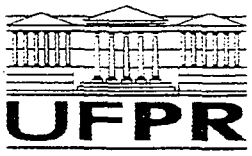
MOZART HASSE

MINERAÇÃO DE DADOS USANDO ALGORITMOS GENÉTICOS

Dissertação apresentada ao Curso de Mestrado em Informática, do Departamento de Informática, Setor de Ciências Exatas da Universidade Federal do Paraná.

Orientadora: Prof.ª Aurora T. R. Pozo

CURITIBA
2000



Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Mozart Hasse, avaliamos o trabalho intitulado **"Mineração de Dados Usando Algoritmos Genéticos"**, cuja defesa foi realizada no dia 03 de agosto de 2000. Após a Avaliação, decidimos pela Aprovação do Candidato.

Curitiba, 03 de agosto de 2000.

Profª Dra. Aurora Trinidad Ramirez Pozo
Presidente - DINF/UFPR

Prof. Dr. Alex Freitas
PUC/PR

Profª. Dra. Silvia Regina Vergílio
DINF/UFPR

Resumo

Este trabalho implementa uma ferramenta para Mineração de Dados. A ferramenta consiste em um classificador que utiliza Algoritmos Genéticos para a indução de regras. Este paradigma foi escolhido devido à grande capacidade dos algoritmos genéticos em lidar com dados inválidos ou imprecisos e a facilidade de adaptá-lo a diferentes aplicações, seja pela configuração de parâmetros ou pela implementação ou modificação de operadores.

O algoritmo genético usa a abordagem de Michigan. Nesta abordagem, o algoritmo busca por uma população inteira de regras, que são posteriormente filtradas e organizadas para formar o classificador. A implementação atual consegue tratar conjuntos de dados com atributos contínuos ou discretos, independente do domínio.

Diferentes aspectos desta ferramenta são discutidos ao longo deste trabalho. Entre eles destacam-se o uso de compartilhamento de recursos no espaço fenotípico com baixo custo computacional, a separação por classes durante a busca de regras, e o uso de um teste de significância na montagem do classificador.

A ferramenta permite a configuração de diversos parâmetros, que podem inclusive ser modificados durante a execução. A busca de regras pode ser parada em qualquer estágio, sendo também possível dedicar mais processamento a classes mais difíceis de classificar de acordo com o desejo do usuário.

A eficiência da ferramenta é comparada com 33 outros algoritmos classificadores em 32 bases de teste, usando os mesmos dados e metodologia. A precisão na classificação, medida pelo percentual de erro, não é significativamente diferente (ao nível de 10%) da precisão do melhor dos outros 33 classificadores. Os resultados obtidos até agora mostram que a ferramenta é robusta e genérica, e está pronta para uso em aplicações

reais de mineração de dados. Futuras implementações pretendem adicionar novos operadores e características a fim de tornar os resultados ainda melhores.

Índice

Índice de Figuras

1. Introdução

Até alguns anos, o principal interesse de muitas empresas no processo de informatização era simplesmente automatizar e controlar tarefas operacionais. Ao longo dos anos, porém, a quantidade, variedade e a qualidade dos dados armazenados aumentou rapidamente, dando oportunidade ao surgimento de novas necessidades dentro das empresas. A existência e a obtenção de informações úteis, com nível de abstração maior e com grande rapidez tornou-se então o novo foco de interesse empresarial [Devlin 1997].

Várias técnicas foram criadas para automatizar a descoberta de informações com grau mais alto de abstração a partir de grandes volumes de dados [Langley 1996]. O principal objetivo é obter estas informações rápido o suficiente para que possam ser usadas para a tomada de decisões estratégicas. A busca de informações deste tipo tem o nome de Mineração de Dados [Devlin 1997].

Existem diversos paradigmas para tratar este tipo de problema [Langley 1996]. Entre os mais conhecidos tem-se:

- Redes neurais, que representam conhecimento através de uma rede interligada de nodos chamados "neurônios". Os neurônios são organizados em camadas ordenadas. A primeira camada de neurônios recebe os dados a classificar, e então passa os dados processados para os neurônios da camada seguinte através de suas conexões. O resultado final é obtido analisando-se a saída da última camada de neurônios. Uma aplicação que usa redes neurais para classificação pode ser encontrada em [Kohonen 1995].

- Classificadores Bayesianos [Weiss & Kulikowski 1991] são usados para classificação em casos onde há muitos exemplos para analisar, com situações em que os mesmos valores para atributos previsores resultam em

classes diferentes. Neste caso a previsão com maior taxa de acerto será a que tiver maior probabilidade. Este paradigma contorna o alto custo de consultar todos os dados para determinar esta probabilidade através de simplificações que tornem o custo aceitável.

- O Aprendizado Baseado em Exemplos, ou Instance-Based Learning (IBL), [Aha et al. 1991] funciona usando vários casos-exemplo com classificação conhecida para classificar dados novos através de comparação de similaridades. O caso-exemplo mais parecido com o dado a classificar, segundo algum critério de comparação, indicará a classe à qual o dado pertence.

- Métodos estatísticos como o método usado em [Kooperberg et al 1997]. Estes métodos se baseiam em estimativas estatísticas sobre a frequência e distribuição dos valores dos atributos. Estas estatísticas podem ser usadas como indicadores de qual é a classe mais provável de um novo exemplo a ser classificado.

- Algoritmos de Indução, como Árvores de Decisão [Quinlan 1993], Beam Search [Holsheimer et al. 1996], CN2 [Clark & Niblett 1989], entre outros. Estes métodos baseiam-se na indução de conjuntos de regras, normalmente no formato "se <condição atendida> então <classe=x>". O conjunto de regras pode ser usado de forma ordenada (uma regra é analisada por vez, em determinada ordem) ou não ordenada (neste caso as regras têm peso e a classe mais indicada pelas regras é escolhida).

No caso do paradigma de Indução de regras, há outros algoritmos que podem ser usados para a busca de regras, e entre eles estão os Algoritmos Genéticos [Mitchell 1997], que são o foco deste trabalho.

Algoritmos Genéticos, um dos ramos da Computação Evolucionária [Mitchell 1997], são inspirados na Teoria da Evolução. Neste algoritmo cada solução é representada

como a "carga genética" de um "indivíduo". Um conjunto de soluções possíveis, chamado "população", é analisado segundo algum critério (função de aptidão ou "fitness"), e novas soluções são criadas e tentadas usando como base o conjunto anterior, através de operadores de recombinação ("cruzamento") e "mutação". A população é continuamente melhorada pela seleção dos indivíduos mais bem adaptados, e o processo de avaliação-seleção-cruzamento-mutação é repetido até que algum critério de término seja atingido. Apesar de suas vantagens, os algoritmos genéticos têm por natureza algumas limitações conhecidas na busca de informações, como não garantir que será encontrada a melhor solução possível durante sua execução, além da dificuldade de ajustar o uso de seus vários operadores para obter um desempenho ótimo para o problema em questão [Mitchell 1997].

Quando usados para descobrir informações, Algoritmos Genéticos apresentam vantagens com relação a algumas das outras técnicas descritas anteriormente, como a grande capacidade de lidar com dados inválidos ou imprecisos ("ruído" [Freitas & Lavington 1998]), o ajuste fino de parâmetros de acordo com o domínio [Mitchell, 1997] e a possibilidade de paralelização e distribuição da carga de processamento [Freitas & Lavington 1998]. Somando-se a isto, o algoritmo genético permite a adaptação e inclusão de novos operadores e parâmetros, ajustados especificamente para uma aplicação (como a mineração de dados). Devido às características acima, dentre estas técnicas de descoberta de informações a que será usada neste trabalho é a baseada em Algoritmos Genéticos [Langley 1996].

Este trabalho tem como objetivo mostrar o uso de Algoritmos Genéticos em problemas de Mineração de Dados, e explorar seu uso em tarefas de classificação através da indução de regras. É apresentada neste trabalho uma ferramenta desenvolvida para a indução de regras e a montagem de um classificador. Analisam-se caminhos que possam melhorar a sua eficiência neste caso. A implementação une em uma única ferramenta características de outros classificadores, procurando melhorar o desempenho do algoritmo. Entre as características presentes em outros algoritmos destacam-se o teste de significância (explicado na seção 3.4), compartilhamento de recursos no espaço fenotípico (mostrado na seção de trabalhos relacionados e detalhado na seção 4.4.1), operação de semeadura (explicada na seção 4.7) e a estimativa de erro de Laplace (explicada na seção 4.4).

1.1. Organização do Trabalho

No Capítulo 2 é dada uma explicação detalhada sobre Algoritmos Genéticos, sua estrutura, funcionamento e principais operadores, e logo em seguida, uma análise sobre seu uso adaptado especificamente para tarefas de mineração de dados. No final do capítulo, na seção 2.4, são citados os trabalhos relacionados.

A Ferramenta compõe-se de duas partes interligadas, o Classificador e o Algoritmo Genético usado na busca de regras. No Capítulo 3, o classificador implementado é explicado em detalhes, incluindo como ele faz uso de algoritmos genéticos e quais as heurísticas de junção de regras que foram implementadas. Devido à importância dos algoritmos genéticos no funcionamento da ferramenta, o Capítulo 4 é reservado ao detalhamento da sua implementacão.

O Capítulo 5 apresenta a ferramenta e os detalhes de implementação, incluindo os resultados de comparações com outros algoritmos em [Lim et al 1999]. Por fim, no Capítulo 6 são apresentadas algumas conclusões e trabalhos futuros.

2. Algoritmos Genéticos

Nesta seção é fornecida uma breve explicação sobre a origem, funcionamento básico e principais componentes de um Algoritmo Genético. Depois disto, mostra-se como utilizar algoritmos genéticos em aplicações de mineração de dados e uma revisão de alguns trabalhos relevantes na área.

2.1. Origem e Conceitos Básicos

Os Algoritmos Genéticos foram concebidos a partir da analogia com a seleção natural, sendo portanto um dos ramos da computação evolucionária. A computação evolucionária teve suas origens nos anos 50-60, e usava a analogia da seleção natural para escolher entre soluções candidatas que eram modificadas aleatoriamente a cada iteração. Esta idéia foi mais desenvolvida e aperfeiçoada principalmente por John Holland, que com sua equipe criou a base dos Algoritmos Genéticos como são conhecidos atualmente.

A idéia básica de funcionamento dos Algoritmos Genéticos é a de tratar as possíveis soluções do problema como "indivíduos" de uma "população". Seleciona-se então um conjunto destes indivíduos como base para a criação de um novo conjunto de possíveis soluções (a nova "geração"). O critério de escolha dos indivíduos deve ser de alguma forma ligado ao valor da função de aptidão. Uma nova geração é obtida aplicando-se operações que misturem as características dos indivíduos (usando-se o operador de "cruzamento"). Estes passos são repetidos até que algum critério de término seja satisfeito.

Por consequência dessas operações, o algoritmo acaba por melhorar gradativamente o conjunto de indivíduos a cada geração. Isto acontece por que parte-se do pressuposto de que uma solução mais adequada é composta pela união das melhores

características dos indivíduos da população que está sendo testada. Ao recombinar características dos melhores indivíduos, implicitamente estão sendo descartadas características indesejáveis e estão sendo tentadas mais combinações de características que até o momento deram melhores resultados.

2.2. Terminologia e Componentes de um Algoritmo Genético

Diversas analogias com a Evolução são usadas para explicar o funcionamento dos componentes dos Algoritmos Genéticos, devido a seu paralelo com a Teoria da Evolução de Darwin.

A idéia básica de funcionamento dos Algoritmos Genéticos é a de tratar as possíveis soluções do problema como "indivíduos" de uma "população", que irá "evoluir" a cada iteração ou "geração". Alguma forma de codificação deve ser criada para poder guardar os dados que se quer otimizar dentro dos indivíduos. A execução do algoritmo pode ser resumida nos seguintes passos:

- Inicialmente escolhe-se uma população inicial, normalmente formada por indivíduos escolhidos aleatoriamente.

- Avalia-se toda a "população" de indivíduos segundo algum critério, determinado por uma função que avalia a qualidade do indivíduo (chamada função de aptidão ou "fitness").

- Em seguida, através do operador de "seleção", escolhem-se os indivíduos de melhor valor (dado pela função de aptidão) como base para a criação de um novo conjunto de possíveis soluções, chamado de nova "geração".

- Esta nova geração é obtida aplicando-se sobre os indivíduos selecionados operações que misturem suas características (chamadas "genes"), através dos operadores de "cruzamento" ("Crossover") e "Mutação".

- Estes passos são repetidos até que uma solução aceitável seja encontrada, até que o número predeterminado de passos seja atingido ou até que o algoritmo não consiga mais melhorar a solução já encontrada.

Com pequenas variações, [Goldberg 1989], [Mitchell 1997] e vários outros autores descrevem um Algoritmo Genético da seguinte forma:

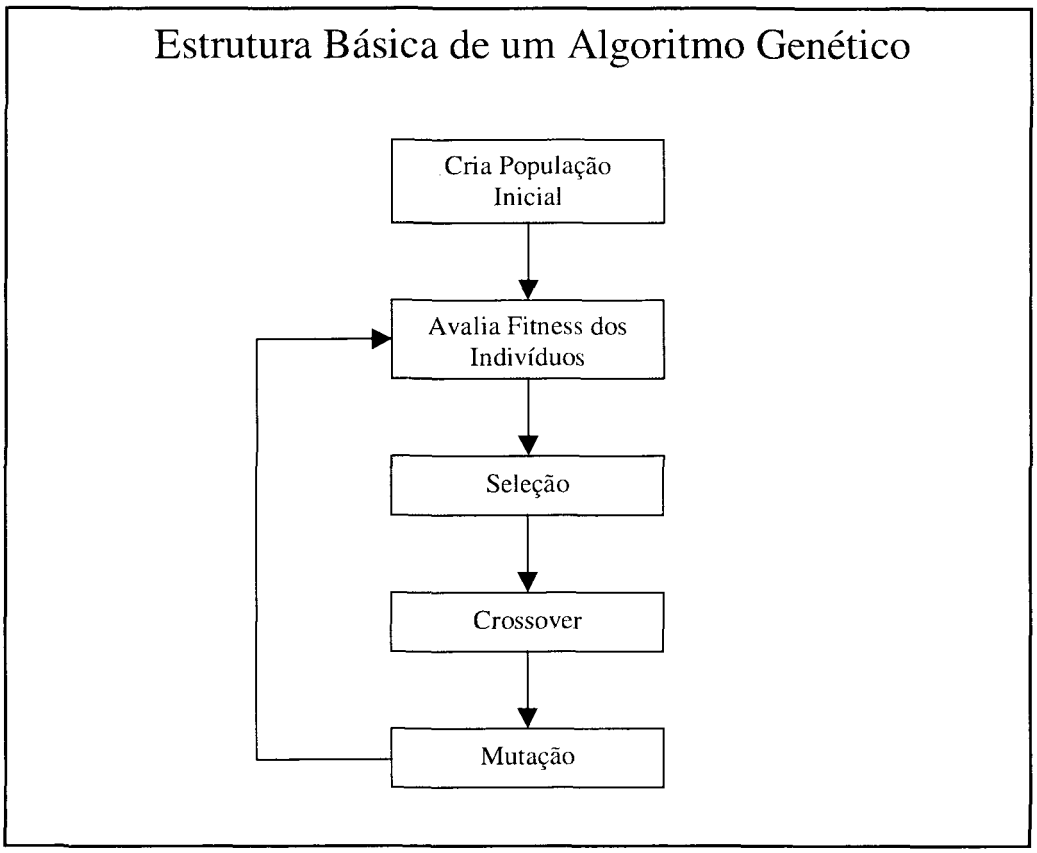


FIGURA 1: ESTRUTURA BÁSICA DE UM ALGORITMO GENÉTICO.

Os principais componentes e operadores mostrados na Figura 1 são descritos a seguir em mais detalhes.

População

A população de um algoritmo genético é o conjunto de indivíduos que estão sendo cogitados como solução e que serão usados para criar o novo conjunto de indivíduos para análise. Populações muito pequenas têm grandes chances de perder a diversidade necessária para convergir a uma boa solução. Entretanto, se a população tiver muitos indivíduos o algoritmo perderá grande parte de sua eficiência pela demora em avaliar todo o conjunto a cada iteração.

Indivíduos

A codificação de cada indivíduo na realidade é composta apenas do "material genético", ou seja, o conjunto de atributos que pode ser considerado como uma das soluções possíveis para o problema analisado.

