrito como estes problemas são resolvidos computacionalmente. A solução para construção de programas capazes de induzir regras a partir de um conjunto

de exemplos é através de um problema de busca.

De acordo com a definição do Aprendizado de Regras, o programa busca aprender um conceito a partir de um conjunto universo de exemplos. As regras induzidas são uma combinação de valores dos atributos da base mais o valor da classe. Assim, pode-se definir a indução de regras formalmente através de três componentes [43]:

Estado Inicial: O estado inicial é uma regra cujos valores podem ser definidos de forma aleatória, ou por alguma técnica de inicialização. A partir desta regra inicial, novos valores dos atributos vão ser combinados para obter bons valores do objetivo escolhido. O espaço de estados é representado pela combinação de todos os valores dos atributos mais os valores da classe.

Teste do Objetivo ou Função Objetivo: O teste do objetivo é executado para saber se as regras induzidas estão produzindo os resultados esperados. No problema de indução de regras um objetivo normalmente é definido através das medidas de avaliação apresentadas na seção anterior. Há a possibilidade da escolha de mais um objetivo por execução, neste caso o teste de objetivo deve levar em consideração a conjunção dos valores. Este trabalho utiliza uma técnica multiobjetivo e na Seção 4.4 será apresentado como a Nuvem de Partículas é aplicada com mais de um objetivo.

Função Sucessor ou Operadores: A função sucessor define quais ações devem ser executadas para a mudança de um estado para outro. Na indução de regras a mudança de estado pode ocorrer alterando os valores para os atributos, incluindo a classe. Neste sentido, pode-se executar diversas ações: trocar o valor de somente um atributo, dois atributos e assim por diante. Há a possibilidade também de retirar o atributo da regra, ou seja, definir um valor vazio para o atributo.

Com estas definições um algoritmo de busca recebe como entrada um ou mais estados iniciais e executa um conjunto de ações para se obter regras que atinjam os seus objetivos. O problema da indução de regras é um problema combinatório, logo diversas técnicas conhecidas da literatura podem ser aplicadas a ele.

Os sistemas tradicionais de Aprendizado de Regras costumam usar uma abordagem de cobertura [37] e algoritmos gulosos. Esta estratégia é um procedimento que é executado

iterativamente. Nesta busca, a cada iteração, o algoritmo encontra a melhor regra e remove todos os exemplos que são cobertos por esta regra. O processo é repetido com os exemplos remanescentes e continua até que todos os exemplos sejam cobertos ou algum critério de parada seja alcançado. No entanto esta abordagem possui diversos problemas. A remoção dos exemplos da base de dados a cada nova descoberta pode causa a super-especialização das regras, ou seja, regras que cobrem somente poucos exemplos. Além disso, um classificador composto por este conjunto de regras é uma lista ordenada onde a interpretação de uma regra depende das regras anteriores. Dentre este algoritmos pode-se destacar o algoritmo ROCCER [38], que possui bons resultado em termos de AUC, quando comparado a outros algoritmos da literatura.

Porém, existem alguns algoritmos que introduzem novas abordagens para a indução de regras. Elas são baseadas em `Metaheurísticas Multiobjetivo`, MHMO. Técnicas de MHMO permitem a concepção de novas estratégias onde as propriedades das regras podem ser expressas em diferentes objetivos e então algoritmos MHMO encontram as regras em uma única execução. Neste sentido, técnicas MHMO permitem a criação de classificadores compostos por regras com propriedades específicas que exploram os conceitos da dominância de Pareto [45, 25]. Estes conceitos serão discutidos na Seção 4.2. As regras geradas por técnicas de MHMO são interpretadas mais facilmente, pois elas podem ser analisadas independentes uma das outras. Além disso, a qualidade das regras induzidas não é afetada durante o processo de aprendizagem, pois a base de dados não é modificada. As próximas seções irão introduzir os conceitos da Otimização por Nuvem de Partículas e da Otimização por Nuvem de Partículas Multiobjetivo, uma técnica MHMO utilizada no desenvolvimento do algoritmo proposto neste projeto.

4.2 Otimização por Nuvem de Partículas

A `Otimização por Nuvem de Partículas`, PSO (do inglês *Particle Swarm Optimization*), desenvolvida por Kennedy e Eberhart [27], é uma metaheurística baseada em população inspirada no comportamento social de pássaros à procura de alimento. Uma técnica é considerada baseada em população caso ela use mais de uma solução inicial

na execução da busca. Nesse tipo de busca toda a população é composta por possíveis soluções para o problema e durante a busca todos os indivíduos da população são atualizados. Ao final da execução, a melhor ou as melhores soluções, de acordo com o teste do objetivo, são definidas como o resultado do problema.

Na Otimização por Nuvem de Partículas, o conjunto das possíveis soluções é um conjunto de partículas, chamada de enxame (*swarm*) ou população, que se movem no espaço de busca. Cada partícula representa uma possível solução e é representada como uma posição no espaço de estados. A idéia da PSO é executar um conjunto de operadores e movimentar cada partícula para regiões promissoras no espaço de busca. A cada iteração um novo conjunto de soluções é obtido. Estes movimentos são executados por um operador que ajusta a `velocidade` da partícula. Como a velocidade deve mover cada partícula para regiões promissoras, este cálculo baseia-se nas melhores soluções encontradas até então por toda a população. O cálculo da velocidade é baseado na melhor posição encontrada por uma vizinhança da partícula e pela melhor posição encontrada pela partícula. A seguir são descritos os principais termos utilizados na Otimização por Nuvem de Partículas:

- ***Swarm* ou Enxame**: População do algoritmo.
- **Partícula**: Indivíduo da população. Cada partícula representa uma solução para o problema que é representada pela posição da partícula.
- **Velocidade**: Operador que movimenta a partícula pelo espaço de estados em direção as melhores soluções do problema. É guiado através das melhores posições da partícula e as melhores posições já encontradas por alguma partícula do enxame.
- p_{best}: Melhor posição já alcançada pela partícula.
- l_{best}: Melhor posição já alcançada por uma partícula pertencente a vizinhança de uma determinada partícula.
- g_{best}: Melhor posição já alcançada por uma partícula em toda população.
- **Líderes**: Partículas da população que possuem os melhores valores da função objetivo para problema.

- **Peso de inércia (w)**: Usado para controlar a influência dos valores anteriores da velocidade no cálculo da velocidade atual.

- **Fator de aprendizado**: Representa a atração que uma partícula terá em direção ao seu próprio sucesso (p_{best}) ou ao sucesso do enxame(g_{best}). Representado por c_1, peso de atração em direção ao sucesso da partícula e c_2, peso de atração em direção ao sucesso dos vizinhos.

- **Topologia de Vizinhança**: Determina o conjunto de partículas que será usado como vizinhança de uma determinada partícula.

A próxima sessão irá discutir algumas topologias de vizinhança existentes. A Seção 4.2.2 apresenta as equações de movimento das partículas pelo espaço de busca.

4.2.1 Topologias de Vizinhança

Como apresentado anteriormente, na Otimização por Nuvem de Partículas um conjunto de possíveis soluções, representadas através de partículas, movimentam-se pelo espaço de busca. Esse movimento é influenciado pela própria história da partícula e por um conjunto de partículas, que é denominado de vizinhança. A forma como essa vizinhança está definida pode alterar como a busca é efetuada [42]. Esta seção irá apresentar as principais topologias de vizinhança utilizadas [42].

- **Grafo vazio**: Nesta vizinhança a partícula está conectada somente à ela mesma. Assim, não há a influência das demais partículas no seu movimento.

- **Melhor local**: Nesta vizinhança a partícula está conectada à k partículas. Assim, o líder é definido como sendo a melhor partícula da vizinhança k, l_{best}. Equivalente à topologia de anel se $k = 2$ (Figura 4.1) e equivalente ao grafo completo caso k seja igual ao número total de partículas. Esta topologia efetua poucas operações para o cálculo do movimento das partículas, porém esse movimento é influenciado por um número pequeno de partículas.

- **Estrela**: Nesta topologia todas as partículas estão conectadas somente à uma partícula, chamada de partícula focal (Figura 4.2). A partícula focal compara a performance de toda as partículas da população e efetua o seu movimento de acordo com a melhor partícula. As demais partículas se movimentam de acordo com a partícula focal, que propaga a escolha da melhor solução para as demais partículas. Esta topologia consegue produzir um movimento com a influência da melhor partícula da população sem executar muitas operações, porém a propagação da melhor posição depende da posição da partícula focal.

- **Árvore**: Todas as partículas estão organizadas em árvore (Figura 4.3). Cada partícula sofre influência de sua partícula pai na árvore, o seu líder é o seu pai na árvore. Se uma partícula em um nó filho encontrar uma melhor posição no espaço de estados que seu pai, é feita uma troca de posição entre essas duas partículas. Esta topologia propicia um acesso rápido às melhores soluções do problema, porém a estrutura dinâmica da árvore torna a busca mais complexa.

- **Grafo completo**: Nesta vizinhança a partícula está conectada à todas as partículas do enxame (Figura 4.4). g_{best} é definido como sendo a melhor partícula entre todas as partículas da população. Possibilita que todas as partículas se influenciem pela melhor partícula da população, mas executa o maior número de operações entre todas as topologias. A topologia do grafo completo foi utilizado no desenvolvimento do algoritmo apresentado no Capítulo 5.

4.2.2 Equações de Movimento das Partículas

A Otimização por Nuvem de Partículas é definida por um conjunto de equações que movimenta as partículas no espaço de estado em direção às melhores soluções do problema. Esta seção irá discutir as equações de movimento das partículas. Os elementos da Otimização por Nuvem de Partículas são definidos a seguir. Cada partícula p_i, em um determinado tempo t (o tempo normalmente é representado pela iteração da execução), possui uma posição $x(t) \in R^n$. Como exposto anteriormente, a posição da partícula re-

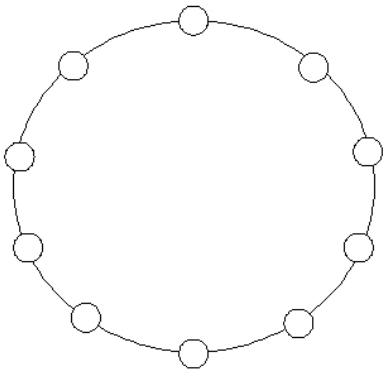

Figura 4.1: Topologia de vizinhança em forma de anel. Nesta topologia, cada partícula é influenciada somente por seus dois vizinhos imediatos. Cada círculo representa uma partícula.

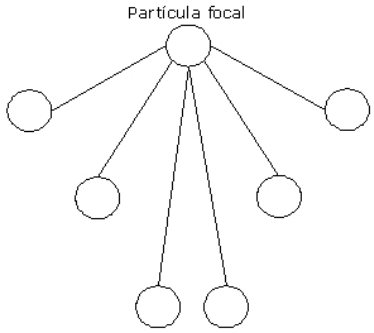

Figura 4.2: Topologia de vizinhança em forma de estrela. A partícula focal está conectada à todas partículas da população, enquanto as demais partículas só estão conectadas à partícula focal. Cada círculo representa uma partícula.

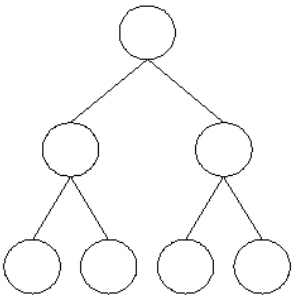

Figura 4.3: Topologia de vizinhança em forma de árvore. As partículas estão organizadas em forma de árvore e cada partícula é influenciada por seu nó pai da árvore. Cada círculo representa uma partícula.

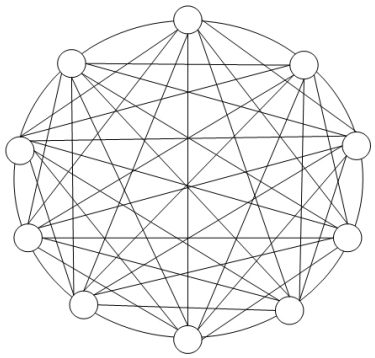

Figura 4.4: Topologia de vizinhança em forma de grafo completo. Nesta topologia todas as partículas estão conectadas e há influência entre todas partículas da população. Cada círculo representa uma partícula.

presenta uma possível solução no espaço de estados e a dimensão do vetor é definida pelo número de atributos da base de dados. A posição da partícula, num tempo $t+1$, é obtida adicionando-se a velocidade, $v(t) \in R^n$, a $x(t)$, tal que:

$$\overrightarrow{x}(t+1) = \overrightarrow{x}(t) + \overrightarrow{v}(t+1) \tag{4.1}$$

A velocidade da partícula p_i é baseada na melhor posição já alcançada pela partícula, $p_{best}(t)$, e pela melhor posição alcançada pelos vizinhos. Como apresentado anteriormente, a definição da melhor solução da vizinhança depende da escolha da topologia, que influência a escolha do líder p_{lider}. A função de atualização da velocidade, no tempo $t+1$, é definida:

$$\overrightarrow{v}(t+1) = \varpi * \overrightarrow{v}(t) + c_1 * \phi_1 * (\overrightarrow{p}_{best}(t) - \overrightarrow{x}(t)) + c_2 * \phi_2 * (\overrightarrow{p}_{lider}(t) - \overrightarrow{x}(t)) \tag{4.2}$$

Onde c_1, c_2, ϕ_1 and $\phi 2$, na Equação 4.2, são os coeficientes que determinam a influência de $p_{best}(t)$ e $p_{lider}(t)$. O coeficiente ϖ é a inércia da partícula e controla o quanto a velocidade anterior irá afetar a velocidade corrente. Após a atualização da velocidade e da posição de todas as partículas o processo é repetido na próxima geração (iteração) até o final da execução do algoritmo.

As melhores partículas da população são encontradas através da função de *fitness*, que

Algoritmo 1 Pseudo-Código da Otimização por Nuvem de Partículas

```
1: Iniciar população
2: Definição dos líderes
3: Enquanto condição de término não ocorrer
4:   Para cada partícula p_i da população faça
5:     Atualizar a posição
6:     Avaliar o fitness
7:     Atualizar p_best
8:   Atualizar o líder
9: Retorno: Melhor solução da população
```

é a função objetivo do problema. O *fitness* de uma partícula p_i, representado como $\alpha(p)$, é uma função em relação à posição da partícula, $\alpha : S \subseteq R^n \rightarrow R$. Então, para problemas de minimização, em um tempo t, p_i é melhor que p_j, se:

$$\alpha(\overrightarrow{x_i}(t)) < \alpha(\overrightarrow{x_j}(t))$$

O inverso é verdade para problemas de maximização. Na indução de regras uma função objetivo normalmente está relacionada com alguma medida de avaliação da regra. Assim quanto maior o valor da medida melhor é a partícula encontrada. Por exemplo, pode-se escolher como função objetivo a precisão, a sensitividade, etc.

O Algoritmo 1 mostra o pseudo-código da Otimização por Nuvem de Partículas. O primeiro passo é a inicialização da população. Esta inicialização pode ser um procedimento aleatório ou pode haver a utilização de diferentes estratégias que podem ser desde algoritmos gulosos ou a utilização de outras metaheurísticas. Após a inicialização da população há a escolha do líder, que é a partícula com maior valor de *fitness*. Após está etapa de configuração inicial o laço evolutivo é executado. Neste laço são executadas as operações descritas anteriormente. Ao final do laço, a partícula como melhor *fitness* é considerada a solução do algoritmo.

4.3 Otimização Multiobjetivo

Problemas de otimização que possuem mais de uma função objetivo são chamados de problema Multiobjetivo. Nestes problemas os objetivos estão normalmente em conflito, ou seja, quando um valor de objetivo é aumentado o valor outros objetivo é diminuído.

Por exemplo, nem sempre é possível obter as regras mais simples e mais precisas, em muitos casos regras mais complexas, com maiores restrições, possuem um maior valor na precisão. Assim, normalmente não há somente uma melhor solução para o problema e busca-se encontrar as melhores soluções que representam o melhor compromisso entre os objetivos [42]

Para a solução de problemas multiobjetivo três abordagens podem ser utilizadas [19]. A primeira abordagem é a redução do conjunto de objetivos em um problema mono-objetivo, através do uso de uma soma ponderada dos objetivos. Nesta solução é construída uma fórmula onde são atribuídos pesos para cada objetivo e em seguida é feita a soma destes valores. Desta forma, os n possíveis valores dos objetivos são reduzidos a somente um. Apesar de esta solução ser bastante simples, ela possui dois grandes problemas. O primeiro é a necessidade de se encontrar os melhores valores possíveis para o peso de forma empírica. A segunda é o fato de o modelo criado ficar especializado somente para os valores definidos pelos pesos, perdendo a oportunidade de encontrar uma melhor configuração que pode ser definida por um outro conjunto de pesos. Além deste dois problemas citados há ainda outros problemas na escolha desta abordagem. A segunda abordagem para a solução de problema multiobjetivos é a `lexicográfica`. Nesta solução são definidas prioridades para os diferentes objetivos e então há o foco no objetivo de acordo com esta prioridade. Assim, na comparação entre duas soluções, primeiro é feita a comparação com o objetivo com maior prioridade, caso não haja uma diferença significante para este valor é feita a análise com o próximo objetivo, e assim por diante. A vantagem deste método é que os objetivos são analisados separadamente e não há uma mistura destes valores durante a comparação das soluções. O problema desta abordagem é que ela introduz novos parâmetros para o problema: a definição da ordem das soluções e a definição de um valor limite que defina se uma solução é melhor que a outra.

A terceira abordagem para a solução de problemas é a definição do conceito de dominância de Pareto [35]. Esta solução é utilizada neste projeto. Nesta solução o objetivo é encontrar a `Fronteira de Pareto`, que contém o conjunto das melhores soluções encontradas pelo algoritmo. O problema multiobjetivo geral, sem restrições, pode ser definido

através da Equação 4.3.

$$f(x) = (f_1(x), ..., f_Q(x)) \tag{4.3}$$

$x \in \Phi$, onde: x é um vetor de variáveis de decisão e Φ é um conjunto finito de soluções aceitáveis. A função $f(x)$ mapeia um conjunto de soluções aceitáveis $X \in \Phi$ para um espaço de objetivos com Q dimensões, sendo $Q > 1$ o número de objetivos. Então, $f : X \to Z$ é uma função que atribui um vetor de objetivos $z = f(x) \in Z$ para cada solução $x \in \Phi$. Sejam $z^1 = (z_1^1, ..., z_Q^1)$ e $z^2 = (z_1^2, ..., z_Q^2), z^1, z^2 \in Z$ dois vetores objetivos. Algumas definições de dominância são:

- $z^1 \succ z^2$ (z^1 domina z^2) se z^1 não é pior que z^2 para nenhum objetivo e é melhor em pelo menos um objetivo.
- $z^1 \succ\succ z^2$ (z^1 domina estritamente z^2) se z^1 é melhor que z^2 para todos os objetivos
- $z^1 \geq z^2$ (z^1 domina fracamente z^2) se z^1 não é pior que z^2 para nenhum objetivo
- $z^1 \mid\mid z^2$ (z^1 e z^2 são incomparáveis um em relação ao outro) se nem z^1 domina z^2 nem z^2 domina z^1
- $z^1 \sim z^2$ (z^1 e z^2 são indiferentes) se z^1 e z^2 são iguais para todos os objetivos.

O objetivo é descobrir soluções que não são dominadas por nenhuma outra no espaço de objetivos. O conjunto de vetores objetivos não dominados é chamado de `Pareto Ótimo` e o conjunto de todos os vetores não-dominados é chamado de `Fronteira de Pareto`. O Pareto Ótimo é útil para problemas reais, por exemplo, problemas da engenharia, e provê informações valiosas sobre o problema [28]. Na maioria das aplicações, a busca pelo Pareto Ótimo é NP-difícil [28], então o problema de otimização foca em encontrar o conjunto mais próximo possível do Pareto Ótimo, um conjunto de aproximação.

Seja $A \subseteq Z$ um conjunto de vetores objetivos. A é dito um conjunto de aproximação se algum elemento de A é incomparável com outro. A relação de dominância pode ser estendida para conjuntos de aproximação. Dados dois conjuntos de aproximação A_1 e

A_2, A_1 domina A_2 ($A1 \succ A2$) se toda solução do vetor A_2 é dominada por pelo menos um vetor presente em A_1. Para a comparação de algoritmos, é útil definir a relação *"é melhor que"*. É dito que a um conjunto de aproximação A_1 *é melhor que* A_2, ($A_1 \triangleright A_2$), se $A_1 \geq A_2$ e $A_1 \neq A_2$. A_1 e A_2 são ditos incomparáveis ($A_1 || A_2$) se nem $A_1 \geq A_2$ e nem $A_2 \geq A_1$. Estas definições são úteis para comparar a saída de algoritmos de aproximação.

4.3.1 Medição de Performance de Algoritmos Multiobjetivos

Esta seção apresenta a metodologia utilizada para a medição de performance de algoritmos multiobjetivos. São descritos os principais passos para a medição da qualidade deste algoritmos, utilizando-se uma metodologia multiobjetivo. As direções gerais da metodologia de mediação de performance utilizados neste trabalho foram obtidas através do estudo apresentado em [28].

