UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

INVESTIGAÇÃO DO MODELO DE APRENDIZADO HÍBRIDO GENÉTICO-BASEADO EM INSTÂNCIAS

LUCIANA DE NARDIN

SÃO CARLOS – SP
2003

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

INVESTIGAÇÃO DO MODELO DE APRENDIZADO HÍBRIDO GENÉTICO-BASEADO EM INSTÂNCIAS

LUCIANA DE NARDIN

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

SÃO CARLOS – SP
2003

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

N278im

De Nardin, Luciana.
Investigação do modelo de aprendizado híbrido genético-baseado em instâncias / Luciana De Nardin. -- São Carlos: UFSCar, 2003.
162 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2003.

1. Aprendizagem de máquina. 2. Aprendizado baseado em instâncias. 3. Algoritmos genéticos. I. Título.

CDD: 006.31 (20ª)

“Os cientistas tentaram entender a felicidade. Pesquisaram-na, fizeram estatísticas, mas ela os confundiu, falando-lhes: ‘A lógica numérica jamais compreendera a lógica da emoção!’ Perturbados, descobriram que o mundo da emoção é indecifrável pelo mundo das idéias. Por isso, os cientistas que viveram uma vida exclusivamente lógica e rígida foram infelizes”.

(Augusto Cury)

Agradecimentos

A Deus, pela força, coragem, luz e paciência em tanto momentos difíceis.

À Profª. Dra. Maria do Carmo Nicoletti, pela orientação, amizade, paciência e, também, por todo conhecimento compartilhado no desenvolvimento deste trabalho.

Ao Prof. Dr. José Fernando Colafemina, do Hospital das Clínicas da Faculdade de Medicina de Ribeirão Preto da Universidade Estadual de São Paulo, pelo fornecimento de uma das bases de dados utilizada nos experimentos.

A meu filho João Pedro que, mesmo sem saber, foi fonte inspiradora nesta etapa de minha vida e no desenvolvimento deste trabalho.

A toda minha família, pelo grande incentivo e por todo carinho. Em especial, a minha irmã Silvana e a meu cunhado Alex, por transformar sua casa em minha casa durante tanto tempo.

A todos os amigos feitos no Departamento de Ciência da Computação da Universidade Federal de São Carlos e especialmente, a Alexandre Charles, Fernando Duarte, Marcelo Módolo e Will Ricardo, que se tornaram grandes amigos e estarão para sempre presentes em meu coração e gravados em minha memória.

A Luiz Garcia Palma Neto, por sua importante colaboração no desenvolvimento do sistema que viabilizou a geração dos arquivos de treinamento e teste.

E, por último, mas não menos importante, a Everton Leandro, por todo carinho e companheirismo demonstrados durante o desenvolvimento deste trabalho e, especialmente, em sua etapa final.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Resumo

Este trabalho investiga a contribuição de algoritmos genéticos a métodos de aprendizado baseado em instâncias, particularmente o *k*-NN, IB1 e IB2. O principal objetivo da investigação é otimizar o desempenho de cada um dos três algoritmos por meio de uma colaboração genética. É descrito um sistema computacional que implementa cada um dos métodos e sua variante genética. Os resultados de experimentos em vários domínios de conhecimento são apresentados e analisados.

Abstract

This research work investigates the contribution of genetic algorithms to instance based learning methods, particularly *k*-NN, IB1 and IB2. The main focus of the investigation is to optimize the performance of three algorithms by means of a genetic algorithm. A computational system that implements each method and its genetic variant is described. Results of experiments in some knowledge domains are presented and analised.

1 capítulo

Introdução

1.1 Considerações sobre Aprendizado de Máquina

A área de Aprendizado de Máquina (AM) é uma subárea da Inteligência Artificial (IA) que investe na pesquisa de métodos e técnicas que viabilizam o aprendizado por computadores. A pesquisa em AM acontece em duas frentes principais: a primeira é a da investigação teórica e criação de algoritmos, que permite a formalização e o estabelecimento de resultados gerais, especificando condições necessárias para o aprendizado acontecer, definindo seus limites de atuação e evidenciando as restrições que devem ser impostas com vistas aos resultados que se quer obter. A segunda frente de pesquisa acontece por meio do uso dos vários modelos de aprendizado automático, implementados como sistemas automatizados, nos mais variados domínios de conhecimento e aplicados às mais variadas tarefas de aprendizado.

Devido talvez ao pouco conhecimento que se tem de como o processo de aprendizado acontece em seres vivos e também devido ao fato do aprendizado poder acontecer das mais variadas maneiras, a automatização desse processo tem sido proposta e implementada como sistemas que são baseados em modelos subsidiados por conceitos das mais variadas áreas, tais como psicologia, estatística, matemática, biologia, teoria da informação, etc.

Não existe uma única proposta de taxonomia para sistemas de AM. Neste trabalho é adotada a sugerida em CARBONELL (1989) que classifica os sistemas de aprendizado em quatro grupos:

- aprendizado indutivo (leia-se simbólico);
- algoritmos genéticos;
- redes neurais;
- aprendizado analítico – aprendizado baseado em explicações.

Uma outra taxonomia, por exemplo, poderia propor dois agrupamentos, nomeados de sistemas indutivos e sistemas dedutivos. De acordo com essa proposta, os três primeiros grupos acima seriam reunidos sob o rótulo de sistemas indutivos e o quarto seria tratado como um sistema dedutivo.

Dentre os vários modelos existentes de aprendizado indutivo de máquina, os que têm tido maior projeção são os métodos "diretos", isto é, os baseados em instâncias e os que usam árvores de decisão, regras ou então, redes neurais. Muito embora árvores de decisão possam ser facilmente "traduzidas" em regras, pode ser observado na literatura que muita pesquisa na área de AM aborda cada um desses modelos de maneira distinta. O grande investimento em pesquisa que envolve esses modelos é justificado pelos bons resultados obtidos em avaliações empíricas de sistemas que os implementam.

Este trabalho de pesquisa investiga a colaboração entre dois modelos indutivos de aprendizado, o simbólico e o genético. O modelo simbólico é representado pelos algoritmos NN e *k*-NN (COVER & HART (1967)) e por algoritmos da família IBL (*Instance Based Learning*)[1] (AHA (1991)). O principal objetivo na investigação dessa colaboração é a de obter um modelo híbrido que, usando um Algoritmo Genético (AG), possa ter um desempenho melhor do que um algoritmo baseado em instâncias quando da classificação de novas instâncias.

Vale lembrar que em algumas referências, o aprendizado baseado em instâncias é caracterizado como simbólico e, em outras, como uma categoria à parte, isto é, aquela de algoritmos baseados em instâncias.

[1] Para facilitar sua identificação na literatura, optamos por manter o nome original da família.

Objetivando o estabelecimento da terminologia utilizada no trabalho, a próxima subseção apresenta e discute os principais conceitos e estratégias relacionados a aprendizado indutivo.

1.2 Aprendizado Indutivo

O modelo mais investigado e mais utilizado de AM é aquele que se caracteriza como o aprendizado que faz uso de processos indutivos para a elaboração da expressão final do conceito que irá representar o conhecimento adquirido via aprendizado. Como visto da seção anterior, enquadram-se nesta categoria o aprendizado simbólico, o genético e o neural.

Para que o aprendizado indutivo aconteça é fundamental que um conjunto de instâncias, que representam os conceitos a serem aprendidos, esteja disponibilizado. Tal conjunto de instâncias é conhecido como conjunto de treinamento – é a informação inicial para um sistema de AM, a partir da qual uma generalização vai acontecer, produzindo a expressão final do conceito. Dependendo do método usado, essa expressão final do conceito pode ser simplesmente um conjunto de regras, uma árvore de decisão ou, então, se o método utilizado for o neural, uma rede neural.

Um outro conjunto de instâncias freqüentemente disponibilizado é o conhecido como conjunto de teste – o conjunto de teste serve para "avaliar" a precisão da expressão induzida do conceito. Empiricamente, "boas" expressões do conceito são aquelas que têm um bom desempenho, quando da classificação de instâncias de teste.

Existem disponibilizados no *University of California at Irvine Repository* (UCI *Repository*) (BLAKE & MERZ (1998)) um grande número de arquivos de dados, que servem como *benchmark* para testes de sistemas de AM. A Figura 1.1 apresenta um esquema de validação comumente utilizado em AM.

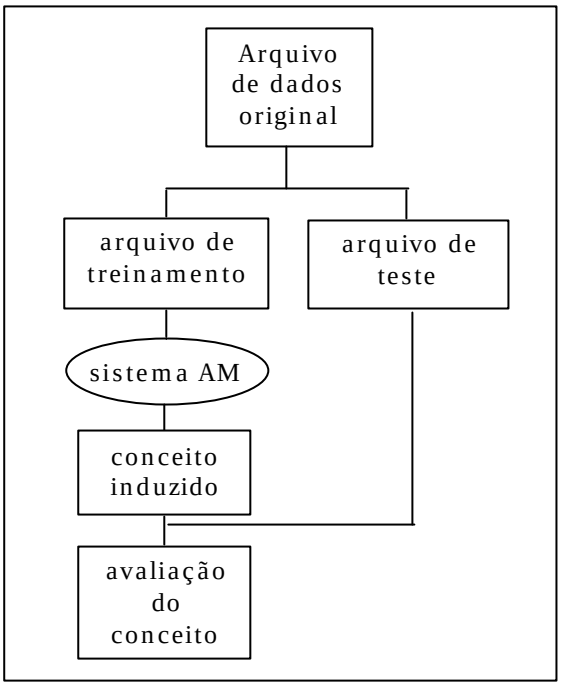

Figura 1.1 Esquema de avaliação de sistemas de AM

Geralmente, arquivos de dados consistem de um conjunto de instâncias (ou exemplos) de treinamento descrito por vetores de pares atributo-valor_do_atributo e de uma classe associada. A determinação da classe é, na maioria dos casos, feita por um especialista humano da área de conhecimento descrita pelos dados. O conjunto de atributos que descreve as instâncias é fixo.