A principal dificuldade no tratamento dos indivíduos é a forma de representar e organizar os dados. Uma das principais formas é representar cada atributo como uma sequência de bits e o indivíduo como a concatenação das sequências de bits de todas as suas características.

Outra forma muito usada de representação é a codificação usando o próprio alfabeto da característica que se quer representar (letras, códigos, números reais, etc.).

Diversas outras formas são possíveis dependendo do tipo de problema. Normalmente a forma ideal está fortemente ligada ao tipo de problema.

Codificações binárias possuem a vantagem de serem simples de implementar, e é fácil adaptar qualquer novo operador genético a uma codificação como esta. Entretanto, existem algumas desvantagens, como a dificuldade em achar uma boa forma de

codificação/decodificação dos bits, o que pode comprometer o desempenho. Outro problema é que, dependendo dos pontos onde houver cruzamento e mutação, podem ser criados indivíduos que contêm genes inválidos, o que exige esforços no sentido de validar o indivíduo gerado.

Codificações com alfabetos mais complexos (adaptados ao tipo de problema que se quer resolver), são mais simples de entender e manipular, apesar da implementação do algoritmo genético e seus operadores torná-la um pouco mais trabalhosa pela adaptação do algoritmo à nova forma de codificação.

Função de aptidão ou "Fitness"

Este é o componente mais importante de qualquer algoritmo genético. É através desta função que se mede quão próximo um indivíduo está da solução desejada ou quão boa é esta solução.

É essencial que esta função seja muito representativa e diferencie na proporção correta as más soluções das boas. Se houver pouca precisão na avaliação, uma ótima solução pode ser posta de lado durante a execução do algoritmo, além de gastar mais tempo explorando soluções pouco promissoras. O critério de distribuição (linear, exponencial, etc.) e o grau de ênfase em cada atributo (peso) são apenas alguns dos tópicos a analisar.

A dosagem entre precisão e tempo de execução da função de aptidão também é importante. Uma função extremamente precisa normalmente tem um custo computacional muito maior, o que dificulta o uso de grandes populações. Já uma função mais simples permite maior exploração, mas tem mais chance de deixar passar ou interpretar de forma incorreta uma possibilidade promissora.

A seguir um exemplo simples de função de aptidão. No caso, o valor da função é determinado simplesmente pela quantidade de bits "1" do indivíduo. Esta população será usada como exemplo mais adiante.

Indivíduos	Função Aptidão	Proporção em relação ao total da população
10101010110101010111	12	23,08%
00001001010101110010	8	15,38%
00001100001011011101	9	17,31%
00000110010010000010	6	11,54%
11100011100010011111	12	23,08%
00010101001000010000	5	9,62%
Total	52	100,00%

TABELA 1: EXEMPLO DE POPULAÇÃO E FUNÇÃO DE APTIDÃO

Seleção

O operador de seleção escolhe quais indivíduos participarão efetivamente na criação da próxima geração da população. Para que a evolução seja possível ao longo das gerações, a seleção deve pegar com mais frequência os indivíduos mais bem adaptados, ou seja, com maior valor na função de aptidão.

Contudo, principalmente nas primeiras gerações não se deve enfatizar excessivamente o uso apenas dos melhores indivíduos. Isso faz com que a diversidade seja rapidamente perdida, o que dificilmente leva a mais do que soluções ótimas locais.

Há diversas formas de seleção, todas argumentando serem as melhores nos problemas para as quais foram usadas como solução. A mais conhecida é a seleção proporcional ao valor de Fitness, ou "Roleta" [Goldberg, 1989]. Uma das implementações é a seguinte:

1. Somar os valores das funções de aptidão de todos os indivíduos da população, armazenando em T.

2. Repetir *N* vezes para selecionar *N* indivíduos:

   2.1. Escolher aleatoriamente um valor *r* tal que 0 <= *r* < T

   2.2. Percorrer sequencialmente os indivíduos da população, em qualquer ordem, calculando a soma acumulada da função de aptidão dos indivíduos já percorridos. Será escolhido o indivíduo no qual a soma acumulada resultar num valor maior ou igual a *r*.

Aplicando este critério sobre a população da Tabela 1, tem-se:

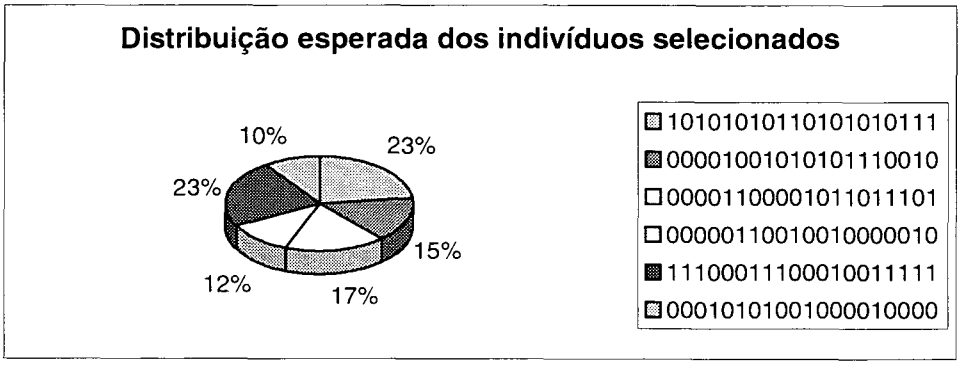


FIGURA 2: SELEÇÃO POR ROLETA APLICADA SOBRE DADOS DA TABELA 2.1.

Esta abordagem teoricamente resulta numa distribuição proporcional de indivíduos, mas em populações pequenas pode acabar resultando em péssimas escolhas. Mesmo que na média espere-se uma distribuição proporcional à função de aptidão, numa determinada geração pode ocorrer a escolha do mesmo indivíduo diversas vezes ou ainda a omissão de um ótimo indivíduo por ele não ter sido "sorteado". Para melhorar um pouco o algoritmo, pode-se escolher apenas um número aleatório que servirá de base para todas as amostragens. Isto ao menos garante uma distribuição o mais proporcional possível à função de aptidão de cada indivíduo. Esta última abordagem é denominada *Stochastic Universal Sampling* [Mitchell, 1997].

Os métodos baseados na seleção por roleta têm o problema de convergir rápido demais, já que a seleção beneficia muito os melhores indivíduos. Isto pode fazer com que uma solução ótima local possa se espalhar rapidamente por toda a população, especialmente nas primeiras gerações, em que a variância é muito alta. Já quando o

algoritmo genético fica perto do final, a variância é quase nula e a seleção torna-se pouco eficiente.

Uma forma de fugir destes problemas é usar métodos que levem em conta outros fatores, como a média da população, desvio padrão, estágio do algoritmo (explorando ou convergindo), entre outros.

A ordem dos indivíduos também é considerada muito importante na seleção. O Elitismo, por exemplo, mostrou-se uma forma importante de melhorar a eficiência do Algoritmo Genético. Nele são sempre selecionados como uma parte da próxima geração os indivíduos mais bem adaptados, enquanto que a outra parte é escolhida por algum outro método.

Pode-se levar em conta como principal critério de escolha a colocação do indivíduo frente aos outros. O problema é o tempo de ordenação, que pode ser muito demorado em grandes populações. Mesmo a seleção via roleta exige duas passagens por todos os indivíduos da população: uma para calcular o total, média, etc e outra para fazer a seleção propriamente dita.

Uma das formas encontradas de tornar pequeno o tempo gasto no processo de seleção é a seleção por Torneio *binária* [Mitchell 1997]. O funcionamento é o seguinte:

1. Escolhe-se aleatoriamente dois indivíduos. O número de indivíduos escolhidos pode variar, mudando com isto a pressão seletiva. Alternativas de variantes da Seleção por torneio podem ser encontradas em [Mitchell 1997].

2. Escolhe-se um número aleatório r entre 0 e 1. O seu valor é comparado com um parâmetro k (próximo a 0.75, por exemplo).

3. Se r for menor do que k, o melhor dos indivíduos é escolhido, caso contrário é escolhido o pior.

4. Retorna-se ao passo 1 até terem sido selecionados indivíduos suficientes.

Apesar de muito mais simples e rápido, este método tem resultados próximos à seleção por ranking, o que o transforma numa boa opção.

Em certos problemas, como os que envolvem a otimização simultânea de múltiplos parâmetros, a convergência rápida e imediata em direção aos melhores indivíduos pode tornar a busca pouco produtiva, principalmente quando o domínio escolhido é mais complexo. Isto ocorre porque o espaço de busca pode conter uma grande quantidade de ótimos locais, que podem se espalhar pela população rápido demais, evitando uma boa exploração. A seleção por torneio binária pode atenuar este problema por ter uma convergência mais lenta e constante, pemitindo maior exploração do espaço de busca.

Cruzamento (Crossover)

O operador de cruzamento cria novos indivíduos misturando características de dois indivíduos "pais". Esta mistura é feita tentando imitar a reprodução de genes em células. Trechos das características de um indivíduo são trocados pelo trecho equivalente do outro. O resultado desta operação é um indivíduo que potencialmente combine as melhores características dos indivíduos usados como base.

Pode-se gerar diversas variações a partir do mesmo par de indivíduos, cada uma delas tendo um percentual diferente de genes de cada um dos "pais".

A proporção de características entre os indivíduos dependerá da estrutura usada para guardar os genes e dos critérios de permutação.

Várias estruturas são usadas para guardar os genes: "haplóide", organizada apenas como uma string de bits, "diplóide", que tem duas strings na forma de um "X" e muitas outras. A criação de novos indivíduos é feita pela troca de segmentos entre os pais.

Alguns exemplos são mostrados na Figura 3:

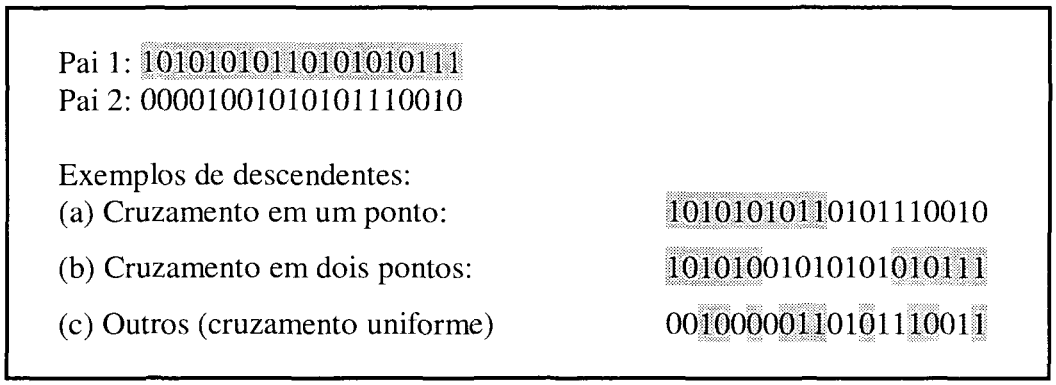


FIGURA 3: EXEMPLOS DE ALGUNS MÉTODOS DE CRUZAMENTO

Existem diversas maneiras de se fazer o cruzamento, sem nenhum consenso sobre qual funciona melhor em cada tipo de problema. Entretanto, vários problemas são apontados em qualquer uma delas. Se os blocos de bits forem grandes ou ocorrerem com muita frequência no mesmo trecho, ocorrerá o problema em que um gene com um ótimo valor pode quase sempre levar consigo um gene com um valor desfavorável simplesmente por que os dois foram colocados um após o outro na sequência de codificação. Exemplos disto ocorrem com os métodos de cruzamento (a) e (b) da Figura 3.

Uma forma de evitar este problema é usar métodos de cruzamento como os da letra (c) [Goldberg 1989], que podem misturar valores dos indivíduos de forma independende de onde está posicionado cada fragmento.

Um método que segue este exemplo é o de cruzamento uniforme. Neste método, cada bit (ou gene) é obtido de um ou do outro "pai" de acordo com o sorteio de um número aleatório entre 0 e 1. Se o número estiver acima de determinado coeficiente, o bit/gene do primeiro dos pais é escolhido, caso contrário o valor do bit/gene é obtido do outro indivíduo.

Quando se escolhem pedaços de indivíduos para o cruzamento normalmente não se escolhem para divisão posições que caiam no meio do valor de um determinado gene,

porque a mistura de partes de genes normalmente resulta num valor para o gene que não possui as características de nenhum dos dois indivíduos usados como base. Este efeito normalmente é indesejável numa operação de cruzamento, apesar de ser útil para manter a diversidade da população. Para manter a diversidade da população usa-se outro operador, o de mutação.

Outra forma de melhorar o cruzamento é restringindo quais pares de indivíduos podem ser usados como pais. Isto pode ser feito impedindo o cruzamento de indivíduos muito similares (evitando o "incesto"). Um outro modo pode ser restringir o cruzamento de indivíduos muito diferentes, fazendo com que indivíduos próximos um do outro no espaço de busca tendam a se cruzar e convergir mais rapidamente.

Ao longo de diversas gerações, esta operação acabará por encontrar combinações que aumentem o valor da função de aptidão acima dos valores de qualquer um dos dois indivíduos originais. A operação de "seleção" usará com mais frequência os indivíduos mais bem adaptados e devido a isso a média do valor da função de aptidão da população terá a tendência de aumentar lentamente.

Mutação

Esta operação simplesmente modifica aleatoriamente alguma característica do indivíduo sobre o qual é aplicada. Esta troca é muito importante, pois acaba por criar novas características que não existiam ou apareciam em pequena quantidade na população em análise.

A existência de variedade entre os indivíduos é essencial para que o algoritmo cumpra sua função. Isto ocorre porque normalmente as soluções são complexas e dependentes de fatores pouco conhecidos. É perfeitamente possível que genes sejam inadequados para certos indivíduos (ou seja, reduzem o valor da função de aptidão), mas sejam extremamente úteis se colocados em indivíduos mais "adaptados" (com "fitness" maior).

A mutação normalmente é controlada por um coeficiente, que indica quantos genes serão trocados aleatoriamente a cada geração. Este coeficiente deve ser alto o suficiente para manter a diversidade, mas não a ponto de alterar significativamente a seleção de genes feita pela função de aptidão.

Geração

A cada passo, um novo conjunto de indivíduos é gerado a partir do anterior. A este novo conjunto dá-se o nome de "Geração". É através da criação de uma grande quantidade de gerações que é possível obter resultados dos Algoritmos Genéticos.

Assim como na natureza, os operadores de seleção, mutação e cruzamento levam tempo para mostrar de forma perceptível seus efeitos. Se o número de gerações for muito pequeno, o algoritmo não terá tempo para "evoluir" até chegar às melhores soluções. Por outro lado se o número de gerações for muito grande, o algoritmo gastará muito tempo para melhorar muito pouco ou absolutamente nada a solução já encontrada.

Algoritmos Genéticos podem ser usados para resolver praticamente qualquer tipo de problema que envolva a exploração de soluções num espaço de busca. Para implementá-lo, basta ter uma representação da solução candidata e uma função de avaliação que seja razoavelmente precisa.

Exemplos de aplicações que usam algoritmos genéticos incluem problemas de otimização de soluções, aprendizado de máquina, desenvolvimento de estratégias e fórmulas matemáticas, análise de modelos econômicos, problemas de engenharia, diversas aplicações na Biologia como simulação de bactérias, sistemas imunológicos, ecossistemas, descoberta de formato e propriedades de moléculas orgânicas, só para citar algumas [Mitchell 1997]. Este trabalho aplica algoritmos genéticos em Mineração de dados, como explicado na seção a seguir.

2.3. Mineração de dados com Algoritmos Genéticos

Fazer uso estratégico e completo da informação gerada dentro da organização é uma forma eficaz de aumentar a produtividade em todos os setores. Tendo os dados à disposição para análise, é importante aproveitar a oportunidade de buscar informações. Este é o objetivo da Mineração de Dados.

É muito importante que o próprio computador analise e descubra estas informações de forma tão automatizada quanto possível, pois em grandes organizações, normalmente a quantidade de informação gerada em uma semana é maior do que um ser humano seria capaz de ler durante a vida toda [Adriaans & Zaatinge 1996].

Existem diversas formas de se usar um algoritmo genético para a mineração de dados [Freitas & Lavington 1998], como:

- Buscar regras de no formato SE-ENTÃO em conjuntos de dados, tais como regras de classificação, onde o antecedente (parte "SE") contém condições de valores de atributos e o consequente (parte "ENTÃO") contém a classe prevista quando as condições do antecedente forem satisfeitas. O Algoritmo genético usa regras neste formato como indivíduos da população. A função de aptidão será calculada pela qualidade da regra gerada, normalmente medida pelos coeficientes de suporte e confiabilidade [Freitas & Lavington 98].