Como um primeiro passo na comparação, quaisquer diferenças significantes entre os algoritmos devem ser consideradas usando a abordagem do ranking de dominância. Esta abordagem possibilita uma afirmação mais forte das diferenças. Após esta comparação, indicadores de qualidade podem ser aplicados para quantificar possíveis diferenças na qualidade dos algoritmos e para detectar diferenças que não são reveladas pelo ranking de dominância. Por fim, a comparação estatística das funções dos indicadores utilizados. Estas abordagens são explicadas a seguir.

O ranking de dominância é um método de medição geral e independente que é baseado na comparação pareada de conjuntos de aproximação. O ranking de dominância pode ser computado usando-se o indicador binário epsilon aditivo [51] e o teste estático de Mann-Whitney [12]. Dados dois algoritmos A e B, o indicador binário epsilon, $I_\xi(A, B)$, obtém o menor fator o qual cada ponto de B pode ser adicionado em uma maneira que o resultado é dominado fracamente por A. Dados dois conjuntos de execuções de algoritmos estocásticos (conjuntos de aproximação), $A = \{A_1, ..., A_k\}$ e $B = \{B_1, ..., B_j\}$, um ranking é dado para cada conjunto de aproximação $Ci \in C$, $C = A \cup B$, sendo o número de conjuntos de aproximação dominados fracamente por C_i mais um, Equação (4.4). Assim, quanto menor é o rank que um conjunto de aproximação C_i recebe, melhor ele é. Cada conjunto

de aproximação é representado pelos valores dos objetivos para cada solução encontrada.

$$rank(C_i) = 1 + |\{C_j \in C : C_j \triangleright C_i\}| \tag{4.4}$$

A Equação 4.4 prove um valor inteiro que é atribuído a cada conjunto de aproximação. Então, um teste estatístico pode ser executado para estabelecer as diferenças estatísticas que existem entre os conjuntos A e B. Em particular, busca-se definir se os rankings atribuídos aos conjuntos de aproximação de um algoritmo são significantemente menores que os rankings associados ao outro algoritmo. O teste de Mann-Withney, Teste-U, também chamado de teste de Mann-Whitney-Wilcoxon é um teste não paramétrico usado para verificar a hipótese nula que dois exemplos fazer parte de uma mesma população [13]. Se, no entanto, a aplicação do Teste-U não apresentar diferença significante entre os dois conjuntos analisados, novos experimentos devem ser executados.

Indicadores de qualidade representam meios de expressar e medir qualidade entre diferentes conjuntos de aproximação, baseados em informações adicionais extraídos destes conjuntos. Neste trabalho, indicadores unários são usados. Indicadores unários são funções, $I(X)$, que mapeiam um conjunto de aproximação, X, em valores reais. Seja um par de conjuntos de aproximação A e B. As diferenças entres os valores da aplicação do indicador $I(A)$ e $I(B)$ revela a diferença de qualidade entre os dois conjuntos. Os indicadores de qualidade unários hipervolume [51], epsilon [50] e $R2$ [23] são os mais usados.

O indicador hipervolume, $I_H(A)$, mede o hipervolume da porção do espaço de objetivos que é dominada fracamente pelo conjunto de aproximação A [51]. Para este indicador ser utilizado, o espaço de objetivos deve ser limitado. Caso contrário, um ponto de referência R (que domina todos os pontos) deve ser definido. O hipervolume da diferença para o ponto de referência R pode ser considerado, sendo que o conjunto de aproximação que obtiver menor valor do indicador apresenta o melhor resultado. O indicador hipervolume é o único indicador que é capaz de detectar se um conjunto de aproximação A não é melhor que outro conjunto de aproximação B para todos os pares de solução [28].

O indicador unário epsilon aditivo $I^1_\epsilon(A)$ provê o fator mínimo ϵ o qual um ponto no

conjunto de referência R pode ser adicionado tal que o conjunto de aproximação transformado é dominado fracamente por A [50], ou seja, o operador mede o fator mínimo que se deve alterar um conjunto de referência até que ele seja dominado por A. Um conjunto de aproximação A é preferível a outro conjunto B, de acordo com o indicador epsilon aditivo, se $I^1_\epsilon(A) < I^1_\epsilon(B)$. Se o hipervolume e o epsilon aditivo indicam preferências opostas para dois conjuntos de aproximação então eles são incomparáveis.

O indicador $R2$, I_{R2} [23], é baseado em um conjunto de funções de utilidade. Uma função de utilidade é um mapeamento de um conjunto de Z dos Q-dimensionais vetores objetivos para um conjunto de números reais.

A Equação 4.5 apresenta a função aumentada de Tchebycheff que foi utilizada no operador R2 neste trabalho. Nesta equação z^*_j é um ponto que é dominado fracamente pelo ponto z_j, $\xi = 0.01$ e $\lambda_j \in \Delta$, o conjunto de pesos contendo $rank(C_i)$(Equação 4.4).

$$u_\lambda = -\left(\max_{j\in 1..q} \lambda_j \left|z^*_j - z_j\right| + \xi \sum_{j=1}^{n} \left|z^*_j - z_j\right|\right) \tag{4.5}$$

O mesmo conjunto de referência é utilizado para todos os indicadores. O conjunto de referência é formado pelos vetores não-dominados de todos os conjuntos de aproximação formados pelos algoritmos em análise. O teste estatístico Kruskal-Wallis [12] é usado para comparar os algoritmos baseados nestes três indicadores de qualidade. O teste de Kruskal-Wallis é uma extensão lógica do Teste-U. Ele é também um teste não-paramétrico usado para comparar três ou mais exemplos testando a hipótese nula que todas as populações possuem distribuições idênticas.

Todos as técnicas de medição de performance utilizadas neste trabalho foram aplicadas através do framework de mediação de performance multiobjetivo PISA PA [4]. No capítulo 6 serão apresentados os detalhes da execução do framework.

4.4 Nuvem de Partículas Multiobjetivo

Na Otimização por Nuvem de Partícula Multiobjetivo (MOPSO) existem várias funções objetivos. Neste sentido, é possível encontrar soluções que explorem os conceitos da do-

minância de Pareto. Baseado neste conceito cada partícula do enxame pode ter diferentes líderes (componente global), mas para aplicação do operador de velocidade somente um deve ser escolhido. Este conjunto de líderes é guardado num repositório, que contém as soluções não-dominadas encontradas. Os componentes do MOPSO são definidos a seguir.

Cada partícula p_i, num tempo t, possui uma posição $x(t) \in R^n$, que representa uma possível solução. A posição da partícula, num tempo $t+1$, é obtida através da Equação 4.1. A velocidade da partícula p_i é baseada na melhor posição alcançada pela partícula até então, $\overrightarrow{p}_{best}(t)$, e a melhor posição encontrada pelos vizinhos de p_i, $\overrightarrow{R_h}(t)$, que é o líder escolhido do repositório. $\overrightarrow{R}_h$ é a posição da partícula do repositório, escolhida como guia global de p_i. Existem várias maneiras de se executar esta escolha, como demonstrado em [42]. Uma maneira possível de realizar a escolha do líder é chamada de `método sigma` [33]. Este método, de acordo com os resultados apresentados em [42], é um dos mais adequados para a técnica MOPSO. O método tenta aumentar a convergência e a diversidade da abordagem MOPSO, logo ele tenta produzir boas e diversificadas soluções num número menor de iterações. No entanto está técnica pode causar a convergência prematura em alguns casos. Este método apresentou bons resultados e isto motivou a escolha desta técnica neste trabalho. A função de atualização da velocidade no tempo $t+1$ é definida por:

$$\begin{aligned}\overrightarrow{v}(t+1) = \varpi * \overrightarrow{v}(t) + (c_1 * \phi_1) * (\overrightarrow{p}_{best}(t) - \overrightarrow{x}(t)) \\ +(c_2 * \phi_2) * (\overrightarrow{R_h}(t) - \overrightarrow{x}(t))\end{aligned} \tag{4.6}$$

As variáveis ϕ_1 e ϕ_2, na Equação 4.6, são os componentes aleatórios do movimento. As constantes c_1 e c_2 indicam o quanto o componente local ou global vai influenciar a velocidade. O coeficiente ϖ é a inércia da partícula e controla o quanto a velocidade anterior afeta a velocidade atual. No final da execução do algoritmo, as soluções presentes no repositório são a solução final.

Para um problema com dois objetivos, o valor sigma de uma partícula é calculado da seguinte maneira:

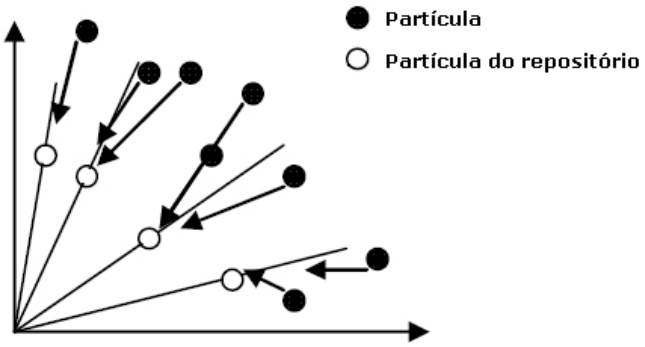

Figura 4.5: Representação do Método Sigma para um espaço com dois objetivos.

$$\sigma = \frac{f_1(x)^2 - f_2(x)^2}{f_1(x)^2 + f_2(x)^2} \tag{4.7}$$

Neste método para cada partícula do enxame, representada como um ponto no espaço de objetivos, é atribuído um valor σ_i. O líder para uma partícula, R_h, é a partícula do repositório que possui a menor distância Euclidiana entre o seu vetor sigma e o vetor sigma da partícula do enxame. A Figura 4.5 mostra um exemplo do método sigma para um estado com dois objetivos. Os círculos escuros representam as partículas da população e os círculos claros as partículas do repositório. As retas indicam o valor sigma das partículas do repositório e cada partícula da população escolhe a partícula do repositório que possui menor distância Euclidiana, em relação ao seu valor sigma.

CAPÍTULO 5

APRENDIZADO DE REGRAS COM A OTIMIZAÇÃO POR NUVEM DE PARTÍCULAS MULTIOBJETIVO

Este capítulo apresenta algoritmo de Aprendizado de Regras Baseado na Otimização por Nuvem de Partículas Multiobjetivo desenvolvido neste projeto. O objetivo inicial é aplicar a técnica MOPSO no Aprendizado de Regras. O algoritmo foi inicialmente desenvolvido baseando-se nos conceitos apresentados em [46] discutidos no Capítulo 2 deste trabalho. Além do desenvolvimento deste algoritmo, são propostas algumas extensões para solucionar problemas que ocorrem na aplicação da técnica MOPSO no AR. Os detalhes do desenvolvimento desta primeira etapa são discutidos na Seção 5.1. Dentre os problemas que surgem durante a extração das regras, este trabalho se concentrou na busca por soluções dos seguintes:

- **Melhora da expressão das regras e melhora na classificação**: Como apresentado anteriormente, o Aprendizado de Regras visa a construção de um conjunto de regras que representem o conhecimento extraído a partir de uma base de dados rotulada. Além disso, este conjunto de regras é utilizado como um classificador. Neste sentido, inicialmente foi adicionada ao algoritmo a possibilidade de se trabalhar com bases de dados com atributos tanto numéricos quanto nominais, visando produzir regras que representem melhor o domínio e produzir um melhor classificador. Esta contribuição foi apresentada nos trabalhos [7], [9] e [8]. Foi implementado também o operador *IN*, que possibilita que atributos nominais possam ter mais de um valor. Por fim, para obter melhores resultados na classificação, alguns esquemas de votação, de acordo com o esquema geral apresentado na Seção 3.3.3, foram implementados.

- **Redução do número de regras sem perda de expressão:** No Aprendizado de Regras, tem-se como objetivo produzir regras simbólicas, ou seja, que possam ser

inteligíveis por humanos. Assim, é interessante que um algoritmo de Aprendizado de Regras não gere um conjunto muito grande de regras, pois assim a inteligibilidade pode ficar comprometida. Assim, para atenuar este problema, um procedimento que retira do modelo regras específicas, sem que haja perda de expressão e sem que haja perda no poder de classificação foi desenvolvido. Este método foi descrito em [6]

- **Execução do algoritmo em base de dados de grande porte:** Um dos problemas do aprendizado de regra na Mineração de Dados é a complexidade da extração das regras, devido à enorme quantidade de dados. Neste sentido, quando se utilizam bases de dados muito grandes, pode haver uma demora muito grande para se obter o resultado do algoritmo. Assim, é apresentada uma proposta de paralelização do algoritmo. Através da paralelização, o algoritmo se torna hábil para executar em diversos processadores, reduzindo-se assim o tempo total de execução. O algoritmo proposto visa diminuir o tempo de execução sem que haja perda da qualidade na soluções geradas. A abordagem paralela foi apresentada em [6]

Todos os algoritmos e suas extensões foram desenvolvidos na linguagem de programação Java. A Seção 5.1 discute os principais aspectos da implementação da técnica da Nuvem de Partículas tais como a representação e a geração das partículas, bem como apresenta um exemplo da execução do algoritmo. Em seguida, na Seção 5.2 são apresentadas as mudanças no algoritmo para trabalhar com dados numéricos. A Seção 5.3 define o operador *IN* e a Seção 5.4 apresenta os esquemas de votação implementados. O procedimento que reduz o número de regras é descrito na Seção 5.5. Por fim, a Seção 5.6 apresenta os principais aspectos da proposta de paralelização do algoritmo. No Capítulo 6 são discutidos experimentos para avaliar cada solução proposta neste capítulo.

5.1 Algoritmo de Aprendizado de Regras MOPSO

Esta sessão apresenta o algoritmo básico de aprendizado de regra proposto, chamado de MOPSO-D. Como mostrado anteriormente, o algoritmo tem como objetivos a maximização do desempenho da classificação e produzir regras a partir de bases de dados

rotuladas. O algoritmo visa construir boas soluções utilizando mais de uma função objetivo. Neste trabalho os objetivos escolhidos para a execução do algoritmo são a sensitividade (3.5) e especificidade (3.6). Este objetivos foram escolhidos, pois estão diretamente relacionados ao cálculo da AUC, principal medida em estudo neste trabalho.

Nesta seção, os principais aspectos do algoritmo MOPSO-D, tais como a representação e geração das partículas são descritos em detalhes e um exemplo de suas funcionalidades é apresentado. Inicialmente será feita uma descrição geral do funcionamento do algoritmo, após cada seção discutirá algum aspecto da busca.

O algoritmo usa a abordagem de Michigan, onde cada partícula representa uma única solução, ou seja, uma regra. Neste contexto, a partícula é um vetor n-dimensional de números reais e contém as restrições para cada atributo. O algoritmo de Aprendizado de Regras funciona da seguinte maneira (Ver Algoritmo 2). Primeiro, um procedimento de inicialização é executado onde a posição de cada partícula é iniciada e todas as partículas são espalhadas aleatoriamente pelo espaço de busca. Neste procedimento todos os demais componentes da partícula, tais como velocidade e melhor local, também são iniciados. O próximo passo é a avaliação das partículas de acordo com os objetivos da busca. Após a avaliação das partículas, o repositório com as melhores soluções é iniciado com as soluções não-dominadas. A partir disso, o espaço dos objetivos é dividido de acordo com o método Sigma [33] e feita a escolha dos melhores globais para cada partícula. O próximo passo na execução do algoritmo é o laço evolutivo, onde as operações discutidas na Seção 4.4 são executadas, todas as partículas são avaliadas e o repositório com as soluções não-dominadas é atualizado com as melhores soluções obtidas após a movimentação das partículas. Por fim, após a execução do laço evolutivo, as partículas presentes no repositório são as soluções obtidas pelo algoritmo.

5.1.1 Representação da Posição das Partículas

Na abordagem proposta, a posição de cada partícula é representada através de um vetor n-dimensional de números reais. Este vetor contém as restrições dos atributos para a regra aprendida. Para este algoritmo cada restrição de atributo deve conter somente um valor e

Algoritmo 2 Algoritmo de Aprendizado de Regras MOPSO.

1: Procedimento de inicialização.
2: Avaliação das partículas de acordo com os objetivos.
3: Encontrar as soluções não-dominadas e escolher líderes.
4: Laço evolucionário
5: Calcular nova velocidade e nova posição de cada partícula do enxame.
6: Avaliar cada partícula.
7: Atualizar repositório das soluções não-dominadas e escolher novos líderes.
8: Retornar partículas do repositório.

o teste de atributo utilizado é o teste por valor, ou seja, a regra verifica se um determinado valor de um exemplo é igual ao seu valor. Neste vetor, um número real representa o valor para cada atributo nominal. Cada atributo pode aceitar o valor vazio "?", que significa que o atributo não apresenta restrições para a regra. Na abordagem proposta o valor da classe não faz parte da partícula, sendo definido através de um parâmetro no inicio da execução.

Por exemplo, considere uma base de dados com atributos: Tempo, Temperatura, Umidade, Vento, PraticarEsportes (classe: sim ou não). Uma possível regra e sua representação em partícula:

Regra: Se (Tempo=sol) E (Temperatura=(menor que 70)) E (Vento=sim) ENTÃO (PraticarEsportes = sim)

Partícula: <sol; menor que 70; ?; sim>

Para representar a partícula como uma possível solução do problema, os valores dos atributos devem ser codificados em números reais. A codificação dos atributos nominais é obtida através de um número real relacionado para cada valor do atributo na base de dados. Normalmente é atribuída uma codificação incremental. Para a partícula apresentada acima, uma possível codificação pode ser: "sol"= 1 (atributo Tempo), "menor que 70"= 2 e sim = 1 (atributo Vento). O valor vazio é representado com o valor 0. Assim, o exemplo apresentado acima pode ser codificado como: $< 1; 2; 0; 1 >$.

5.1.2 Procedimento de Inicialização

O procedimento de inicialização espalha aleatoriamente as partículas pelo espaço de busca e inicializa todos os componentes da partícula. Este procedimento deve distribuir as partículas de forma diversificada pelo espaço de busca. Neste processo, os atributos nominais são definidos através de um procedimento de roleta [20], onde os valores mais freqüentes da base de dados possuem maior probabilidade. A probabilidade de o valor vazio ser escolhido é definida em função do número de possíveis valores para o atributo. Assim, quanto maior o número de valores, menor é a probabilidade de um valor vazio ser escolhido pela roleta.

Após a inicialização da posição, a velocidade é iniciada aleatoriamente e o líder local, p_{best}, é definido como a posição inicial. Na atualização do líder local ele é definido através dos conceitos de dominância de Pareto, mas, se não há dominância entre duas soluções ele é definido aleatoriamente entre as duas. Como exemplo da inicialização do algoritmo, suponha uma nova partícula com a posição:

> Regra: Se (Tempo=sol) E (Temperatura=(menor que 70)) E (Umidade = (menor que 68)) ENTÃO (PraticarEsportes = sim)
>
> Partícula: $< 1; 2; 1; 0 >$

No início do algoritmo, o melhor local é definido como o valor posição inicial e a velocidade é definida de forma aleatória:

> Melhor Local: $< 1; 2; 1; 0 >$
>
> Velocidade: $< 3; 1; 4; 0 >$

Após a inicialização, todas as partículas são avaliadas de acordo com os objetivos. Suponha que para a partícula descrita acima os objetivos sejam:

> Sensitividade $= 0,55$
>
> Especificidade $= 0,1$

O próximo passo é a seleção das partículas não-dominadas. As melhores partículas de acordo com os conceitos de dominância de Pareto são guardadas no repositório e o melhor local de cada partícula pode ser escolhido através das partículas do repositório. Para este exemplo, o líder escolhido é:

Líder: $< 0; 1; 0; 2 >$

Ao final deste procedimento, o laço evolutivo está pronto para ser executado.

5.1.3 Laço Evolutivo

Uma vez realizada a configuração inicial, o laço evolutivo é executado até que um critério de parada seja alcançado. Neste trabalho o critério de parada utilizado é um número máximo de gerações. Em cada iteração as operações discutidas na sessão anterior são implementadas. A velocidade de cada partícula é atualizada (4.6) e então, as novas posições (4.1) são calculadas. No processo de atualização da posição da partícula, um operador de `mod` é aplicado. Este operador é usado para limitar a partícula dentro do espaço de busca. O operador de `mod` foi escolhido para promover igual probabilidade de seleção para cada valor dos atributos. Para os atributos nominais, os valores são limitados no total de valores para cada atributo.