Considerando uma situação genérica na qual instâncias de treinamento são descritas pelos atributos: $atrib_1$, $atrib_2$, $atrib_3$,..., $atrib_N$ e por uma classe associada, $classe_i$ (dentre M possíveis classes), e supondo que o número de possíveis valores de $atrib_i$ seja notado por $|atrib_i|$, uma representação geral do valor da instância de treinamento é dada pelo vetor:

$$V_{1i_1} \quad V_{2i_2} \quad V_{3i_3} \quad \ldots \quad V_{Ni_N} \quad classe_j$$

onde $i_k \in \{1, 2,..., |atrib_k|\}$, $k = 1, 2,..., N$ e $j \in \{1, 2,..., M\}$.

Dentre as principais características associadas aos métodos indutivos de AM, estão:

1. aprendizado supervisionado e não supervisionado: no aprendizado supervisionado cada uma das instâncias de treinamento

é descrita por um vetor de atributo_valor e uma classe associada. O conceito é induzido usando também a informação da classe a qual cada instância pertence. No aprendizado não supervisionado isso não acontece; a informação sobre a classe de cada instância não existe, e consequentemente, apenas a informação sobre os valores de atributos é usada para agrupar instâncias que, empiricamente, pertencem a uma mesma classe;

2. aprendizado incremental e não-incremental: no aprendizado incremental, a expressão do conceito é construída exemplo a exemplo, o que implica em constante revisão, uma vez que a classificação de um novo exemplo pode fazer com que seja necessário um rearranjo da expressão do conceito. Dessa forma, a expressão do conceito vai sofrendo modificações à medida que os exemplos vão se tornando disponíveis. Isto não acontece no caso não-incremental, pois neste o conjunto de treinamento deve estar disponível desde o inicio do processo, considerando que a expressão do conceito é induzida levando em consideração todos os exemplos de uma só vez (NICOLETTI (1994));

3. aprendizado por reforço: permite que máquinas e agentes de *software* automaticamente determinem o comportamento ideal de acordo com um contexto específico, com o objetivo de maximizar o desempenho. A resposta ao aprendizado é simplesmente um valor escalar que representará o sucesso ou falha;

4. aprendizado de um conceito *versus* aprendizado de vários conceitos: diz respeito à capacidade de um sistema aprender a expressão de apenas um ou então de vários conceitos de cada vez;

5. uso de Teoria do Domínio: se um sistema não possui informação a respeito do problema de aprendizado a ser tratado, a expressão do conceito deve ser induzida com base apenas nos exemplos disponíveis. Para que soluções de problemas complexos de aprendizado possam ser encontradas, entretanto, muitas vezes é necessário que um volume substancial de conhecimento sobre o problema esteja disponível, a fim de subsidiar a indução do conceito. Este conhecimento é referenciado como Teoria do Domínio.

Particularmente, o modelo de aprendizado indutivo categorizado como Programação Lógica Indutiva (MUGGLETON (1992)) é fortemente baseado no uso de Teoria do Domínio, para a indução da expressão apropriada dos conceitos;

6. linguagens de descrição: é a linguagem usada para expressar os exemplos, a Teoria do Domínio bem como os exemplos induzidos. Geralmente são usadas linguagens formais. Existem situações de aprendizado onde exemplos são mais convenientemente expressos em uma determinada linguagem, enquanto que Teoria do Domínio e conceitos em uma outra. As linguagens são denominadas:

- linguagem de descrição de instâncias: utilizada para descrever os exemplos do conjunto de treinamento;
- linguagem de descrição de conceitos: utilizada pelo algoritmo de aprendizado para construir as descrições do conceito sendo aprendido;
- linguagem de descrição da Teoria do Domínio: utilizada para descrever o conhecimento disponível ao sistema de aprendizado.

Como apontado em SANTOS (1997), além dessas características é fundamental que um sistema de AM possa ser caracterizado em função:

- dos domínios de problemas onde possui bom/mau desempenho;
- de implementar (ou não) mecanismos para o tratamento de ruídos nos dados;
- dos tipos de dados que consegue (ou não) tratar.

O aprendizado indutivo simbólico é o aprendizado mais naturalmente "entendido" pelo ser humano, dado que a expressão induzida do conceito é, dentre as várias representações de conceitos adotadas por métodos de AM, a que mais se aproxima da linguagem natural. Por essa razão, a próxima seção descreve uma situação trivial de aprendizado, usando um método de AM simbólico.

1.3 Um Exemplo Trivial de Aprendizado Simbólico

O algoritmo ID3 proposto em QUINLAN (1987) foi um dos pioneiros da área de aprendizado simbólico que, a partir de um conjunto de treinamento induz o conceito na forma de uma árvore de decisão.

O exemplo descrito nesta subseção mostra o funcionamento geral do algoritmo ID3 em uma situação de aprendizado em um domínio médico. É uma reescrita do exemplo apresentado em QUINLAN (1987). No exemplo, o atributo classe pode assumir os valores {hipo-primário, hipo-compensado, negativo}, com relação a hipotireoidismo.

Na Tabela 1.1 são mostrados alguns dos exemplos de treinamento; a descrição de cada um deles é feita por seis atributos e uma classe associada. A primeira linha de dado da Tabela 1.1, por exemplo, descreve dados relativos a um paciente do sexo feminino que não sofreu cirurgia de tireóide, cujos valores de dosagem hormonal TSH, FT1 e TT4 tiveram os valores de 5.8, 66.2 e 156.2, respectivamente, não tem tiroxina e que foi diagnosticada, por um médico, como da classe negativa.

Para a descrição dos exemplos, alguns dos atributos utilizados são contínuos, tais como TSH, FTI e TT4 e outros são discretos: sexo, cirurgia_tireoides e tiroxina, que têm como possíveis valores, respectivamente, {fem, mas}, {f,v} e {f,v} (que representam os conjuntos de valores {feminino, masculino}, {falso, verdade}, {falso, verdade}, respectivamente).

Tabela 1.1 Conjunto de dados sobre pacientes que é usado como conjunto de treinamento

Sexo	Cirurgia_tireoides	TSH	FTI	TT4	Tiroxina	Classe
fem	f	5.8	66.2	156.2	f	negativo
fem	f	7.3	70.2	152.9	v	negativo
mas	v	8.8	60.2	148.4	f	hipo-primário
fem	f	6.9	65.4	132.7	f	hipo-compensado
mas	f	5.4	65.2	150.9	f	negativo
...	...	...	...	...	...	...

O ID3 generaliza os exemplos de treinamento da Tabela 1.1 como a árvore de decisão mostrada na Figura 1.2, onde cada nó da árvore corresponde a um teste de atributo e as folhas, às classes.

```
TSH < 6.05: negativo
TSH > 6.05:
        FTI < 64.5: hipo-primário
        FTI > 64.5:
                tiroxina = v: negativo
                tiroxina = f:
                        cirurgia_tireoides = v: negativo
                        cirurgia_tireoides = f:
                                TT4 < 150.5: hipo-compensado
                                TT4 > 150.5: negativo
```

Figura 1.2 Árvore de classificação de hipotireoidismo induzida usando dados da Tabela 1.1

A Figura 1.3 é uma representação pictórica da descrição da árvore induzida pela ID3 mostrada na Figura 1.2.

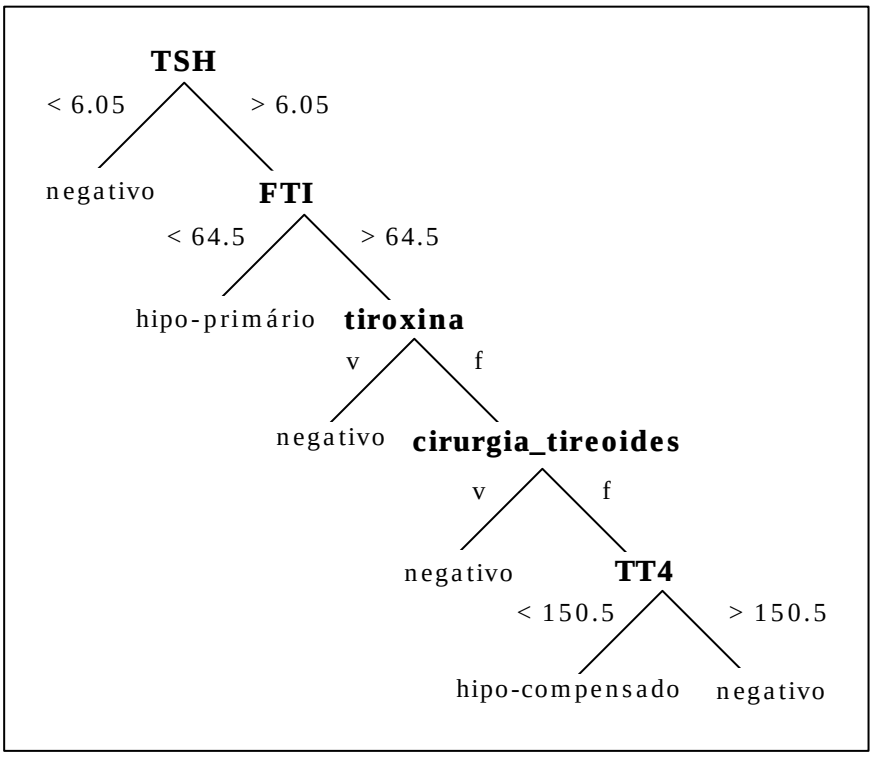

Figura 1.3 Estrutura de classificação pictoricamente representada como árvore

Uma reescrita da árvore de decisão da Figura 1.2 como regras de decisão está mostrada na Figura 1.4.

1. **Se** (TSH < 6.05)
   então negativo
2. **Se** (TSH > 6.05) **e** (FTI < 64.5)
   então hipo-primário
3. **Se** (TSH > 6.05) **e** (FTI > 64.5) **e** (tiroxina = verdadeiro)
   então negativo
   .
   .
   .
6. **Se** (TSH > 6.05) **e** (FTI > 64.5) **e** (tiroxina = falso) **e** (cirurgia_tireoides = falso) **e** (TT4 > 150.5)
   então negativo.

Figura 1.4 Regras de decisão para classificação de hipotireoidismo

Uma vez induzida a expressão do conceito (no caso, a árvore de decisão da Figura 1.2), o procedimento padrão adotado na área de AM é avaliar o conceito em um conjunto de teste, de maneira a validar, empiricamente, a expressão induzida.