- Cada indivíduo pode ser também um conjunto de "protótipos" para fins de classificação. Neste caso a tarefa do Algoritmo Genético consiste em encontrar o melhor conjunto de protótipos de acordo com o grau de acerto deste conjunto, de acordo com o paradigma de Classificação baseado por semelhança entre instâncias, ou Instance-Based Learning (IBL). A implementação deste tipo de Algoritmo Genético é discutida em [Knight &

Sem 1995], enquanto que o paradigma IBL é explicado em detalhes em [Aha et al. 1991].

- Pode-se também usar um Algoritmo Genético indiretamente, para determinar por exemplo a importância de cada atributo para fins de classificação em um algoritmo como IBL [Kelly & Davis 1991].

- Variações das formas acima e outras formas são discutidas mais detalhadamente em [Freitas & Lavington 1998].

A tarefa de mineração de dados usada neste trabalho é chamada de classificação. Dado um conjunto de exemplos de classe conhecida (exemplos de treinamento), a tarefa de classificação consiste em analisar este conjunto e criar uma descrição lógica capaz de classificar corretamente novos exemplos de classe desconhecida. Neste caso, a descrição lógica gerada pode ser considerada um classificador.

Um dos maiores problemas em tarefas de classificação é a tolerância a ruído [Freitas & Lavington 98] nos dados. Em quase todas as aplicações práticas, as medições e os dados analisados podem conter ruído, ou seja, imprecisões cuja origem não pode ser determinada [Brazdil & Clark 1990]. A existência de ruído pode levar o algoritmo a criar um classificador excessivamente especializado aos dados analisados. Se o classificador especializar-se demais aos dados, sua precisão será comprometida quando for usado em dados posteriores não usados durante o treinamento.

2.4. Trabalhos Relacionados

A Indução de regras com algoritmos genéticos pode ser dividida em duas abordagens significativamente diferentes, chamadas abordagem de Pittsburgh [Smith 1983] e de Michigan [Holland 1986].

Na abordagem de Pittsburgh cada indivíduo da população contém um classificador completo. A maior vantagem desta abordagem é a simplicidade do algoritmo, que não precisa de formas de criação de nichos ou heurísticas para criar conjuntos de regras. O principal problema está na grande redundância dentro da população, que pode levar à necessidade de grandes populações e indivíduos com uma codificação muito grande [Giordana & Neri 1995]. Um exemplo de uso da abordagem de Pittsburgh é o sistema LS-1 [Smith 1983].

Já na abordagem de Michigan cada indivíduo da população corresponde a uma regra simples, que classifica apenas um subconjunto dos dados. Neste caso é necessário desenvolver algum tipo de estratégia para extrair da população um conjunto de regras não redundantes [Wilson 1987][Holland 1986]. Uma das formas de extrair um conjunto de regras não redundantes é com a formação de "nichos". Técnicas de formação de nichos [Watson 1998] forçam a competição entre indivíduos similares, criando assim subpopulações que exploram regiões potencialmente distintas do espaço de busca.

Alguma técnica de formação de nichos precisa ser usada a fim de explorar e manter a diversidade dentro da população. A maioria dos métodos de formação de nichos são baseados em "Crowding" [DeJong 1975][Mahfoud 1992] ou em Compartilhamento de Recursos ("Sharing") [Goldberg & Richardson 1987][Deb & Goldberg 1989].

Quando se usa "Crowding" para criar nichos, a substituição de indivíduos na população é modificada para fazer com que novos indivíduos substituam outros similares com menor valor para a função de aptidão dentro da população. Neste caso a seleção e demais operações como cruzamento e mutação não são modificadas. De acordo com os testes feitos por [Watson 1998] este método costuma chegar mais próximo dos valores ótimos locais, mas não se comporta tão bem quanto o "sharing" para manter subpopulações estáveis de indivíduos. Implementações futuras podem incluir o operador de "crowding" para confirmar estes resultados em tarefas de classificação que façam uso de algoritmos genéticos com a abordagem de Michigan.

Já no caso do compartilhamento de recursos, a formação de nichos é causada pela redução do valor da função de aptidão proporcionalmente à presença de indivíduos similares dentro da população. A similaridade entre indivíduos pode ser medida tanto no espaço genotípico quanto no espaço fenotípico [Mitchell 1997].

O espaço genotípico compreende a semelhança entre as codificações dos indivíduos, medida por exemplo pela quantidade de bits iguais entre eles. Medir a similaridade entre regras no espaço genotípico não parece ser uma boa alternativa para um classificador, já que é possível (e muito provável) que possam existir regras muito diferentes (quanto à codificação e atributos usados) que cubram exatamente os mesmos exemplos de treinamento. Segundo [Giordana & Neri 1995] não existe nenhuma métrica geral de distância entre duas regras que seja satisfatória para a tarefa de classificação.

O compartilhamento de recursos no espaço fenotípico permite uma comparação mais adequada de similaridade. Isto ocorre porque a similaridade é medida pelo número de exemplos de treinamento que são cobertos simultaneamente pelas regras. Quanto mais exemplos duas regras cobrem, mais a função de aptidão delas será reduzida.

Um sistema que usa uma abordagem bastante similar ao compartilhamento de recursos no espaço fenotípico é o operador "universal suffrage" da ferramenta REGAL [Giordana & Neri 1995]. Este operador modifica a operação de seleção para promover a criação de nichos e a competição entre as regras. O operador de seleção do REGAL trata as regras como "candidatos" que concorrem pelos votos dos "eleitores", representados pelos exemplos de treinamento positivos. Cada exemplo positivo vota em apenas uma das regras que o cobre, sendo que a escolha é definida por um processo semelhante à seleção por roleta [Mitchell 1997]. As regras com maior número de votos são usadas no processo de cruzamento. A função de aptidão leva em conta a simplicidade e consistência das regras.

Um algoritmo genético classificador que usa um compartilhamento de recursos fenotípico foi proposto por [McCallum & Spackman 1990]. No referido artigo a função de aptidão leva em consideração tanto os exemplos positivos quanto os negativos, e é

determinada simplesmente pelo número de acertos menos o número de erros. Uma adaptação é necessária para evitar indivíduos com valores negativos da função de aptidão.

O compartilhamento de recursos também pode ser feito levando-se em conta apenas os exemplos positivos cobertos por mais de uma regra. A análise do compartilhamento de recursos no espaço fenotípico usando apenas os exemplos positivos pode ser encontrada em [Horn, Deb & Goldberg 1994].

A abordagem desta dissertação tem algumas diferenças importantes em relação aos trabalhos citados, pois une as seguintes características:

- O compartilhamento de recursos leva em consideração apenas os exemplos positivos para a redução do valor da função de aptidão. Os motivos desta abordagem serão explicados porteriormente na Seção 3.2.

- A função de aptidão foi adaptada para levar em conta tanto a precisão[1] da regra quanto a sua abrangência[2]. A função de aptidão é explicada em detalhes na Seção 4.4.

- O compartilhamento de recursos proposto tem baixo custo computacional, pois permite que os exemplos de treinamento sejam analisados por cada regra apenas uma vez por geração, como explicado na Seção 4.4.1.

Desta forma, nesta dissertação o uso de Algoritmos Genéticos para a mineração de dados será feito criando um classificador que usa a abordagem de Michigan, ou seja, buscando por regras individuais que, unidas de forma organizada, resultarão em um classificador. O capítulo seguinte explica como e onde algoritmos genéticos são usados no processo.

[1] A precisão de uma regra é medida neste trabalho pelo seu percentual de acerto.
[2] A abrângência de uma regra é medida pelo número de exemplos de treinamento que ela cobre.

3. O Classificador

Em linhas gerais, o funcionamento do classificador pode ser dividido em quatro etapas. Primeiramente, algumas informações básicas são extraídas dos dados (passo 1), para em seguida poder criar um algoritmo genético para cada classe a analisar (passo 2). Depois disto, os algoritmos genéticos são executados (passo 3) para encontrar regras de classificação. Finalmente, as regras são agrupadas para formar o classificador (passo 4). Os detalhes sobre cada um destes passos são explicados nas seções subsequentes.

3.1. Análise e Obtenção dos dados

Os dados de treinamento e teste seguem o formato usado pela ferramenta C4.5 [Quinlan 1993], usando três arquivos: o arquivo com extensão denominada **NAMES**, que contém os nomes, ordem e tipos dos atributos, o arquivo com extensão denominada **DATA**, que contém os dados de treinamento e o arquivo com extensão denominada **TEST**, (opcional) que contém dados de teste para medir a precisão do classificador em dados novos ou desconhecidos.

Para aumentar a eficiência dos algoritmos genéticos, todos os exemplos dos dados de treinamento são analisados antes da busca de regras.

Algumas informações importantes que precisam ser obtidas de antemão são a distribuição de frequência das classes (número de exemplos por classe), os valores possíveis de cada atributo (no caso de atributos contínuos, são obtidos os valores máximo e mínimo), e, para cada classe, os dados são separados entre exemplos positivos e negativos. Esta informação pode ser obtida do arquivo NAMES (com os nomes e tipos dos atributos) e do arquivo DATA (assume-se que os exemplos de treinamento contêm todos os valores possíveis). Isto é extremamente útil durante a execução do algoritmo, pois permite, entre outras coisas, uma distribuição mais balanceada da carga de trabalho entre os subprocessos.

Após a preparação e pré-processamento dos dados, é possível subdividir a tarefa de classificação em partes mais especializadas, como explicado a seguir.

3.2. Criação de subprocessos por classes

Para cada classe do conjunto de dados analisado, um novo algoritmo genético independente será criado. Em cada um desses algoritmos genéticos, os dados a analisar são divididos simplesmente entre exemplos "positivos" (pertencentes à classe analisada) e "negativos" (exemplos pertencentes a todas as outras classes). As vantagens desta abordagem tornam-se mais visíveis quando são feitas comparações entre a busca de regras com e sem a separação por classes.

Implementações anteriores deste classificador, que usavam uma única população de regras para todas as classes, tiveram dificuldades para gerar regras úteis, ou seja, regras capazes de cobrir um ou mais exemplos de treinamento. Isto ocorreu porque grande parte dos indivíduos continha pares de atributos com valores que não ocorriam simultaneamente em nenhum exemplo de treinamento. Eram necessários vários cruzamentos até se chegar a uma regra válida, ou seja, capaz de classificar ao menos um exemplo de treinamento. Isto ocorreu em todos os testes, realizados em vários conjuntos de dados diferentes.

Mesmo incluindo na função de aptidão um estímulo a regras contendo valores "potencialmente úteis", ou seja, valores que ocorriam em exemplos positivos existentes, a evolução das regras continuou lenta, pois muitas regras não chegavam a este estágio. Dezenas de gerações eram necessárias até que houvesse grande quantidade de regras válidas na população, e, mesmo assim, vários cruzamentos resultavam em regras inválidas.

A separação das populações por classe tornou a evolução das regras muito mais rápida, pois apenas valores válidos de atributos eram usados, fazendo com que apenas alguns raros cruzamentos resultassem em regras que não cobriam nenhum exemplo de treinamento.

Devido aos problemas acima, foi feita a separação das regras por classes. As principais vantagens encontradas durante os testes foram:

- O espaço de busca pode ser bastante reduzido pelo uso apenas de valores "potencialmente úteis" [Noda, Freitas & Lopes 1999] dos dados de treinamento. Quando a busca é dirigida apenas a uma classe, é necessário analisar apenas valores de atributos que apareçam em exemplos positivos daquela classe. Esta condição pode ser implementada tanto na criação da população inicial quanto na mutação. A implementação corrente mostrou que isto reduz drasticamente a quantidade de regras da população que não são capazes de cobrir nenhum exemplo positivo.

- A divisão permite um controle simplificado sobre o tempo de processamento gasto na busca. Por exemplo, cada subprocesso pode ter uma população de regras proporcional à quantidade de exemplos positivos existentes em cada classe. Além disso, seria possível usar a média da função de aptidão de cada subpopulação como um indicador de onde é necessário aplicar mais tempo de processamento, a fim de que todas as classes tenham soluções com aproximadamente o mesmo nível de precisão. A média da função de aptidão de cada classe (ou alguma outra medida semelhante) poderia inclusive ser usada como critério de término.

- O uso de uma população inteira de regras para a busca de uma única classe elimina o problema de misturar regras de classes diferentes, o que muito provavelmente resulta em regras de baixa qualidade.

- A implementação torna-se mais simples e a execução mais rápida, porque qualquer valor de atributo escolhido cobrirá ao menos um exemplo positivo, formando portanto uma regra com valores *potencialmente* úteis, já que mesmo uma regra contendo apenas valores potencialmente úteis pode não ser capaz de cubrir nenhum exemplo positivo dos dados de treinamento.

- A paralelização e distribuição de processamento torna-se fácil, pois cada algoritmo genético de cada classe pode rodar de maneira independente em máquinas diferentes.

- O processo de busca pode dar bons resultados parciais. A execução pode ser parada em qualquer geração, e as regras encontradas até aquele momento já podem ser usadas para criar bons classificadores. Se o usuário não ficar satisfeito com os resultados até o momento, pode simplesmente deixar o algoritmo buscando regras por mais algumas gerações até chegar a um resultado adequado.

Depois de efetuada a divisão, é possível executar cada um dos algoritmos genéticos separadamente.

3.3. Execução dos Algoritmos Genéticos

Para iniciar a execução de cada algoritmo genético, é necessário definir o tamanho da população de cada um deles. Na implementação atual, o tamanho da população de regras que cada algoritmo genético irá criar pode ser guiado por dois critérios, cuja escolha depende do conjunto de dados analisado :

- **Distribuição entre as Classes**: o número de exemplos pertencentes a cada classe é usado como base para criar populações de regras com aproximadamente a mesma proporção. Caso o número de regras em uma determinada classe seja pequeno demais (menos de 10 regras, por exemplo), uma quantidade mínima é usada para permitir uma busca útil. Esta alternativa é comumente usada quando a complexidade de classificação de cada classe é mais ou menos proporcional ao número de exemplos de treinamento de cada classe.

- **Número de Classes**: neste caso o número de regras da população de cada subprocesso é o mesmo para cada classe. Esta alternativa é mais adequada em casos em que é mais eficiente buscar por regras que cubram exceções, ou seja, pesquisar e prever com maior precisão os exemplos que pertencem às classes menos frequentes.

Apenas um critério de término foi implementado até agora. Os algoritmos genéticos irão parar quando o número de gerações especificado pelo usuário for atingido por todos os subprocessos. Entretanto, é permitido ao usuário continuar a execução por mais gerações ou interromper a busca de regras antes do limite especificado.

Uma vez que uma população de regras tenha sido gerada para cada classe, é possível seguir para o próximo passo: escolher e organizar as melhores regras em um classificador.

3.4. Geração do Classificador

Um classificador precisa organizar as regras de forma adequada para ter uma precisão satisfatória. A forma de agrupamento das regras pode ser em conjuntos ordenados ou não ordenados.

Quando se usa conjuntos não ordenados de regras, todas as regras são usadas simultaneamente para classificar uma instância. A classe escolhida será determinada pelas regras que cobrem a instância a classificar. Se, dentre as regras que cobrem a instância, existirem regras que indiquem classes diferentes, a classe escolhida dependerá da soma dos valores ou pesos atribuídos a cada regra. Esta abordagem pode se tornar difícil de entender e interpretar quando muitas regras cobrem subconjuntos comuns de exemplos.

Conjuntos de regras ordenados têm uma abordagem diferente. Todas as regras são organizadas em uma determinada ordem. Para classificar uma nova instância, cada uma das regras é tentada, uma por vez, até que seja encontrada alguma regra que cubra a instância em questão. Quando isto acontece, a classificação é obtida diretamente da regra e nenhum outro processamento é necessário. Caso nenhuma regra cubra a instância em questão, uma classe padrão é escolhida.

Conjuntos ordenados de regras têm a propriedade de serem *lógicos*, no sentido de que não é complicado saber qual regra aplicar e consequentemente qual classe atribuir a

um determinado caso. Não há a necessidade de se incluir cálculos probabilísticos ou pesos, o que facilita a interpretação. Devido a estes fatores, esta é a abordagem adotada neste trabalho. Contudo, futuras implementações podem permitir o uso das duas abordagens.