Após o cálculo da nova posição a partícula é avaliada de acordo com os objetivos da busca. Após a avaliação dos objetivos é feita a atualização do melhor local da partícula. Ao final da iteração, após a atualização da posição de todas as partículas, as soluções não-dominadas atualizam o repositório. Por fim, os novos líderes globais são obtidos e uma nova iteração é executada. Como exemplo deste procedimento, considere a partícula iniciada na seção anterior. A nova velocidade é calculada de acordo com (4.6) e considerando os coeficientes das partículas como:

$\omega = 0,27,\ \phi_1 = 0,55,\ \phi_2 = 0,66,\ c_1 = c_2 = 2,05$

$v = <0,81;\ 0,27;\ 1,08;\ 0 > + < 0;\ 0;\ 0;\ 0 > + < 1,35;\ 1,35;\ 1,35;\ 2,70>$

$v = < 2,16;\ 1,62;\ 2,43;\ 2,70 >$

Esta velocidade é truncada e o novo valor é:

$$v = < 2; 2; 2; 3 >$$

O próximo passo é a adição da nova velocidade à posição atual da partícula:

$$x = < 1; 2; 1; 0 > + < 2; 2; 2; 3 >$$

$$x = < 3; 4; 3; 3 >$$

O operador de `mod` é então aplicado para restringir a partícula dentro do espaço da busca. Neste exemplo, o atributo Temperatura possui somente 3 possíveis valores, logo o valor do atributo na partícula excede o valor máximo do atributo. Aplicando-se o operador `mod`, o valor final da posição da partícula nessa iteração é:

$$x = < 3; 1; 3; 3 >$$

Este procedimento é repetido para todas as partículas até o critério de parada ser alcançado. Ao final do laço evolutivo, as regras aprendidas pelo algoritmo MOPSO são as soluções não dominadas presentes no repositório.

5.2 Trabalhando com Atributos Numéricos e Nominais

O algoritmo que foi apresentado na seção anterior somente trabalha com dados nominais e para ser utilizado em um domínio com atributos reais, uma etapa prévia de discretização dos dados é necessária. A discretização é um processo não trivial que muitas vezes leva à perda de informação. Assim, para evitar esta perda de informação, gerar um conjunto de regras mais fiel aos dados originais e produzir um melhor classificador foi desenvolvida uma extensão que permite que o algoritmo trabalhe com dados numéricos. O novo algoritmo desenvolvido, chamado de MOPSO-N, possui uma característica importante que é a habilidade de trabalhar tanto com atributos reais quanto atributos nominais. Para cada atributo real da base de dados, o MOPSO-N aprende um intervalo de valores apropriado para um determinado padrão. A construção dos intervalos de valores ocorre durante o

procedimento de busca, ao contrário da discretização que é um procedimento isolado que ocorre antes da execução do algoritmo. Como estes intervalos são definidos durante a execução do algoritmo, eles são construídos visando a maximização dos objetivos escolhidos.

O MOPSO-N possui os mesmos passos do Algoritmo 2, porém possui diferenças na definição do formato da partícula e nas operações de movimento. Como apresentado anteriormente, a posição de cada partícula é representada através de um vetor n-dimensional de números reais. Este vetor contém as restrições dos atributos para a regra aprendida. Agora, este vetor deve conter informações tanto de atributos nominais quando numéricos. Os atributos nominais continuam com o mesmo formato anteriormente, um número real que representa um valor para o atributo. Os atributos numéricos são representados por dois números reais em duas células do vetor. Estes números representam um intervalo de valores para os atributos numéricos. O intervalo é definido por um limite inferior e um limite superior. Cada atributo pode aceitar o valor vazio "?", que significa que o atributo não apresenta restrições para a regra.

Como exemplo, considere uma base de dados com atributos apresentada na Tabela 3.2: Tempo (nominal), Temperatura (numérico), Umidade (numérico), Vento (nominal), PraticarEsportes (classe: sim ou não). Uma regra e sua representação em partícula com atributos numéricos é a seguinte:

Regra: Se (Tempo=chuvoso) E ($67 < \text{Temperatura} < 85$) E (Vento=sim) ENTÃO (PraticarEsportes = sim)

Partícula: < sol; 67; 85; ?; ?; sim >

Para os atributos nominais o mesmo procedimento de codificação é efetuado. Os atributos numéricos não necessitam de codificação, pois já possuem valores reais. O valor vazio para um atributo numérico recebe como o valor o limite mínimo do atributo presente da base de dados. Assim, se este atributo for novamente utilizado pela regra os novos limites começam a partir do valor mais baixo da base de dados para o atributo em questão. Novamente, na abordagem proposta o valor da classe não faz parte da partícula, sendo

definido através de um parâmetro no inicio da execução. Assim, o exemplo apresentado acima pode ser codificado como: $< 1; 67; 85; 65; 65; 1 >$, onde o limite inferior para o atributo Umidade é 65.

No procedimento de inicialização as características de espalhar as partículas de forma aleatória e diversificada são mantidas. É aplicado o mesmo procedimento para os atributos nominais. Para os atributos numéricos, inicialmente, é executado um sorteio para definir se o atributo será vazio ou não. Para este sorteio é definida uma probabilidade, obtida através de um parâmetro, chamado de $prob_{vazio}$. Se um atributo é definido como não vazio, os limites, inferior e superior, são espalhados aleatoriamente pelo intervalo definido pelo valor mínimo e máximo do atributo, obtidos através da base de dados. Na abordagem proposta, o valor de $prob_{vazio}$ foi definido com um valor baixo para favorecer a criação de regras com diferentes intervalos iniciais e assim explorar um maior conjunto de intervalos do espaço de estados. Um exemplo da inicialização da partícula é apresentado a seguir:

Posição (x): <1; 64,0; 64,0; 67,86; 88,61; 0>

O procedimento de escolha do melhor local e a definição da velocidade são os mesmos do algoritmo anterior: melhor local igual a posição inicial e velocidade aleatória. Em seguida ocorre a escolha do novo líder.

Melhor Local ($\overrightarrow{p}_{best}$): <1; 64,0; 64,0; 67,86; 88,61; 0>

Velocidade (v): <3; 74,42; 75,45; 79,05; 93,46; 0>

Líder: <0; 64,0; 64,0; 66,55; 88,29; 2>

Na execução das equações de velocidade e da posição são efetuados os mesmos passos do algoritmo apresentado anteriormente. Porém, um novo operador de `mod` é proposto para limitar os atributos numérico dentro do espaço de busca. Ele é executado usando os valores máximos e mínimos para cada atributo, obtido através da base de dados. Se um novo valor de limite excede o valor máximo, o excesso é adicionado ao valor mínimo e este é um novo limite. O mesmo procedimento é executado para o limite inferior. Após este

processo, o menor valor é definido como limite inferior e o maior como o limite superior do atributo. Se ambos os valores excederem os limites, o atributo é definido como vazio. Um exemplo da aplicação do operador é:

Seja o valor da posição de uma partícula após a execução do movimento:

x = <1; 83,99; 84,27; 88,22; 113,51; 1>

O operador de `mod` é então aplicado para restringir a partícula dentro do espaço da busca. O valor $113,51$ excede o valor máximo do atributo, que neste exemplo é de 96. Assim o operador adiciona o excesso, $113,51 - 96 = 17,51$, ao valor mínimo do atributo, que é 65. O novo valor é $82,51$. Este valor é menor que o valor mínimo definido anteriormente. Neste caso os limites são trocados de posição, e o valor final da posição da partícula nessa iteração é:

x = <1; 83,99; 84,27; 82,51; 88,22; 1>

5.3 Operador *IN*

O operador *IN* tem com objetivo aumentar a expressividade de regras que contenham atributos nominais. O objetivo do operador é possibilitar que uma regra possa ter mais de um valor para as restrições dos atributos nominais. O operador *IN* é equivalente ao teste de atributo por subconjunto apresentado na Seção 3.3. Com a adição deste operador, restrições para atributos nominais passam a guardar um conjunto maior de informações, desta maneira, uma regra pode expressar informações que originalmente eram definidas por um conjunto de regras. No exemplo abaixo, as Regras 1 e 2 podem ser condensadas somente na Regra 3 com a aplicação do operador.

Regra 1: Se (Tempo=chuvoso) E (Vento=sim) ENTÃO (PraticarEsportes = nao)

Regra 2: Se (Tempo=nublado) E (Vento=sim) ENTÃO (PraticarEsportes = nao)

Regra 3: Se (Tempo= chuvoso ou nublado) E (Vento=sim) ENTÃO (PraticarEsportes = nao)

A idéia para a implementação do operador é alterar a codificação dos atributos nominais na representação das partículas. Além disso, o teste de atributo agora deve verificar se um determinado valor pertence a um subconjunto e não somente se é igual a um valor. No algoritmo inicial cada atributo nominal era representado por um valor no vetor posição da partícula e cada valor para o atributo era codificado como um número incremental. Porém, com a introdução das restrições com subconjunto esta codificação deve guardar informações sobre a combinação de valores. As demais etapas do algoritmo continuam a mesma da versão com somente um valor por atributo.

Como exemplo da combinação de valores, utilizando a base de dados da Tabela 3.2, os possíveis valores e a combinação destes valores que uma regra pode possuir para o atributo Tempo são apresentados na Tabela 5.1. A codificação deve possibilitar que a combinação seja identificada através de um número real. Se novamente for utilizada uma codificação incremental, como a apresentada na tabela, a princípio não há uma maneira de se obter a combinação através do número. Assim, a solução desenvolvida foi definir uma codificação que seja possível obter a combinação a partir de um possível número e obter uma combinação a partir de uma codificação. Para isso, as combinações de valores foram transformadas em números binários.

Na codificação proposta cada valor é definido por um índice do número binário. Por exemplo, na Tabela 5.1, o valor Sol possui o índice 2, o valor Chuva o índice 1 e o valor Nublado o índice 0. Estes índices são utilizados para identificar a qual casa do número binário o valor corresponde. Na codificação utilizada se o bit da casa estiver ligado (possuir o valor 1) o valor do atributo faz parte da combinação, caso contrário, ele não faz parte. Assim, a combinação Sol ou Nublado é definida pelo número binário 101. Como para fazer parte da partícula este número deve ser real ele é transformado para o sistema decimal, logo a codificação é definida como 5. Para se obter a combinação a partir da codificação basta transformar um número decimal para binário. Na execução das equações do movimento o operador `mod` que é aplicado para limitar a regra dentro do

Tabela 5.1: Possíveis combinação de valores para o atributo Tempo e duas possíveis codificações.

Sol	**Chuva**	**Nublado**	**Incremental**	**Binário**
0	0	0	0	$000 = 0$
1	0	0	1	$100 = 4$
0	1	0	2	$010 = 2$
0	0	1	3	$001 = 1$
1	1	0	4	$110 = 6$
1	0	1	5	$101 = 5$
0	1	1	6	$011 = 3$
1	1	1	7	$111 = 7$

espaço de busca passa a ter como limite o número total de combinações que é 2^{n-1}, onde n é o número de valores para o atributo.

5.4 Implementação de Diferentes Esquemas de Votação

Como apresentado na Seção 3.3.3, para a construção de um classificador de regras não-ordenado, deve haver um esquema de votação para classificar cada exemplo entre as classes analisadas. Na Figura 3.1 é apresentado um esquema geral de votação que identifica cada passo deste processo. Neste trabalho, foram implementados três esquemas de votação: Votação Simples, Votação por Precisão e Votação por Laplace. Esta seção também apresenta como a AUC é obtida após a execução do processo de votação.

O primeiro esquema, intitulado Votação Simples (VL), não utiliza todos os passos do esquema geral apresentado. Nesta votação, basicamente são contadas quantas regras de cada classe cobrem o exemplo. O primeiro passo deste esquema é a identificação das regras que cobrem o exemplo para cada classe. Após está identificação não é efetuada nenhuma ordenação nem há a ponderação na votação. Após a identificação das regras, a classe que obtiver o maior número de votos, ou seja, possuir o maior número de regras é definida como a classe do exemplo. Este método tem como vantagem o fato de todas as regras que cobrem o exemplo participarem da votação. Assim, se for gerado um bom conjunto de regras pelo algoritmo a classificação será boa para a maioria dos exemplos. Porém, como não há ponderação na votação, nem a seleção das melhores regras, caso haja um grande número de regras ruins o classificador terá resultados ruins, apesar de poder

possuir regras com bom potencial para a classificação.

O segundo esquema, Votação por Confiança (VC), também não executa todas as etapas do esquema geral de votação. Neste método, como na Votação Simples, novamente não há a ordenação das regras, porém a votação final é ponderada. Inicialmente a VC executa a identificação das regras que cobrem o exemplo, para cada classe. Após isto, é somada para cada classe o valor da confiança (3.9) de cada regra obtida durante o treinamento. Este esquema possui as mesmas vantagens da Votação Simples, com a vantagem que com a ponderação, regras pouco precisas não influenciam muito a escolha da classe. Como desvantagem, novamente, se há um conjunto de regras muito grande para uma classe (isto normalmente ocorre quando a base de dados é desbalanceada [37], ou seja, quando a base de dados possui muito mais exemplos de um classe do que das outras), mesmo com a ponderação o classificador efetua uma classificação errônea.

Por fim, o terceiro esquema implementado (esquema utilizado nos experimentos do Capítulo 6) é a Votação por Laplace (VL). Este esquema é proposto em [48] e apresenta bons resultados, principalmente em base de dados desbalanceadas. Neste esquema todas as etapas são efetuadas. Primeiro, é efetuada a divisão das regras que cobrem o exemplo de acordo com as classes da base de dados. Após isto é feita ordenação de cada conjunto de regras de acordo com a precisão de Laplace (3.4). Após a ordenação é efetuada a votação ponderada, novamente utilizando-se a precisão como peso. Nesta votação somente as k melhores regras votam, onde k é o número de classes da base de dados. Como este estudo só contempla bases de dados binárias, este número sempre é definido com o valor 2. Este método tem como vantagem o fato de que a ordenação das regras e a seleção de somente as melhores regras fazem com que a votação seja definida pela qualidade das regras ao invés do número de regras. Assim, quando aplicado em domínios desbalanceados o número total de regras para cada classe não irá influenciar. Vale ressaltar que nestes domínios normalmente é gerado um número maior de regras para a classe majoritária. Neste método o que define a classe do exemplo em análise é a qualidade das regras selecionadas para votação. Assim quanto mais precisas as regras para uma determinada classe, maior será a probabilidade de esta classe ser escolhida. Como desvantagem, o fato de selecionar

as regras pode retirar da votação regras que poderiam produzir uma classificação mais correta caso elas fizessem parte deste processo.

O calculo da AUC é realizado baseado nesta votação. Depois que todas as regras votam, cada exemplo possui um valor associado. Então, para cada instância de base de dados existe um ranking numérico com valores associados. Este ranking pode ser utilizado como um limiar para produzir um classificador binário. Se o ranking de uma determinada instância estiver acima do limiar, então o classificador produz um "sim", caso contrário, produz um "não". Cada valor de limiar gera um ponto diferente no plano ROC. Então, a variação do limiar de $-\infty$ para $+\infty$ produz uma curva no plano ROC, sendo possível calcular a área abaixo da curva (AUC) [15]. Este valor dá uma boa visualização do desempenho do classificador. Numa classificação perfeita, os exemplos positivos têm os maiores valores agrupados no topo do ranking e o classificador possui o maior valor de AUC (igual a um). No Capítulo 6 será apresentado um estudo comparando a precisão e a AUC de classificadores gerados utilizando cada um dos esquemas de votação discutidos, para bases de dados balanceadas e desbalanceadas.

5.5 Procedimento de Remoção de Regras Específicas

O desenvolvimento deste procedimento visa a redução do número total de regras sem que haja a perda de expressão do modelo gerado. A idéia é retirar do repositório as regras que expressem um mesmo conhecimento. Assim, o procedimento somente deixa no repositório das soluções não dominadas as soluções mais genéricas. Uma regra r_1 é dita genérica em relação a outra regra r_2 se r_1 possui menos restrições de atributos que r_2 e todas as restrições de r_1 estão contidas em r_2. Além disso, as duas regras devem cobrir o mesmo conjunto de instâncias, ou seja, suas matrizes de contingência devem ser iguais. Na relação oposta, é dito que r_2 é mais específica que r_1. Como exemplo, dadas as Regras 1 e 2:

Regra 1: Se (Tempo=chuvoso) ENTÃO (PraticarEsportes = nao)

Regra 2: Se (Tempo=chuvoso) E (Vento=sim) ENTÃO (PraticarEsportes =

nao)

Se as Regras 1 e 2 possuem a mesma Matriz de Contingência, é dito que a Regra 1 é mais genérica que a Regra 2, pois possuem menos restrições de atributo. Pode-se dizer também que a Regra 2 é mais específica que a Regra 1, pois possui mais restrições.

O procedimento, chamado de Remoção de Regras Específicas, funciona da seguinte maneira: para cada regra que será adicionada no repositório procurar por uma regra mais genérica ou mais específica que ela no repositório. Caso encontre uma regra mais genérica o procedimento é encerrado e a regra não entra no repositório. Caso uma regra mais específica seja encontrada ela deve ser retirada do repositório. Ao final, se nenhuma regra genérica foi encontrada a regra em questão é adicionada. Assim, somente a regra mais genérica que continua no repositório. Este procedimento é executado a cada geração, no final do processo de seleção das melhores soluções (passo 7 do Algoritmo 2). No Capítulo 6 será apresentado um estudo que mostra o percentual de redução no número total de regras e a influência desta redução nos resultados da classificação.

5.6 Proposta de um Algoritmo Paralelo Baseado na Metaheurística MOPSO

Apesar de algumas técnicas evolutivas terem sido propostas e implementadas com sucesso, como os algoritmos GAssist [2] e XCS [47], as Metaheurísticas Multiobjetivo GRASP-PR [25] e MOPSO [45], entre outras, poucos são capazes de trabalhar com grandes bases de dados. Esta sessão descreve um algoritmo, chamado MOPSO-P, que utiliza as vantagens da técnica da Otimização por Nuvem de Partículas e da Otimização Multiobjetivo e as aplica no contexto da Mineração de Dados, com o intuito de trabalhar com grandes bases de dados.

O algoritmo segue a mesma metodologia apresentada nas Seções 5.1 e 5.2. Ele é baseado no algoritmo MOPSO-N, porém ele tem a idéia de dividir o conjunto de treinamento em p subconjuntos, ao invés de somente um. O número p e o número de nós distribuídos disponíveis para a tarefa. Assim como o MOPSO-N, MOPSO-P tenta gerar

um bom conjunto de regras usando atributos tanto numéricos quando nominais e tentar construir um bom classificador em termos de AUC. Porém, o principal objetivo da proposta do algoritmo é a redução do tempo de execução distribuindo o processo em diversos nós, principalmente quando o algoritmo estiver trabalhando com grandes bases de dados. Na abordagem proposta, o algoritmo é executado de maneira sequencial, isto é, ele não foi aplicado num ambiente paralelo. Assim, problemas de execução tais como de sincronização e balanceamento de carga não são considerados neste momento.

Inicialmente os principais aspectos do MOPSO-P são descritos na Seção 5.6.1. Na Seção 6.2.5, MOPSO-P é avaliado através de um conjunto de experimentos onde os resultados e o tempo de execução do MOPSO-N e do MOPSO-P em bases de dados grandes, são comparados utilizando-se a AUC.

5.6.1 Algoritmo MOPSO-P

O MOPSO-P possui os principais passos apresentados nas Seções 5.1 e 5.2. Ele executa os mesmos passos apresentados no Algoritmo 2. O principal objetivo do algoritmo é permitir a execução múltipla da técnica MOPSO. Para fazer isto, dois novos procedimentos são desenvolvidos: um para separar os conjunto de dados original e efetuar múltiplas execuções do MOPSO-N e o segundo para juntar o conjunto de regras geradas e assim obter o classificador. A Figura 5.1 apresenta um diagrama com os principais detalhes do algoritmo MOPSO-P. A seguir os principais passos do algoritmo são descritos.

Primeiro, o MOPSO-P recebe como entrada os conjuntos de treinamento e teste e o número p de partições. Então, ele executa um procedimento que divide o conjunto de treinamento em p partições, cada partição tendo um número igual de instâncias de treinamento, respeitando a proporção de exemplos por classe para cada partição. Cada partição é distribuída para um nó de execução. Após cada partição ter sido alocada para um nó é executado o algoritmo MOPSO-N. A seguir, após todos os nós terem terminado a execução, existem p conjuntos de regras. Estas regras irão formar o classificador. Assim, um procedimento para a junção das regras deve ser executado. Neste passo diferentes decisões podem ser tomadas para evitar redundância nas regras. Por exemplo, é possível

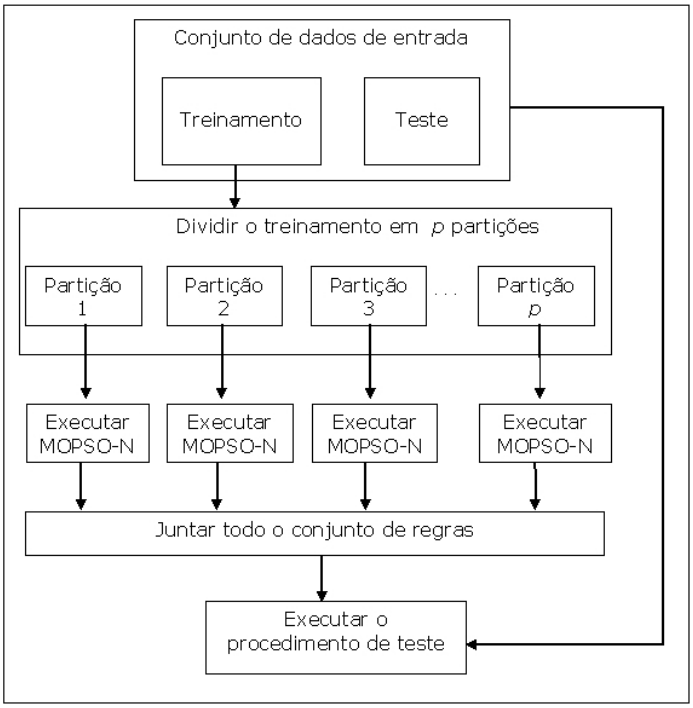

Figura 5.1: MOPSO-P, Abordagem paralela

executar um procedimento de seleção para identificar as soluções não-dominadas e para retirar as regras equivalentes. Neste trabalho nenhuma seleção é aplicada, então não há procedimento para evitar a redundância de regras e todas as regras são usadas para compor o classificador final. Trabalhos futuros investigarão diferentes métodos para a junção das regras. Por fim, após o procedimento de junção, o classificador está pronto para classificar novas instâncias. Para efeitos de avaliação do classificador uma etapa de teste é efetuada. Na etapa de testes o classificador é testado com o conjunto de dados de teste original, não há divisão deste conjunto.