Se o conceito induzido tiver um bom desempenho na fase de testes, ele é candidato a ser usado para a categorização de novas instâncias, isto é, instâncias que têm classes desconhecidas. Considere então as duas instâncias descritas na Tabela 1.2, cujas classes são desconhecidas.

Tabela 1.2 Exemplos para fase de classificação

Sexo	Cirurgia_tireoides	TSH	FTI	TT4	Tiroxina	Classe
mas	f	6.1	64.9	133.1	f	?
mas	v	6.5	67.3	132.2	v	?

Usando a expressão do conceito induzida pelo ID3, isto é, a árvore de decisão da Figura 1.2, as duas instâncias têm suas respectivas classes determinadas pela árvore de decisão, como hipo-compensado e negativo, respectivamente.

É interessante mencionar que durante a construção da árvore, o atributo sexo foi descartado, mesmo fazendo parte da descrição dos exemplos de treinamento. Isto se deve ao fato de tal atributo não contribuir para a determinação da classe de uma instância, neste domínio.

1.4 Objetivos do Trabalho

As seções anteriores buscaram contextualizar a área de pesquisa onde a proposta deste trabalho se insere. O trabalho focaliza a investigação de uma colaboração, dada por um AG a algoritmos de aprendizado baseado em instâncias. Os principais objetivos do trabalho são:

- investigar o uso de AGs na busca de vetores de pesos que produzam um melhoramento de desempenho de métodos de aprendizado baseado em instâncias, particularmente do *k*-vizinho mais próximo com distância ponderada;
- implementar um sistema de aprendizado cooperativo genético - baseado em instâncias.

1.5 Organização da Dissertação

Esta dissertação está dividida em cinco capítulos. Neste capítulo foram feitas considerações sobre a área de AM caracterizando, especialmente, o Aprendizado Indutivo, de maneira a contextualizar a área de pesquisa em questão. Neste mesmo capítulo são estabelecidos os objetivos do trabalho e descrita sua organização.

No segundo capítulo é descrito um levantamento e estudo sobre a família IBL - suas principais características e principais algoritmos. Antes de focalizar a família IBL, entretanto, o trabalho apresenta e discute em detalhes o algoritmo NN - *Nearest Neighbor* (COVER & HART (1967)) dado que é o primeiro algoritmo representativo do modelo baseado em instâncias. São discutidas também algumas variações do NN, tais como o *k*-NN e o *k*-NN com distância ponderada, dado que o *k*-NN é um dos algoritmos abordados no trabalho.

O terceiro capítulo apresenta os Algoritmos Genéticos como uma técnica de aprendizado e de otimização. Discute possíveis representações dos dados, principais operadores genéticos e parâmetros relevantes ao problema tratado nesta pesquisa.

O quarto capítulo apresenta, em detalhes, a proposta da elaboração genética – baseada em instâncias investigada e desenvolvida neste trabalho.

Descreve a arquitetura do sistema desenvolvido HIGEBI (Híbrido Genético – Baseado em Instâncias) que implementa a colaboração genética no aprendizado baseado em instâncias. Apresenta e discute os resultados obtidos pelo sistema em alguns domínios de conhecimento e apresenta algumas propostas de análises para pesquisas futuras.

A apresentação das referências bibliográficas utilizadas na pesquisa durante o desenvolvimento deste trabalho finaliza a dissertação.

2 capítulo

Aprendizado Baseado em Instâncias – Caracterização e Principais Algoritmos

2.1 Introdução

Como comentado no capítulo anterior, a vasta maioria dos métodos de AM induz, de alguma forma, a(s) expressão(ões) que representa(m) o conceito sendo aprendido a partir de um conjunto de instâncias do conceito, conhecido como conjunto de treinamento. Este é o caso de métodos baseados em árvores de decisão ou regras, cujo processo de aprendizado que implementam consiste basicamente da construção de uma árvore de decisão ou, então, de um conjunto de regras que representa o conceito aprendido. No caso de redes neurais, a representação do conceito não é feita simbolicamente por meio de uma árvore ou então, regras. O conceito aprendido a partir do conjunto de treinamento é representado pela própria rede, por meio de sua topologia, pesos das conexões, além de outras informações.

Em contraste a algoritmos de aprendizado que generalizam o conjunto de treinamento em uma árvore, um conjunto de regras ou então uma rede, existem os chamados algoritmos de aprendizado baseado em instâncias, conhecidos como algoritmos IBL (*Instance Based Learning*) (Aha *et al* (1991)), introduzidos no capítulo anterior. Tais algoritmos não realizam qualquer generalização; eles simplesmente armazenam todas as instâncias de treinamento – a expressão do conceito é, pois, o próprio conjunto de treinamento.

Como comentado em MITCHELL (1997), “métodos baseados em instâncias são algumas vezes referenciados como métodos preguiçosos *(‘lazy’)* de aprendizado, uma vez que adiam o processamento até o ponto em que uma determinada instância deve ser classificada. A vantagem chave dessa espécie de aprendizado adiado, ou preguiçoso é que, ao invés de estimar o conceito alvo para o espaço todo de instâncias, esses métodos podem estimá-lo localmente e diferentemente para cada nova instância a ser classificada”.

Dessa forma, quando da classificação de uma nova instância, os algoritmos baseados em instâncias buscam determinar qual dentre as instâncias armazenadas que representam o conceito, é a mais “similar” à nova instância a classificar. Uma vez que para tais algoritmos similaridade é abordada como proximidade, a classe da instância mais “próxima” é atribuída à nova instância.

“Entre algumas das dificuldades de sistemas baseados em instâncias estão o aumento do tempo de classificação à medida que muitos exemplos vão sendo adicionados à memória e a possibilidade de exceder a capacidade de memória devido ao grande número de exemplos” (SANTOS & NICOLETTI (1997)). Deve ser lembrado que praticamente todo o processo acontece em tempo de classificação de uma nova instância e não quando os exemplos de treinamento são disponibilizados ao sistema, uma vez que a fase de treinamento consiste apenas de armazenamento.

Considerando que os algoritmos da família IBL assumem que instâncias similares têm características similares, uma heurística local pode ser utilizada para classificação de novas instâncias utilizando a técnica do "vizinho mais próximo" – NN. Dessa forma, a abordagem do aprendizado baseado em instâncias pode ser considerada como uma extensão do algoritmo NN.

O algoritmo NN proposto em COVER & HART (1967) apresenta algumas limitações que os algoritmos da família IBL procuram contornar. Segue uma lista das principais limitações de classificadores baseados no NN, citadas em BREIMAN (1984):

- são classificadores computacionalmente caros, uma vez que armazenam todas as instâncias de treinamento;
- são intolerantes a atributos com ruídos e a atributos irrelevantes;
- são sensíveis à escolha da função de similaridade;

- não trabalham naturalmente com atributos que tenham valores nominais (ou simbólicos) ou então com atributos que, por alguma razão, não têm valores associados;
- fornecem pouca informação útil com relação à estrutura dos dados.

2.2 A Família NN

O classificador NN é a base de muitos algoritmos de aprendizado *"lazy"* e é o algoritmo que fundamenta a família IBL. Ele é muito simples e permite a classificação de novas instâncias com alto grau de precisão. Segundo WETTSCHERECK (1997), geralmente os algoritmos que se fundamentam no NN têm três características principais de funcionamento:

- adiar: eles armazenam todas as instâncias de treinamento e adiam a classificação de uma nova instância para o momento em que tal classificação é solicitada;
- responder: a interrogação sobre a classe de uma nova instância é respondida combinando as instâncias de treinamento, tipicamente usando uma abordagem local de aprendizado, onde: (1) as instâncias são definidas como pontos em um espaço, (2) a função de similaridade é definida sobre todos os pares dessas instâncias, e (3) uma função de predição define uma resposta que é uma função monotônica da similaridade;
- descartar: depois da classificação de uma nova instância, qualquer resultado intermediário é descartado.

Como comentado em WETTSCHERECK (1997), "essa definição é intencionalmente vaga; ela não define como as instâncias de treinamento são armazenadas ou representadas nem como serão combinadas para a determinação da classe de uma nova instância. O classificador *k*-NN é puramente um algoritmo de aprendizado adiado (ou *'pure lazy'*)".

2.2.1 Aprendizado via NN

Considere um espaço bidimensional (definido pelos atributos a_1 e a_2) onde a Classe 1 está representada por 10 instâncias de treinamento, a Classe 2 por 12 instâncias e a Classe 3 por 9 instâncias. Suponha, então, que se deseja determinar a classe de uma nova instância, identificada como x. A Figura 2.1 ilustra essa situação.

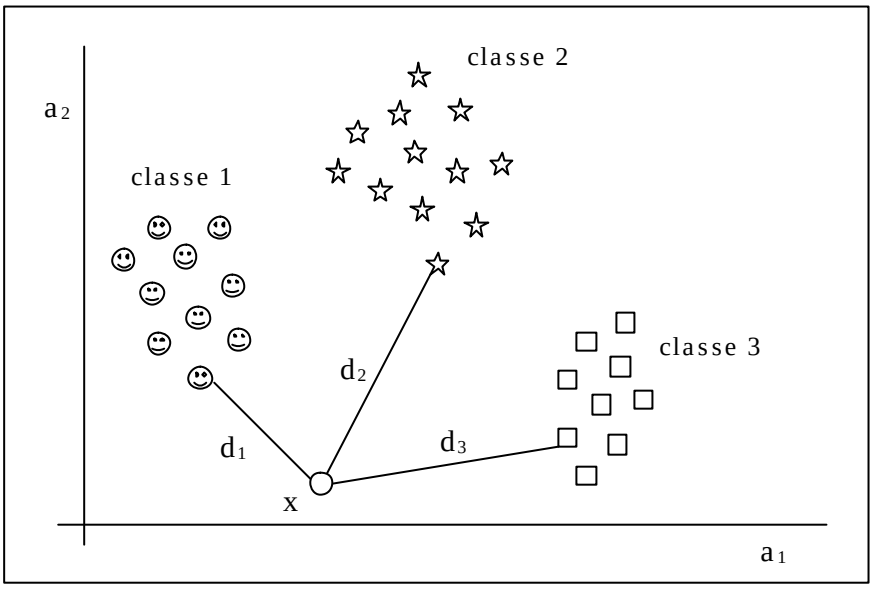

Figura 2.1 Processo de classificação realizado pelo algoritmo NN onde: x: instância a ser classificada, d_1: menor distância à classe 1, d_2: menor distância a classe 2 e d_3: menor distância à classe 3. Os conceitos são representados por ☺, ☆ e □.