Já que se trata de um conjunto ordenado de regras, é necessário adotar algum critério para definir quais regras devem ser incluídas primeiro. O valor da função de aptidão de cada regra é adotado como métrica, *ignorando-se o decréscimo pela redundância*, ou seja, de forma totalmente independente de quais exemplos são cobertos pelas outras regras. Isto é feito porque neste estágio não há mais a necessidade de se criar subpopulações, regras boas serão adicionadas ou mantidas se melhorarem o classificador. Isto descarta também regras que cobrem todos os exemplos (positivos ou negativos) e regras que não cobrem nenhum exemplo positivo.

As regras úteis pertencentes a todas as classes são agrupadas em um único conjunto para análise e ordenadas por importância. Algumas heurísticas podem ser usadas para juntar as regras [Clark & Niblett 1989][Giordana & Neri 1995]. Diversas variações baseadas em algoritmos semelhantes como o CN2 foram implementadas e testadas para tentar determinar a melhor abordagem para o classificador. Uma dessas variações, obtida empiricamente, é a seguinte:

- Criar um classificador com um conjunto de regras vazio
- Enquanto o conjunto de regras a analisar não estiver vazio:
  - Retirar a melhor regra do conjunto a analisar segundo a métrica adotada.
  - Adicionar a regra escolhida no final da lista de regras
  - Verificar a eficiência do classificador após a adição da nova regra. Se o classificador ficar melhor (de acordo com a métrica adotada), a nova regra é mantida, caso contrário é descartada. Algumas métricas foram testadas, e serão explicadas mais adiante.
- Ajustar a classe padrão para a classe que resultar no menor índice de erro.

Uma outra alternativa possível é dividir o conjunto de regras a analisar por classe, e aplicar um subconjunto de cada vez. Com isto, as regras de cada classe ficam

agrupadas, facilitando a compreensão, apesar de normalmente resultar num pequeno decréscimo na precisão. Diversas variantes foram testadas, com pequenas variações de acordo com o conjunto de dados e o tamanho da população. Não foram determinados os fatores que fizeram uma variante sair-se melhor do que as outras em cada conjunto de dados.

Um cuidado especial é necessário na escolha da melhor métrica para definir se a adição de uma regra melhorou ou piorou o classificador. Os principais dados usados para medir a qualidade de um classificador são:

- Verdadeiros Positivos (**VP**): é o número de exemplos positivos corretamente classificados como positivos pelo classificador.

- Verdadeiros Negativos (**VN**): é o número de exemplos negativos corretamente classificados como negativos.

- Falsos Positivos (**FP**): é o número de exemplos negativos que foram incorretamente considerados pelo classificador como positivos.

- Falsos Negativos (**FN**): é o número de exemplos positivos incorretamente classificados como negativos.

Os critérios de medição da qualidade do classificador na implementação atual são:

- Percentual de acerto: critério definido como (**VP+VN**)/(**VP+VN+FP+FN**). Se este valor permanecer o mesmo antes e depois da adição de uma nova regra, um critério de desempate pode ser o número de falsos positivos, que deve sempre ser minimizado. Este critério não leva em consideração quantos exemplos não foram cobertos por nenhuma regra da lista, ou seja, os exemplos que pertencem à classe padrão.

- Percentual de acerto com análise da regra padrão: critério definido como **(VP+VN)/(VP+VN+FP+FN+DN)** onde **DN** é o número de exemplos negativos corretamente classificados mesmo sem ter sido cobertos por nenhuma regra (portanto com classificação dada pela classe padrão). O objetivo é permitir que regras de todas as classes (incluindo regras da classe padrão) sejam incluídas no classificador.

Um otimização que, em testes empíricos, melhorou a precisão do classificador nos dados de treinamento foi o uso do teste de significância [Clark & Niblett, 1989]. O teste de significância impede que sejam usadas regras que provavelmente são provenientes de ruído nos dados de treinamento.

Na implementação atual, a significância de uma regra é medida pela distribuição dos exemplos positivos e negativos. Compara-se a distribuição dos exemplos cobertos pela regra com a distribuição de exemplos de todo o conjunto de treinamento. Se a regra está classificando exemplos ao acaso, é de se esperar que a proporção de acertos e erros dentre os exemplos que a regra cobre, **VP/(VP+FP)** , seja próxima à proporção de acertos e erros de uma regra que classifique todos os exemplos de treinamento como positivos, **(VP+FN)/(VP+FP+VN+FN)**. Quanto mais próximas forem as duas medidas, mais provável que a regra esteja escolhendo exemplos ao acaso. Quanto mais exemplos positivos forem classificados corretamente pela regra, mais provável que a regra mostre uma correlação genuína nos dados de treinamento. A fórmula implementada no classificador foi:

$$Sig = 2 * VP * (DR * \mathrm{Log}(DR/DP))$$

Onde :

- ***Sig*** é o coeficiente de significância da regra

- **VP** é o total de exemplos positivos corretamente classificados pela regra analisada

- **DP** é a distribuição esperada de uma regra que escolhende exemplos ao acaso, ou seja, **(TP+FN)/(TP+FP+TN+FN)**

- **DR** é a distribuição observada na regra analisada, ou seja, **TP/(TP+FP)**

Se determinada regra tem o valor de ***Sig*** menor do que o coeficiente de significância especificado pelo usuário, ela é descartada.

Quanto à quantidade de regras geradas, a ferramenta assume que o maior número possível de regras seja usado, desde que cada regra adicionada melhore a precisão do classificador. É possível ao usuário informar um limite máximo de regras que possam ser analisadas para criar o classificador. Isto pode ser útil para limitar a complexidade do resultado, para diminuir o tempo de processamento ou para evitar o uso de regras que tenham pouca chance de melhorar o resultado quando o classificador for usado em dados de teste (não vistos durante a montagem do classificador).

O uso de outras heurísticas ou métricas pode melhorar a precisão do classificador. Entretanto, a parte mais importante para a eficiência da ferramenta é a busca de regras. O capítulo a seguir explica o algoritmo genético mais detalhadamente.

4. Descrição do Algoritmo Genético

A utilização de algoritmos genéticos para uma tarefa específica torna necessária a adaptação e o refinamento do algoritmo padrão da Figura 1 do Capítulo 2. O algoritmo genético implementado serve para buscar regras que diferenciem corretamente exemplos positivos e negativos de uma determinada classe. Cada classe dos dados de treinamento será analisada por um algoritmo genético independente. Neste capítulo discutem-se quais os principais tópicos que foram especializados para atingir este objetivo.

As principais modificações com relação ao algoritmo genético padrão são explicadas nas seções que seguem. Na Seção 4.1 mostra-se como os indivíduos da população, que representam as regras, são codificados. Posteriormente, os parâmetros de configuração para ajuste fino do algoritmo são detalhados na Seção 4.2. A Seção 4.3 explica como a população inicial é criada. A Seção 4.4 mostra as possíveis funções de aptidão e o método usado para a criação de nichos. As Seções 4.5., 4.6 e 4.7 mostram como os operadores de seleção, cruzamento e mutação foram implementados. A Seção 4.8 apresenta um operador usado como opção no lugar da mutação.

4.1. Representação e codificação das regras

O algoritmo genético usa como população regras com o seguinte formato:

$$r_1 \wedge r_2 \wedge \ldots \wedge r_n \Rightarrow C$$

onde *n* é o número de atributos, $\wedge$ é o operador lógico "E" (conjunção) e $\mathbf{r_i}$ é a cláusula de restrição para o valor do atributo **i,** sendo $\mathbf{1 \leq i \leq n}$. **C** é a classe que será dada como resultado se e somente se todas as restrições da regra forem atendidas.

A codificação permite que restrições sobre qualquer atributo possam ser nulas, ou seja, pode haver uma ou mais restrições em uma regra que considerem válidos quaisquer valores pertencentes ao atributo sobre o qual a restrição se aplica.

A representação e formato das restrições varia de acordo com o tipo de atributo:

- Para atributos discretos, cada restrição r_i é representada como $A_i = v_i$, onde A_i é o nome do atributo e v_i é um dos valores válidos possíveis.

- Para atributos contínuos, cada restrição é representada como $v_{min} \leq A_i \leq v_{max}$. Os valores v_{min} e v_{max} são os limites mínimo e o máximo para o atributo A_i.

Antes da execução do algoritmo começar, alguns parâmetros precisam ser configurados.

4.2. Parâmetros do Algoritmo Genético

Vários parâmetros podem ser configurados, incluindo os seguintes:

- Taxa de renovação da População: especifica a proporção de regras que será mantida a cada geração. Este número geralmente é alto, em torno de 80%, indicando que, a cada geração, 20% das regras (as piores da população) serão substituídas. Como resultado, apenas as piores regras são descartadas, fazendo com que a evolução da população ocorra aos poucos, de forma mais estável.

- Taxa de Cruzamento: apenas o cruzamento uniforme foi implementado [Mitchell 1997]. Implementações futuras podem incluir outros tipos de cruzamento.

- Número máximo de gerações: o usuário pode manualmente para a execução em qualquer geração, ou deixar que o algoritmo seja executado até este número de gerações ser atingido.

- Taxa de Mutação: indica a probabilidade de ocorrer uma mutação a cada operação de cruzamento.

- Método de Seleção: a implementação atual permite o uso da Seleção por Torneio (ou seja, a escolha do melhor entre dois indivíduos escolhidos aleatoriamente) ou Stochastic Universal Sampling [Mitchell 1997].

- Função de Aptidão: a melhor alternativa pode variar dependendo do domínio e de outros parâmetros do algoritmo. As duas alternativas possíveis são mostradas na Seção 4.4.

- Probabilidade de criação de restrições vazias: este coeficiente regula a quantidade de restrições que em média serão deixadas vazias a cada nova regra criada. Este parâmetro é usado para controlar a mutação e a criação da população inicial.

O tamanho da população de regras é determinado pelo classificador, conforme mostrado na Seção 3.3.

Após a configuração de todos os parâmetros, a população inicial é criada.

4.3. População Inicial

A população inicial é criada através da seleção aleatória de exemplos dos dados de treinamento. Cada exemplo escolhido é usado como semente para a geração de uma regra, regra esta que irá cobrir o exemplo escolhido como base e terá algumas restrições deixadas vazias, conforme a proporção especificada pelo usuário. Este método garante a

criação de regras que irão cobrir ao menos um exemplo positivo dentre os dados de treinamento.

O coeficiente de criação de restrições vazias é um parâmetro importante e pode afetar de forma significativa o comportamento do algoritmo. Caso o valor seja muito alto, serão criadas regras com muitas restrições vazias e a busca de regras irá de regras gerais para regras específicas. Já se o coeficiente for muito baixo, a busca começará com regras bastante específicas que serão gradativamente substituídas à medida que regras mais gerais entrem na população. Nenhuma tentativa foi feita para tentar encontrar o valor ótimo para este parâmetro. A melhor aproximação é dada pela quantidade de atributos dos dados de treinamento. Um número maior de atributos no domínio indica a necessidade de começar com regras mais abrangentes. Os valores para o coeficiente de criação de restrições vazias tipicamente oscilou entre 0,80 e 0,95. Estes valores significam que, para cada regra da população inicial, 80 a 95% dos atributos estavam sem restrições.

Após a criação de toda a população inicial, a avaliação da função de aptidão de todas as regras é efetuada.

4.4. Função de Aptidão ("Fitness")

A função de aptidão para este problema precisa ser capaz de classificar as regras como classificadores parciais, portanto a precisão da regra é mais importante do que a sua capacidade de cobrir todos os exemplos positivos encontrados nos dados de treinamento. Entretanto, a precisão da regra (medida pelo percentual de acerto) não pode ser a única medida a levar em consideração. Algum tipo de compensação deve ser dada pela abrangência da regra a fim de evitar o uso de regras que cubram poucos exemplos e portanto tenham maior probabilidade de representar apenas ruído nos dados.

A implementação atual permite o uso de duas funções de aptidão: Estimativa de erro de Laplace [Niblett 1987] e a soma ponderada de precisão e abrangência, uma métrica

criada para tentar melhorar os resultados em alguns conjuntos de dados através do ajuste preciso dos seus coeficientes.

A estimativa de erro de Laplace é uma métrica que procura evitar o favorecimento excessivo de regras que cobrem poucos exemplos, e é dada pela fórmula:

$$\text{LaplaceAccuracy} = (n_c + 1) / (n_{tot} + K)$$

onde:

- **K** é o número de classes no domínio (no caso duas: "positivo" e "negativo").
- n_c é o total de exemplos positivos cobertos pela regra
- n_{tot} é o total de exemplos "cobertos" pela regra, ou seja, classificados como positivos.

A estimativa de erro de Laplace é um caso especial da estimativa desenvolvida por Cestnik [Cestnik 1990], chamada de *m-probability-estimate*:

$$\text{mPAccuracy} = (n_c + p_0(c)\, m) / (n_{tot} + m)$$

A estimativa de erro de Laplace assume que $p_0(c)$ é igual a $1/\mathbf{K}$ e que o parâmetro ajustável m é fixado como igual a **K** [Niblett 1987]. Usando-se a terminologia da Seção 3.4, o termo n_c é equivalente a **VP** e o termo n_{tot} é equivalente a (**VP+FP**). Com isto, a fórmula aplicada pelo algoritmo genético torna-se:

(VP+1) / (VP+FP+2)

A outra função de aptidão implementada é uma soma ponderada da precisão e abrangência da regra, definida como

Prec*VP/(VP+FP) + Abr*VP/(VP+FN)

onde:

- **Prec** é o peso dado à precisão da regra (percentual de acerto *dentro das regras cobertas*)

- **Abr** é o peso dado à abrangência da regra (ou seja, seu percentual de cobertura de exemplos positivos)

- Os valores devem ser tais que **Prec + Abr = 1**.

O uso de uma soma ponderada permite o ajuste fino da função de acordo com o domínio. Domínios que necessitam de uma grande quantidade de regras (em que cada uma cobre poucos exemplos) podem precisar de valores menores para o coeficiente **Abr**, já que no caso em questão é mais vantajoso ter regras precisas. Por outro lado domínios com muito ruído precisam dar um peso maior a **Abr** para evitar o uso de regras que cubram poucos exemplos.

4.4.1. Criação de Nichos

Um algoritmo genético simples, quando confrontado com um problema que tenha múltiplas soluções, irá convergir, na melhor das hipóteses, para uma população contendo apenas uma destas soluções [Mahfoud 1994].

Para que seja possível a criação de um bom classificador, não basta ter simplesmente a melhor regra de acordo com critérios individuais. Em métodos de indução

de regras, um classificador é composto por um conjunto de regras, e cada regra deve cobrir uma região diferente do espaço de busca.

É portanto essencial para o classificador que a população de cada algoritmo genético tenha regras que cubram todos ou quase todos os exemplos do conjunto de dados. Isto quer dizer que o objetivo de cada algoritmo genético é encontrar múltiplas soluções dentro de uma mesma população. Este tipo de problema é resolvido usando-se métodos para a criação de nichos [Watson 1998].

A grande maioria dos métodos modernos de criação de nichos é baseado ou no método de "crowding" [DeJong 1975][Mahfoud 1992] ou em "sharing" [Goldberg e Richardson 1987].

Métodos de compartilhamento de recursos ou "sharing" usam a analogia com um ambiente de recursos limitados, em que os indivíduos da população competem entre si pelos recursos disponíveis no ambiente. Isto é feito reduzindo-se o valor da função de aptidão de indivíduos que façam uso dos mesmos "recursos" do ambiente, isto é, que sejam similares.

O compartilhamento de recursos no espaço genotípico faz a comparação de similaridade entre a codificação das regras, ou seja, baseia-se no número de bits iguais entre os indivíduos. Isto não parece ser uma boa alternativa em tarefas de classificação, pois duas regras podem cobrir exatamente os mesmos exemplos, tendo codificações totalmente diferentes.

Medir a similaridade entre regras no espaço fenotípico pode ser muito mais útil em tarefas de classificação, pois a similaridade é medida pela quantidade de exemplos de treinamento cobertos pelas regras. Quanto mais regras cobrem o mesmo exemplo de treinamento, menos este exemplo contribui para o valor da função de aptidão destas regras.

Na implementação atual, a medição de similaridade entre indivíduos é baseada somente nos exemplos *positivos* corretamente classificados nos dados de treinamento, já que nos casos de erros de classificação o valor da função de aptidão já é reduzido.