5.7 Considerações Finais

Neste capítulo foram apresentados o algoritmo e as extensões desenvolvidas neste trabalho. Todas as extensões tiveram como objetivo a solução de alguns problemas que surgem quando a técnica MOPSO é aplicada no contexto do Aprendizado de Regras na Mineração de Dados. Basicamente, buscou-se melhorar a expressividade das regras geradas, obter um melhor resultado do classificador e reduzir o tempo de execução do algoritmo. O

próximo capítulo apresentará os experimentos que validam as extensões propostas. Para cada algoritmo desenvolvido será apresentado um conjunto de experimentos que verificará se a extensão alcançou o objetivo esperado. Neste capítulo a principal medida utilizada para a verificação da qualidade do classificador será a AUC, pois o algoritmo desenvolvido tem como objetivo a criação de um bom classificador em termos desta medida.

CAPÍTULO 6

EXPERIMENTOS

Este capítulo apresenta os experimentos para a validação do algoritmo proposto e suas extensões. Inicialmente, na Seção 6.1, será apresentada a metodologia que inclui as informações sobre as bases de dados utilizadas, os algoritmos utilizados na comparação e o ajuste de parâmetros da abordagem MOPSO. Após a metodologia, os resultados experimentais da validação do algoritmo proposto neste trabalho e suas extensões serão apresentados na Seção 6.2.

6.1 Metodologia

Para a execução e avaliação do algoritmo desenvolvido neste projeto, apresentado no Capítulo 5, foram utilizados um conjunto de bases de dados de livre acesso. Essas bases foram obtidas através do repositório de Aprendizado de Máquina UCI [1] e são detalhadas na Seção 6.1.1. Para a execução da etapa de testes, foi necessária a preparação das bases analisadas.

O algoritmo MOPSO foi implementado na linguagem de programação Java. Para a execução do algoritmo proposto foram desenvolvidas rotinas para o carregamento da base de dados, cálculo das medidas de avaliação e obtenção dos resultados finais do algoritmo. Para o carregamento da base de dados foram utilizados os métodos do *framework* Weka, discutido no Capítulo 3. Para a utilização deste conjunto de métodos foram implementadas rotinas de conversão de diversos formatos de dados para o formato .ARFF. Para a avaliação das regras e do classificador foram implementadas todas as medidas apresentadas na Sessão 3.3.2.2. Após a obtenção das medidas de avaliação há a execução de uma rotina para a obtenção dos resultados finais. Estes resultados são utilizados na etapa da análise dos experimentos, que será apresentada na próxima seção. O Algoritmo 3 apresenta um resumo de todos os passos necessários para a execução do MOPSO-N.

Algoritmo 3 Pseudo-Código com todos os passos para a execução do MOPSO

1: Preparação da base.
2: Carregar base usando o framework *Weka*.
3: Execução do MOPSO para a classe positiva.
4: Execução do MOPSO para a classe negativa.
5: Cálculo das medidas de avaliação.
6: Obtenção das medidas de avaliação utilizadas nos experimentos.

Os passos apresentados no Algoritmo 3 resumem as etapas discutidas nesta seção. O primeiro passo para a execução do MOPSO-N é a preparação das bases obtidas. Como dito anteriormente as bases devem estar no formato .ARFF. Além disso, pode-ser necessário a execução de rotinas para a redistribuição das instâncias e discretização de alguns atributos. Após a preparação, há o carregamento da base através do *framework* Weka.

Com as bases carregadas o MOPSO é executado. Como apresentado anteriormente, o algoritmo proposto só é aplicado em bases de dados com somente duas classes, normalmente uma classe é dita a positiva e a segunda é dita negativa. Como o estudo presente neste trabalho só contempla bases de dados binárias, as bases de dados que continham mais de duas classes, foram reduzidas selecionado-se a classe com menor frequencia a classe positiva e juntando os demais exemplos como a classe negativa. Os experimentos foram executados usando a validação estratificada cruzada com 10 partições [32], exceto para a validação do algoritmo MOPSO-P.

Além disso, como o valor da classe não faz parte da partícula o algoritmo é executado duas vezes, cada vez gerando regras para cada classe. Após a execução do algoritmo e a obtenção das regras geradas são calculadas as medidas utilizadas para a validação dos experimentos. Por fim, com todas as medidas já calculadas há a execução de um procedimento que obtém os valores das medidas que são utilizadas para a comparação dos algoritmos.

6.1.1 Bases de dados

Para o experimentos foram consideradas um total de 18 bases de dados do repositório de Aprendizado de Máquina UCI [1]. As bases de dados são apresentadas na Tabela 6.1. As bases de dados 1, 7 e 15 possuem todos os seus atributos nominais, enquanto as demais

Tabela 6.1: Descrição das bases de dados utilizadas no experimentos

#	Base de dados	# Atributos	# Exemplos	% da Classe Majoritária	Tipo dos atributos
1	adult	15	48842	76,07	nominais
2	breast	10	683	65,00	numéricos
3	bupa	7	345	57,97	numéricos
4	digits	17	9892	90,40	nominais
5	ecoli	8	336	89,58	numéricos e nominais
6	flag	29	174	91,23	numéricos e nominais
7	german	21	1000	70,00	nominais
8	glass	10	214	92,05	numéricos
9	haberman	4	306	73,52	numéricos
10	heart	14	270	55,00	numéricos
11	ionosphere	34	351	64,10	numéricos
12	kr-vs-kp	37	3196	52,00	numéricos
13	lettera	17	19999	96,05	numéricos
14	new-thyroid	6	215	83,72	numéricos
15	nursery	9	12960	97,45	nominais
16	pima	9	768	65,51	numéricos
17	satimage	36	6435	90,27	numéricos
18	vehicle	19	846	76,47	numéricos

bases possuem tanto atributos numéricos quanto discretos.

6.1.2 Algoritmos

Nos experimentos executados, o algoritmo proposto é comparado com alguns algoritmos de Aprendizado de Regras conhecidos da literatura. A execução e a obtenção das medidas utilizadas na comparação para algoritmo foram efetuadas através ferramenta Weka [21]. Os algoritmos selecionados e os parâmetros da execução (valores padrão da ferramenta) são:

- C4.5 e C4.5 Sem Poda (SP): Algoritmos de Árvore de Decisão. Em ambos algoritmos o número mínimo de instâncias por folha foi definido como 2 e o fator de confiança da poda foi de 25%. Os algoritmos são técnicas de Aprendizado de Regras.

- NNge: Algoritmo vizinhos mais próximos de Aprendizado de Regras que usa cópias generalizadas não-aninhadas. O número de tentativas para a generalização foi definido como 5.

- RIPPER: Algoritmo de Aprendizado de Regras proposicionais. Sem configuração de parâmetros.

Para a mediação das diferenças entre os resultados de cada algoritmo foi utilizado o teste estatístico não paramétrico de Wilcoxon [13], Teste-U, com nível de confiança de 5%. Neste teste os conjuntos valores de obtidos na execução dos algoritmos em todos as partições são comparados, dois a dois algoritmos. Caso haja uma diferença significante

entre este valores, dentro do nível adotado e observando a média destes valores, é dito que uma técnica é melhor que a outra. Este teste será utilizado em todas as comparações apresentadas na Seção 6.2.

6.1.3 Configuração dos Parâmetros do MOPSO

Para a configuração dos parâmetros do MOPSO algumas opções foram avaliadas. Inicialmente, os parâmetros foram definidos de forma empírica, baseados no desenvolvimento do algoritmo apresentando em [46]. Porém, para execução dos experimentos apresentados neste trabalho, os parâmetros foram definidos através da análise de alguns trabalhos relacionados [42] [5]. Assim, através dos experimentos inicias e da análise dos métodos da literatura a configuração utilizada foi definida é apresentada na Tabela 6.2. O peso de inércia, ϖ, é definido dinamicamente a cada atualização de posição das partículas num intervalo entre $[0,0,8]$. Uma fórmula alternativa de atualização de velocidade [5], usando um fator constrição ao invés do peso de inércia foi testada, mas não apresentou bons resultados. Os coeficientes de aprendizado foram configurados como $c_1 = c_2 = 2.05$. ϕ_1 e ϕ_2 também foram definidos de forma dinâmica a cada geração, variando entre $[0,1]$. Por fim, o parâmetro $prob_{vazio}$, utilizado na inicialização de atributos numéricos, foi definido com o valor $0,1$ para facilitar a criação de intervalos para os atributos numéricos das regras da população inicial. Os números de gerações e partículas foram definidos de forma empírica. O algoritmo é executado um total de 100 gerações e com um número total de 500 partículas. Números maiores de partículas e gerações tornam a execução do algoritmo muito longa e não há uma melhora significante no resultado final. Utilizando-se estes parâmetros com números menores os resultados apresentados não são satisfatórios. Esta configuração de parâmetros apresentou os melhores resultados e é utilizada em todos os experimentos apresentados a seguir.

Tabela 6.2: Parâmetros utilizados na execução do MOPSO

Parâmetro	**Configuração**
ϖ	dinâmico [0, 0,8]
c_1 e c_2	2,05
ϕ_1 e ϕ_2	dinâmico [0, 1]
$prob_{vazio}$	0,1
partículas	500
gerações	100

6.2 Resultados

A seguir são apresentados os resultados dos experimentos para a validação do algoritmo proposto e suas extensões. Todos os experimentos seguem a metodologia apresentada na seção anterior. O primeiro conjunto de experimentos, Seção 6.2.1 refere-se à comparação entre as abordagens numérica e discreta do MOPSO. Após isto, a Seção 6.2.2 apresenta alguns resultados da aplicação do operador *IN*. Em seguida, os métodos de votação implementados são comparados na Seção 6.2.3. A Seção 6.2.4 discute os resultados obtidos pela aplicação do procedimento Remoção de Regras Específicas. O algoritmo MOPSO-P é avaliado na Seção 6.2.5. Por fim, a Seção 6.2.6 apresenta um comparação geral do algoritmo desenvolvido neste trabalho, a abordagem MOPSO e todas suas extensões, em dois contextos: comparação dos resultados da classificação com alguns algoritmo da literatura em termos de AUC e acurácia e a comparação utilizando a abordagem multiobjetivo apresentada na Seção 4.3.1.

6.2.1 Abordagem Numérica x Discreta

Este primeiro experimento confronta os resultados obtidos pelo algoritmo MOPSO-D, que trabalha somente com atributos nominais, com os resultados do algoritmo MOPSO-N, que trabalha com dados numéricos e nominais. Ambos algoritmos tem como objetivo construir um conjunto de regras que representem o domínio em análise e produzir um bom classificador em termos de AUC. As regras geradas pelo MOPSO-N constroem os intervalos de valores de acordo com a maximização dos objetivos da busca, ou seja, o algoritmo visa sempre construir bons intervalos. Esta seção visa mostrar que através de

dados numéricos é possível a geração de um bom classificador. Para este experimento ambos algoritmos utilizaram o método de Votação por Laplace, Seção 5.4, e foi executado o procedimento de Remoção de Regras Específicas.

Cada algoritmo foi executado um total 30 vezes e resultados finais foram obtidos através da média destas execuções. Neste experimento foram analisadas um total de 8 bases de dados: `breast`, `bupa`, `ecoli`, `flag`, `haberman`, `glass`, `new-thyroid` e `pima`. Estas bases de dados foram escolhidas pois formam um conjunto diversificado de bases de dados com diferentes números de atributos e exemplos.

Para a execução do MOPSO-D é necessário que os atributos numéricos sejam discretizados e assim tornar todas as bases com valores nominais. Para esta tarefa inicialmente foi utilizado um filtro, obtido através da ferramenta Weka, que define para cada atributo numérico o número de intervalos, chamado de *bins*, o qual será dividido o atributo [17]. Porém está técnica não apresentou bons resultados, pois para a maioria das bases de dados os atributos são aglutinados em uma só faixa de valor. Assim, uma novo filtro foi utilizado. Este filtro, também disponível na ferramenta Weka, discretiza o intervalo dos atributos numéricos da base para um conjunto de valores pre-definidos. O processo é definido por uma simples divisão, isto é, o filtro divide todo o intervalo em partes de mesmo tamanho. Foram utilizados os parâmetros padrão e diferentes discretizações para cada base de dados. Cada base foi discretizada com 3, 5 e 7 *bins*.

A Tabela 6.3 apresenta a média dos valores de AUC para cada base de dados, os números entre parênteses representam o desvio padrão. Na tabela são apresentados os resultados do MOPSO-N e o MOPSO-D com as bases discretizadas com 3 *bins*, MOPSO-D-3, com 5 *bins* MOPSO-D-5 e com 7 *bins*, MOPSO-D-7. As células marcadas com cinza claro indicam que o algoritmo obteve um melhor resultado de acordo com o Teste-U. Caso duas células estejam marcadas para uma mesma bases de dados, os algoritmos marcados obtiveram resultados equivalentes.

Analisando os resultados, percebe-se que para a maioria das bases de dados o MOPSO-N apresentou melhores resultados que o MOPSO-D, para todas as discretizações. A abordagem numérica apresentou melhores resultados em todas as bases de dados. O

Tabela 6.3: Média da AUC para o MOPSO-N e MOPSO-D com diferentes discretizações

Base de dados	MOPSO-N	MOPSO-D-3	MOPSO-D-5	MOPSO-D-7
breast	98,94 (1,32)	98,19 (1,61)	97,45 (2,77)	97,48 (1,32)
bupa	70,73 (9,28)	56,96 (4,78)	59,01 (7,18)	64,80 (9,87)
ecoli	78,62 (31,59)	74,38 (22,77)	81,61 (11,27)	73,56 (20,37
flag	80,30 (14,65)	72,49 (22,82)	67,35 (27,42)	65,49 (23,20)
glass	75,23 (16,36)	52,87 (13,35)	67,09 (10,00)	65,49 (23,20)
haberman	69,63 (10,72)	60,97 (8,92)	64,68 (8,92)	71,03 (8,38)
new-thyroid	93,93 (12,61)	96,01 (4,78)	97,89 (2,62)	89,35 (8,98)
pima	91,17 (3,88)	64,87 (8,54)	69,54 (6,14)	68,41 (6,65)
Média	82,31	72,09	75,57	71,97

Tabela 6.4: Número de regras geradas pelo MOPSO-N e MOPSO-D-3

Base de dados	MOPSO-N	MOPSO-D-3
breast	26,53	26,85
bupa	164,97	77,06
ecoli	31,50	22,40
flag	123,18	35,27
glass	61,11	34,63
haberman	77,50	25,37
new-thyroid	17,45	13,51
pima	73,28	44,25
Média	65,14	38,55

MOPSO-D obteve resultados equivalentes para a base de dados haberman, com 7 bins, e para a base new-thyroid, com 3 e 5 bins. Com este resultado percebe-se também que a forma de discretização dos dados pode influenciar bastante no resultado da classificação, o que torna mais complexo usar um algoritmo que trabalhe somente com atributos nominais. Por fim, pode-se concluir que é possível a geração de um melhor classificador utilizando-se os dados originais, sem a necessidade de um pre-processamento de discretização. A próxima análise verifica o número de regras geradas por cada algoritmo. Nesta etapa só será utilizada a base de dados com 3 bins, a Tabela 6.4, o número médio de regras gerado por cada algoritmo em análise.

Embora o MOPSO-N possua melhores resultados em relação à AUC, ele produz mais regras que o MOPSO-D. É possível observar o maior número de regras para o MOPSO-N. Este resultado pode complicar o processo de interpretação dos dados minerados. Para redução do número de regras foi proposto o o procedimento de Remoção de Regras Es-

Tabela 6.5: Exemplos de regras obtidas sem e com a execução do operador *IN*

parents	has_nurs	form	children	housing	finance	social	health	Classe
Regras sem o operador								
usual	less_proper	?	?	convenient	convenient	?	recommended	positive
usual	proper	?	?	convenient	convenient	?	recommended	positive
Regra com o operador								
usual	less_proper ou proper	?	?	convenient	convenient	?	recommended	positive

pecíficas, que é validado na Seção 6.2.4.

6.2.2 Aplicação do Operador *IN*

O operador *IN* é aplicado ao MOPSO com o intuito de aumentar a expressividade das regras geradas. A idéia é fazer que as restrições para os atributos nominais passem a ter um conjunto de valores ao invés de somente um. Assim, informações que antes só poderiam ser obtidas com um conjunto de regras agora podem ser representadas em somente uma. Os experimentos apresentados nesta seção inicialmente mostram na prática o funcionamento do operador. Além disso, os testes têm como objetivo mostrar que a inserção do operador não reduz a qualidade do classificador gerado.

O algoritmo MOPSO foi executado de duas formas: com e sem a presença do operador *IN*. Os experimentos seguiram a metodologia descrita na Seção 6.1. O algoritmo utiliza o procedimento de Remoção de Regras Específicas e é utilizada a Votação Por Laplace. Para a execução dos testes só foram utilizadas bases de dados com somente atributos nominais. Ambos os algoritmo foram executado um total de 10 vezes.

A Tabela 6.5 apresenta um conjunto de regras obtidas para a base de dados `nursery`, *fold* 0, de uma execução. A tabela mostra o resultado da aplicação do operador no algoritmo. As duas primeiras regras possuem os mesmos valores para os atributos exceto para o atributo "has_nurs". Assim, com aplicação do operado *IN* estas duas regras podem ser aglutinadas através da combinação dos valores para este atributo. O resultado final é compilado em uma só regra facilitando a interpretação do classificador. A Tabela 6.6 mostra o resultado médio de AUC para as 10 execuções, os números entre parênteses indicam os desvios padrões. As células marcadas com cinza claro indicam qual algoritmo obteve o melhor resultado, de acordo com o Teste-U.

Tabela 6.6: Resultado da AUC da execução do algoritmo MOPSO com e sem o operador *IN*.

Base de dados	sem o operador	com operador
adult	87,79 (0,61)	87,34 (0,48)
digits	94,88 (1,84)	93,57 (1,96)
german	73,13 (6,04)	73,66 (4,52)
nursery	85,23 (4,04)	94,64 (2,91)

Observando os resultados da comparação da AUC dos dois métodos, percebe-se que não há uma diferença significante entre os resultados. O algoritmo MOPSO executado sem o operador *IN* obteve melhores resultados nas bases `adult` e `digits`, enquanto o algoritmo com o operador foi melhor nas bases `german` e `nursery`. Na base `nursery` o MOPSO com a execução do operador obteve um resultado bem melhor. Assim, esta comparação mostrou que adição do operador *IN* melhora a expressividade regra e não reduz a qualidade do classificador gerada, sendo quem em algumas bases houve uma melhora na AUC.

6.2.3 Análise dos Diferentes Métodos de Votação

No processo de classificação executado pelo classificador não ordenado de regras desenvolvido neste trabalho, o método de votação é determinante para a escolha feita. Na Seção 5.4, foram apresentados os esquemas de votação desenvolvidos neste projeto: Votação Simples, Votação por Confiança e Votação por Laplace.

Este experimento tem como objetivo verificar a qualidade da classificação efetuada por cada um dos métodos em termos de AUC. Serão comparados três versões do algoritmo MOPSO, cada uma com uma método de votação diferente. Nos experimentos, foi utilizada a metodologia da Seção 6.1. O algoritmo MOPSO trabalha com dados numéricos e executa o procedimento de Remoção de Regras Específicas. Foram utilizadas um conjunto de 8 bases de dados e cada algoritmo foi executado um total de 30 vezes. A Tabela 6.7 mostra o valor médio da AUC para todas execuções e os desvios padrões. As células marcadas com cinza claro indicam qual algoritmo obteve um melhor resultado de acordo com o Teste-U.