O vizinho mais próximo da instância x pode ser calculado usando a distância Euclidiana padrão. Para o exemplo da Figura 2.1, o NN vai classificar a instância x como pertencente à classe 1, uma vez que, dentre as instâncias armazenadas, a mais próxima de x pertence à classe 1. A definição formal do algoritmo NN, como descrita em GATES (1972), é mostrada na Figura 2.2.

Sejam:

- Espaço n-dimensional de atributos
- M classes, numeradas de 1, 2, ..., M
- p instâncias de treinamento, cada uma expressa como um par (x_i, θ_i) para $1 \leq i \leq p$, onde:

a) x_i é uma instância de treinamento, expressa por um vetor de valores (associados aos respectivos atributos)

$$x_i = (x_{i_1}, x_{i_2}, ..., x_{i_n})$$

b) $\theta_i \in \{1, 2, ..., M\}$ e denota a classe correta da instância x_i.

Seja $T_{NN} = \{(x_1, \theta_1), (x_2, \theta_2), ..., (x_p, \theta_p)\}$ o conjunto de treinamento do NN.
Dada uma instância desconhecida x, a regra de decisão decide que x está na classe θ_j se:

$$d(x, x_j) \leq d(x, x_i) \text{ para } 1 \leq i \leq p$$

onde d é alguma métrica n-dimensional de distância.

Figura 2.2 Descrição formal do NN

O NN descrito na Figura 2.2 é conhecido como 1-NN, pois para a determinação da classe da nova instância, utiliza apenas "um vizinho mais próximo", isto é, calcula a distância da nova instância a cada um dos exemplos que estão armazenados.

2.2.2 Aprendizado via *k*-NN

A partir do 1-NN, uma variante conhecida como *k*-NN ou *k* vizinho mais próximo (onde $k > 1$), identifica as *k* instâncias mais próximas $\{i_1, i_2, ..., i_k\}$ de x e decide pela escolha da classe que comparece com maior freqüência no conjunto $\{i_1, i_2, ..., i_k\}$.

O *k*-NN é simples, rápido e bastante eficiente; entretanto a qualidade do *k*-NN depende de quais instâncias são identificadas como mais próximas; o que, por sua vez, depende da métrica de distância adotada.

Em MITCHELL (1997), o aprendizado *k*-NN é abordado como um aprendizado de uma aproximação de uma função $f: R^n \rightarrow V$, cujo domínio é o espaço n-dimensional de valores reais e cujo contra-domínio é um conjunto discreto de valores, $V = \{v1, v_2, ..., v_S\}$.

A Figura 2.3 descreve o pseudocódigo do algoritmo *k*-NN para uma representação discreta de f.

Treinamento:

Para cada exemplo de treinamento (x,f(x)), inclua o exemplo na lista exemplos_treinamento.

Classificação

Seja x_q uma instância a ser classificada,

- Sejam $x_1,\ldots, x_k$ as *k* instâncias da lista exemplos_treinamento mais próximas de x_q.

- Retorne

$$\hat{f}(x_q) \leftarrow \operatorname{argmax}_{v \in V} \sum_{i=1}^{k} d(v, f(x_i))$$

onde $\delta(a,b)=1$ se a=b, caso contrário, $\delta(a,b)=0$

Figura 2.3 Pseudocódigo do *k* vizinho mais próximo para a representação de funções com valores discretos f: $R^n \rightarrow V$

Como discutido em MITCHELL (1997), o algoritmo descrito na Figura 2.3 pode ser facilmente adaptado para a aproximação de funções com valores contínuos. Para fazer isso, o algoritmo deve calcular o valor médio dos valores associados aos k exemplos de treinamento mais próximos, ao invés de calcular o seu valor mais comum. Mais precisamente, para aproximar uma função contínua f: $R^n \rightarrow R$, a linha final do algoritmo descrito na Figura 2.3 deve ser substituída por:

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

A formalização e análise do *k*-NN apresentadas a seguir estão baseadas na proposta descrita em WETTSCHERECK (1997).

Seja **F** o conjunto de atributos que descrevem as instâncias de treinamento e considere que as classes associadas às instâncias estão agrupadas no conjunto **J**. Assim sendo, cada instância x que faz parte da descrição do conceito é representada como $\mathbf{x} = \{x_1, x_2, \ldots, x_{|F|}\}$, onde cada x_i, $1 \leq i \leq |F|$ é um valor do conjunto de valores associado ao atributo i. Além disso, a classe associada a x é notada por $\mathbf{x_c} \in \mathbf{J}$.

Cada atributo é contínuo ou então assume como valores os elementos de um conjunto discreto.

A entrada para o classificador *k*-NN é uma nova instância **q**. A saída é uma atribuição de classe à **q**.

O objetivo do classificador é minimizar a "perda esperada" ou o "risco de classificação incorreta" para cada classe $c_j \in \mathbf{J}$, de acordo com a fórmula:

$$R_j = \sum_{c_{j'} \in \mathbf{J}} L_{c_j c_{j'}} \; p(c_{j'} \mid \mathbf{q})$$

onde $L_{c_j c_{j'}}$ é o custo associado com a classificação incorreta de uma instância da classe c_j como uma instância da classe $c_{j'}$ ($j \neq j'$) e $p(c_{j'} \mid \mathbf{q})$ é a probabilidade de classificar **q** na classe $c_{j'}$. O *k*-NN geralmente assume que todas as classificações incorretas têm custo igual, de forma que:

$$L_{c_j c_{j'}} = \begin{cases} 0 & \text{se} \quad j = j' \\ 1 & \text{se} \quad j \neq j' \end{cases}$$

Uma vez que a classe exata de **q** é desconhecida, o algoritmo fornece como saída a classe mais provável de **q**, dada por:

$$k\text{-NN}(\mathbf{q}) = \max_{c_j \in \mathbf{J}} \; p(c_j \mid \mathbf{q})$$

O classificador *k*-NN difere de outros classificadores na forma como define a probabilidade posterior da classe:

$$p(c_j \mid \mathbf{q}) = \frac{\sum_{\mathbf{x} \in \mathbf{K_q}} 1(x_c = c_j) * K(d(\mathbf{x}, \mathbf{q}))}{\sum_{\mathbf{x} \in \mathbf{K_q}} K(d(\mathbf{x}, \mathbf{q}))}$$

onde:

- a função 1( ) tem valor 1, se e somente se, seu argumento for verdadeiro. O resultado desta função só será 1 se a classe da instância $\mathbf{x}$ coincide com c_j.
- K( ) é a função *kernel* definida como:

$$K(d(\mathbf{x}, \mathbf{q})) = \frac{1}{d(\mathbf{x}, \mathbf{q})}$$

- $\mathbf{K_q}$ é o conjunto dos k vizinhos mais próximos de $\mathbf{q}$, extraídos de um conjunto $\mathbf{X}$ de instâncias de treinamento e determinados pela função de distância $d( )$.

Para a definição do $\mathbf{K_q}$, o k-NN calcula a distância $d(\mathbf{x},\mathbf{q})$ de $\mathbf{q}$ a cada $\mathbf{x} \in \mathbf{X}$ usando:

$$d(\mathbf{x},\mathbf{q}) = \left(\sum_{f \in \mathbf{F}} w(f) . d(x_f, q_f)^r \right)^{1/r}$$

onde r=2 (distância Euclidiana) e a função $\delta( )$ define como os valores de uma dada característica diferem, sendo:

$$d(x_f, q_f) = \begin{cases} |x_f - q_f| & \text{se f é contínuo} \\ 0 & \text{se f é discreto e } x_f = q_f \\ 1 & \text{se f é discreto e } x_f \neq q_f \end{cases}$$

e $\omega(f)$ define a função de peso das características. O k-NN define esta função como uma função constante:

$$\omega(f) = \omega_f$$

Finalmente, k-NN define:

$$\forall_{f \in \mathbf{F}} \quad \omega_f = s$$

para algum valor escalar constante s.

2.2.3 Aprendizado via CNN

O NN tem uma outra variante conhecida como CNN (*Condensed Nearest Neighbor*). Ambas diferem quando do aprendizado do conceito, representado pelo conjunto de treinamento. Enquanto a expressão do conceito no NN (referenciada nesta seção como T_{NN}) consiste de todo o conjunto de treinamento, no CNN a expressão do conceito (referenciada como T_{CNN}) consiste de um subconjunto de T_{NN}.

No caso específico do CNN, "a eventual queda no desempenho, devido a escolha de um conjunto bem menor de pontos, é compensada pela redução de memória necessária para o armazenamento do conjunto de treinamento e pelo menor tempo gasto na classificação de uma nova instância" (GATES (1972)).

Para a construção do T_{CNN} é empregado o conceito de subconjunto consistente, que é definido como um subconjunto de T_{NN}, mas que classifica todas as instâncias de T_{NN}. O menor dos subconjuntos consistentes é denominado subconjunto minimal. Este conjunto é o que irá classificar mais eficiente e apropriadamente toda instância que esteja presente em T_{NN}. O algoritmo CNN é consistente, mas não necessariamente minimal.

Em GATES (1972) encontramos a descrição em pseudocódigo do CNN (Figura 2.4) e em SANTOS & NICOLETTI (1997) um exemplo de aplicação.

```
T_CNN ← primeira instância de T_NN
INSTANCIA ← primeira instância de T_NN

while todas as instâncias de T_NN não forem corretamente classificadas
por CNN do
        if CNN classifica INSTANCIA
                then INSTANCIA ← próxima instância de T_NN
                else
                        T_CNN ← T_CNN ∪ {INSTANCIA}
                        INSTANCIA ← primeira instância de T_NN
                end
end-while.
```

Figura 2.4 Algoritmo CNN

No algoritmo da Figura 2.4, o CNN classifica uma INSTANCIA se a classe da instância é a mesma da classe da instância que lhe é mais próxima e que já faz parte da descrição do conceito.