O fator de redução da função de aptidão para uma determinada regra está definido como:

$$\frac{VP}{\sum_{i=1,n} (vp_i)}$$

Onde:

- **VP** é o número de exemplos positivos corretamente classificados pela regra analisada (conforme nomenclatura da Seção 3.4),
- **n** é o total de regras na população (incluindo a regra analisada)
- $\mathbf{vp}_i$ é o número de exemplos positivos cobertos simultaneamente pela *i*-ésima regra da população e pela regra sendo analisada.

Esta métrica também é boa em termos de custo computacional, pois pode ser calculada juntamente com a avaliação da função de aptidão da população, tendo portanto baixo custo computacional. Isto significa que, para calcular o valor da função de aptidão de toda a população de regas, é necessário passar apenas uma vez por todos os exemplos de treinamento a cada geração (já que esta é a operação mais demorada). Nenhuma memória adicional é necessária caso os dados de treinamento tenham um volume maior.

4.4.2. Análise empírica da criação de nichos

Testes empíricos realizados durante a implementação mostraram uma enorme diferença entre o uso da ferramenta com e sem o compartilhamento de recursos,

especialmente quando a execução se estende por muitas gerações ou quando a complexidade dos dados aumenta (neste caso a "complexidade" medida pela quantidade de regras necessárias para uma boa solução).

Em qualquer caso, o algoritmo genético foi capaz de achar regras abrangentes e precisas. Contudo, sem o compartilhamento de recursos a população de regras converge rapidamente a variações da melhor regra, o que faz toda a população cobrir apenas uma pequena parte dos dados de treinamento.

A evolução da população de regras sem usar o compartilhamento de recursos mostra diversos problemas:

- o algoritmo não consegue melhorar a população de regras após muitas gerações. Toda a população de regras fica cada vez mais concentrada numa parcela dos dados de treinamento, tornando infrutífera a busca por regras precisas para os exemplos não cobertos pelas regras da população atual.

- Dentre os casos positivos, menos da metade é coberto por regras significativas (regras que passam no teste de significância a 99%), o que mostra a concentração das regras na busca de um mesmo subconjunto de exemplos. Isto é mostrado também pela enorme quantidade de exemplos não cobertos por nenhuma regra.

- O resultado de cada execução teve grandes variações no resultado obtido até a estabilização da população, com diferenças absolutas acima de 20%. Diversos testes foram feitos, e em nenhum dos casos a precisão do classificador melhorou após a 50ª geração (considerando uma população de 200 regras e elitismo de 80%). Em todos os casos a precisão do classificador foi muito baixa, poucas vezes superando o que se esperaria da classificação aleatória dos exemplos.

Estes efeitos apareceram de forma ainda mais clara quando populações menores foram usadas. Populações maiores não atenuaram significativamente os problemas

mostrados. Execuções em vários conjuntos de dados mostraram os mesmos efeitos em graus variados mas significativos.

Sem a criação de nichos, o uso do operador de "seeding" (explicado na Seção 4.8) no lugar da mutação não ajudou a melhorar os resultados de forma significativa. Assim que uma nova regra que cobria novos exemplos era adicionada na população, quase sempre era descartada na geração seguinte, por não atingir um valor na função de aptidão tão alto quanto as outras regras da população.

Quando o algoritmo foi executado com os mesmos parâmetros e habilitando o compartilhamento de recursos, a abrangência do conjunto de regras aumentou dramaticamente. Para ilustrar este caso, tem-se um exemplo de execução na Tabela 2, representada graficamente na Figura 4. O conjunto de dados escolhido foi o **DNA**, por ter uma grande quantidade de atributos, o que exige uma busca de regras mais abrangente (a lista completa está na Seção 5.2). Foi usada uma população de 200 regras, elitismo de 80%, cruzamento uniforme com taxa 0,75, seleção por torneio binário, função de aptidão dada por Laplace, uma mutação simples (sem o uso do operador de Seeding) por geração e execução variando de 10 a 200 gerações. A geração de regras foi feita sem limite máximo, com teste de significância a 99%, ordenação pelo valor da função de aptidão e sem agrupamento de regras de mesma classe. A Tabela 2 apresenta o percentual de acerto nos dados de treinamento.

Geração	% Acerto
10	65,10%
20	75,60%
30	83,65%
40	88,25%
50	91,00%

Geração	% Acerto
60	92,10%
70	92,70%
80	92,95%
90	93,90%
100	93,00%

Geração	% Acerto
110	92,60%
120	93,85%
130	94,85%
140	94,50%
150	94,25%

Geração	% Acerto
160	92,95%
170	93,85%
180	93,55%
190	93,65%
200	94,65%

TABELA 2: PROGRESSO DA EVOLUÇÃO DO CLASSIFICADOR USANDO NICHOS

Os resultados mostram que, mesmo usando uma população pequena, a precisão do classificador não sofreu variações grandes após alcançar a estabilidade, indicando que a criação de nichos permitiu a coexistência estável de subpopulações de regras.

A evolução da população de regras com o uso do compartilhamento de recursos mostrou-se eficaz tanto no exemplo mostrado como em todos os conjuntos de dados analisados. A população de regras se manteve diversificada em todos os testes, independente do conjunto de dados ou do tamanho da população. O uso do operador de Seeding no lugar da mutação resultou numa pequena melhora, mais visível em populações pequenas.

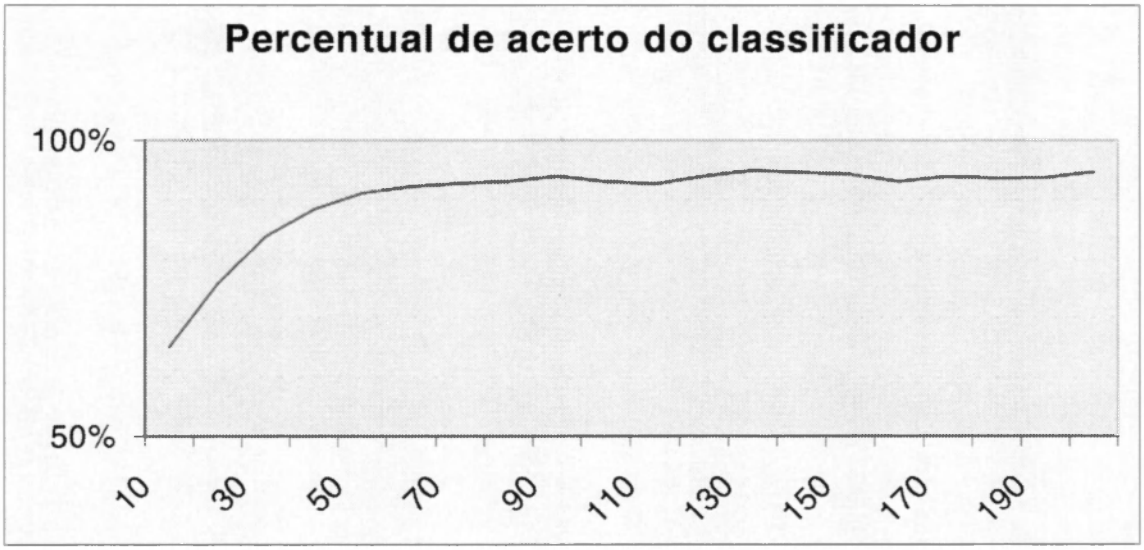


FIGURA 4: EXEMPLO DE EXECUÇÃO DO CLASSIFICADOR USANDO FORMAÇÃO DE NICHOS

4.5. Operador de Seleção

Dois métodos de seleção estão implementados: Stochastic Universal Sampling e Seleção por Torneio (Bínária) [Mitchell 1997]. Testes empíricos realizados durante a implementação mostraram que a Seleção por Torneio permitiu uma evolução mais estável apesar de ligeiramente mais lenta.

Estes dois métodos de seleção são os mais comuns em aplicações de algoritmos genéticos, mas existem outras alternativas que não foram implementadas neste trabalho. Um operador de seleção que possui uma abordagem diferente e voltada para a tarefa de classificação é o Universal Suffrage, usado na ferramenta REGAL [Giordana & Neri 1995].

O operador citado tem como objetivo favorecer o surgimento de subpopulações de regras, através de um sistema no qual os exemplos de teste "votam" em uma das regras que os cubra, de forma proporcional à qualidade da regra.

4.6. Cruzamento

A operação de cruzamento é feita a nível de atributos. Uma nova regra é criada pegando-se um atributo por vez e escolhendo que restrições serão aplicadas sobre ele. A restrição usada virá de uma ou da outra regra usada como base dependendo da taxa de cruzamento.

Uma pequena modificação foi adicionada para evitar processamento desnecessário. Caso o cruzamento de duas regras resulte em uma regra igual a uma das regras usadas como base para o cruzamento, o cruzamento é tentado novamente até que uma nova regra seja criada. Se o número de tentativas exceder um limite (por exemplo, 5 tentativas) uma mutação é aplicada para tentar gerar uma regra diferente.

4.7. Mutação

Quando aplicado sobre uma regra, o operador de mutação simplesmente escolhe uma restrição e muda seu valor. A probabilidade desta restrição se tornar vazia é dada pelo coeficiente de criação de restrições vazias, explicado na Seção 4.2. A freqüência com que ocorrem mutações é dada pela taxa de mutação, também explicada na Seção 4.2. Durante os testes da ferramenta a taxa de mutação foi ajustada para efetuar em média uma mutação por geração.

4.8. Operador de "criação de sementes"

Outro operador foi implementado como opção à operação de mutação, para tentar auxiliar na estabilidade da população de regras. Ao invés de modificar aleatoriamente um atributo de alguma regra, o operador de mutação foi substituído por uma outra operação. Esta operação consiste na substituição de uma regra por outra nova, cujo objetivo é cobrir algum exemplo de treinamento não coberto por outras regras da população. O comportamento desta operação é similar ao operador de "criação de sementes" da ferramenta REGAL [Giordana & Neri 1995]. A cada geração, depois da avaliação da população, verifica-se quais exemplos de treinamento são cobertos por menos regras, e então uma nova regra é criada para cobrir um destes exemplos (escolhido aleatoriamente entre os menos cobertos, com a mesma quantidade de restrições vazias da população inicial). Testes empíricos mostraram uma pequena melhora, bem mais significativa em populações pequenas (menos de 100 indivíduos).

Após analisar cada parte do algoritmo genético e do classificador, é possível mostrar como estas partes foram colocadas na ferramenta. Isto é discutido no próximo capítulo.

5. A Ferramenta

A ferramenta consiste em um classificador baseado na indução de regras, que é feita pelo algoritmo genético. Todos os passos são manipulados através de uma interface gráfica simples, que permite o acompanhamento e intervenção durante todo o processo. É também possível a consulta e alteração de diversos parâmetros. A janela inicial da ferramenta é mostrada na Figura 5.

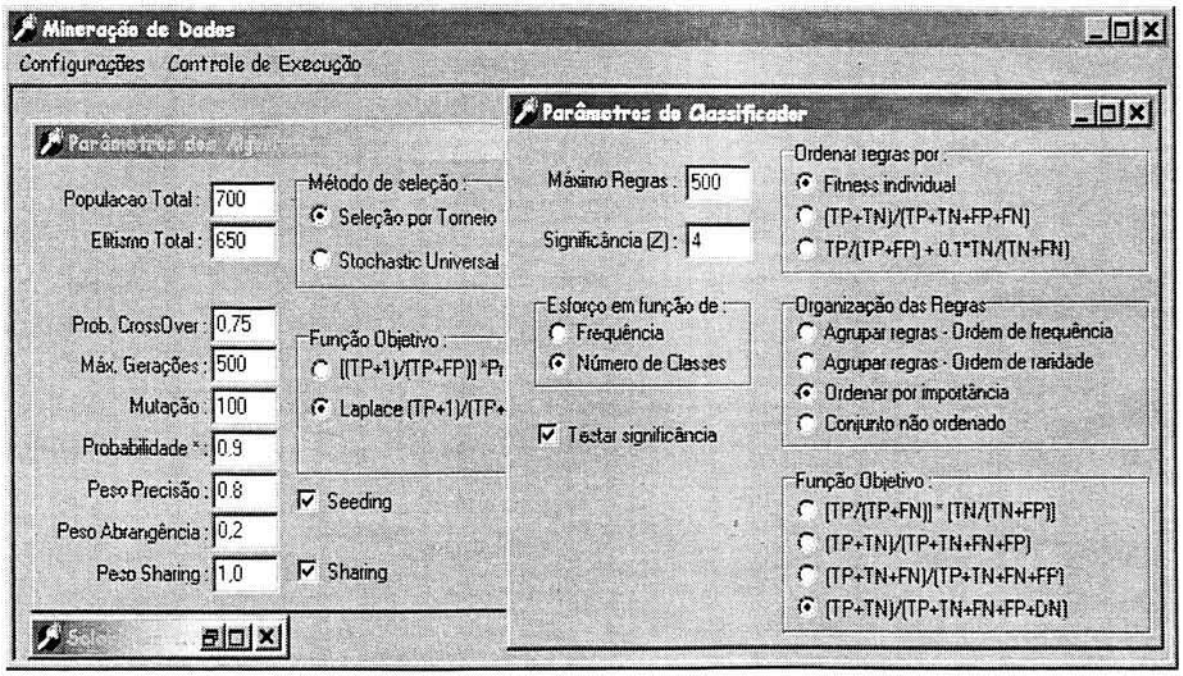


FIGURA 5. JANELA DA FERRAMENTA MOSTRANDO AS OPÇÕES DE CONFIGURAÇÃO.

A implementação foi feita inteiramente em Pascal orientado a objetos (Delphi[3]). O uso da linguagem Pascal permitiu maior simplicidade e clareza, e também tornou o código bastante modular devido à organização das classes dos objetos. O uso de um ambiente gráfico permitiu o uso de recursos visuais, que facilitaram a análise e a manipulação dos dados.

[3] Borland Delphi é marca registrada da Inprise.

Algumas classes abstratas foram criadas para a separação e organização do código:

- A classe "Problema" contém o algoritmo genético e seus parâmetros, além de todo o controle de execução.

- A classe "População" contém o conjunto de indivíduos de cada algoritmo genético.

- A classe "Indivíduo" contém a codificação, os dados e o código para avaliação da função de aptidão.

- A classe "Gene" contém dados específicos sobre cada componente do indivíduo (restrições sobre cada atributo, por exemplo).

- A classe auxiliar "ConjuntoInstâncias" contém os exemplos de treinamento e/ou teste. Esta separação permite independência da localização e formato dos dados, estejam eles na memória, num arquivo texto ou num servidor SQL remoto. Apenas a decodificação dos dados no formato C4.5 foi implementada.

A implementação dos algoritmos genéticos foi dividida em camadas. Na camada mais abstrata, implementou-se apenas o esqueleto básico, que pode ser especializado para qualquer aplicação que use um algoritmo genético. Na camada mais específica foi implementada a indução de regras com compartilhamento de recursos, que pode ser usada tanto em tarefas de classificação quanto na busca de regras úteis independentes. Uma outra camada foi implementada para tratar a heurística de agrupamento das regras. Isto foi feito para que futuramente possa ser tentada alguma variante da abordagem de Pittsburgh.

Quanto à parametrização, a implementação atual atribui como padrão os mesmos valores para os parâmetros dos algoritmos genéticos. O usuário pode a qualquer

tempo suspender a execução de qualquer um dos algoritmos separadamente para colocar valores específicos se desejado.

Implementações futuras ajustarão automaticamente a maioria destes parâmetros de acordo com os dados a analisar. Ajustes incluem o tamanho da população e as probabilidades de cruzamento e mutação, de acordo com a quantidade e tipo dos atributos dos dados de treinamento.

Cada passo do programa é acompanhado e controlado por uma janela específica. O primeiro passo é a escolha do conjunto de dados, como mostra a Figura 6. No caso de validação cruzada[4] (feita pela média de diversos subconjuntos), é possível abrir e ver todos os conjuntos de dados simultaneamente. Também é possível escolher onde gravar os resultados do classificador.