Tabela 6.7: Valor médio da AUC obtido através de três esquemas de votação diferentes

Base de dados	Votação Simples	Votação por Confiança	Votação por Laplace
breast	98,84 (1,37)	99,07 (1,22)	98,94 (1,32)
ecoli	93,40 (4,04)	93,01 (3,68)	93,92 (6,68)
flag	70,73 (21,63)	68,81 (21,06)	79,82 (20,47)
glass	66,61 (20,34)	63,18 (20,81)	71,91 (16,05)
haberman	70,40 (9,82)	70,34 (9,32)	70,18 (8,64)
ionosphere	89,03 (6,23)	91,42 (5,90)	87,58 (7,34)
new-thyroid	97,30 (7,52)	97,01 (8,01)	96,18 (9,71)
pima	79,81 (4,62)	79,83 (4,67)	80,90 (4,50)

Na comparação dos métodos de votação não houve uma técnica que se destacou em relação às demais, porém a Votação por Laplace obteve melhores resultados. Este esquema foi melhor nas bases de dados `breast, ecoli, flag, glass, haberman` e `pima`. Na base de dados `flag` este resultado foi significantemente melhor que os demais métodos. O esquema de Votação por Confiança obteve melhores resultados nas bases de dados `breast, haberman, ionosphere, new-thyroid` e `pima`. A Votação Simples foi melhor na base `breast, haberman` e `new-thyroid`. Estes resultados mostram que as técnicas produzem resultados de classificação semelhantes, em temos de AUC, porém a Votação por Laplace foi a técnica que apresentou melhores resultados.

6.2.4 Remoção de Regras Específicas

Este seção apresenta o conjunto de experimentos para a validação do procedimento de Remoção de Regras Específicas. O objetivo do procedimento é reduzir o número total de regras do classificador sem que haja a redução da qualidade da classificação. A redução no número de regras é interessante pois torna o modelo gerado mais simples e inteligível. O experimento usa a metodologia apresentada na Seção 6.1 e utiliza com o método de votação a Votação por Confiança. O algoritmo utilizado trabalha com dados numéricos. Foram utilizadas 8 bases de dados nesta comparação.

O algoritmo foi executado em duas formas diferentes: sem a remoção das regras específicas e com a remoção das regras. Cada forma foi executada um total de 30 vezes. A Tabela 6.8 apresenta os valores médios de AUC e o número médio de regras gerado. Os números em parênteses indicam os desvios padrões.

Tabela 6.8: Número de regras e AUC, procedimento Remoção de Regras Específicas

Base de dados	**com regras específicas**		**sem regras específicas**		% de redução
	AUC	# de regras	AUC	# de regras	
breast	98,97 (1,33)	334	98,94 (1,32)	91	72,75
ecoli	78,62 (31,59)	182	93,92 (6,68)	52	71,42
flag	80,30 (14,65)	974	79,82 (20,47)	54	94,45
glass	75,23 (16,36)	83	71,91 (16,05)	71	14,45
haberman	69,63 (10,72)	412	70,18 (8,64)	155	62,37
ionosphere	86,37 (8,17)	357	87,58 (7,34)	324	9,24
new-thyroid	93,93 (12,61)	96	96,18 (9,71)	40	58,33
pima	91,17 (3,88)	352	80,90 (4,50)	260	26,13

Os resultados apresentados na Tabela 6.8 mostram que o procedimento Remoção de Regras específicas reduz de forma significante o número de regras geradas pelo algoritmo MOPSO. A média de redução paras as 8 bases de dados foi de 51,14%. Além disso, o MOPSO executado juntamente com o procedimento não perde performance na classificação. Das 8 bases de dados utilizadas, em 6 o algoritmo com o procedimento conseguiu alcançar o melhor resultado em termos de AUC

6.2.5 Validação Abordagem Paralela

Este conjunto de experimento tem como objetivo validar a abordagem paralela do MOPSO, MOPSO-P, proposta neste trabalho, Seção 5.6.1. Nesta investigação, os experimentos não são executados num domínio paralelo e cada partição é executada sequencialmente.

Nestes experimentos são efetuadas algumas alterações na metodologia utilizada nos demais experimentos apresentados neste capítulo. O algoritmo é executado utilizando as maiores bases de dados em análise: `adult, digits, lettera` e `nursery`. A forma de alimentação é modificada, assim não é utilizada a validação estratificada cruzada com 10 partições. Para todas as bases de dados, a base de dados de treinamento é composta por 2/3 das instâncias enquanto o resto, 1/3 compõem a base de dados de teste. O número de partições foi definido como 4 nós e o algoritmo foi executado um total de 10 vezes. A votação utilizada no cálculo da AUC para ambos algoritmos foi a Votação por Laplace 5.4. Os demais parâmetros seguem a metodologia adotada.

O primeiro estudo efetuado compara a os resultados da classificação do MOPSO-P, em

Tabela 6.9: Valor médio de AUC e número de regras, algoritmos MOPSO-P and MOPSO-S

Bases de dados	**MOPSO-P**		**MOPSO-S**	
	AUC	# regras	AUC	# regras
adult	88,13 (0,17)	785,9	87,82 (0,33)	203,7
digits	97,55 (0,71)	223,4	97,27 (0,65)	64,3
lettera	96,47 (0,26)	302	96,67 (0,19)	94,1
nursery	95,74 (0,20)	64,1	87,20 (0,18)	19,9

termos de AUC, com os resultados da abordagem sequencial do MOPSO, aqui chamado de MOPSO-S. Os valores médios de AUC para todas as execuções são mostrados na Tabela 6.9 e os números entre parênteses indicam os desvios padrões. Além disso o número médio de regras em cada execução é apresentado. As células marcadas com cinza claro indicam qual algoritmo obteve melhor resultado de acordo com Teste-T.

Pode ser observado que o MOPSO-P obtém bons valores de AUC. Em duas bases de dados, `adult` e `nursery`, o MOPSO-P obteve melhores resultados que o MOPSO-S, em duas bases, `digits` e `lettera`, o resultado encontrado foi equivalente. Uma explicação para estes melhores resultados obtidos é o o maior número de regras gerado pela abordagem paralela. Como dito anteriormente, a algoritmo MOPSO desenvolvido neste trabalho gera uma aproximação da fronteira de Pareto considerando a sensitividade e a especificidade como objetivos, tendo em vista obter bons resultados de AUC. Como o MOPSO-P é um algoritmo que efetua execuções paralelas do algoritmo MOPSO, ele produz um conjunto maior de boas regras, que é obtido pela junção das regras de cada nó. A Figura 6.1 apresenta o resultado de ambos os algoritmos para a base de dados `nursery`. Os pontos escuros representam as regras geradas pelo MOPSO-P em uma execução. Os pontos mais claros são as regras geradas pelo MOPSO-S, também em uma execução. Nota-se que o MOPSO-P gera um maior número de regra com bons valores de sensitividade e especificidade.

O algoritmo MOPSO-P apresentou bons resultados de AUC, mas o principal objetivo da abordagem proposta é a redução de tempo, especialmente quando trabalhando com grandes bases de dados. A Tabela 6.10 apresenta o tempo médio para as 10 execuções de ambos algoritmos em todos as bases de dados. A Tabela 6.11 apresenta o tempo

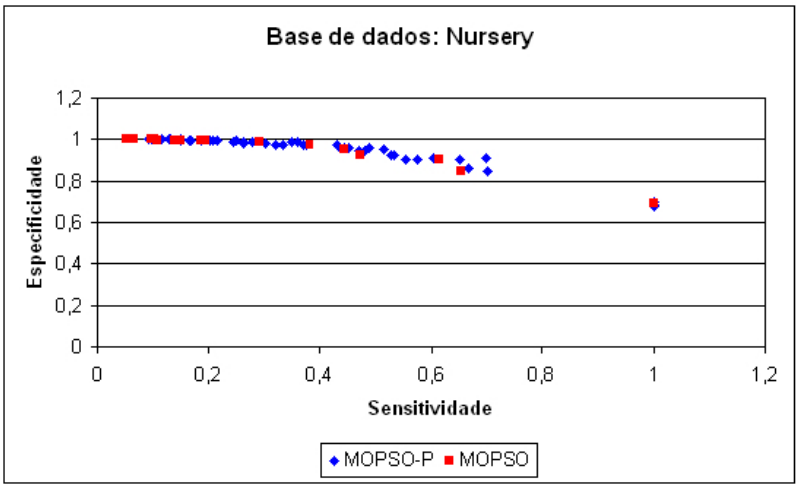

Figura 6.1: Conjunto de regras do MOPSO-P e MOPSO-S, base de dados Nursery.

Tabela 6.10: Tempo médio de execução, MOPSO-P e MOPSO-S

Bases de dados	MOPSO-P	MOPSO-S
adult	606,71 seg	584,23 seg
digits	249,88 seg	166,47 seg
lettera	364,47 seg	251,20 seg
nursery	166,81 seg	166,47 seg

médio de execução de somente uma partição do MOPSO-P. A princípio, o MOPSO-P executa por mais tempo que o MOPSO-S, pois nesta etapa o algoritmo executa várias vezes o MOPSO-S de forma sequencial. Porém, analisando a execução de cada partição, o algoritmo paralelo pode executar em um menor tempo que o sequencial. Por exemplo, para a base de dados `digits`, todas as partições são executadas num total de 223,36 segundos. assim, o tempo para os procedimentos de divisão e junção foi de 26,52 segundos. Para esta base de dados, a execução dos procedimentos de junção e partição mais o algoritmo de Aprendizado de Regras foi de 82,36. Esse tempo é mais rápido que o executado pelo MOPSO-S, pois o MOPSO-P executa com menores quantidades de dados. Uma execução do MOPSO-P gastou menos da metade do tempo que o MOPSO-S executou.

Tabela 6.11: Tempo médio de execução, uma partição do MOPSO-P

Bases de dados	MOPSO-P
adult	101,00 seg
digits	55,84 seg
lettera	71,89 seg
nursery	41,65 seg

6.2.6 Comparação Geral

Este conjunto de experimentos têm como objetivo comparar os resultados do algoritmo MOPSO e suas extensões, desenvolvidos neste trabalho, com alguns classificadores conhecidos da literatura. Além disso, os resultados do MOPSO são analisados utilizando a metodologia multiobjetivo apresentada na Seção 4.3.1. A Seção 6.2.6.1 apresentará a comparação dos classificadores utilizando como medidas a AUC e a acurácia (3.9) e a Seção 6.2.6.2 discute a análise multiobjetivo dos resultados do MOPSO.

6.2.6.1 Comparação dos classificadores

Esta seção apresenta a validação do algoritmo MOPSO desenvolvido neste trabalho e suas extensões, em termos de AUC e acurácia. Como apresentado anteriormente, o algoritmo proposto visa construir um classificador de regras não-ordenado com bons valores de AUC. Assim, os valores de AUC do MOPSO serão confrontados com o resultados de quatro algoritmos de Aprendizado de Regras conhecidos da literatura. Além da AUC será comparada também a acurácia, uma medida largamente utilizada na comparação de classificadores.

Nesta comparação o MOPSO será confrontado com os algoritmos apresentados na Seção 6.1.2. A comparação será efetuada com todas as bases de dados da Tabela 6.1. O algoritmo MOPSO utilizando contém todos as extensões apresentadas no Capítulo 5: trabalho com dados numéricos e nominais, aplica o operador *IN*, executa o procedimento de Remoção de Regras Específicas e utiliza o esquema de Votação por Laplace.

A execução do MOPSO segue toda a metodologia apresentada na Seção 6.1. O MOPSO foi executado um total de 10 vezes. Os parâmetros do algoritmo são descritos na Tabela 6.2 e a medição das diferenças entre os algoritmos foram obtidas através do Test-U. Os valores médios da AUC e da acurácia de cada algoritmo para todos as 10 partições são apresentados na Tabela 6.12 e os números entre parênteses indicam os desvios padrões. As células marcadas com cinza claro indicam a técnica com melhor resultado em termos de AUC para base de dados em questão. As células marcadas com cinza escuro indicam a técnica que foi melhor em acurácia.

Tabela 6.12: Comparação geral: AUC e Acurácia

Bases de dados	MOPSO		C4.5		C4.5 SP		RIPPER		NNge	
	AUC	Acurácia	AUC	Acurácia	AUC	Acurácia	AUC	Acurácia	AUC	Acurácia
1	87,79 (0,61)	80,01 (0,32)	89,08 (0,63)	86,10 (0,40)	86,07 (0,95)	84,74 (0,55)	75,11 (0,90)	84,49 (0,49)	72,66 (0,55)	79,55 (0,62)
2	98,94 (1,32)	95,44 (1,72)	95,84 (3,04)	95,31 (2,03)	95,33 (3,03)	95,31 (2,26)	96,98 (2,30)	96,19 (3,52)	96,71 (2,20)	96,34 (2,20)
3	70,73 (9,28)	66,71 (5,24)	63,60 (7,40)	66,64 (5,08)	64,2 (6,35)	66,06 (4,66)	66,51 (6,44)	68,72 (6,62)	60,32 (8,11)	62,04 (8,69)
4	94,88 (1,84)	20,45 (0,16)	99,27 (0,45)	99,69 (0,07)	99,30 (0,44)	99,71 (0,10)	98,99 (0,54)	99,59 (0,11)	99,54 (0,40)	99,80 (0,11)
5	93,92 (6,68)	91,46 (2,46)	78,90 (19,85)	92,27 (2,86)	79,12 (19,65)	92,27 (2,86)	79,90 (15,30)	92,25 (2,88)	75,17 (12,20)	92,55 (3,47)
6	79,82 (20,47)	90,76 (3,38)	50,00 (0,0)	91,26 (4,32)	54,07 (24,31)	85,60 (5,09)	55,26 (15,47)	88,21 (5,74)	64,44 (20,81)	92,28 (4,32)
7	73,13 (6,04)	70,89 (2,30)	69,30 (6,49)	72,30 (4,62)	61,05 (6,28)	66,60 (4,27)	63,18 (5,68)	72,60 (4,99)	61,57 (5,36)	71,00 (4,73)
8	71,91 (16,05)	90,61 (3,69)	75,07 (21,06)	93,00 (5,40)	75,82 (21,48)	92,53 (5,39)	56,00 (16,08)	90,23 (3,90)	49,5 (1,05)	91,14 (2,54)
9	70,18 (8,64)	73,75 (4,36)	59,59 (9,30)	72,88 (7,76)	62,28 (9,66)	72,22 (7,17)	61,41 (8,49)	74,55 (6,68)	56,14 (7,24)	67,33 (6,30)
10	87,47 (7,44)	78,74 (8,25)	76,55 (7,49)	74,07 (7,61)	75,30 (7,25)	71,85 (7,44)	77,75 (10,15)	78,14 (9,94)	76,33 (8,00)	76,66 (8,19)
11	87,58 (7,34)	83,15 (3,50)	86,60 (6,70)	88,61 (4,83)	88,22 (6,28)	89,17 (3,91)	90,97 (4,94)	90,03 (5,07)	90,47 (4,07)	91,16 (3,91)
12	96,60 (2,53)	86,37 (6,70)	99,85 (0,24)	99,43 (0,46)	99,86 (0,18)	99,24 (0,51)	99,50 (0,36)	99,09 (0,52)	98,50 (0,81)	98,52 (0,81)
13	95,21 (1,23)	96,29 (0,34)	98,30 (1,04)	99,59 (0,16)	98,48 (1,26)	99,57 (0,19)	97,68 (1,15)	99,66 (0,08)	95,68 (1,19)	99,52 (0,18)
14	96,18 (9,71)	92,32 (3,99)	91,80 (12,06)	95,41 (4,79)	91,89 (12,11)	95,41 (4,79)	92,77 (12,17)	96,79 (4,32)	92,50 (12,23)	96,34 (5,59)
15	85,23 (4,04)	86,94 (3,71)	98,93 (1,27)	98,85 (0,29)	98,79 (1,98)	99,32 (0,23)	82,70 (5,27)	98,24 (0,30)	90,91 (6,44)	99,25 (0,29)
16	80,90 (4,50)	74,11 (4,59)	75,57 (3,77)	76,68 (5,17)	75,46 (4,54)	75,64 (4,79)	71,30 (4,07)	74,73 (2,56)	68,51 (7,17)	71,74 (6,59)
17	91,57 (1,29)	90,26 (0,08)	73,03 (6,85)	91,70 (1,00)	75,95 (7,32)	91,21 (0,86)	74,13 (3,38)	92,40 (0,68)	72,22 (4,07)	92,58 (0,58)
18	95,59 (1,97)	86,12 (2,74)	93,49 (3,33)	94,20 (1,81)	93,34 (3,31)	94,20 (1,98)	92,61 (4,83)	94,19 (2,88)	85,82 (7,85)	90,89 (3,76)
Média	86,53	80,80	81,93	88,22	81,91	87,25	79,59	88,33	78,16	87,14

A Tabela 6.12 mostra que o MOPSO possui valores muito bons de AUC, quando comparado aos demais algoritmos. Nas bases de dados 2 ,3 ,5 ,6 ,7 ,9, 10, 14, 16, 17 e 18 o MOPSO apresentou o melhor resultado, de acordo com o Teste-U. Além disso, em algumas bases de dados como 5, 6 e 17 o algoritmo proposto apresentou um resultado bom na classificação.

Para os demais algoritmos, o C4.5 apresentou melhores resultados nas bases de dados 1, 8, 12, 13 e 15. O C4.5 Sem Poda alcançou os melhores valores de AUC nas bases 8, 12 e 13. O NNge obteve melhores resultados nas bases de dados 4 e 11, enquanto o RIPPER obteve melhor resultado somente na base 11. O MOPSO obteve a melhor média considerando os valores de AUC de todas as bases de dados. Estes resultados mostram que a técnica MOPSO, trabalhando com dados originalmente numéricos e utilizando como objetivos a sensitividade e a especificidade, consegue obter resultados muito bons de classificação em termos de AUC.

Na comparação da acurácia, o MOPSO só conseguiu atingir o melhor resultado na base de dados 10. Porém para a demais bases de dado, exceto a base 4, os resultados obtidos são bastante competitivos com os demais algoritmos. As técnicas que apresentaram os melhores resultados foram o NNge, que foi melhor nas bases 2, 4, 5, 6, 11 e 16, e o RIPPER que foi melhor nas bases 3, 7, 9, 13, 14 e 18. O C4.5 foi melhor nas bases de dados 1, 8, 12, 16 e 18, enquanto o C4.5 Sem Poda obteve um melhor resultado nas bases de dados 15 e 18. Na comparação das médias da acurácia para todas as bases o RIPPER conseguiu um melhor resultado com o valor de 88,33.

6.2.6.2 Análise Multiobjetivo

O segundo conjunto de experimentos é relacionado com a qualidade dos conjuntos de aproximação produzidos pelo MOPSO. O ideal neste caso é comparar a aproximação obtida pelo algoritmo com a Fronteira de Pareto. Porém, a fronteira de pareto é desconhecida para as bases de dados que possuem atributos numéricos, pois não há a possibilidade de se obter todas soluções possíveis do problema. Por esta razão, diferentes discretizações são consideradas. Após a discretização, todas as regras possíveis serão geradas, e selecionadas

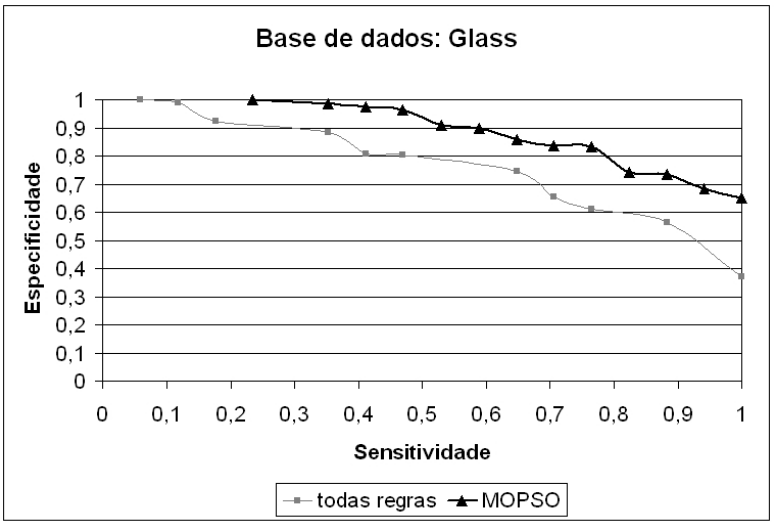

Figura 6.2: Fronteira de Pareto para a base de dados Glass e conjunto de aproximação gerado pelo MOPSO.

para a construção da Fronteira de Pareto.

Esta análise foi limitada a um número pequeno de bases de dados (`haberman, ecoli` e `new-thyroid`) devido ao custo da geração de todas as regras para uma base de dados. A discretização foi realizada utilizando-se um filtro da ferramenta Weka. O filtro utilizado é simples e divide cada atributo numérico em intervalos iguais. O número de partições é definido como parâmetro e neste trabalho foram utilizadas três diferentes partições: três, cinco e sete partições.

As regras geradas pelo MOPSO e pelo algoritmo exaustivo foram obtidas utilizando todos os exemplos de cada base de dados. O MOPSO foi executado um total de 30 execuções. A comparação foi efetuada para as classes positiva e negativa, separadamente.