Considere uma situação em que a expressão do conceito aprendido pelo NN é o conjunto:

T_{NN} = { (-13,1), (-16,1), (-19,1), (-6,2), (-4,2), (-2,2), (0,2), (2,2), (4,2), (6,2), (13,1), (16,1), (19,1) }

No conjunto acima, cada instância é descrita por um atributo e uma classe associada. O primeiro elemento de T_{NN} é lido como: a instância cujo valor de abscissa é –13 pertence à classe 1. A representação gráfica do T_{NN} está mostrada na parte superior da Figura 2.6 e o conjunto que representa a expressão do conceito está mostrada da parte inferior da Figura 2.6.

Considere agora a "execução" do algoritmo CNN da Figura 2.4, mostrada na Figura 2.5.

$T_{CNN} \leftarrow$ (-13,1)	
INSTANCIA ← (-13,1)	
	$T_{CNN} \leftarrow$ {(-13,1)}
INSTANCIA ← (-16,1)	
INSTANCIA ← (-19,1)	
INSTANCIA ← (-6,2)	
	$T_{CNN} \leftarrow$ {(-13,1), (-6,2)}
INSTANCIA ← (-4,2)	
INSTANCIA ← (-2,2)	
INSTANCIA ← (0,2)	
INSTANCIA ← (2,2)	
INSTANCIA ← (4,2)	
	$T_{CNN} \leftarrow$ {(-13,1), (-6,2), (4,2)}
INSTANCIA ← (6,2)	
INSTANCIA ← (13,1)	
	$T_{CNN} \leftarrow$ {(-13,1), (-6,2), (4,2), (13,1)}
INSTANCIA ← (16,1)	
INSTANCIA ← (19,1)	

Figura 2.5 Execução do algoritmo CNN

A parte inferior da Figura 2.6 exibe a representação gráfica do conceito aprendido pelo CNN.

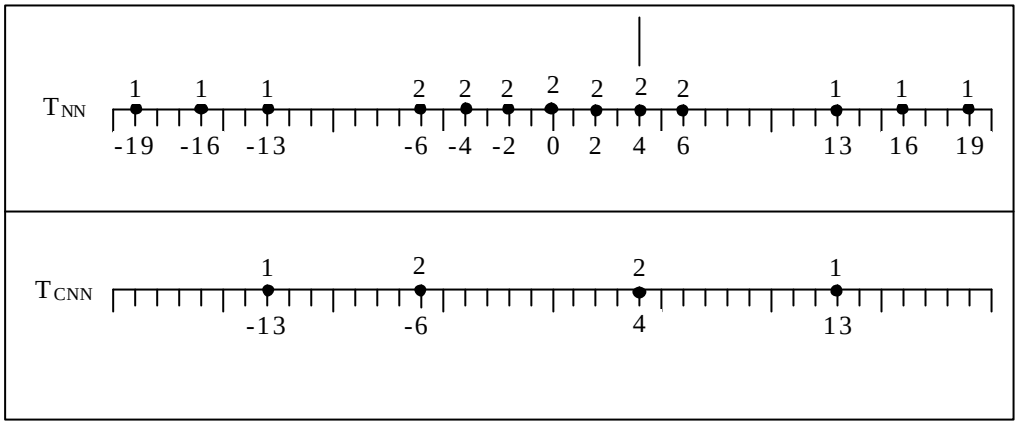

Figura 2.6 Conjunto obtido pela aplicação do CNN

2.2.4 Aprendizado via *Wk*-NN

Um refinamento do algoritmo *k* vizinho mais próximo pode ser feito por meio de um esquema de atribuição de pesos. Essa atribuição pode acontecer de duas formas: pesos associados a instâncias que representam a relevância da instância na classificação de outras ou, então, pesos associados a atributos, para expressar a contribuição do respectivo atributo na definição da classe da instância.

Na descrição da Figura 2.3, o peso de cada vizinho pode ser estabelecido como o inverso do quadrado de sua distância à instância x_q. Isso se traduz na substituição da linha final do algoritmo pela Equação (2.1):

$$\hat{f}(x_q) \leftarrow \operatorname{argmax}_{v \in V} \sum_{i=1}^{k} ?_i \, d(v, f(x_i)) \text{ onde } \omega_i \equiv \frac{1}{d(x_q, x_i)^2} \qquad (2.1)$$

Segundo MITCHELL (1997), para tratar o caso onde a instância a ser classificada x_q coincide exatamente com uma das instâncias de treinamento x_i, tornando o denominador $d(x_q,x_i)^2$ zero, assume-se que $\hat{f}(x_q)$ é $f(x_i)$. Para o aprendizado de funções com valores reais, o recurso de pesos associados às

distâncias também pode ser adotado. Isso implica a substituição da linha final da Figura 2.3 pela Equação (2.2):

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} ?_i \, f(x_i)}{\sum_{i=1}^{k} ?_i} \qquad (2.2)$$

onde ω_i é definido na equação 2.1. Note que o denominador da equação 2.2 é uma constante que normaliza as contribuições dos vários pesos (isto é, garante que se $f(x_i) = c$ para todos os exemplos de treinamento, então c será também atribuído à $\hat{f}(x_q)$). Uma vez que pesos tenham sido adicionados às distâncias, não existe problema em permitir que todos os exemplos de treinamento tenham influência na classificação de x_q, uma vez que exemplos de treinamento bem distantes pouco influenciarão $\hat{f}(x_q)$. A única desvantagem em considerar todos os exemplos é que o classificador levará mais tempo na classificação de uma nova instância.

Um aspecto prático na aplicação do *k* vizinho mais próximo é que a distância entre instâncias é calculada usando todos os atributos que descrevem a instância, isto é, todos os eixos do espaço Euclidiano nos quais as instâncias estão contidas. Isto contrasta com outros métodos de aprendizado, tais como o aprendizado de regras de decisão (ou árvores de decisão), que selecionam apenas um subconjunto dos atributos que descrevem as instâncias, quando construindo a expressão da hipótese que as representam.

O efeito disso pode ser evidenciado, por exemplo, numa situação em que cada instância é descrita por 20 atributos, onde apenas dois deles são relevantes na determinação da classificação de uma determinada função. Neste caso, instâncias que têm valores idênticos para esses dois atributos relevantes, podem estar bem distantes uma da outra, no espaço de instâncias 20-dimensional. Conseqüentemente, a métrica de similaridade usada pelo *k* vizinho mais próximo - que é dependente dos 20 atributos - vai direcionar o aprendizado de maneira errada - a distância entre vizinhos será dominada pelos atributos irrelevantes, que estão presentes em maior número. Esta dificuldade que surge quando muitos atributos irrelevantes estão presentes é conhecida como problema da dimensionalidade.

2.3 A Família IBL

A família IBL é composta por cinco algoritmos. A diversidade entre estes algoritmos se deve principalmente a diferentes estratégias adotadas para:

- armazenamento das instâncias de treinamento (quais instâncias armazenar?);
- avaliação da similaridade entre as instâncias;
- número de instâncias usadas, quando da classificação de uma nova (quantas instâncias considerar?).

O conjunto de treinamento usado por algoritmos IBL é uma seqüência de instâncias onde cada instância é representada por um vetor de pares do tipo atributo-valor e uma classe associada.

Uma classe é definida pelo conjunto de todas as instâncias do espaço de instâncias que têm o mesmo valor para o atributo classe. Além disso, assume-se que cada instância pertence a exatamente uma classe e que classes são disjuntas.

Uma descrição de conceito baseada em instâncias inclui um conjunto de instâncias armazenadas e possivelmente, informação a respeito do desempenho ocorrido durante classificações anteriores, ou seja, o número de classificações corretas e incorretas. A descrição do conceito pode também incorporar funções de similaridade e de classificação (além do conjunto de instâncias armazenado). O objetivo destas funções é determinar como o conjunto de instâncias (que faz parte da descrição de conceito) será utilizado para predizer a classe de novas instâncias.

Segundo AHA (1991), as funções de similaridade e classificação e o conjunto de instâncias são os três componentes que caracterizam todos os algoritmos desta família, onde:

- função de similaridade: é responsável por calcular a similaridade (valores numéricos) entre uma instância de treinamento i e as instâncias que participam da descrição do conceito;
- função de classificação: é responsável por classificar a instância i. Para isto, utiliza os registros de desempenho de classificações anteriores das

instâncias que descrevem o conceito e o resultado da função de similaridade;

- atualizador da descrição do conceito: é responsável por manter registros de desempenho de classificação e decidir qual instância incluir na descrição do conceito. Como entrada recebe a instância i, os resultados da função de classificação e a descrição atual do conceito. É o responsável pela modificação da descrição do conceito.

Nas subseções seguintes são descritos os diversos algoritmos da família IBL.

2.3.1 Algoritmo IB1

O algoritmo IB1 é o algoritmo mais simples da família IBL. Tem como principais características:

- processamento das instâncias incrementalmente;
- utilização de uma política de tolerância à ausência de valores de atributos.

O IB1 armazena todas as instâncias de treinamento, que são processadas incrementalmente. O conceito aprendido pelo IB1 é representado, pois, pelo próprio conjunto de treinamento. A descrição em pseudocódigo do algoritmo IB1 extraído de AHA (1991) é apresentada no Algoritmo 2.1.

```
DC ← ∅
for_each x ∈ Conjunto de Treinamento do
        begin
            1.  for_each y ∈ DC do
                sim[y] ← similaridade(x,y)
            2.  y_max ← y ∈ DC com a maior sim[y]
            3.  if classe(x) = classe(y_max)
                then classificação ← correta
                else classificação ← incorreta
            4.  DC ← DC ∪ {x}
```

Algoritmo 2.1 Algoritmo IB1

Nesta descrição do IB1:

- conjunto de Treinamento é o conjunto de instâncias de treinamento, onde cada uma delas é representada por um vetor de pares: atributo/valor_do_atributo e uma classe associada;
- DC é um conjunto que representa a descrição do conceito criada até então. É inicializado vazio e, ao final do algoritmo é o próprio conjunto de treinamento;
- *similaridade*(x,y) é uma função que "mede" o grau de similaridade entre duas instâncias. No contexto do IB1, o conceito de "similaridade" é implementado por meio de uma métrica de "proximidade".