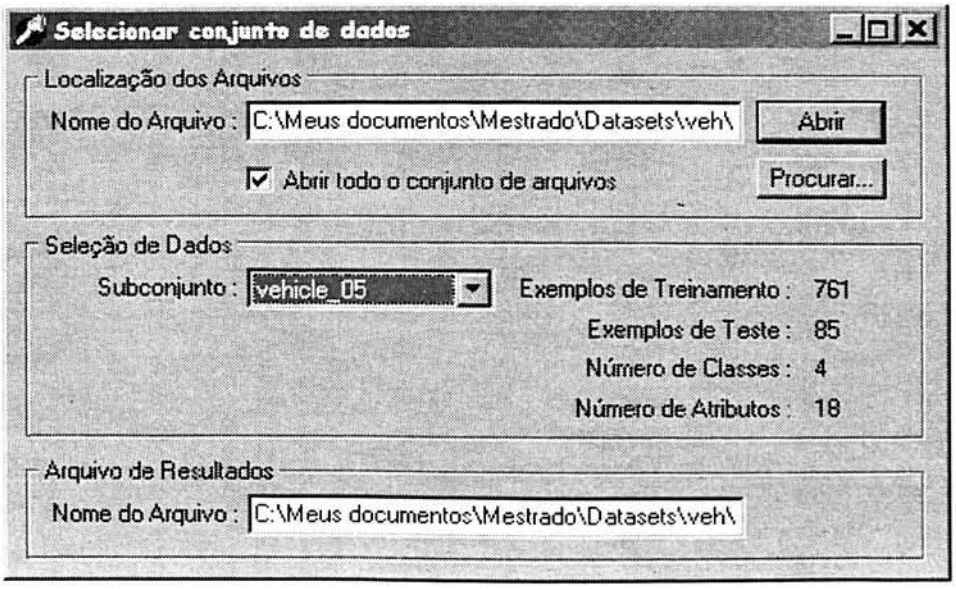


FIGURA 6: SELEÇÃO DO CONJUNTO DE DADOS.

Após a escolha do conjunto de dados, é preciso verificar os parâmetros do classificador e do algoritmo genético, mostrados nas Figuras 7 e 8 respectivamente.

[4] Também conhecida como "n-fold cross validation". Este termo é explicado em detalhes mais adiante junto com a metodologia empregada, na seção 5.1.

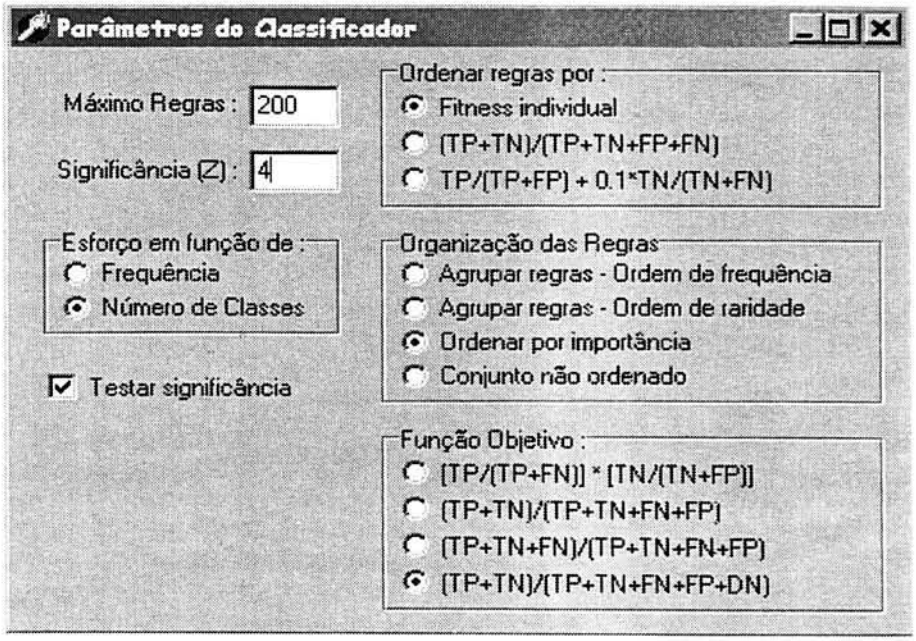


FIGURA 7: PARÂMETROS DE CONFIGURAÇÃO DO CLASSIFICADOR.

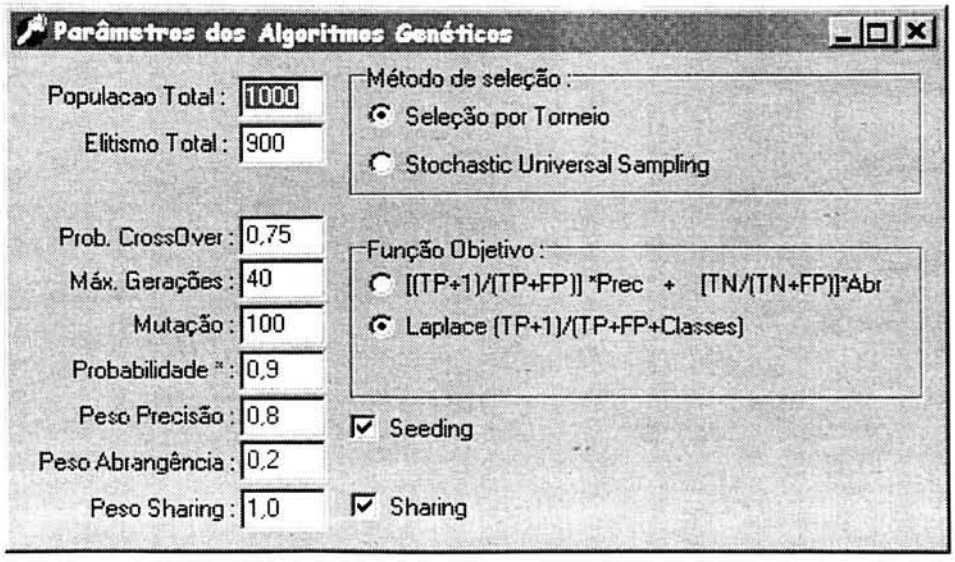


FIGURA 8: PARÂMETROS DE CONFIGURAÇÃO DOS ALGORITMOS GENÉTICOS.

Pode-se ver na Figura 9 um algoritmo genético durante a execução. Uma janela é aberta para cada classe analisada. Os principais parâmetros e situação da geração atual são mostrados, enquanto o andamento do algoritmo nas gerações anteriores é apresentado na parte inferior da janela.

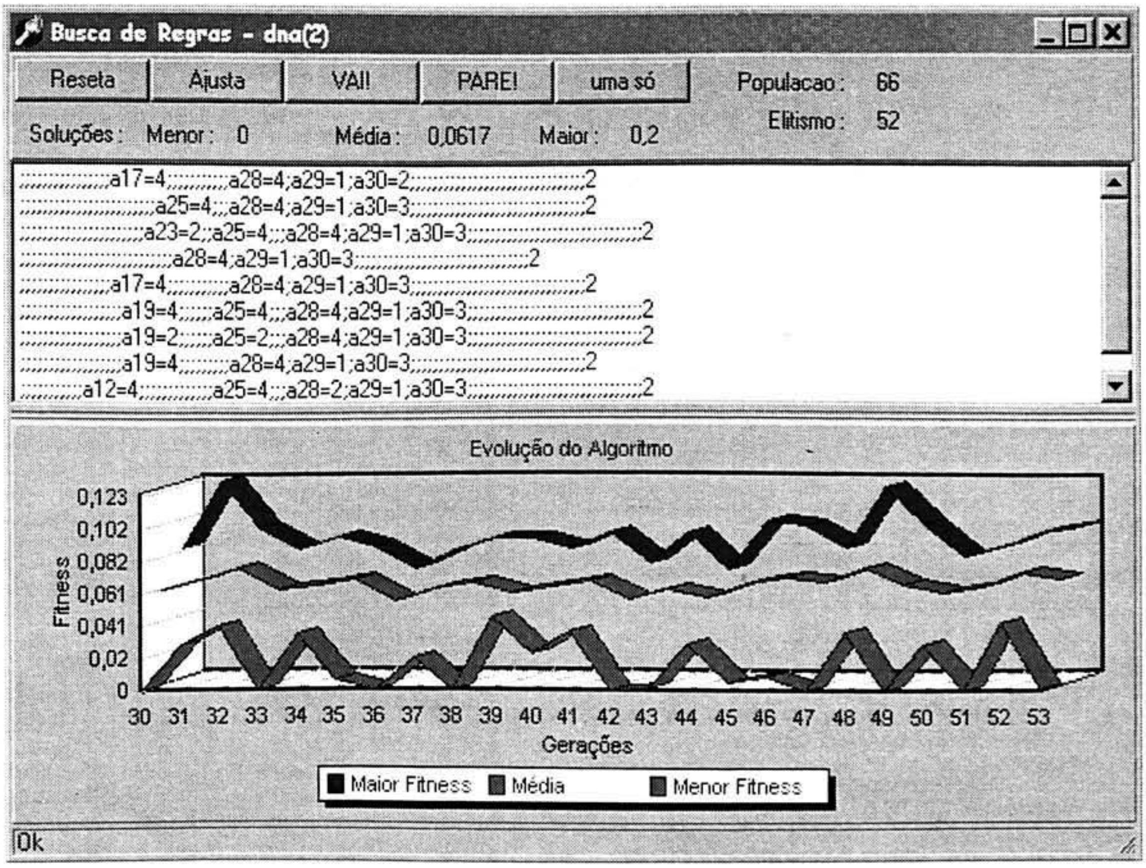


FIGURA 9: EXEMPLO MOSTRANDO O ALGORITMO GENÉTICO EM AÇÃO.

Um dos problemas visíveis pela figura é a dificuldade em criar um critério de término para a execução dos algoritmos genéticos. Nenhum critério simples como a média da população ou a variação do valor da função de aptidão do melhor indivíduo pode ser aplicado de maneira confiável para determinar quando a população atingiu a estabilidade. Mesmo que a população mantenha uma boa distribuicão de regras em diversas regiões do espaço de busca devido ao compartilhamento de recursos, os indivíduos sempre sofrerão pequenas variações causadas pelo operador de seleção. Esta variação, constatada durante os testes da ferramenta, é chamada de "genetic drift" [Goldberg & Segrest 1987][Mahfoud 1994A].

Um exemplo de execução do classificador pode ser visto na Figura 10. A figura mostra a janela de controle da ferramenta, onde é possível supervisionar o andamento da execução. No topo da janela aparecem as soluções de ordenação das regras para cada classe

padrão possível (cada número indica uma regra dentre todas as regras escolhidas para a montagem do classificador). O gráfico da janela mostra o progresso na taxa de acerto.

Devido ao compartilhamento de recursos feito pelos algoritmos genéticos e o "genetic drift", as populações de regras sofrem variação quanto aos exemplos de treinamento cobertos. Esta variação permanece mesmo quando o algoritmo já chegou perto do resultado ideal.

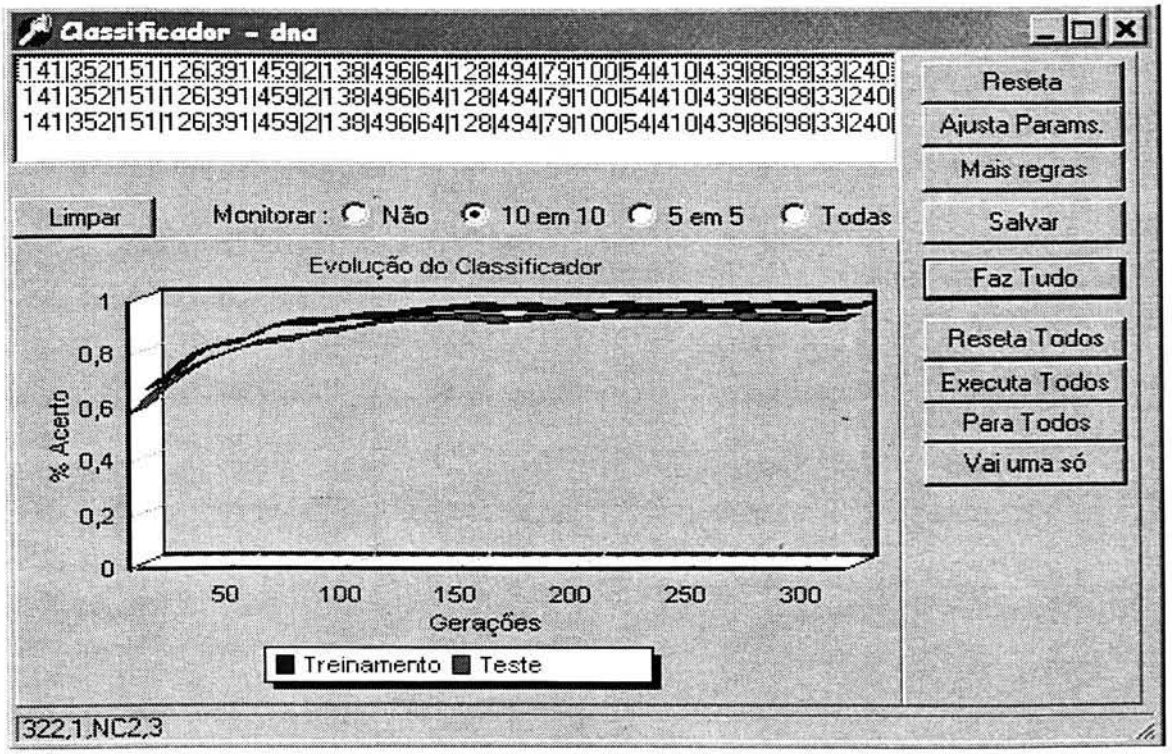


FIGURA 10: JANELA DE JUNÇÃO DE REGRAS MOSTRANDO O PROGRESSO DA EXECUÇÃO.

A implementação atual tem como critério de parada apenas o número de gerações especificado pelo usuário. Nenhum critério de término automático se mostrou satisfatório para indicar ao algoritmo quando parar. O percentual de acerto sempre sofre alguma variação, mesmo depois de várias tentativas de execução por mais de 2000 gerações. Após cerca de 50 a 250 gerações (dependendo do tamanho da população e do conjunto de dados), a taxa de acerto manteve-se em uma média mais estável, apesar de continuar variando a cada geração.

As seções a seguir mostram como a ferramenta se saiu ao ser comparada com outros classificadores em diversos conjuntos de dados. A Seção 5.1 explica a metodologia empregada e a Seção 5.2 dá informações sobre os conjuntos de dados. A Seção 5.3 dá informações básicas sobre os outros algoritmos e a Seção 5.4 mostra os resultados obtidos.

5.1. Metodologia empregada

A fim de tornar comparações mais precisas, os mesmos dados de treinamento e de teste de [Lim et al 1999] foram usados. Para obter a maior fidelidade possível nas comparações, a mesma metodologia de aferição da taxa de acerto foi aplicada. No trabalho de [Lim et al 1999], 33 algoritmos classificadores diferentes (detalhados na Seção 5.3) foram comparados em 32 tarefas de classificação com domínios variados. As comparações se baseiam na precisão de classificação nos dados de teste, na velocidade de execução e no número de regras, no caso de algoritmos que geram regras para classificação.

A metodologia usada em [Lim et al 1999] para medir o grau de precisão de um classificador em um conjunto de dados é baseada no percentual de classificações incorretas nos conjuntos de dados de teste. O valor usado nas comparações é o percentual total de erros, ou *taxa* de erro. A metodologia para determinar este valor é a seguinte:

- Para conjuntos de dados grandes (com mais de 1000 exemplos de teste), o conjunto de dados *de teste* (que não foi usado pelo classificador) é usado para medir a precisão do classificador.

- Para conjuntos de dados menores, a taxa de erros de classificação é medida usando-se o procedimento chamado de validação cruzada ou "ten-fold cross validation", mostrado abaixo:

    - Os dados originais são divididos em 10 subconjuntos distintos, sem exemplos em comum. Cada um dos subconjuntos contém aproximadamente a mesma proporção de exemplos de cada classe do conjunto original.

- Para cada subconjunto, um classificador é criado usando-se como dados de treinamento os outros 9 subconjuntos. O classificador gerado é então avaliado usando-se como teste o subconjunto não incluído nos dados de treinamento, resultando em uma taxa de erro parcial.

- A taxa de erro é obtida pela média de todas as 10 estimativas dos subconjuntos.

Além da estimativa ou taxa de erro, outra medida importante é a *margem* de erro, dada por:

$$\sqrt{p(1-p)/n}$$

onde p é a estimativa de erro para o conjunto de dados analisado e n é o número de exemplos de treinamento. O resultado de um classificador é considerado próximo ao melhor de todos se a diferença absoluta entre as taxas de erro é menor do que o valor da margem de erro [Lim et al 1999].

5.2. Dados usados nos experimentos

Os conjuntos de dados analisados pertencem a 16 domínios diferentes, sendo 14 deles obtidos de casos reais e 2 gerados artificialmente.