A Figura 6.2 mostra a Fronteira de Pareto real, gerada pelo algoritmo exaustivo ("todas as regras"), para a base de dados `glass`, com três partições e o conjunto de aproximação gerado pelo MOPSO, para a mesma base, mas somente com valores numéricos. Ambos os conjuntos de regras foram obtidos para o fold 0 e a aproximação obtida pelo MOPSO é o conjunto das melhores regras das 30 execuções. Através do gráfico pode-se perceber a melhor performance do MOPSO-N.

Para a comparação das fronteiras foi utilizada uma metodologia de medição do desempenho apresentada na Seção 4.3.1. Na abordagem utilizada, inicialmente os algoritmos

são comparados através do teste do ranking de dominância. As diferenças são medidas através do teste estatístico de Mann-Withney [3]. Após isto, alguns indicadores de qualidade são utilizados para reforçar os resultados do teste inicial e para identificar algumas diferenças que não são obtidas com o teste do raking de dominância. Novamente o teste de Mann-Withney é utilizado para medir a significância entre os resultados de cada indicador.

Na comparação entre as fronteiras do MOPSO-N e o algoritmo exaustivo inicialmente o teste de ranking de dominância não mostrou nenhuma diferença significante entre ambos os algoritmos, para todas as diferentes discretizações. Assim, como não houve nenhuma relação de dominância definida, alguns indicadores de qualidade foram utilizados para medir a diferença entre ambos os algoritmos. Os indicadores utilizados foram o Epsilon aditivo, o Hipervolume e o R2.

Nesta análise, para as três bases de dados, `haberman, ecoli` e new-`thyroid`, o MOPSO-N apresentou melhores resultados. Em todos os indicadores, na comparação do MOPSO com todas as três diferentes discretizações, os conjuntos de aproximação gerados pelo MOPSO superaram os resultados obtidos através das bases de dados discretas. Estes resultados mostram, que para esse conjunto de bases de dados, com a utilização de atributos numéricos ao invés da execução de um processo de discretização é possível a obtenção de regras com melhores valores para os objetivos escolhidos e enfatizam a qualidade das fronteiras geradas pelo algoritmo MOPSO proposto.

CAPÍTULO 7

ESTUDO DE CASO: APLICAÇÃO NA PREDIÇÃO DE DEFEITOS EM SOFTWARES

Software possui um papel crucial em todas as áreas e atividades humanas. Devido a isto, tarefas de garantia de qualidade, como o teste de software, podem ser consideradas fundamentais. No entanto, teste é uma tarefa muito cara, que consome muito esforço e tempo no desenvolvimento de software. Para reduzir custos no teste, uma boa estratégia é focar o esforço de teste em algumas partes do software, tais como módulos, classes ou métodos, com maior probabilidade de serem defeituosos. Isto pode ser resolvido extraindo-se conhecimento de projetos passados e construindo-se um modelo para a predição de tendência a defeitos a ser aplicado em projetos futuros.

A maioria dos modelos de predição encontrados na literatura é estatístico e baseada em um conjunto de métricas de projeto Orientado a Objeto (POO), por exemplo: o conjunto proposto por Chidamber and Kemerer (CK-metric suite) [11], o conjunto de McCabe, Halstead e métricas de linhas de código. No entanto, a precisão na predição destes modelos é insatisfatória. Devido a isto, os trabalhos mais recentes têm introduzido o uso de técnicas de Aprendizado de Máquina. Redes Neurais e Redes Bayesianas são as técnicas mais exploradas [18, 30, 34, 39, 44, 49]. Pai e Dugan [34] predizem defeitos em classes usando métricas de POO e redes Bayesianas. As mesmas idéias são exploradas em [14] usando Máquinas de Vetores Suporte [26]. Estes trabalhos apresentam resultados promissores, mas o modelo obtido não é simbólico e devido a isto há dificuldade na interpretação dos resultados.

Resumindo, até agora, os trabalhos baseados no AM possuem a desvantagem de produzirem modelos que não são facilmente interpretados. Além disso, os resultados reportados pela maioria dos trabalhos somente aborda poucos aspectos dos modelos de predição, tais como a precisão.

Para reduzir estas desvantagens este capítulo apresenta a aplicação do algoritmo desenvolvido neste trabalho no contexto da predição de defeitos em software. Os resultados da aplicação do algoritmo MOPSO são descritos em dois diferentes contextos: predição de defeitos em classes e em métodos. As regras geradas são analisadas para identificar relacionamento interessantes entre as métricas de POO e a tendência a defeitos. Em todos os experimentos foram utilizadas as medidas: AUC (Seção 3.3.4), Acurácia (3.9), Precisão (3.10), *Recall* (3.11) e F-Measure (3.12). Estas medidas são definidas na Seção 3.3.

O resto deste capítulo está estruturado da seguinte maneira. Na Seção 7.1 é mostrado como o algoritmo proposto é introduzindo nos dois contextos de predição de defeitos. É descrita a metodologia usada nos experimentos, são apresentadas as métricas e as bases de dados utilizadas. Nas Seções 7.2 e 7.3 são apresentadas as análises dos resultados obtidos na predição de defeitos em métodos e em classes, respectivamente.

7.1 Predição de Defeitos

Nesta seção é explorada a abordagem que usa o algoritmo MOPSO descrito anteriormente para determinar a probabilidade de classes e métodos terem defeito. Para isto, duas avaliações são feitas usando diferentes métricas de software. Na primeira avaliação, um conjunto de experimentos é conduzido para predizer a tendência a defeitos de métodos. Na segunda, os experimentos são relacionados com a tendência a defeitos em classes. Nas duas avaliações o algoritmo MOPSO é comparado com outros algoritmos conhecidos da literatura, que já são usados no contexto da predição de falhas. Ambos os estudos usam as mesmas medidas de avaliação de classificadores, apresentadas na Seção 3.3.

7.1.1 Bases de Dados

Os estudos apresentados neste capítulo usam bases de dados de acesso livre, disponibilizadas pelo Projeto de Métricas de Dados da NASA [40]. Na primeira avaliação um conjunto de cinco bases de dados é utilizado, todas as bases contendo somente métricas de métodos. Três bases de dados são obtidas através de projetos escritos na linguagem de

Tabela 7.1: Descrição das bases de dados.

	Projeto	Linguagem	% Módulos defeituosos	% Módulos não defeituosos
Nível de método	PC1	C	77	1032
	CM1	C	49	449
	JM1	C	2106	8779
	KC1	C++	326	1783
	KC2	C++	107	415
Nível de classe	KC1	C++	247	1814

programação procedural C (PC1, CM1 e JM1) e dois na linguagem orientada a objetos C++ (KC1 e KC2). Todos os módulos são associados com duas classes, defeituoso e não defeituoso. No segundo experimento é usada uma base de dados, contendo métricas no nível de classe. Esta base está relacionada a um projeto em C++ (KC1). À partir desta base, uma base de dados de trabalho foi construída utilizando-se somente defeitos de implementação, deixando de fora erros de configuração ou defeitos do Sistema Operacional. O valor da classe também foi definido como defeituoso e não defeituoso.

A Tabela 7.1 mostra uma breve descrição de cada base de dados, tanto as com informações dos métodos (nível de método), quanto com informações de classes (nível de classe). Um aspecto das bases de dados que deve ser realçado é o desbalanceamento na distribuição das classes. Por exemplo, a base de dados CM1 é desbalanceada pois $90,20\%$ exemplos pertencem à classe não defeituosa e somente $9,8\%$ dos exemplos são métodos defeituosos.

7.1.2 Métricas de Software

Para a base de dados composta pelas métricas de classe, cada exemplo é composto por um subconjunto das métricas CK e o atributo meta. Este subconjunto é detalhado na Tabela 7.2. Para as bases de dados com informações sobre os métodos são 21 métricas mais o atributo meta. Este conjunto é formado por oito métricas de Halstead, quatro métricas de McCabe, contadores de linhas e operadores de contagem. Todas as métricas são descritas na Tabela 7.3.

O objetivo deste trabalho é identificar módulos defeituosos, então, é importante o somente o uso de métricas, que de alguma maneira, estão relacionadas com os defeitos. Assim, uma técnica de seleção de atributos baseada em correlação é aplicada para seleci-

Tabela 7.2: Descrição das métricas de classe.

Métrica	Descrição
Coupling Between Objects (CBO)	O número distinto de classes que não tenham relação de herança entre si.
Depth in Inheritance Tree(DIT)	O tamanho do maior caminho de uma classe até o nó raiz na herança.
Lack of Cohesion of Methods (LCOM)	O % de métodos que usam o atributo, para cada campo da classe.
Num of Children (NOC)	O número de classes herdadas de uma determinada classe.
Response for Class (RFC)	O número de métodos que podem ser executados numa determinada classe.
Weighted Methods per Class (WMC)	O número de métodos implementados numa classe.

Tabela 7.3: Descrição das métricas de método.

#	Métrica	Descrição
1	Comprimento de Halstead (N)	# de ocorrências de operadores e operandos.
2	Volume de Halstead (V)	# mínimo de bits necessários para o programa.
3	Nível de Halstead (L)	Nível de inteligibilidade do programa.
4	Dificuldade de Halstead (D)	Nível de dificuldade no programa.
5	Conteúdo Inteligente de Halstead (I)	Complexidade de uma dado algoritmo.
6	Esforço de Programação de Halstead (E)	Esforço mental estimado.
7	Erro estimado de Halstead (B)	Número de erros estimados.
8	Tempo de Programação de Halstead (T)	Quantia de tempo estimada para a programação.
9	Complexidade Ciclomática de McCabe (V(g))	O número de caminhos de execução lineares independentes.
10	Complexidade Essencial de McCabe (EV(g))	Medida de construções não-estruturadas.
11	Complexidade de Projeto de McCabe (IV(g))	Quantifica o esforço de teste para a integração.
12	LOC de McCabe	# total de linhas de código
13	LOCode	# de linhas do comando.
14	LOComment	# de linhas de comentários.
15	LOBlank	# de linhas em branco.
16	LOCodeAndComment	# de linhas de código e comentários.
17	UniqOp	# de operadores únicos.
18	UniqOpnd	# de operandos únicos.
19	TotalOp	Total de operadores.
20	TotalOpnd	Total de operandos.
21	BranchCount	# de ramos (desvios)

onar as melhores métricas para cada base de dados. Este processo é utilizado na maioria dos trabalhos de predição de defeitos. O filtro utilizado seleciona um subconjunto de atributos que estão altamente correlacionados com o defeito [22]. Este filtro foi aplicado somente nas bases de dados no nível de método, devido ao grande número de atributos da base. As métricas selecionadas para cada base são apresentadas na Tabela 7.4.

7.1.3 Algoritmos

Os resultados obtidos pela predição do algoritmo MOPSO são comparados com conjunto de algoritmos da literatura: um conjunto de conhecidos classificadores e um conjunto

Tabela 7.4: Métricas de método selecionadas.

Base de Dados	Métricas Selecionadas
PC1	5, 9, 14, 15, 16, e 18
CM1	11, 12, 14, 15, 17 e 18
JM1	5, 9, 10, 11, 12,14, 15 e 16
KC1	2, 4, 5, 13, 14, 15, 18 e 21
KC2	5, 7, 10, 17 e 18

de técnicas de Aprendizado de Máquina, utilizadas no contexto da predição de defeitos. Todos os algoritmos foram executados através da ferramenta Weka [21]. Os algoritmos selecionados e os parâmetros de execução (valores padrão das ferramentas) são definidos a seguir:

- C4.5 e C4.5 Sem Poda (SP): Algoritmos de Árvore de Decisão. Em ambos algoritmos o número mínimo de instâncias por folha foi definido como 2 o fator de confiança da poda foi de 25%. Algoritmos de Aprendizado de Regras.

- NNge: Algoritmo vizinhos mais próximos de Aprendizado de Regras que usa cópias generalizadas não-aninhadas. O número de tentativas para a generalização foi definido como 5.

- RIPPER: Algoritmo de Aprendizado de Regras proposicionais. Sem configuração de parâmetros.

- BayesNet (BN): Algoritmo de aprendizado de Redes Bayesianas usando o algoritmo Simple Estimator. O algoritmo K2 foi utilizado para efetuar as buscas na rede.

- Naive Bayes (NB): Não houve configuração de parâmetros.

- Redes Neurais (RN): Algoritmo Multi-Layer Perceptron usando backpropagation. O número de neurônios da camada escondida é uma média entre o número de atributos e o número de classes de cada base de dados. A taxa de aprendizado foi definida com o valor 0,3 e o momento com o valor 0,2. O algoritmo foi executado por 500 épocas.

- Máquinas de Vetores Suporte (SVM): Algoritmo seqüencial mínimo de John Platt para treinamento de um classificador de vetor suporte. O parâmetro de regularização C foi definido como 1. A função de kernel utilizada foi a RBF e o parâmetro do kernel foi definido com o valor 0,5.

Os experimentos foram executados usando a validação estratificada cruzada com 10 partições e para todos os algoritmos foram aplicados os mesmos conjuntos de treinamento e teste. A Rede Neural foi executada trinta vezes, pois o algoritmo é não-determinístico.

O algoritmo MOPSO foi executado um total de 100 gerações e com 500 partículas, para cada classe. O algoritmo foi executado um total de 30 vezes. Os parâmetros ω, ϕ_1 and ϕ_2, são definidos aleatoriamente pelo algoritmo em cada atualização da posição. ω varia no intervalo $[0, 0, 8]$, ϕ_1 e ϕ_2 variam entre $[0, 1]$. Os coeficientes c_1 e c_2 são iguais a $2, 05$. O parâmetro $prob_{vazio}$ foi definido com o valor $0, 1$.

7.2 Predizendo Defeitos em Métodos

Esta seção apresenta os resultados dos experimentos obtidos na análise das bases de dados com métricas de método. Os resultados das medidas de avaliação do classificador para cada base de dados são apresentados nas Tabelas 7.5, 7.6, 7.7, 7.8 e 7.9. Os valores correspondem à média da execução em cada partição e os números entre parênteses indicam o desvio padrão. Para a comparação de diferentes algoritmos, como utilizado no Capítulo 6, o teste estatístico de Wilcoxon, com 5% de nível de confiança, foi utilizado.

Duas comparações são executadas. Primeiro uma comparação geral é feita. Nesta comparação o melhor algoritmo, dentre todos os algoritmos em análise, é obtido, considerando todas as medidas. Os melhores valores são indicados através das células marcadas com cinza escuro. A segunda comparação é restrita somente aos algoritmos de Aprendizado de Regras. Neste caso, os resultados do MOPSO são comparados com o C4.5, C4.5 Sem Poda, RIPPER e NNge. Nesta comparação, os melhores resultados estão definidos com um cinza claro nas tabelas.

7.2.1 Comparação Geral

Para a base de dados KC1, o MOPSO apresenta o melhor valor de AUC. A RN apresentou o melhor valor de acurácia. Para as demais medidas, primeiro para as derivadas da classe com o valor "defeituoso", o MOPSO possui bons valores de precisão, mas um valor baixo de *recall* e F-*measure*. A RN apresentou o melhor valor de precisão e o BN os melhores valores de *recall* e F-*measure*. Para a classe com o valor "não defeituoso", o MOPSO possui bons valores de precisão, porém o BN alcançou o melhor resultado. Para as outras

duas medidas, o algoritmo proposto conseguiu os melhores resultados. Estes resultados mostram, que para a base KC1, o MOPSO tem dificuldade na identificação de defeitos, porém não produz classificações erradas, ou seja, produz uma classificação segura.

Tabela 7.5: Medidas de avaliação para a base de dados KC1, nível de método

			Defeituoso			Não defeituoso		
Algoritmo	**AUC**	**Acurácia**	**Precisão**	**Recall**	**F-*measure***	**Precisão**	**Recall**	**F-*measure***
MOPSO	80,24 (4,30)	84,81 (0,71)	48,43 (37,94)	4,54(4,19)	8,11 (7,19)	85,07 (0,62)	99,49 (0,54)	91,72 (0,38)
Redes Neurais	78,98 (4,52)	85,79 (1,47)	69,52 (19,57)	18,56(9,45)	27,66 (11,62)	86,84 (1,30)	98,10 (1,66)	92,11 (0,822)
Bayes Net	79,49 (4,63)	75,43 (3,37)	34,45 (5,03)	63,80(7,75)	44,66 (5,84)	92,13 (1,69)	77,56 (3,25)	84,19 (2,36)
Naive Bayes	79,67 (3,95)	82,40 (2,34)	42,22 (8,51)	36,53(10,37)	38,75 (8,68)	88,69 (1,68)	90,80 (2,28)	89,71 (1,39)
SVM	50,58 (1,09)	83,83 (0,60)	26,63 (29,32)	2,45(2,30)	4,36 (3,99)	84,69 (0,36)	98,71 (0,61)	91,17 (0,34)
C45	67,95 (8,54)	84,06 (1,17)	48,70 (13,12)	18,67(7,12)	25,89 (7,76)	86,61 (0,78)	96,02 (2,07)	91,05 (0,74)
C45 NP	75,18 (4,99)	84,30 (1,32)	48,98 (11,94)	23,23(8,91)	30,55 (9,25)	87,21 (1,02)	95,45 (2,18)	91,13 (0,82)
RIPPER	57,12 (3,75)	84,59 (1,65)	52,73 (15,98)	17,49(7,65)	25,51 (8,57)	86,53 (1,07)	96,86 (1,80)	91,39 (0,95)
NNge	63,10 (4,07)	83,35 (1,67)	44,84 (6,25)	33,77(8,72)	38,10 (7,73)	88,43 (1,31)	92,42 (1,91)	90,37 (1,02)

Tabela 7.6: Medidas de avaliação para a base de dados KC2, nível de método

			Defeituoso			Não defeituoso		
Algoritmo	**AUC**	**Acurácia**	**Precisão**	**Recall**	**F-*measure***	**Precisão**	**Recall**	**F-*measure***
MOPSO	84,77 (4,46)	84,62 (3,05)	83,03 (17,67)	32,28(11,19)	45,48 (12,68)	84,93 (2,30)	98,10 (1,99)	91,02 (1,78)
Redes Neurais	84,53 (3,79)	83,92 (4,80)	70,08 (18,22)	43,60(12,86)	52,35 (13,05)	86,68 (2,83)	94,31 (4,84)	90,27 (3,09)
Bayes Net	85,02 (3,58)	81,43 (4,64)	55,02 (9,43)	74,81(6,95)	62,76 (6,09)	92,78 (1,83)	83,12 (6,23)	87,55 (3,50)
Naive Bayes	85,06 (4,14)	84,87 (4,95)	71,40 (17,56)	47,45(10,71)	56,34 (12,83)	87,48 (2,22)	94,46 (5,01)	90,79 (3,25)
SVM	64,60 (5,01)	82,75 (2,50)	70,04 (14,23)	33,87 (11,44)	43,48 (11,32)	84,90 (2,03)	95,33 (3,25)	89,76 (1,61)
C45	77,55 (4,83)	84,10 (2,67)	73,73 (15,25)	37,45(5,99)	49,17 (7,20)	85,62 (1,54)	96,13 (2,44)	90,56 (1,65)
C45 NP	77,91 (4,71)	82,38 (2,79)	59,89 (25,06)	36,36(15,21)	43,70 (15,95)	85,31 (2,46)	94,21 (4,68)	89,43 (1,84)
RIPPER	70,67 (6,57)	84,49 (3,30)	74,51 (18,70)	46,36 (15,74)	53,86 (11,65)	87,42 (2,77)	94,21 (5,14)	90,56 (2,22)
NNge	68,06 (5,99)	78,53 (5,70)	50,83 (11,48)	50,36 (14,41)	48,59 (10,11)	87,18 (2,60)	85,77 (8,54)	86,19 (4,45)

Tabela 7.7: Medidas de avaliação para a base de dados CM1

			Defeituoso			Não defeituoso		
Algoritmo	**AUC**	**Acurácia**	**Precisão**	**Recall**	**F-*measure***	**Precisão**	**Recall**	**F-*measure***
MOPSO	76,36 (9,61)	77,79 (7,03)	25,07 (11,19)	57,5(22,31)	34,64 (14,53)	94,40 (3,18)	80,05 (6,04)	86,56 (4,56)
Redes Neurais	69,78 (13,97)	89,58 (1,13)	2,75 (9,73)	1,6(5,43)	1,98 (6,78)	90,24 (0,65)	99,18 (1,59)	94,49 0,65)
Bayes Net	78,59 (10,15)	74,89 (6,47)	23,39 (9,24)	64,00 (21,57)	33,88 (12,03)	94,98 (3,00)	76,15 (6,11)	84,42 (4,33)
Naive Bayes	78,59 (10,15)	74,89 (6,47)	23,39 (9,24)	64,00 (21,57)	33,88 (12,03)	94,98 (3,00)	76,15 (6,11)	84,42 (4,33)
SVM	50,0 (0,0)	90,16 (0,56)	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	90,16 (0,56)	100,0 (0,0)	94,82 (0,30)
C45	50,0 (0,0)	90,16 (0,56)	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	90,16 (0,56)	100,0 (0,0)	94,82 (0,30)
C45 NP	54,72 (14,16)	89,55 (0,84)	3,33 (10,01)	2,0 (6,01)	2,5 (7,51)	90,27 (0,66)	99,10 (1,48)	94,47 (0,49)
RIPPER	50,55 (2,45)	89,55 (0,84)	3,33 (10,01)	2,0 (6,01)	2,5 (7,51)	90,27 (0,66)	99,10 (1,48)	94,47 (0,49)
NNge	49,99 (4,50)	83,75 (6,33)	3,24 (6,56)	8,0 (16,02)	4,60 (9,26)	90,23 (1,15)	91,98 (8,39)	90,87 (4,15)