A suposição básica que o IB1 faz é a de que a similaridade entre duas instâncias é inversamente proporcional à distância euclidiana entre elas, como pode ser verificado na Figura 2.7.

$$similaridade(x,y) = -\sqrt{\sum_{i=1}^{n} f(x_i, y_i)}$$

onde

- n: número de atributos usados para descrever as instâncias
- $f(x_i, y_i) = (x_i - y_i)^2$, para atributos numéricos
- $f(x_i, y_i) = (x_i \neq y_i)$, para atributos com valores simbólicos e/ou *booleanos*. Retornando um valor 0 se os dois valores simbólicos e/ou *booleanos* são iguais, ou retornando 1 caso contrário (TING (1997)).
- $f(x_i, y_i) = 1$, para atributos que tenham valores ausentes tanto na representação de x quanto na de y. Nos casos em que apenas um determinado valor estiver ausente, o IB1 assume como este valor o mais diferente possível do valor corrente.

Figura 2.7 Função de similaridade adotada pelo algoritmo IB1

Considerando novamente a descrição do Algoritmo 2.1, no passo:

1. é calculada a similaridade entre a instância x, do conjunto de treinamento e cada uma das instâncias y que já faz parte da expressão do conceito ($y \in DC$);
2. é eleita a instância $y_{max} \in DC$ que tem maior similaridade com x;
3. é registrado o desempenho da instância $y_{max} \in DC$, conforme sua classe seja ou não a mesma da instância x;
4. a instância de treinamento x é incorporada à expressão do conceito.

Devido ao pouco nível de detalhe com que o pseudocódigo foi descrito em AHA (1991), é possível observar que no passo 3 o registro de desempenho da instância $y_{max} \in$ DC não é armazenado, quando deveria ser, para tornar o algoritmo mais claro. Uma possível reescrita do passo 3, poderia ser:

if classe (x) = classe (y_{max})
then classificação_correta(y_{max}) ← classificação_correta (y_{max}) + 1
else classificação_incorreta (y_{max}) ← classificação_incorreta (y_{max}) + 1

onde os vetores classificação_correta e classificação_incorreta teriam um tamanho máximo igual ao número de elementos do conjunto de treinamento.

2.3.2 Algoritmo IB2

O algoritmo IB2 difere fundamentalmente do IB1; durante o aprendizado seu atualizador de descrição de conceito armazena apenas as instâncias classificadas incorretamente. O IB2, entretanto, apresenta as mesmas funções de similaridade e de classificação do IB1.

Segundo SANTOS & NICOLETTI (1997), "a incorporação dessa estratégia de armazenamento foi subsidiada pela intuição de que a maioria das instâncias classificadas incorretamente encontra-se na fronteira do conceito e, de certa forma, o delimitam". O Algoritmo 2.2 descreve o pseudocódigo do IB2, extraído de AHA (1991).

```
DC ← ∅
for_each x ∈ Conjunto de Treinamento do
        begin
            1.  for_each y ∈ DC do
                        sim[y] ← similaridade(x,y)
            2.  ymax ← y ∈ DC com a maior sim[y]
            3.  if classe(x) = classe(ymax)
                then classificação ← correta
                else
                  begin
                        3.1 classificação ← incorreta
                        3.2 DC ← DC ∪ {x}
                  end
        end.
```

Algoritmo 2.2 Algoritmo IB2

Note que para a descrição do IB2 valem as mesmas observações feitas quando da descrição do IB1, relativas a Conjunto de Treinamento, DC e função *similaridade*(x,y). A diferença básica entre os dois algoritmos, entretanto, se deve às instâncias que são armazenadas e que, ao final, vão representar a expressão do conceito. Como visto, o IB1 incorpora à expressão do conceito toda instância de treinamento. Já o IB2 apenas vai incorporar cada uma das instâncias de treinamento que difere, com relação à classe, de sua instância mais próxima, já armazenada.

É óbvio que com a estratégia de apenas armazenar determinadas instâncias, a necessidade de armazenamento do IB2 se torna bem menor do que a do IB1. Entretanto, essa estratégia tem o efeito colateral de ser bastante sensível a ruídos nas instâncias de treinamento. Considerando que instâncias com ruídos geralmente são classificadas incorretamente, via de regra, tais instâncias, via IB2, passarão a fazer parte da descrição do conceito.

Em AHA (1992) alguns experimentos mostram que em domínios com alta incidência de ruídos, tais instâncias têm grande participação na expressão final do conceito, afetando de forma negativa o desempenho do sistema durante a fase de classificação. Tais experimentos buscaram evidenciar o quanto o IB2 sacrificaria a precisão de classificação para obter um baixo nível de armazenamento. Foi possível concluir que embora o IB2 reduza, às vezes significativamente, o número de instâncias armazenadas, ele realmente sacrifica a precisão de classificação, principalmente quando os domínios têm ruídos ou instâncias “excepcionais” (isto é, instâncias caracterizadas por atributos que não as descrevem apropriadamente).

A característica do IB2 de eleger principalmente as instâncias que estão próximas à fronteira do conceito, como as instâncias que o descrevem, pode também ser observada com relação ao algoritmo CNN (ver Seção 2.2.3).

Vale observar que em domínios sem a presença de ruídos, o IB2 reduz significativamente a necessidade de armazenamento.

2.3.3 Algoritmo IB3

O IB3 foi proposto em AHA (1992) com o intuito de contornar o problema de sensibilidade a ruídos (abordado como valor incorreto de atributo) nos dados. A

idéia que subsidia o IB3 é a de “filtrar” as instâncias armazenadas de maneira a neutralizar a participação, quando da classificação de novas instâncias, daquelas que provavelmente têm ruído.

O IB3 assume que instâncias com ruído terão precisão de classificação pobre devido ao fato de seus vizinhos similares, no espaço de instância, invariavelmente terem outras classificações. Conseqüentemente, o critério que o IB3 usa para a identificação de uma instância com ruído é a sua baixa precisão de classificação. Com o objetivo de colecionar informação para poder evidenciar tais instâncias, o IB3:

- mantém registros de classificação de todas as instâncias armazenadas, isto é, armazena o número de tentativas de classificação corretas e incorretas. O registro de classificação de uma instância exibe o seu desempenho de classificação em instâncias de treinamento subseqüentemente apresentadas;
- utiliza um teste de significância para determinar quais instâncias são boas classificadoras e quais não são (candidatas a serem instâncias com ruídos). As instâncias consideradas “boas” continuam participando da descrição do conceito e são usadas para classificar novas instâncias apresentadas subseqüentemente. Aquelas consideradas “não boas” classificadoras são descartadas da descrição do conceito.

O IB3, durante a fase de treinamento, usa um teste baseado em intervalos de confiança para decidir se “incorpora” ou não uma instância de treinamento à expressão do conceito. A cada instância estão associados dois intervalos de confiança: o intervalo de precisão de classificação da instância (IP) e o intervalo de freqüência observada da classe (IF). Intervalos são construídos em torno da precisão da classificação da instância atual (isto é, sua porcentagem de tentativas de classificação correta) e a freqüência observada de sua classe (isto é, a porcentagem de instâncias de treinamento processados que são elementos dessa classe).

O IB3 aceita uma instância se a sua precisão de classificação for significativamente maior que a sua freqüência observada de classe e remove a instância da descrição se a sua precisão for significativamente menor. Quando os dois intervalos se sobrepõem, a instância permanece na expressão do conceito, mas não é usada em tentativas de classificação (AHA (1997)).

O fato do IB3 categorizar uma instância como aceitável não significa que tal instância continuará aceitável durante o resto do treinamento. O IB3 repetidamente reavalia a aceitabilidade de cada instância, com base em seus registros de classificação, à medida que o aprendizado prossegue.

A Tabela 2.1 mostra as fórmulas, extraídas de HOGG & TANIS (1983), que são usadas para a construção do intervalo de precisão de classificação (IP) de uma instância e para o intervalo de freqüência observada da classe (IF):

Tabela 2.1 Fórmulas para a construção do IP e IF

	Fórmula Utilizada	**Significado das Variáveis**
Limite Inferior (IP)	$\frac{p + z^2/2n - z\sqrt{p(1-p)/n + z^2/4n^2}}{1 + z^2/n}$	Onde: - p: precisão da instância observada - n: número de tentativas de classificação da instância
Limite Superior (IP)	$\frac{p + z^2/2n + z\sqrt{p(1-p)/n + z^2/4n^2}}{1 + z^2/n}$	
Limite Inferior (IF)	$\frac{p + z^2/2n - z\sqrt{p(1-p)/n + z^2/4n^2}}{1 + z^2/n}$	Onde: - p: freqüência da instância observada - n: número de instâncias de treinamento previamente processadas.
Limite Superior (IF)	$\frac{p + z^2/2n + z\sqrt{p(1-p)/n + z^2/4n^2}}{1 + z^2/n}$	

Para cada instância armazenada o IB3 mantém um registro de classificação que indica o número de tentativas de classificação corretas e incorretas que usam a respectiva instância, ou seja, um registro do tipo:

<instância, #tentativas_classificação_correta, #tentativas_classificação_incorreta>

Com as informações desse registro os valores de p e n usados para a construção do intervalo IP da instância podem ser obtidos.

Para a construção do intervalo de freqüência, para cada classe, devem ser contabilizados a freqüência da classe (p) e o número de instâncias processadas previamente (n). O intervalo IF é o mesmo para instâncias da mesma classe. Como comentado em SANTOS & NICOLETTI (1997), "durante o processo de treinamento, existe um intervalo de freqüência para cada uma das classes, isto é, se as instâncias de treinamento estão distribuídas entre três classes, existem apenas três intervalos de freqüência diferentes. Por outro lado, cada instância tem associada a

ela um intervalo de precisão; existem, pois, tantos intervalos de precisão quantas forem as instâncias".

Os testes de freqüência e precisão foram projetados com o intuito de dificultar a aceitação de uma instância. Dessa forma, de acordo com a intenção de aceitar ou rejeitar uma instância é utilizado um valor alto ou baixo para a variável z (utilizada nas fórmulas). Assim, se a intenção for de aceitar apenas as instâncias que sejam efetivamente boas classificadoras, assume-se um valor alto para z (por exemplo, 0.9); entretanto, z pode assumir, por exemplo, 0.75 quando da eliminação de uma instância, se a intenção for descartar inclusive instâncias que apresentem um desempenho moderado.