Para aumentar a quantidade de dados analisados e também testar os algoritmos quanto à presença de ruído, para cada domínio foi criado um outro conjunto com todos os dados do conjunto original, cada um acrescido de alguns atributos semelhantes mas com valores aleatórios. Os conjuntos de dados acrescidos de ruído são diferenciados pela presença do sinal "+" após o nome abreviado.

Uma breve descrição dos dados e suas características são dadas na Tabela 3. Informações mais detalhadas e referências sobre a origem destes dados podem ser encontrados em [Lim et al 1999].

Nome	Tam.	Classes	Atrib. Contínuos	Atrib. Discretos	Breve descrição do domínio
bcw	683	2	9		Diagnóstico de Câncer no seio obtido da
bcw+	683	2	18		Universidade de Wisconsin
cmc	1473	3	2	7	Escolha de método contraceptivo, dados obtidos
cmc+	1473	3	8	7	de uma pesquisa com mulheres da indonésia
dna	2000	3		60	Diagnóstico de tipos de junções em sequências
dna+	2000	3		80	de DNA
hea	270	2	7	6	Diagnóstico de doença cardíaca, dados obtidos
hea+	270	2	17	6	de Cleveland Clinic Foundation
bos	506	3	12	1	Análise do custo de vida nos subúrbios de
bos+	506	3	24	1	Boston
led	2000	10		7	Identificação de dígitos pelos leds acesos
led+	2000	10		24	(Domínio criado artificialmente)
bld	345	2	6		Diagnóstico de doenças de fígado em homens
bld+	345	2	15		via testes de sangue e consumo de álcool
pid	532	2	7		Diagnóstico de diabetes de pacientes da Índia
pid+	532	2	15		Diagnóstico pelo estado psicológico + testes
sat	4435	6	36		Classificação de uma imagem de satélite, dadas
sat+	4435	6	50		características das regiões vizinhas
seg	2310	7	19		Classificação de fragmentos de imagens de
seg+	2310	7	28		Outdoors
smo	1855	3	3	5	Previsão da atitude de pessoas diante de
smo+	1855	3	10	5	restrições a fumantes no local de trabalho.
thy	3772	3	6	15	Diagnóstico de problemas na tireóide
thy+	3772	3	10	25	
veh	846	4	18		Classificação de veículos usando como dados
veh+	846	4	30		as medidas da silhueta
vot	435	2		16	Classificação de um político como republicano
vot+	435	2		30	ou democrata de acordo com suas votações
wav	600	3	21		Diferenciação entre 3 formatos de onda
wav+	600	3	40		Dados criados artificialmente
tae	151	3	1	4	Avaliação de professores assistentes, usando
tae+	151	3	6	4	Como dados informações sobre o curso

TABELA 3: BREVE EXPLICAÇÃO SOBRE OS CONJUNTOS DE DADOS ANALISADOS

5.3. Algoritmos usados nas comparações

Esta seção dá apenas uma breve descrição dos algoritmos usados nas comparações feitas por [Lim et al 1999]. As referências indicadas contêm dados mais detalhados sobre cada algoritmo. Foram incluídos 22 algoritmos baseados em árvores de decisão, 9 algoritmos estatísticos e 2 algoritmos de redes neurais.

Árvores de Decisão:

- CART – Versão do algoritmo CART [Breiman et al 1984] tendo um critério especial de escolha de atributos. Dois métodos de poda foram testados.

- Árvore S-Plus – variante do método CART escrito na linguagem S [Becker et al 1988]. É descrita em detalhes em [Clark & Pregibon 1993]

- C4.5 – foi usada a release 8 [Quinlan 1993][Quinlan 1996], com valores padrão, gerando árvores de decisão e usando o programa do pacote para a geração de conjuntos de regras.

- FACT – é um algoritmo rápido de classificação, detalhado em [Loh & Vanichsetakul 1988]. Duas variantes foram testadas conforme o critério de escolha dos atributos.

- QUEST – este algoritmo é descrito em [Loh e Shih 1997]. Quatro variações diferentes foram testadas, conforme o critério de poda e escolha dos atributos. A versão usada foi a 1.7.10.

- IND – este algoritmo é detalhado em [Buntine 1992]. A versão usada é a 2.1, com valores padrão. Quatro variações são analisadas.

- OC1 – Este algoritmo é detalhado em [Murthy, et al 1994]. Três variações são analisadas. A versão usada nos testes foi a 3.

- LMDT – detalhado em [Brodley & Utgoff 1995], usando os valores padrão do programa.

- CAL5 – criado principalmente para valores numéricos, é detalhado em [Muller & Wysotzki 1994][Muller & Wysotzki 1997]. Usou-se uma ferramenta do pacote para encontrar os valores ótimos.

- T1 – Árvore de decisão baseada na escolha de apenas 1 atributo [Holte 1993].

Algoritmos Estatísticos

- LDA – análise linear do discriminante. Foi usado o SAS PROC DISCRIM [SAS 1990] com valores padrão.

- QDA – análise quadrática do discriminante. Foi usado o SAS PROC DISCRIM [SAS 1990] com valores padrão.

- NN - Implementação do método do vizinho mais próximo do pacote SAS PROC DISCRIM [SAS 1990].

- LDA – Polytomous Logistic Regression [Agresti 1990]. Análise logística do discriminante.

- FDA – análise flexível do discriminante [Hastie et al 1994]. Duas variações de [Friedman 1991] foram analisadas.

- PDA – trata-se de uma variante do algoritmo LDA mostrada em [Hastie et al 1995]

- MDA – análise mista do discriminante. Implementada usando S-Plus e a biblioteca `mda` [Hastie & Tibshirani 1996].

- POL – este é o algoritmo POLYCLASS, detalhado em [Kooperberg et al 1997]. Indicado por [Lim et al 1999] como um dos melhores de todos os algoritmos analisados.

Redes Neurais

- LVQ – Learning Vector Quantization, explicado detalhadamente em [Kohonen 1995]

- RBF – Radial Basis Function Network implementada usando [Sarle 1994]. Mais detalhes sobre o algoritmo em [Bishop 1995] e [Ripley 1996].

5.4. Resultados obtidos

	Tam. Trein.	Tam. Teste	Classes	Resultados esperados Melhor	Pior	Default	Resultados da Ferramenta	Pos.	Tam. Pop.	Margem Erro	Próx. Melhor
bcw	683	683	2	0,0278	0,0848	0,3500	0,0380	26,0	200	0,0063	Não
bcw+	683	683	2	0,0293	0,0760	0,3500	0,0446	13,0	200	0,0065	Não
cmc	1473	1473	3	0,4340	0,6010	0,5730	0,4840	17,0	800	0,0129	Não
cmc+	1473	1473	3	0,4320	0,5770	0,5730	0,4880	15,0	800	0,0129	Não
dna	2000	1186	3	0,0472	0,3790	0,4920	0,0523	10,0	2000	0,0062	Sim
dna+	2000	1186	3	0,0438	0,3790	0,4920	0,0514	11,5	3500	0,0059	Não
hea	270	270	2	0,1410	0,3410	0,4440	0,1790	10,0	800	0,0212	Não
hea+	270	270	2	0,1480	0,3110	0,4440	0,2074	20,0	900	0,0216	Não
bos	506	506	3	0,2210	0,3140	0,6570	0,2660	25,5	1000	0,0184	Não
bos+	506	506	3	0,2250	0,4220	0,6570	0,2726	14,0	1000	0,0186	Não
led	2000	4000	10	0,2680	0,8160	0,8900	0,2670	1,0	600	0,0070	Sim
led+	2000	4000	10	0,2650	0,8130	0,8900	0,2760	11,0	1000	0,0070	Não
bld	345	345	2	0,2790	0,4320	0,4190	0,2750	1,0	300	0,0241	Sim
bld+	345	345	2	0,2860	0,4410	0,4190	0,3130	2,0	600	0,0243	Não
pid	532	532	2	0,2210	0,3100	0,3333	0,2370	16,5	1000	0,0180	Sim
pid+	532	532	2	0,2170	0,3180	0,3333	0,2620	28,0	1000	0,0179	Não
sat	4435	2000	6	0,0980	0,4000	0,7650	0,1735	29,0	4000	0,0066	Não
sat+	4435	2000	6	0,1160	0,4100	0,7650	0,1950	29,0	4500	0,0072	Não
seg	2310	2310	7	0,0221	0,5150	0,8570	0,1030	29,0	1000	0,0031	Não
seg+	2310	2310	7	0,0264	0,5740	0,8570	0,1120	29,0	1200	0,0033	Não
smo	1855	1000	3	0,3040	0,4500	0,3050	0,3040	1,0	2000	0,0145	Sim
smo+	1855	1000	3	0,3050	0,4500	0,3050	0,3090	18,0	2200	0,0146	Sim
thy	3772	3428	3	0,0055	0,8900	0,0729	0,0170	17,0	400	0,0013	Não
thy+	3772	3428	3	0,0047	0,8750	0,0729	0,0228	19,0	600	0,0012	Não
veh	846	846	4	0,1450	0,4870	0,7390	0,3580	27,0	1000	0,0121	Não
veh+	846	846	4	0,1550	0,4870	0,7390	0,3450	27,0	1200	0,0124	Não
vot	435	435	2	0,0364	0,0617	0,3860	0,0435	9,0	200	0,0231	Sim
vot+	435	435	2	0,0412	0,0662	0,3860	0,0460	16,0	200	0,0236	Sim
wav	600	3000	3	0,1510	0,4770	0,6667	0,2470	18,0	600	0,0065	Não
wav+	600	3000	3	0,1600	0,4660	0,6667	0,2690	24,0	1000	0,0067	Não
tae	151	151	3	0,3250	0,6930	0,6560	0,5429	25,0	1500	0,0381	Não
tae+	151	151	3	0,4450	0,6960	0,6560	0,5350	19,0	1500	0,0404	Não
Totais							0,2292	17,4			8

TABELA 4: RESULTADOS OBTIDOS E A COMPARAÇÃO COM OS OUTROS CLASSIFICADORES

Todos os testes foram feitos usando-se um Pentium[5] 166Mhz com 80 Mb de memória RAM e Windows 95[6]. A plataforma e o equipamento são diferentes dos usados em [Lim et al 1999], o que dificulta a comparação de tempo de execução com os outros algoritmos.

A Tabela 4 mostra os resultados obtidos. A ferramenta foi testada em todos os 32 conjuntos de dados, obtendo uma média geral de taxa de erro igual a 0,2292. Comparando-se este valor às médias de todos os outros 33 classificadores, o resultado situa-se em 21° lugar. Segundo [Lim et al 1999] este resultado não é significativamente diferente (considerando-se uma margem de 10%) do melhor dos 33 classificadores, que chegou a uma estimativa de erro média de 0,195.

Em nenhum dos conjuntos de dados analisados a ferramenta obteve a pior colocação. Além disto, sua precisão foi próxima ou equivalente ao melhor dos algoritmos em 8 dos conjuntos de dados (`dna`, `led`, `bld`, `pid`, `smo`, `smo+`, `vot` e `vot+`). Apenas outros 12 algoritmos conseguiram desempenho próximo do ideal em 8 ou mais conjuntos de dados.

A Tabela 5 mostra uma comparação da ferramenta com algoritmos baseados em regras. Os dados já mostrados na Tabela 4 foram omitidos, exceto os resultados da Ferramenta, para facilitar a comparação. Na média o algoritmo ficou em 12° lugar, e conseguiu desempenho próximo do ideal em 10 conjuntos de dados. Comparando as taxas de erro, o algoritmo conseguiu a 15ª colocação dentre os 22 algoritmos analisados. A média do melhor algoritmo baseado em árvores, o QUEST (variante QL0), foi de 0,207. O C4.5 obteve média 0,220 contra 0,229 da ferramenta.

[5] Pentium é marca registrada da Intel
[6] Windows é marca registrada da Microsoft

	Resultados esperados			Resultados da	Pos.	Margem	Próx.
	Melhor	Pior	Default	Ferramenta		Erro	Melhor
bcw	0,0308	0,0848	0,3500	0,0380	5,5	0,0066	Não
bcw+	0,0293	0,0760	0,3500	0,0446	8,0	0,0065	Não
cmc	0,4400	0,5180	0,5730	0,4840	14,0	0,0129	Não
cmc+	0,4450	0,5230	0,5730	0,4880	10,0	0,0129	Não
dna	0,0472	0,3790	0,4920	0,0523	2,0	0,0062	Sim
dna+	0,0489	0,3790	0,4920	0,0514	2,0	0,0063	Sim
hea	0,1480	0,2700	0,4440	0,1790	5,0	0,0216	Não
hea+	0,1480	0,2700	0,4440	0,2074	9,5	0,0216	Não
bos	0,2210	0,2870	0,6570	0,2660	16,5	0,0184	Não
bos+	0,2250	0,3580	0,6570	0,2726	11,5	0,0186	Não
led	0,2710	0,8160	0,8900	0,2670	1,0	0,0070	Sim
led+	0,2660	0,8130	0,8900	0,2760	4,0	0,0070	Não
bld	0,2790	0,4320	0,4190	0,2750	1,0	0,0241	Sim
bld+	0,3130	0,4410	0,4190	0,3130	1,5	0,0250	Sim
pid	0,2210	0,3100	0,3333	0,2370	9,5	0,0180	Sim
pid+	0,2210	0,3180	0,3333	0,2620	19,0	0,0180	Não
sat	0,1380	0,4000	0,7650	0,1735	19,0	0,0077	Não
sat+	0,1400	0,4000	0,7650	0,1950	20,0	0,0078	Não
seg	0,0247	0,3600	0,8570	0,1030	21,0	0,0032	Não
seg+	0,0264	0,3600	0,8570	0,1120	21,0	0,0033	Não
smo	0,3040	0,4500	0,3050	0,3040	1,5	0,0145	Sim
smo+	0,3050	0,4500	0,3050	0,3090	16,0	0,0146	Sim
thy	0,0055	0,7260	0,0729	0,0170	16,0	0,0013	Não
thy+	0,0047	0,7500	0,0729	0,0228	18,5	0,0012	Não
veh	0,2060	0,4870	0,7390	0,3580	19,0	0,0139	Não
veh+	0,2340	0,4870	0,7390	0,3450	19,0	0,0146	Não
vot	0,3640	0,5800	0,3860	0,0435	6,5	0,0231	Sim
vot+	0,4120	0,6620	0,3860	0,0460	12,0	0,0236	Sim
wav	0,1760	0,4770	0,6667	0,2470	8,0	0,0070	Não
wav+	0,1620	0,4290	0,6667	0,2690	15,0	0,0067	Não
tae	0,3250	0,6930	0,6560	0,5429	18,0	0,0381	Não
tae+	0,4510	0,6960	0,6560	0,5350	11,0	0,0405	Não
Totais				0,2292	11,3		10

TABELA 5: COMPARAÇÃO COM OS OUTROS CLASSIFICADORES BASEADOS EM REGRAS

Todos os testes usaram como padrão a opção de tentar usar todas as regras geradas pelos algoritmos genéticos. Neste caso, o número de regras usadas em cada conjunto de dados variou de 45 a 92 em quase todos os conjuntos analisados. Exceções ocorreram nos seguintes casos especiais:

- nos conjuntos **vot** e **vot+** foram necessárias apenas 3 regras devido à grande simplicidade dos dados.

- em conjuntos contendo enormes espaços de busca (como **sat** e **dna**) foram necessárias de 197 a 337 regras, devido à grande quantidade de regras possíveis e à dificuldade de classificação. Casos intermediários foram os conjuntos **veh** e **wav**, que precisaram de 102 a 123 regras.

Na maioria dos casos, 40 gerações foram suficientes para chegar a um resultado próximo do "ideal", ou seja, quando a execução por mais gerações não alterou significativamente a precisão do classificador (a taxa de acerto não variou mais do que 1%), exceto em conjuntos de dados com um grande espaço de busca como o **dna**. O número de gerações, escolhido manualmente, variou de 40 a 200, dependendo do domínio. Nos experimentos realizados, o tempo de execução variou de 135 segundos (no caso do conjunto **led**, com um espaço de busca minúsculo, usando uma população de 600 regras e 50 gerações) até 3 horas no caso do conjunto **dna+**, devido ao seu tamanho ($2{,}48 \times 10^{56}$ combinações com 2000 exemplos de treinamento) e aos parâmetros usados (população de 1500 regras e 200 gerações).