Tabela 7.8: Medidas de avaliação para a base de dados PC1

			Defeituoso			Não defeituoso		
Algoritmo	**AUC**	**Acurácia**	**Precisão**	**Recall**	**F-*measure***	**Precisão**	**Recall**	**F-*measure***
MOPSO	78,84 (5,83)	93,15 (1,00)	33,81 (43,36)	7,08(9,68)	11,32 (14,75)	93,49 (0,77)	99,57 (0,52)	96,43 (0,52)
Redes Neurais	77,75 (5,78)	93,44 (1,12)	61,66 (34,75)	15,41(10,20)	23,67 (13,67)	94,02 (0,82)	99,26 (0,70)	96,57 (0,59)
Bayes Net	78,20 (5,44)	92,15 (2,38)	44,64 (15,07)	25,89 (9,57)	31,41 (9,80)	94,62 (0,66)	97,09 (2,68)	95,82 (1,36)
Naive Bayes	75,56 (4,46)	89,17 (2,13)	28,34 (9,68)	32,32 (12,40)	28,80 (7,65)	94,90 (0,83)	93,41 (2,80)	94,12 (1,24)
SVM	50,0 (0,0)	93,05 (0,40)	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	93,05 (0,40)	100,0 (0,0)	96,40 (0,21)
C45	67,82 (11,96)	93,59 (1,58)	54,66 (37,12)	20,71 (15,32)	29,46 (20,53)	94,37 (1,08)	99,03 (0,86)	96,64 (0,82)
C45 NP	76,66 (13,74)	93,05 (1,50)	48,33 (27,88)	23,57 (15,19)	30,53 (17,56)	94,51 (1,14)	98,25 (1,13)	96,34 (0,79)
RIPPER	57,83 (7,27)	93,05 (0,89)	35,71 (25,77)	16,96 (15,35)	22,24 (17,92)	94,10 (1,09)	98,73 (0,87)	96,36 (0,46)
NNge	65,32 (7,71)	93,41 (1,33)	53,64 (24,64)	32,67 (15,53)	39,30 (16,47)	95,12 (1,18)	97,96 (0,91)	96,51 (0,69)

Tabela 7.9: Medidas de avaliação para a base de dados JM1

			Defeituoso			Não defeituoso		
Algoritmo	**AUC**	**Acurácia**	**Precisão**	**Recall**	**F-*measure***	**Precisão**	**Recall**	**F-*measure***
MOPSO	71,81 (1,86)	80,90 (0,22)	74,12 (16,93)	2,23(0,79)	4,32 (1,50)	80,96 (0,14)	99,77 (0,19)	89,39 (0,12)
Redes Neurais	71,24 (1,87)	80,94 (0,52)	58,51 (11,65)	9,78(6,35)	15,71 (7,96)	81,93 (0,83)	98,01 (1,78)	89,24 (0,39)
Bayes Net	71,62 (1,81)	75,77 (0,82)	39,09 (1,85)	45,25 (3,47)	41,91 (2,37)	86,35 (0,70)	83,09 (1,01)	84,69 (0,55)
Naive Bayes	66,91 (1,79)	80,39 (0,58)	48,59 (3,15)	22,31 (1,94)	30,55 (2,26)	83,50 (0,35)	94,32 (0,57)	88,58 (0,35)
SVM	52,70 (1,19)	80,82 (0,58)	52,72 (10,48)	6,84 (2,30)	12,06 (3,84)	81,52 (0,40)	98,57 (0,37)	89,23 (0,31)
C45	66,63 (1,43)	80,96 (0,45)	52,90 (4,24)	14,61 (4,41)	22,51 (6,35)	82,55 (0,58)	96,87 (1,03)	89,14 (0,27)
C45 NP	69,40 (1,43)	79,84 (0,70)	44,66 (4,96)	19,51 (4,74)	26,91 (5,83)	83,01 (0,66)	94,31 (1,24)	88,29 (0,44)
RIPPER	56,46 (1,61)	81,22 (0,46)	55,31 (4,30)	15,985 (3,87)	24,51 (4,58)	82,78 (0,57)	96,87 (0,87)	89,27 (0,27)
NNge	59,91 (1,51)	76,90 (0,85)	38,42 (2,60)	32,19 (2,90)	35,00 (2,68)	84,34 (0,58)	87,62 (0,77)	85,95 (0,53)

Para a base KC2, os resultados são similares aos obtidos na KC1. Novamente, MOPSO obteve um valor muito bom de AUC. Ele obteve o melhor valor de AUC, junto com outros algoritmos. O NB obteve o melhor valor de acurácia, porém o MOPSO obteve um valor muito próximo. Para as medidas obtidas para classe defeituosa, o MOPSO obteve o melhor valor de precisão. O BN obteve o melhor *recall* e F-*measure*. Para a classe não defeituosa, o BN obteve o melhor valor de precisão, o MOPSO o melhor valor de *recall* e a RN o melhor valor de F-*measure*.

A base de dados CM1 é a única base que o MOPSO não apresentou o melhor valor de AUC. O algoritmo BN obteve o melhor valor de AUC, mas o MOPSO atingiu um valor muito próximo. Os melhores valores de acurácia foram obtidos pelo SVM e pelo C4.5. No entanto, o MOPSO obteve valores muito bons para as demais medidas, especialmente as derivadas da classe com valor "defeituoso". Ele obteve os melhores valores de precisão e F-*measure*. O BN obteve o melhor valor de *recall*. Para as medidas da classe "não defeituoso", o algoritmo proposto e o BN obtiveram os melhores valores de precisão, enquanto o SVM, C4.5 e o RIPPER obtiveram os melhores valores de *recall* e F-*measure*. A análise desta base de dados ilustra os problemas de usar somente a acurácia para verificar os resultados da classificação. Os algoritmos que possuem os melhores valores de acurácia possuem um valor pobre considerando as medidas da classe "defeituoso"e a AUC. CM1 é a base de dados mas desbalanceadas, dentre as escolhidas. Portanto, a influência da classe minoritária no cálculo da acurácia é baixa. Os melhores valores de acurácia foram obtidos por algoritmos que têm boas medidas para a classe majoritária (não defeituoso). Porém, quando os resultados das medidas da classe minoritária são analisadas, estes algoritmos apresentam resultados ruins. SVM e C4.5 possui todas a medidas da classe com valor "defeituoso"igual a 0. Então, apesar destes algoritmo possuírem bons valores de acurácia, eles não identificam nenhum método defeituoso, o objetivo da predição. Por outro lado, os algoritmos MOPSO e BN, que possuem os melhores valores de AUC e os piores valores de acurácia, possuem bons valores para todas as medidas.

Para a base de dados PC1, novamente o MOPSO obteve o melhor valor de AUC, juntamente com o C4.5 Sem Poda. Quase todos os algoritmos obtiveram valores de

acurácia próximos, mas a RN, o C4.5 e o NNge obtiveram os melhores resultados. Nas medidas para a classe com o valor "defeituoso", o MOPSO obteve resultados similares aos encontrados na base KC1. Valores médios de precisão e *recall*, ou seja, não produz classificações erradas, mas tem dificuldade na identificação dos defeitos. A Rede Neural obteve o melhor valor da precisão, o NNge os melhores valores de *recall* e F-*measure*. Nas medidas da classe com valor "não defeituoso", todos os algoritmos obtiveram bons valores. O NNge obteve o melhor valor de precisão, SVM o melhor valor de *recall*, e cinco algoritmos obtiveram os melhores valores de F-*measure*, incluindo o MOPSO.

Para a base de dados JM1, o MOPSO obteve resultados muito bons. Ele obteve o melhor valor de AUC e um bom valor de acurácia. O RIPPER apresentou o melhor valor de acurácia. Para as medidas da classe com métodos defeituosos, o MOPSO obteve o melhor valor de precisão e o BN obteve os melhores valores de *recall* e F-*measure*. Para os métodos não defeituosos, novamente, o BN obteve o melhor valor da precisão. Por fim, o MOPSO obteve os melhores valores de F-*measure* e *recall*.

Em resumo, o algoritmo MOPSO obteve bons resultados na predição em bases de dados com métricas de métodos. Em quase todas as bases de dados, ele obteve o melhor valor de AUC, a medida principal utilizada nesta análise. Ele também obteve bons resultados de acurácia. Para as medidas obtidas para a classe dos métodos defeituosos, o MOPSO obteve bons resultados na predição, embora em algumas bases de dados ele tenha obtido um valor baixo de *recall*. Quando analisando a classe dos métodos não defeituosos, o MOPSO obteve melhores resultados na maioria dos casos. Concluindo, o algoritmo MOPSO produz poucos erros na predição de defeitos, porém apresenta dificuldade para classificar métodos defeituosos. A Figura 7.1 apresenta um resumo da comparação de todos os algoritmos. A figura mostra quantas vezes cada algoritmo obteve o melhor resultado para cada medida, em todas as bases de dados. Pode-se observar que o MOPSO é bastante competitivo com os demais algoritmos, que são largamente utilizados no contexto da predição de defeitos em software, como as Redes Neurais e as Redes Bayesianas.

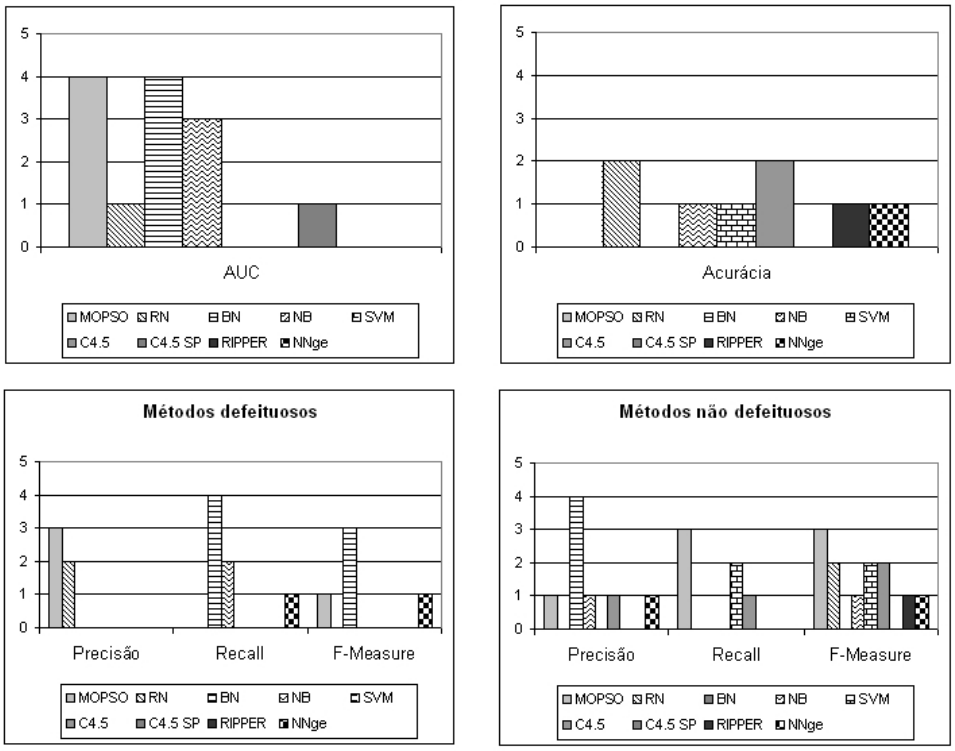

Figura 7.1: Resumo da comparação de todos os algoritmos

7.2.2 Comparação dos Algoritmo de Aprendizado de Regras

A seção anterior mostrou que o MOPSO é bastante competitivo quando comparado à algoritmos como as Redes Bayesianas e Neurais. No entanto, uma característica importante no MOPSO é que os seus resultados são mais simples de interpretar que os obtidos pelo BN e a RN. Neste sentido, é interessante comparar o MOPSO com algoritmos de Aprendizado de Regras como o C4.5, o C4.5 Sem Poda o RIPPER e o NNge. Esta investigação é uma comparação mais justa, pois todos os algoritmos induzem um mesmo tipo de modelo. Utilizando-se modelos de classificação baseados em regras é possível identificar algumas características sobre as predições que em classificações em "caixa-preta", como no BN e na RN, são difíceis de obter, como por exemplo, como cada decisão específica da classificação foi alcançada.

A Figura 7.2 apresenta um resumo considerando somente os algoritmos de Aprendizado de Regras. Esta figura mostra quantas vezes cada algoritmo obteve o melhor resultado para cada figura. Pode ser observado que o MOPSO obteve o melhor valor de AUC para todos as bases de dados em relação aos demais algoritmos, e obteve bons valores de

acurácia também.

Os dois primeiros gráficos mostram que o MOPSO obteve o melhor valor de AUC para todas as 5 bases de dados. Os melhores valores de acurácia estão distribuídos entre todos algoritmos. O terceiro gráfico mostra o resumo dos resultados das medidas obtidos para a classe dos métodos defeituosos. O MOPSO obteve o melhor valor de precisão em quatro base de dados, C4.5 SP em duas e os demais obtiveram o melhor resultado em uma base. Para o *recall* o inverso é observado, MOPSO foi melhor em somente uma base, enquanto o NNge foi melhor em 4 bases de dados. A análise do F-*measure* é similar ao *recall*. MOPSO e o RIPPER são melhores em uma base de dados, enquanto o NNge é melhor em três.

Por fim, o quarto gráfico mostra o resumo das medidas da classe com métodos não defeituosos. Os resultados são opostos aos resultados da outra classe. NNge obteve o melhor valor de precisão em 4 bases de dados, enquanto o MOPSO e o RIPPER foram melhores em uma base. O MOPSO obteve os melhores valores de *recall* em 4 bases, enquanto o C4.5 foi melhor em uma. Para o F-*measure*, o MOPSO foi melhor em 4 bases, o C4.5 e o RIPPER em duas e NNge em uma.

Concluindo, quando comparando somente os algoritmos de aprendizado de regra, o MOPSO obteve um resultado muito bom. Ele obteve os melhores valores de AUC e foi bastante competitivo em acurácia, as duas principais medidas em análise neste trabalho.

7.2.3 Obtendo uma predição a partir do MOPSO

Esta seção apresenta um exemplo de como o MOPSO pode ser usado por um desenvolvedor de software num ambiente de testes. Um módulo da base de dados KC2 é submetido ao MOPSO e o resultado da predição é avaliado. Neste exemplo o MOPSO irá treinar com as instâncias do KC2, exceto o módulo que será predito. O módulo é apresentado na Tabela 7.10. Suponha que a informação de defeito do módulo não seja conhecida. Note que o exemplo foi escolhido para mostrar como a predição pode ser executada com o MOPSO.

Na predição, o módulo é submetido ao MOPSO sem a informação da presença do

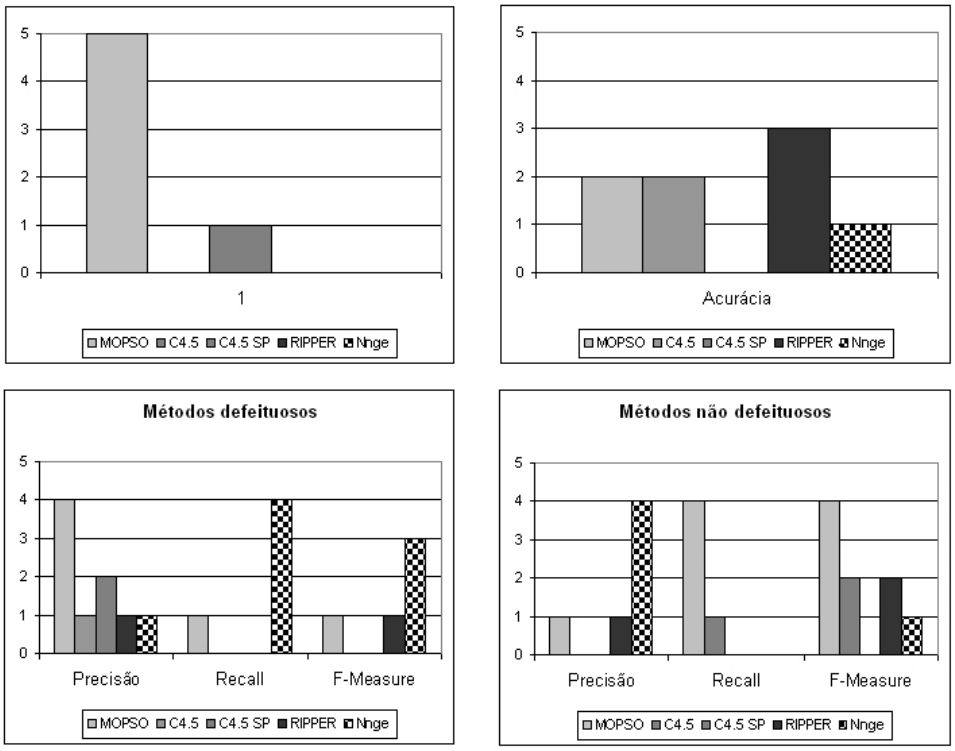

Figura 7.2: Comparação dos algoritmo de Aprendizado de Regras

Tabela 7.10: Módulo selecionado.

I	B	EV(g)	UniqOp	UniqOpnd	Classe
6	83,01	0,88	18	50	defeituoso

defeito. Como discutido anteriormente, o MOPSO irá aprender um conjunto de regras através do conjunto de entrada. Após o processo de aprendizado, o algoritmo irá usar o conjunto de regras para predizer se o módulo contém ou não defeito. Esta predição é executada através do processo de votação apresentado na Seção 3.3.3. O método utilizado foi a Votação por Laplace.

Este processo de votação não é apresentado ao engenheiro de software, apenas as regras que votaram no exemplo são apresentadas. Para o exemplo discutido, o MOPSO classificou o módulo como defeituoso e as regras que votaram este exemplo como defeituoso são apresentadas na Tabela 7.11. Através destas regras é possível construir conclusões mais gerais sobre a predição (veja a Seção 7.3.1) ou usar as medidas de qualidade da regra, apresentadas na Seção 3.3.2, para garantir o resultado da predição.

Neste exemplo percebe-se que ambas regras da Tabela 7.11 possuem valores similares para as métricas EV(g) e UniqOpnd e estas métricas parecem ser determinantes para a

Tabela 7.11: Regras utilizadas na predição.

Regra						Medidas			
I	**B**	**EV(g)**	**UniqOp**	**UniqOpnd**	**Classe**	**Sens**	**Espec**	**Prec**	**Sup**
?	?	4,18<EV(g)<23	?	39,14<UniqOpnd < 268,47	defeituoso	0,30	0,98	0,83	0,04
?	?	4,08<EV(g)<66.02	9,48<UniqOp < 31.23	33,38<UniqOpnd< 92.88	defeituoso	0,23	0,99	0,88	0,06

predição do módulo como defeituoso. Uma segunda análise que pode ser obtida através das regras é sua qualidade. Ambas as regras possuem um alto valor de especificidade, assim as regras não fazem muitos erros na classificação. Além disso, a precisão das regras é alta, assim maior é a probabilidade da regra ter feito uma predição correta.

7.3 Predizendo Defeitos em Classes

O segundo experimento executado foi com as bases de dados com métricas de classe. Primeiro, todos os algoritmos apresentados no experimento anterior são novamente avaliados, utilizando-se a mesma metodologia. Após, é feita uma análise empírica nas regras geradas pelo MOPSO para identificar relacionamentos interessantes entre as métricas e a tendências à defeitos nas classes. Este estudo é apresentado na Seção 7.3.1.

A Tabela 7.12 apresenta os resultados das medidas de avaliação usadas nesta comparação. Novamente os resultados representam a média da execução dos algoritmos para todas as partições e os números entre parênteses indicam o desvio padrão. Para medição das diferenças foi utilizado o teste estatístico de Wilcoxon com 5% de confiança. Na tabela, as células marcadas com cinza escuro indicam os melhores valores na comparação entre todas as técnicas, enquanto as células marcadas com cinza claro indicam o melhor resultado entre os algoritmos de Aprendizado de Regras. Nesta comparação foi utilizada somente uma base de dados, projeto KC1 da NASA, com as métricas especificadas na Tabela 7.2.