A justificativa do IB3 para realizar a comparação entre a precisão das instâncias armazenadas e a freqüência relativa de suas classes, é o fato desse algoritmo normalizar a aceitação de uma instância com relação à distribuição da freqüência do conceito, buscando diminuir sua sensibilidade a distribuição tendenciosa. Assim, como comentado em SANTOS & NICOLETTI (1997), “espera-se que instâncias de conceitos com alta freqüência relativa observada tenham também precisão de classificação relativamente alta, uma vez que uma porcentagem relativamente alta de suas tentativas de classificação serão de instâncias de suas classes, ou seja, serão bem sucedidas. De maneira análoga, espera-se que instâncias de conceitos com baixa freqüência relativa observada tenham precisão de classificação relativamente baixa”.

O registro de classificação mantido pelo IB3 é atualizado da seguinte maneira:

- Para cada instância de treinamento t, encontrar seu vizinho aceitável mais próximo (considerando seus intervalos de confiança construídos):

    1. Se existir pelo menos uma instância aceitável, os registros de classificação são atualizados para todas as instâncias armazenadas que estiverem na região delimitada por uma hiperesfera cujo centro é a própria instância t e cujo raio é a distância normalizada entre t e o vizinho aceitável mais próximo de t. A Figura 2.8 mostra um espaço bidimensional composto por sete instâncias rotuladas y_i, i = 1, ..., 7, e mostra um exemplo de quais instâncias terão seus registros de classificação atualizados levando em consideração uma nova instância de treinamento t. A similaridade entre as instâncias já existentes e a nova instância t é mostrada na Tabela 2.2. No

exemplo apresentado y_2, y_4 e y_5 são instâncias aceitáveis. Como para a instância t, deve ser encontrado seu vizinho aceitável mais próximo, dentre as instâncias aceitáveis, a instância mais próxima a t é y_2, uma vez que:

$$(sim(t, y_2) = -1.4) \geq (sim(t, y_4) = -2.6) \geq (sim(t, y_5) = -3.1).$$

Sendo assim, todas as instâncias y_i que tiverem $sim(t, y_i) \geq sim(t, y_2)$, onde i = 1, ..., 7, terão seus registros de classificação atualizados, ou seja, y_1, y_2 , y_3 e y_6.

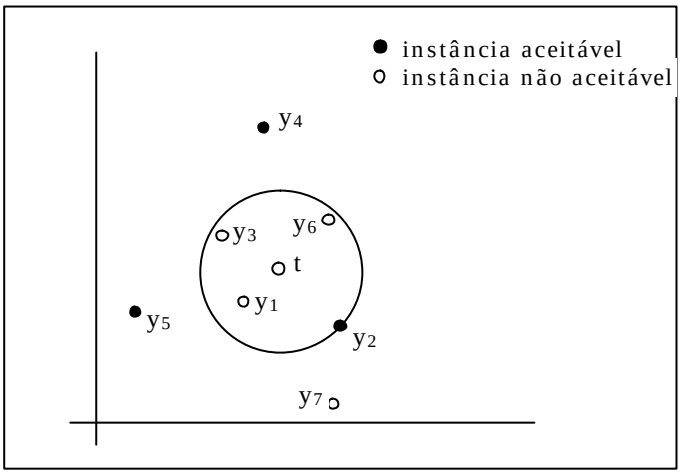

Figura 2.8 Situação 1: instâncias escolhidas pelo IB3 que terão seus registros de classificação atualizados

Tabela 2.2 Tabela de similaridades das instâncias apresentadas na Figura 2.8

	Similaridades para a instância t	**Similaridades em ordem crescente**
$sim(t, y_1)$	-0.6	1ª
$sim(t, y_2)$	-1.4	4ª
$sim(t, y_3)$	-1.2	3ª
$sim(t, y_4)$	-2.6	6ª
$sim(t, y_5)$	-3.1	7ª
$sim(t, y_6)$	-0.8	2ª
$sim(t, y_7)$	-2.3	5ª

2. Se dentre as instâncias armazenadas, nenhuma for aceitável, um número aleatório r ∈ ao intervalo [1, m] (onde m é o número de instâncias armazenadas) é gerado. Dessa forma, serão atualizados os registros de classificação das r instâncias mais similares à nova instância t. A Figura 2.9 ilustra essa situação, onde no espaço bidimensional composto por sete instâncias (y_i, i = 1,..., 7), não existe, nesse momento de aprendizado, nenhuma instância aceitável. Supondo que o número aleatório r ∈ [1, m] (m =

7) gerado seja 5, ou seja, a 5ª instância com a maior similaridade será tratada como instância aceitável mais similar a nova instância t. Pela Tabela 2.3 nota-se que a 5ª instância mais similar a t seria y_1. Dessa forma, pela Figura 2.10 é possível perceber que todas as instâncias y_i, i = 1,..., 7, que satisfaçam a condição sim(t, y_i) ≥ sim(t, y_1) (ou seja, y_3, y_7, y_4, y_2 e y_1) terão seus registros de classificação atualizados.

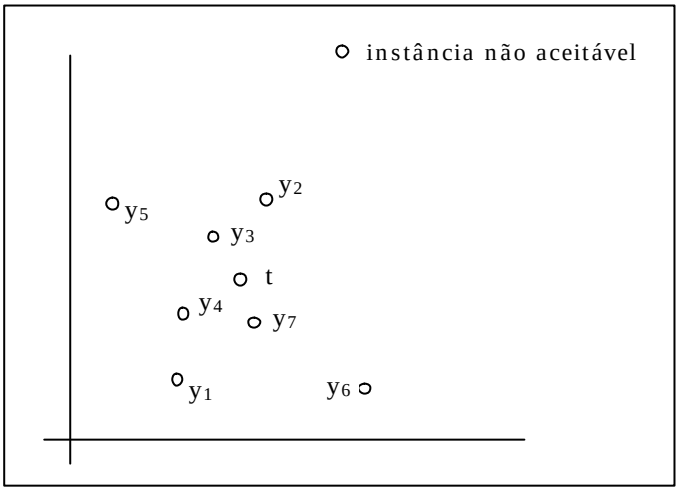

Figura 2.9 Situação 2: Nenhuma instância aceitável no espaço bidimensional

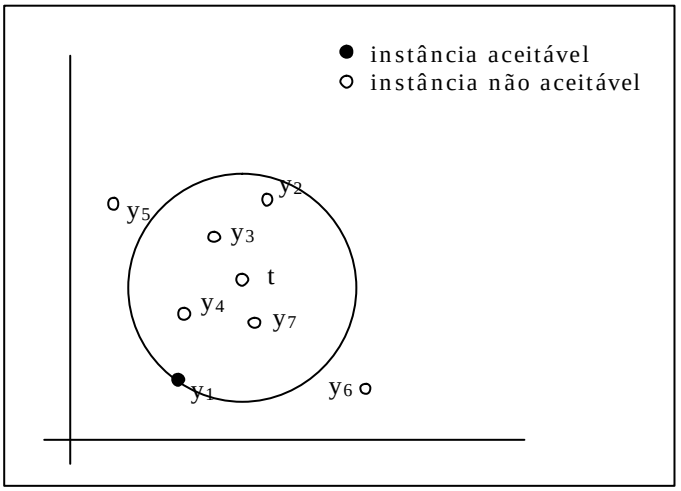

Figura 2.10 Situação 2: Instâncias escolhidas pelo IB3 que terão seus registros de classificação atualizados

Tabela 2.3 Tabela de similaridades das instâncias apresentadas na Figura 2.9

	Similaridades para a instância t	**Similaridades em ordem crescente**
sim(t, y_1)	-1.4	5ª
sim(t, y_2)	-1.1	4ª
sim(t, y_3)	-0.5	1ª
sim(t, y_4)	-1.0	3ª
sim(t, y_5)	-1.6	6ª
sim(t, y_6)	-1.9	7ª
sim(t, y_7)	-0.9	2ª

A existência ou não de instâncias aceitáveis não altera o comportamento do IB3, uma vez que no caso de nenhuma das instâncias armazenadas serem aceitáveis, o algoritmo "elege" uma delas como aceitável (através da escolha aleatória) e simula uma situação em que existe pelo menos uma instância aceitável e passa a se comportar como tal (SANTOS & NICOLETTI (1997)).

Pelo uso destes mecanismos de atualização dos registros de classificação e "filtragem" das instâncias que apresentarem ruídos, o IB3 apresentou desempenho consideravelmente melhor que os algoritmos IB1 e IB2, especialmente em domínios com grande quantidade de ruídos, pois, a maioria das instâncias com ruídos é descartada da descrição do conceito, não participando assim, da expressão final, que é utilizada na fase de classificação. No Algoritmo 2.3 é apresentada a descrição do IB3.

```
DC ← ∅
for_each x ∈ Conjunto de Treinamento do
        begin
          1.  for_each y ∈ DC do
                sim[y] ← similaridade(x,y)
          2.  if ∃ { y ∈ DC | aceitável (y) }
                then y_max ← y ∈ DC com a maior sim[y] | aceitável (y)
                else
                  begin
                      2.1. r ← valor aleatório ∈ [1, | DC |]
                      2.2. y_max ← y ∈ DC que é a r-ésima instância mais
                           similar a y
                  end
          3.  if classe(x) = classe(y_max)
                then classificação ← correta
                else
                  begin
                      3.1. classificação ← incorreta
                      3.2. DC ← DC ∪ {x}
                  end
          4.  for_each y ∈ DC do
```

```
                begin
                  if sim[y] ≤ sim[y_max]
                    then
                      begin
                          4.1. atualiza o registro de classificação
                          4.2. if registro de classificação de y é significativamente
                               pobre
                                 then DC ← DC – {y}
                      end
                end
        end
```

Algoritmo 2.3 Algoritmo IB3

Os algoritmos IB1, IB2 e IB3 assumem que todos os atributos que descrevem as instâncias de treinamento têm a mesma relevância na descrição do conceito. Isso faz com que esses algoritmos, particularmente o IB3, tenham seu desempenho degradado rapidamente à medida que atributos irrelevantes são usados na descrição das instâncias de treinamento. Motivado por esse fato foi proposto em AHA (1992) o algoritmo IB4, que é apresentado e discutido a seguir.