O tempo de execução é determinado principalmente pelo domínio (número de exemplos de treinamento e o número de atributos contínuos e discretos), número de gerações e tamanho da população. Se o usuário não ficar satisfeito com o resultado obtido até determinado momento, sempre é possível deixar o algoritmo sendo executado por mais algumas gerações para melhorar o resultado.

Não foram feitas tentativas de descobrir a melhor combinação de parâmetros para cada conjunto de dados. O objetivo principal foi encontrar uma solução não muito distante da que seria obtida pela combinação ideal, ao mesmo tempo procurando minimizar o tempo de execução. Os parâmetros usados com mais frequência são mostrados nas Figuras 7 e 8.

A análise e os testes feitos até agora já permitem tirar algumas conclusões, e indicam muitos caminhos para trabalhos futuros, como mostrado no próximo capítulo.

6. Conclusões

A criação de um ambiente de trabalho para o uso da ferramenta permitiu a intervenção e análise dos resultados em qualquer estágio do processo. Estas características foram muito úteis durante os testes para determinar o que era necessário para melhorar o desempenho do classificador em cada um dos conjuntos de dados analisados.

A criação de um algoritmo genético por classe também resultou em grande melhora no desempenho, especialmente em conjuntos de dados com um grande espaço de busca. Isto se deve em grande parte à economia no tempo que foi gasto analisando regras que não cobriam nenhum exemplo nos dados de treinamento.

Os experimentos mostraram que uma das etapas mais importante é o ajuste dos parâmetros. Poucas combinações foram tentadas, indicando que a precisão em muitos conjuntos de dados ainda pode ser melhorada significativamente se melhores combinações de parâmetros forem encontradas. Parâmetros bem escolhidos também reduzem o tempo gasto pelo algoritmo.

Os critérios de ordenação e seleção para formar os conjuntos de regras tiveram grande variação dependendo do domínio. Muitos domínios precisaram de escolhas específicas para atingir um grau de precisão adequado. Isto indica que seria uma boa alternativa usar um outro algoritmo genético ou algum outro tipo de heurística diferente da citada na Seção 3.4 para agrupar as regras.

O compartilhamento de recursos no espaço fenotípico teve um ótimo desempenho e melhorou muito a eficiência em todos os casos. Esta operação forçou a população de regras a cobrir todos ou quase todos os exemplos de treinamento.

A média de erro da ferramenta nos conjuntos de dados analisados foi de 0,2292. De acordo com [Lim et al 1999] este resultado não é significativamente diferente do melhor

dos outros 33 classificadores. A precisão na classificação não teve diferenças significativas entre as execuções com os mesmos parâmetros no mesmo conjunto de dados.

A ferramenta apresentada mostrou-se capaz de tratar adequadamente conjuntos de dados com e sem ruído, e não é afetada pela ordenação dos exemplos de treinamento. Os resultados obtidos até agora mostram que a ferramenta é robusta e genérica, e pode ser usada diretamente numa grande variedade de aplicações. Resultados parciais podem ser obtidos a qualquer momento, e o usuário pode escolher livremente quanto tempo deseja gastar na busca de uma solução através da escolha do número de gerações.

Muitas características do algoritmo podem ser melhoradas para atingir resultados melhores, como a inclusão de novos operadores genéticos, ajuste automático de parâmetros e outros melhoramentos citados na seção de trabalhos futuros.

6.1. Trabalhos futuros

Há diversas partes da ferramenta que podem ser melhoradas, em diversos aspectos. Os principais melhoramentos propostos distribuem-se pelas seguintes áreas:

Agrupamento das regras

- Criar um critério de término confiável que não dependa do usuário.

- Testar e comparar outras heurísticas para a ordenação e escolha das regras que farão parte do classificador (Seção 3.4).

- Criar outro algoritmo genético para buscar conjuntos de regras. Neste caso pode-se usar heurísticas para criar uma boa população inicial, e também é possível intercalar a execução dos algoritmos que buscam as regras com a

busca de conjuntos de regras usando as melhores regras das populações assim que forem surgindo.

- Unir técnicas de junção de regras de outros algoritmos, como a heurística do Algoritmo CN2 [Clark & Nibblet 1989], que buscam uma regra por vez e excluem os exemplos cobertos pelas regras já usadas. Isto implica em recalcular a função de aptidão dos indivíduos restantes a cada regra adicionada, o que torna a montagem do classificador muito mais lenta.

Busca de regras com Algoritmos Genéticos

- Tornar os parâmetros do algoritmo auto-ajustáveis, testando valores de estudos sobre casos genéricos [Spears 1995] [Cantú-Paz 2000] [Horn, Deb & Goldberg 1994]. Pode-se procurar métricas e analisar características da população, a fim de determinar quais parâmetros precisam ser modificados a cada geração.

- Usar outros métodos de formação de nichos [Watson 1998], também no espaço fenotípico.

- Implementar outros critérios de término para os algoritmos genéticos de cada classe, a fim de tornar a ferramenta mais automatizada.

- Adaptar a ferramenta para execução em vários processadores [Freitas & Lavington 1998].

Outros trabalhos futuros relacionados

- Analisar detalhadamente o tempo de execução do algoritmo em comparação com os outros em [Lim et al 1999] e também seu comportamento em conjuntos de dados maiores.

- Usar as regras encontradas para outras aplicações de mineração de dados, como descobrir regras "interessantes" [Noda, Freitas & Lopes 1999].

- Análise teórica do método de formação de nichos utilizado, de forma similar a [Horn, Deb & Goldberg 1994].

- Determinar em que tipos de domínio a Ferramenta se sai melhor, através da análise das características dos conjuntos de dados e a comparação de seus resultados com outras ferramentas.

Bibliografia

[Adriaans & Zaatinge 1996] ADRIAANS, Pieter & ZANTINGE, Dolf. Data mining. Harlow. Addinson Wesley Longman. 1996,159p.

[Agresti 1990] AGRESTI, A. Categorical data analysis. John Wiley & Sons, New York, NY, 1990.

[Aha et al. 1991] AHA, D.W. ; KIBLER, D. ; ALBERT, M. K. Instance-Based learning algorithms. Machine Learning 6, 1991, 37-66.

[Becker et al 1988] BECKER, R. A. ; CHAMBERS, J. M. ; WILKS, A. R. The new S language. Wadsworth, 1988.

[Bishop 1995] BISHOP, C. M. Neural networks for pattern recognition. Oxford University Press, New York, NY, 1995.

[Brazdil & Clark 1990] BRAZDIL, Pavel & CLARK, Peter. Learning from Imperfect Data. In: Machine Learning, Meta-reasoning and Logics, pp. 207-232, eds. P. B. Brazdil e K. Konolige (1990), Boston: Kluwer.

[Breiman et al 1984] BREIMAN, L. ; FRIEDMAN, J. ; OLSHEN, R. ; STONE, C. Classification and regression trees. Chapman and Hall, New York, NY, 1984.

[Brodley & Utgoff 1995] BRODLEY, C. E. & UTGOFF, P. E. Multivariate decision trees. Machine Learning 19:45-77, 1995.

[Buntine 1992] BUNTINE, W. Learning classification trees. Statistics and Computing, 2:63-73, 1992.

[Cantú-Paz 2000] CANTÚ-PAZ, Eric. Selection Intensity in Genetic Algorithms with Generation Gaps. in 2000 Genetic and Evolutionary Computation Conference (GECCO), 2000, Las Vegas, Morgan Kaufmann.

[Cestnik 1990] CESTNIK, B. Estimating probabilities: a crucial task in machine learning. in ECAI-90, 1990.

[Clark & Pregibon 1993] CLARK, L. A. ; PREGIBON, D. Tree-based models. In CHAMBERS, J. M. & HASTIE, T. J. eds. Statistical models in S, pp. 377-419. Chapman & Hall, New York, NY, 1993.

[Clark & Niblett 1989] CLARK, P. & NIBLETT, T. The CN2 induction algorithm. Machine Learning, 3/4, Netherlands, Kluwer (1989) pp.261-283.

[Deb & Goldberg 1989] DEB, K. & GOLDBERG, D. An investigation of niches and species formation in genetic function optimization. Proc. 3rd Int. Conf. On Genetic Algorithms, Morgan Kaufmann (Fairfax, VA), pp. 42-50.

[DeJong 1975] DEJONG, K. A. An Analysis of the behavior of a class of genetic adaptative systems. PhD thesis, University of Michigan, 1975.

[Devlin 1997] DEVLIN, Barry. Data warehouse: from architecture to implementation. Addinson Wesley Longman. 1997, 432p.

[Freitas & Lavington 1998] FREITAS, Alex A. & LAVINGTON, Simon H. Mining very large databases with parallel precessing. Boston, Kluwer Academic, 1998, 208p.

[Friedman 1991] FRIEDMAN, J. Multivariate adaptive regression splines (with discussion). Annals of Statistics, 19:1-141, 1991.

[Giordana & Neri 1995] GIORDANA, Attilio & NERI, Filippo. Search-intensive concept induction in Evolutionary Computation vol.3 n.4, 1995, pp 375-416.

[Goldberg 1989] GOLDBERG, David E. Genetic algorithms in search, optimization, and machine learning. Alabama, Addinson-Wesley, 1989, 413p.

[Goldberg & Richardson 1987] GOLDBERG, D. E. & RICHARDSON, J. Genetic Algorithms with Sharing for Multimodal function optimization. In Proceedings of the Second International Conference on Genetic Algorithms, 1987.

[Goldberg & Segrest 1987] GOLDBERG, D. E. & SEGREST, P. Finite markov chain analysis of genetic algorithms. Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms, 1987, 1-8.

[Hastie et al 1995] HASTIE, T. ; BUJA, A. ; TIBSHIRANI, R. Penalized discriminant analysis. Annals of Statistics, 23:73-102, 1995.

[Hastie & Tibshirani 1996] HASTIE, T. & TIBSHIRANI, R. Discriminant analysis by gaussian mixtures. Journal of the Royal Statistical Society, Series B, 58:155-176, 1996.

[Hastie et al 1994] HASTIE, T. ; BUJA, A. ; TIBSHIRANI, R. Flexible discriminant analysis by optimal scoring. Journal of the American Statistical Association., 89:1255-1270, 1994.

[Holland 1986] HOLLAND, J. H. Escaping Brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems in MICHALSKI, R. ; CARBONELL J. ; MITCHELL, T. (Eds.), Machine Learning: an AI Approach, vol II. Morgan Kaufmann, Los Altos, CA, pp. 593-623.

[Holsheimer et al. 1996] HOLSHEIMER, M. ; KERSTEN, M. ; Siebes, A. Data surveyor: searching the nuggets in parallel in: FAYYAD, U. M. et al. (Eds.) Advances in knowledge discovery and data mining, 447-467. AAAI Press, 1996.

[Holte 1993] HOLTE, R. C. Very simple classification rules perform well on most commonly used datasets. Machine Learning, 11:63-90, 1993.

[Horn, Deb & Goldberg 1994] HORN, J., DEB, K. & GOLDBERG, D. E. Implicit Niching in a Learning Classifier System: Nature's way. Evolutionary Computation, 2, (1994) 37-66. (Illigal Report No. 94001 in Illinois Genetic Algorithm Laboratory, University of Illinois)

[Kelly & Davis 1991] KELLY Jr., J.D. & DAVIS, L. A hybrid genetic algorithm for classification. Proc. 12th IJCAI-95, 645-650, 1991.

[Knight & Sem 1995] KNIGHT, L. & SEM, S. PLEASE: a prototype learning system using genetic algoritms. Proc. 6th Int. Conf. Genetic Algorithms, 429-435, 1995.

[Kohonen 1995] KOHONEN, T. Self-organizing maps. Springer-Verlag, Heidelberg, 1995.

[Kooperberg et al 1997] KOOPERBERG, C. , BOSE, S. & STONE, C. J. Polychotomous regression. Journal of the American Statistical Association, 92:117-127, 1997.

[Langley 1996] LANGLEY, Pat. Elements of machine learning. San Francisco, Morgan Kaufmann, 1996, 419p.

[Lim et al 1999] LIM, T.-S. , LOH, W.-Y. & SHIH, Y.-S. A comparison of prediction accuracy, complexity and training time of 33 old and new classification algorithms in Machine Learning Journal. Kluwer Academic, Boston.

[Loh & Shih 1997] LOH, W.-Y. & SHIH, Y.-S. Split selection methods for classification trees. Statistica Sinica, 7: 815-840, 1997.

[Loh & Vanichsetakul 1988] LOH, W.-Y. & VANICHSETAKUL, N. Tree-structured classification via generalized discriminant analysis (with discussion). Journal of the American Statistical Association, 83: 715-728, 1988.

[Mahfoud 1992] MAHFOUD, Samir W. Crowding and preselection revisited. In R. Manner & B. Manderick (Eds.), Parallel Problem Solving from Nature, 2. North-Holland, 1992, pp. 27-36.

[Mahfoud 1994] MAHFOUD, Samir W. Population Sizing for Sharing Methods. Illigal Report No. 94005 in Illinois Genetic Algorithm Laboratory, University of Illinois, 1994.

[Mahfoud 1994A] MAHFOUD, Samir W. Genetic Drift in Sharing methods. Proceedings of the first IEEE conference on Evolutionary Computation, World Congress on Computational Intelligence, 1994, pp. 67-72.

[McCallum & Spackman 1990] McCALLUM, R. A. & SPACKMAN, K. A. Using genetic algorithm to learn disjunctive rules from examples. Proc. Int. Conf. on Machine Learning (Austin, Texas), 1990, pp. 149-152.

[Mitchell 1997] MITCHELL, Melanie. An introduction to genetic algorithms. Cambridge. MIT Press. 1997, 207p.

[Muller & Wysotzki 1994] MULLER, W. & WYSOTZKI, F. Automatic construction of decision trees for classification. Annals of Operations Research, 52:231-247, 1994.

[Muller & Wysotzki 1997] MULLER, W. & WYSOTZKI, F. The decision tree algorithm CAL5 based on a statistical approach to its splitting algorithm. In G. Nakhaezadeh & C. C. Taylor, eds. Machine Learning and Statistics: the interface, pp. 45-65. John Wiley & Sons, New York, NY, 1997.

[Murthy et al 1994] MURTHY, S. K. ; KASIF, S. ; SALZBERG, S. A system for induction of oblique decision trees. Journal of Artificial Intelligence Research, 2:1-33, 1994.

[Niblett 1987] NIBLETT, T. Constructing Decision Trees in noisy domains. In I.Bratko & N. Lavrac, eds, Progress in Machine Learning (proceedings of the 2nd European Working Session on Learning), pages 67-78. Sigma, Wilmslow, UK, 1987.

[Noda, Freitas & Lopes 1999] NODA, E., FREITAS, A. A. & LOPES, H. S. Discovering interesting prediction rules with a genetic algorithm. in Proc. Congress on Evolutionary Computation (CEC-99), Washington D. C., em Julho de1999, pp. 1322-1329.

[Quinlan 1993] QUINLAN, J. R. C4.5: programs for machine learning. San Mateo. Morgan Kaufmann, 1993, 302p.

[Quinlan 1996] QUINLAN, J. R. Improved use of continuous attributes in C4.5. Journal of Artificial Intelligence Research, 4:77-90, 1996.

[Ripley 1996] RIPLEY, R. D. Pattern recognition and neural networks. Cambridge University Press, Cambridge, 1996.

[Sarle 1994] SARLE, W. S. Neural networks and statistical models. In Proceedings of the Nineteenth Annual SAS Users Groups International Conference, pp. 1539-1550, Cary, NC, 1994.

[SAS 1990] SAS Institute Inc. SAS/STAT user's guide, version 6. Volume 1 & 2. SAS Institute Inc, Cary, NC, 1990.

[Smith 1983] Smith, S. Flexible Learning of Problem Solving Heuristics through Adaptative Search. Proc. 8th Int. Conf. on Artificial Intelligence. Karlsruhe, Alemanha, 1983, pp. 422-425.

[Spears 1995] SPEARS, W. M. Recombination parameters. In Handbook of Evolutionary Computation. IOP Publishing Ltd and Oxford University Press, 1995, seção E1.3.

[Watson 1998] WATSON J.-P. A performance Assessment of Modern Niching Methods for Parameter Optimization Problems. Colorado State University, Fort Collins. (1998)

[Weiss & Kulikowski 1991] WEISS, S. M. & KULIKOWSKI, C. A. Computer systems that learn. Morgan Kaufmann, 1991.

[Wilson 1987] WILSON, S. Classifier Systems and the Animat Problem. Machine Learning, 2. Kluwer Academic Publishers, 1987, pp. 199-228.