Para esta base de dados, o MOPSO obteve resultados bastante competitivos, embora não tenha obtido o melhor resultado em nenhuma medida. Para AUC, o C4.5 Sem Poda obteve o melhor resultado, mas o MOPSO obteve um resultado bastante próximo. O C4.5 Sem Poda e o SVM obtiveram os melhores valores de acurácia. Para as medidas para as classe defeituosas, novamente os dois algoritmos obtiveram os melhores valores de

precisão. O BN obteve os melhores valores de *recall* e F-*measure*. Para as medidas das classes não defeituosas, todos os algoritmos obtiveram resultados similares. O algoritmo BN obteve a melhor precisão, o C4.5 o melhor *recall* e o RIPPER e a RN obtiveram os melhores valores de F-*measure*. A análise dos algoritmo de Aprendizado de Regras foi similar. O C4.5 SP obteve o melhor valor de AUC, RIPPER a melhor acurácia, a melhor precisão para as classes defeituosas e o melhor *recall* para as classes não defeituosas. O NNge obteve os melhores resultados para as demais medidas.

Em resumo, para a predição de defeitos em classes, através de métricas de projeto OO, o MOPSO é bastante competitivo com os demais algoritmos da literatura. Ele obteve um bom valor de AUC e acurácia, as medidas principais em análise neste trabalho.

7.3.1 Conjunto de Métricas CK e tendência a defeitos

As regras geradas pelo algoritmo MOPSO são simples e intuitivas. Isto auxilia os programadores a estabelecer hipóteses para métricas do conjunto CK no contexto da tendência a defeitos de uma classe e na seleção das classes a serem testadas. Nesta estratégia, o programador tem o interesse em detectar corretamente o defeito em classes.

Neste sentido, este trabalho utiliza a seguinte estratégia: na predição de defeitos uma regra que possui um alto valor de especificidade (3.6) é mais desejável que uma regra com maior sensitividade (3.5), pois assim predições errôneas não serão efetuadas.

Tabela 7.12: Medidas de avaliação para a base de dados KC1, nível de classe

			Defeituoso			Não defeituoso		
Algoritmo	**AUC**	**Acurácia**	**Precisão**	**Recall**	**F-*measure***	**Precisão**	**Recall**	**F-*measure***
MOPSO-N	73,22 (4,25)	79,92 (14,01)	20,60 (24,63)	27,27 (31,28)	16,99 (16,64)	90,40 (2,85)	87,09 (19,64)	86,96 (12,69)
Neural Network	71,21 (4,69)	88,03 (0,51)	9,95 (20,86)	3,90(8,73)	5,47 (11,89)	88,39 (0,89)	99,48 (1,12)	93,60 (0,28)
Bayesian Network	72,83 (5,25)	85,20 (2,51)	38,57 (8,54)	36,85 (9,45)	37,31 (8,29)	91,44 (1,24)	91,78 (2,43)	91,59 (1,50)
Naive Bayes	69,05 (4,51)	82,43 (2,18)	11,41 (7,62)	6,45 (5,13)	7,71 (5,36)	87,93 (0,52)	92,77 (2,70)	90,27 (1,33)
SVM	31,13 (3,87)	88,50 (1,08)	53,73 (8,31)	25,13 (7,73)	33,94 (8,63)	90,50 (0,93)	97,13 (0,64)	93,70 (0,57)
C45	61,12 (9,53)	87,87 (0,29)	28,01 (23,05)	9,35 (7,99)	13,96 (11,77)	88,88 (0,83)	98,56 (1,18)	93,46 (0,18)
C45 NP	74,17 (4,62)	88,40 (1,07)	52,94 (8,04)	26,78 (8,37)	35,09 (8,57)	90,66 (1,02)	96,80 (0,91)	93,62 (0,57)
RIPPER	57,44 (5,62)	87,91 (1,25)	38,42 (21,48)	17,36 (12,08)	23,54 (14,99)	89,68 (1,26)	97,51 (1,50)	93,42 (0,69)
NNge	60,17 (6,84)	83,98 (1,45)	29,32 (9,39)	28,83 (12,18)	28,32 (12,18)	90,48 (1,78)	91,50 (2,61)	90,94 (0,91)

Tabela 7.13: Melhores regras aprendidas para a base de dados KC1, nível de classe

	Regra							Medidas			
#	**CBO**	**DIT**	**LCOM**	**NOC**	**RFC**	**WMC**	**Classe**	**Sens**	**Espec**	**Prec**	**Sup**
1	?	?	66<LCOM<98	?	27 < RFC < 157	?	faulty	0,700	0,541	0,172	0,083
2	?	?	?	?	37<RFC<152	?	faulty	0,672	0,561	0,172	0,080
3	?	?	?	NOC≤2	?	21 < WMC < 99	faulty	0,668	0,573	0,175	0,080
4	6 < CBO ≤ 24	?	?	?	35 < RFC < 134	?	faulty	0,635	0,610	0,181	0,076
5	?	?	89<LCOM<96	?	9 < RFC ≤ 222	?	faulty	0,603	0,658	0,194	0,072
6	?	?	?	?	?	27 < WMC < 99	faulty	0,574	0,717	0,216	0,068
7	8 < CBO ≤ 24	?	?	?	?	28 < WMC < 99	faulty	0,514	0,778	0,240	0,061
8	?	?	?	NOC<1	65 < RFC < 176	?	faulty	0,481	0,803	0,250	0,057
9	?	?	93<LCOM≤100	?	73 < RFC ≤ 222	?	faulty	0,417	0,869	0,303	0,049
10	?	?	?	?	72<RFC<99	?	faulty	0,331	0,911	0,338	0,03

No entanto, isto não descarta a importância da sensitividade, pois é interessante que os módulos com defeitos sejam identificados para reduzir os custos na tarefa de teste. Assim, a análise das regras deve considerar um balanço entre altos valores de especificidade e bons valores de sensitividade. Além disso, o suporte (3.8) e a confiança (3.1), ou precisão, são importantes medidas para medir diferenças em regras com resultados equivalentes utilizando a análise proposta. Este conhecimento extraído pode ser usado por um gerente de software para eficientemente alocar recursos de teste para módulos com maior tendência à defeito e, aumentar a qualidade dos módulos de software produzidos.

Esta análise usa as medidas apresentadas na Seção 3.3.2.2 para definir a significância de cada métrica. As regras induzidas pelo MOPSO permitem facilmente executar a análise apresentada, já que, a sensitividade e a especificidade são objetivos da busca utilizados neste trabalho. Se uma métrica é parte de uma regra que possui bons valores para as medidas em análise, ela é dita significante. As melhores regras foram escolhidas de acordo com suas medidas. Regras que possuem um valor baixo de sensitividade e especificidade abaixo de um determinado limiar são descartadas. Estes limiares foram definidos empiricamente com o valor $0,3$ para a sensitividade e $0,5$ para a especificidade. Como foi dito, é melhor ter regras que cubram poucos exemplos ao invés de regras que cubram muitos exemplos de forma errada. Foram analisadas um total de 53 regras, obtidas através da execução do MOPSO e as melhores regras selecionadas que deram uma maior quantidade de informação são apresentadas na Tabela 7.13. O resultado da influência de cada métrica de acordo com a metodologia discutida é apresentado a seguir.

RFC e WMC são as métricas mais importantes para indicar defeitos em classes, de acordo com as regras 6 e 10. Valores nos intervalos apresentados nas regras implicam numa maior probabilidade de defeitos. Estas regras são as únicas com somente um atributo, boa sensitividade e altos valores de especificidade, o que significa que as regras não cobrem muitos exemplos não defeituosos. Observando as demais regras, pode-se concluir que altos valores de CBO e LCOM, associados com altos valores de RFC e WMC, estão relacionados com a presença de defeitos nas classes. Considerando a métrica NOC, existe uma regra que a associa às métricas RFC e WMC. Esta regra é muito específica e cobre poucos

exemplos. Não existem regras relacionadas à métrica DIT, assim o seus valores não são significantes para a presença de defeitos em classes.

CAPÍTULO 8

CONCLUSÕES

Neste trabalho, foi apresentado o desenvolvimento de um algoritmo para a solução do problema do Aprendizado de Regras no contexto da Mineração de Dados baseado na metaheurística da Otimização por Nuvem de Partículas Multiobjetivo, MOPSO. O aprendizado indutivo de regras é bastante importante na área de Mineração de Dados, pois regras são uma das formas mais utilizadas para a representação do conhecimento extraído. Além disso, o conjunto de regras pode ser usado como um classificador não-ordenado. A técnica MOPSO é uma Metaheurística Multiobjetivo, uma nova abordagem para a construção de classificadores, nos quais as propriedades das regras podem ser avaliadas através de diferentes objetivos.

No desenvolvimento do algoritmo MOPSO buscou-se diagnosticar alguns problemas que surgem na aplicação da técnica no contexto da Mineração de Dados. Um dos principais problemas é a falta de expressividade das regras geradas. Neste sentido, foi adicionado ao algoritmo a possibilidade da geração de regras a partir de dados numéricos, sem a necessidade de uma etapa de discretização. Assim, as regras representam o conjunto de dados original, sem a execução de uma etapa de pré-processamento dos dados. Esta extensão do algoritmo também teve como objetivo melhorar os resultados na classificação. Os resultados da validação desta extensão mostram que a técnica MOPSO com dados numéricos produz um classificador preciso em termos de AUC. Além disso, o conjunto de regras geradas alcança melhores valores nos objetivos do que usar os dados discretizados.

Uma outra solução para a falta de expressividade das regras, foi a implementação do operador *IN* que possibilita a combinação de valores para os atributos nominais. Desta maneira, as restrições para os atributos nominais guardam informações que originalmente só podem ser obtidas por um conjunto de regras. A validação mostrou que o algoritmo não perde performance quando o operador é aplicado. Por fim, para melhorar a expressi-

vidade de todo o conjunto de regras foi proposto o procedimento de Remoção de Regras Específicas que é usado para reduzir o número total de regras geradas. Nos resultados apresentados, o classificador final obteve uma redução significativa nas regras e não houve alteração nos resultados da classificação em termos de AUC.

Um outro problema diagnosticado é em relação à qualidade do classificador gerado. Como já dito, a extensão para trabalhar com dados numéricos teve como objetivo solucionar este problema. Uma segunda extensão está relacionada com o processo de votação de novos exemplos efetuado pelo classificador formado pelas regras. Foram apresentadas três soluções: Votação Simples, Votação por Precisão e Votação por Laplace. Nos experimentos foi mostrado que a Votação por Laplace atingiu os melhores resultados e quando aplicada ao MOPSO produz bons resultados comparados com métodos conhecidos da literatura.

O terceiro problema estudado foi em relação à execução do algoritmo em bases de dados de grande porte. Para isto foi apresentada a proposta de paralelização do algoritmo MOPSO, que possibilita que a técnica seja usada em problemas reais da MD. Para a validação desta proposta foram apresentados um conjunto de experimentos que mostram que a técnica consegue obter resultados muito bons de AUC e que tem a possibilidade de executar mais rápido do que o algoritmo sequencial.

Além dos experimentos de validação das propostas, o algoritmo MOPSO com todas as extensões implementadas (trabalhar com dados numéricos, execução do procedimento Remover Regras Específicas, execução do operador *IN* e execução da Votação por Laplace) foi comparado com alguns algoritmos de Aprendizado de Regras conhecidos da literatura. O MOPSO foi comparado com o C4.5, C4.5 Sem Poda, o NNge e o RIPPER em termos de AUC e acurácia. Os resultados mostraram que o algoritmo proposto neste trabalho consegue obter resultados muito bons de AUC e é muito competitivo em termos de acurácia. Além desta comparação, o MOPSO foi avaliado utilizando-se uma abordagem multiobjetivo e os resultados mostraram que a técnica desenvolvida produz um conjunto de regras muito bom.

Por fim, foi apresentado um estudo de caso em que o algoritmo MOPSO desenvolvido

foi aplicado no contexto da predição de defeitos em softwares. O algoritmo é utilizado como um preditor de defeitos em novos módulos a partir de métricas de software. Esta tarefa se mostra muito importante, pois uma predição correta pode reduzir os custos associados à tarefa de teste de software.

O algoritmo foi aplicado em dois contextos: predição de defeitos em métodos e em classes. Nos dois contextos os resultados do algoritmo foram comparados com técnicas geralmente utilizadas nos trabalhos de predição de defeitos e se mostrou uma técnica promissora. Além da análise dos resultados da predição foi feita uma análise empírica da influência de cada métrica do conjunto CK na existência de defeito na classe.

Como trabalhos futuros há a investigação da técnica MOPSO com novos objetivos. A introdução de outros objetivos adiciona complexidade na validação do algoritmo e aumenta a complexidade na geração das soluções mas pode introduzir uma quantidade maior de informação na geração das regras e assim melhorar o conjunto gerado.

Além disso, há a investigação de novos métodos para a redução do número total de regras geradas e a extensão do algoritmo para trabalhar com mais de duas classes. Em relação à proposta de paralelização do algoritmo, o estudo pode ser estendido para a execução num ambiente paralelo e há a necessidade de mapeamento de novos problemas como de sincronização e balanceamento de carga.

BIBLIOGRAFIA

[1] A. Asuncion e D. J. Newman. UCI Machine Learning Repository, [http://www.ics.uci.edu/∼mlearn/MLRepository.html]. Irvine, CA: University of California, School of Information e Computer Science, 2007.

[2] Jaume Bacardit e Josep Maria Garrell. *Bloat Control and Generalization Pressure Using the Minimum Description Length Principle for a Pittsburgh Approach Learning Classifier System*, volume 4399/2007, páginas 59–79. Springer Berlin / Heidelberg, 2007.

[3] Stefan Bleuler, Marco Laumanns, Lothar Thiele, e Eckart Zitzler. PISA — A platform and programming language independent interface for search algorithms. *Evolutionary Multi-Criterion Optimization (EMO 2003)*, Lecture Notes in Computer Science, páginas 494 – 508, Berlin, 2003. Springer.

[4] Stefan Bleuler, Marco Laumanns, Lothar Thiele, e Eckart Zitzler. The pisa homepage. http://www.tik.ee.ethz.ch/pisa/, 2003.

[5] D. Bratton e J. Kennedy. Defining a standard for particle swarm optimization. *Swarm Intelligence Symposium,*, páginas 120–127, 2007.

[6] André B. de Carvalho e Aurora Pozo. *Mining Rules: A Parallel Multiobjective Particle Swarm Optimization Approach.* Springer, 2008.

[7] André B. de Carvalho e Aurora Pozo. Non-ordered data mining rules through multi-objective particle swarm optimization: Dealing with numeric and discrete attributes. *Poceedings of Hybrid Intelligent Systems, 2008. HIS '08. Eighth International Conference*, páginas 495–500, 2008.

[8] André B. de Carvalho e Aurora Pozo. *A Non-Ordered Rule Induction Algorithm Through Multi-Objective Particle Swarm Optimization: Issues and Applications.* Springer-Verlag, 2008.

[9] André B. de Carvalho e Aurora Pozo. Um algoritmo multiobjetivo de aprendizado de regras baseado na otimização por nuvem de partículas. *II Workshop on Computational Intelligence (WCI)*, 2008.

[10] André B. de Carvalho, Aurora Pozo, Silvia Vergilio, e Alexandre Lenz. Predicting fault proneness of classes trough a multiobjective particle swarm optimization algorithm. *Poceedings of 20th IEEE International Conference on Tools with Artificial Intelligence*, 2008.

[11] S. R. Chidamber e C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, 1994.

[12] W. J. Conover. *Practical nonparametric statistics*. Wiley, 1971.

[13] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, 2006.

[14] Karim O. Elish e Mahmoud O. Elish. Predicting defect-prone software modules using support vector machines. *J. Syst. Softw.*, 81(5):649–660, 2008.

[15] Tom Fawcett. Using rule sets to maximize ROC performance. *IEEE International Conference on Data Mining*, páginas 131–138. IEEE Computer Society, 2001.

[16] Tom Fawcett. Roc graphs: Notes e practical considerations for researchers. 2004.

[17] Usama M. Fayyad e Keki B. Irani. Multi-interval discretization of continuousvalued attributes for classification learning. *Thirteenth International Joint Conference on Articial Intelligence*, páginas 1022–1027, 1993.

[18] Norman Fenton, Martin Neil, William Marsh, Peter Hearty, David Marquez, Paul Krause, e Rajat Mishra. Predicting software defects in varying development lifecycles using bayesian nets. *Inf. Softw. Technol.*, 49(1):32–43, 2007.

[19] Alex A. Freitas. A critical review of multi-objective optimization in data mining: a position paper. *University of Kent, UK*, 2004.

[20] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, first edition, 1986.

[21] Waikato Machine Learning Group. Weka machine learning project, http://www.cs.waikato.ac.nz/ml/weka, 2007.

[22] M. Hall. Correlation-based feature selection for machine learning, 1998.

[23] Michael Pilegaard Hansen e Andrzej Jaszkiewicz. Evaluating the quality of approximations to the non-dominated set. Relatório Técnico IMM-REP-1998-7, Technical University of Denmark, março de 1998.

[24] Celso Ishida. *Explorando Abordagens Inovadoras para Geração de Classificadores*. Tese de Doutorado, Universidade Federal do Paraná, Agosto de 2008.

[25] Celso Ishida, André B. de Carvalho, e Aurora Pozo. Exploring Multi-objective PSO and GRASP-PR for rule induction. *Poceedings of EvoCOP 2008*, páginas 73–84. LNCS 4972. Springer Berlin / Heidelberg, 2008.

[26] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, e K. R. K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 13(3):637–649, 2001.

[27] J. Kennedy e R. C. Eberhart. Particle swarm optimization. *IEEE International Conference on Neural Networks*, páginas 1942–1948. IEEE Press, 1995.

[28] Joshua Knowles, Lothar Thiele, e Eckart Zitzler. A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers. 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland, fevereiro de 2006. revised version.

[29] N. Lavrac, P. Flach, e B. Zupan. Rule evaluation measures: A unifying view. S. Dzeroski e P. Flach, editors, *Ninth International Workshop on Inductive Logic Programming (ILP'99)*, páginas 174–185. Springer-Verlag, June de 1999.

[30] Hakim Lounis e Lynda Ait-Mehedine. Machine-learning techniques for software product quality assessment. *QSIC '04: Proceedings of the Quality Software, Fourth International Conference*, páginas 102–109, Washington, DC, USA, 2004. IEEE Computer Society.

[31] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[32] M. C. Monard e J. A. Baranauskas. *Indução de Regras e Árvores de Decisão*. Editora Manole, 2003.

[33] S. Mostaghim e J. Teich. Strategies for finding good local guides in multi-objective particle swarm optimization. *SIS '03 Swarm Intelligence Symposium*, páginas 26–33. Proceedings of the 2003 IEEE Swarm Intelligence Symposium. IEEE Computer Society, 2003.

[34] Ganesh J. Pai e Joanne Bechta Dugan. Empirical analysis of software fault content and fault proneness using bayesian methods. *IEEE Transaction on Software Engineering*, 33(10):675–686, October de 2007.

[35] V. Pareto. Manuel d"economie politique, 1927.

[36] Adriano D. Pila. *Computação Evolutiva para a Construção de Regras de Conhecimento com Propriedades Específicas*. Tese de Doutorado, ICMC/USP, Abril de 2007.

[37] Ronaldo Prati. *Novas abordagens em aprendizado de máquina para a geração de regras, classes desbalanceadas e ordenação de casos*. Tese de Doutorado, ICMC/USP, julho de 2006.

[38] Ronaldo C. Prati e Peter A. Flach. ROCCER: An algorithm for rule learning based on ROC analysis. Leslie Pack Kaelbling e Alessandro Saffiotti, editors, *International Joint Conferences on Artificial Intelligence*, páginas 823–828. Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, 2005.

[39] Elena Pérez-Miñana e Jean-Jacques Gras. Improving fault prediction using bayesian networks for the development of embedded software applications: Research articles. *Softw. Test. Verif. Reliab.*, 16(3):157–174, 2006.

[40] NASA IV&V Facility Metrics Data Program. Metrics data repository, http://mdp.ivv.nasa.gov/.

[41] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1993.

[42] M. Reyes-Sierra e C. A. C. Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International Journal of Computational Intelligence Research*, 2(3), 2006.

[43] Stuart Russel e Peter Norvig. *Inteligência Artificial*. Editora Campus, second edition, 2004.

[44] Mie Mie Thet Thwin e Tong-Seng Quah. Application of neural networks for software quality prediction using object-oriented metrics. *J. Syst. Softw.*, 76(2):147–156, 2005.

[45] Augusto Toracio e Aurora Pozo. Multiple objective particle swarm for classification-rule discovery. *Proceedings of CEC 2007*, páginas 684–691. IEEE Computer Society, 2007.

[46] Augusto Toracio e Aurora Pozo. Aprendizado de regras de classificação com otimização por nuvem de particulas multiobjetivo. Dissertação de Mestrado, Universidade Federal do Paraná, March de 2008.

[47] Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[48] Qin Xin Yanbo J. Wang e Frans Coenen. A novel rule ordering approach in classification association rule mining. *International Journal of Computational Intelligence Research*, 2(3):287–308, 2006.

[49] Yuming Zhou e Hareton Leung. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *IEEE Transaction on Software Engineering*, 32(10):771–789, October de 2006.

[50] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, e da V. G. Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7:117–132, abril de 2003.

[51] Eckart Zitzler e Lothar Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.