2.3.4 Algoritmo IB4

O algoritmo IB4 pode ser abordado como uma versão mais elaborada do IB3, que procura incorporar ao processo de aprendizado a relevância dos atributos na expressão do conceito que representam. O IB4, para cada uma das classes participantes do conjunto de treinamento, vai procurar criar um vetor de pesos de atributos, que reflita a "relevância" dos atributos na expressão da classe em questão.

O IB4 criará, então, tantos vetores de pesos quantas forem as classes participantes do conjunto de treinamento. No caso específico do IB4 a função de similaridade inclui também como parâmetro um conceito; calcula a similaridade entre duas instâncias levando em consideração os pesos de atributos associados a um determinado conceito. O IB4 difere do IB3 principalmente com relação a:

- função de similaridade: é dependente do conceito. O cálculo da similaridade entre as instâncias x e y, para determinar a classificação de x com relação a um determinado conceito c, utiliza a definição mostrada na Figura 2.11;

$$similaridade(x,y) = -\sqrt{\sum_{i=1}^{n} peso_{c_i}^2 * f(x_i, y_i)}$$

onde:
- c_i: peso do atributo i no conceito c, para i=1,..., n
- $f(x_i,y_i)$: definida como no IB1 – ver Tabela 2.2.

Figura 2.11 Função de similaridade adotada pelo IB4

O cálculo da similaridade entre instâncias é também dependente do conceito e justificado em AHA (1992) por meio do seguinte exemplo: “para qualquer tigre t e gato g, a *similaridade*(animal,t,g) deve ser maior que a *similaridade*(animal_estimação,t,g)”. Dessa forma, o IB4 (descrito no Algoritmo 2.4) "aprende" funções de similaridade "customizadas” para cada conceito, sendo que:

- função de classificação: no IB4 as instâncias são classificadas com relação a um determinado conceito. Assim, cada descrição de um conceito agrupa todas as instâncias incorretas juntas, independente de suas outras classificações com relação a outros conceitos;
- atualizador da descrição do conceito: este atualizador aprende os pesos dos atributos para cada conceito através de um mecanismo onde, toda vez que uma instância x é classificada, o seu vizinho mais próximo y, com relação a descrição do conceito c, é usado para atualizar os pesos.

Sejam:
- x: instância sendo classificada
- y_{max}: vizinho aceitável mais próximo
- c: conceito alvo
- λ: maior valor entre a freqüência de classe de x e de y_{max}

```
DC ← ∅
for_each x ∈ Conjunto de Treinamento do
  begin
    1. for_each y ∈ DC do
       sim[y] ← similaridade(x,y)
    2. if ∃ { y ∈ DC | aceitável(y) }
         then y_max ← algum y ∈ DC com a maior sim[y] | aceitável(y)
         else
           begin
             2.1. r ← valor aleatório ∈ [1, |DC|]
             2.2. y_max ← algum y ∈ DC que é a r-ésima instância mais
                  similar a y
           end
    3. if classe(x) = classe(y_max)
         then classificação ← correta
        else
```

begin
3.1. classificação ← incorreta
3.2. DC ← DC ∪ {x}
end
4. ***for_each*** y ∈ DC ***do***
begin
if sim[y] ≥ sim[y_{max}]
then
begin
4.1. atualiza o registro de classificação de y
4.2. ***if*** registro de classificação de y é significativamente pobre
then DC ← DC – {y}
end
end
5. ***for_each*** atributo i ***do***
begin
5.1. diferença = $| x_i - y_{max_i} |$
5.2. ***if*** classe(x)=classe(y_{max})
then $pesoacumulado_{C_i} = pesoacumulado_{C_i} + (1 - \lambda) * (1 - diferença)$
else $pesoacumulado_{C_i} = pesoacumulado_{C_i} + (1 - \lambda) * (diferença)$
5.3. $pesonormalizado_{C_i} = pesonormalizado_{C_i} + (1 - \lambda)$
5.4. $peso_{C_i} = \max\left(\frac{pesoacumulado_{C_i}}{pesonormalizado_{C_i}} - 0.5, 0 \right)$
end
end

Algoritmo 2.4 Algoritmo IB4

Neste algoritmo, a atualização dos pesos dos atributos é realizada no passo 5.

O IB4 utiliza um método para tratar instâncias com valores relativos a atributos ausentes, porém, como comentado em SANTOS & NICOLETTI (1997) este método traz um problema para o IB4, pois ao atualizar os pesos dos atributos com valores ausentes o IB4 "aprende" um conjunto de pesos de atributos, eventualmente incorretos, uma vez que o algoritmo assume valores para esses atributos que nem sempre refletem uma situação plausível. Assim que as novas instâncias com os novos atributos são introduzidas, o IB4 precisa novamente aprender os pesos dos atributos, agora de maneira correta, após essas instâncias serem processadas, o que implica em um grande tempo de processamento do algoritmo.

Para contornar este problema foi proposto o algoritmo IB5, discutido a seguir, que permite que novos atributos sejam tolerados de maneira mais eficiente.

2.3.5 Algoritmo IB5

Normalmente, em situações de aprendizado nem sempre as instâncias são inicialmente descritas por todos os possíveis atributos. Muitas vezes, valores ausentes de atributos são introduzidos após o início do processo de aprendizado; sendo assim, é interessante que algoritmos de aprendizado permitam a introdução desses valores para que estes possam ser incorporados posteriormente à expressão do conceito.

O algoritmo IB5 é tido como uma extensão do IB4. Em AHA (1992) são descritos experimentos onde foram aplicados os algoritmos IB4 e IB5 a um domínio de dados artificial contendo seis atributos com valores booleanos, dos quais apenas um atributo é relevante, e uma classe associada.

O IB4 reagiu mais lentamente à introdução desses novos atributos porque aprendeu um conjunto incorreto para o peso do atributo, o que fez com que este necessitasse de muito mais processamento, pois à medida que as próximas instâncias (que introduziram o novo atributo) foram processadas, o conjunto incorreto de peso dos atributos teve que ser reaprendido.

No caso do IB5, essa reação mais rápida é justificada pela utilização do método de tolerância a novos atributos, onde são considerados apenas atributos cujos valores são conhecidos durante as tentativas de classificação, ignorando atributos cujos valores estão ausentes. Dessa forma, ao adotar tal procedimento o IB5 não precisa reaprender os pesos dos atributos.

Ao calcular a similaridade entre as instâncias x e y, com relação ao conceito c, se o valor de um determinado atributo estiver ausente (seja em x ou em y), esse atributo é ignorado pela função, como pode ser visto através da Figura 2.12.

$$similaridade(x,y) = -\sqrt{\sum_{i=1}^{n} peso_{c_i}^2 * dif(x_i, y_i)^2}$$

onde:

$$dif(x_i, y_i) = \begin{cases} |x_i - y_i| & \text{se } ambos\_conhecidos(x_i, y_i) = 1 \\ 0 & \text{caso contrário} \end{cases}$$

- $ambos\_conhecidos(x_i, y_i) = 1$ se ambos os valores do atributo, tanto em x quanto em y são conhecidos e 0 caso contrário.

Figura 2.12 Função de similaridade adotada pelo IB5

O algoritmo do IB4 descrito na seção 2.5 (Algoritmo 2.4) é praticamente idêntico ao do IB5, diferindo apenas com relação à função de similaridade e o Passo 5 do algoritmo.

No IB5, os passos 5.1 ao 5.4 são processados somente se ambos os valores são conhecidos, ou seja, *ambos_conhecidos*(x_i, y_i) = 1 e a função utilizada para o cálculo da similaridade é a definida na Figura 2.12.

Quando comparado o IB5 ao IB4, o fator determinante para a reação mais rápida do primeiro, é que este atualiza os pesos apenas quando os valores dos atributos são conhecidos, tanto em x quanto em y. Isso permite que os pesos dos atributos sejam aprendidos corretamente e que na presença de novos atributos não seja necessário "aprender" novamente os pesos dos tributos, como acontece no IB4. Dessa forma, como os pesos dos atributos serão corretamente aprendidos, o uso desses pesos no cálculo da similaridade permitirá similaridades mais precisas.

Em SANTOS & NICOLETTI (1997) encontramos um resumo das principais características dos algoritmos da família IBL mostrados nas seções 2.2 a 2.6. Este resumo é apresentado aqui nas Tabelas 2.4 e 2.5.

Neste capítulo foram descritos o algoritmo NN e duas de suas versões e a família IBL e seus algoritmos. Todos esses algoritmos são inst6ancias do modelo de aprendizado baseado em instâncias. No próximo capítulo será introduzida a técnica de busca e otimização denominada Algoritmos Genéticos, que será incorporada a algoritmos da família NN e IBL com o objetivo de melhorar seu desempenho.

Tabela 2.4 Estrutura dos algoritmos da Família IBL

Estrutura	IB1	IB2	IB3	IB4	IB5
Função de Similaridade	▪ n: número de atributos ▪ $f(x_i,y_i) = (x_i - y_i)^2$, para atributos numéricos ▪ $f(x_i,y_i) = (x_i \neq y_i)$, para atributos simbólicos ▪ $f(x_i, y_i) = 1$, para atributos que tenham valores ausentes tanto na representação de x quanto na de y. Se apenas um estiver ausente, o IB1 assume como este valor mais diferente possível do corrente	mesma do IB1	mesma do IB1	$sim(c,x,y) = -\sqrt{\sum_{i=1}^{n} peso_{c_i}^2 * (x_i - y_i)^2}$ ▪ similaridade é dependente do conceito (c)	$sim(c,x,y) = -\sqrt{\sum_{i=1}^{n} peso_{c_i}^2} * dif(x_i, y_i)^2$ $dif(x_i, y_i) = \begin{cases} \lvert x_i - y_i \rvert & ambos\_conhecidos\ (x_i, y_i) \\ 0 & \text{caso contrário} \end{cases}$ ▪ ambos $(x_i, y_i) = 1$ se ambos valores são conhecidos e 0, caso contrário
Função de Classificação	vizinho mais próximo	mesma do IB1	vizinho aceitável mais próximo	mesma do IB3	mesma do IB3
Atualizador da Descrição do Conceito	▪ salva todas as instâncias	▪ salva apenas as instâncias classificadas incorretamente	▪ salva apenas as instâncias incorretas ▪ usa apenas instâncias boas classificadoras ▪ descarta instâncias que não são boas classificadoras	mesmo do IB3, e: ▪ introduz pesos nos atributos e mecanismo de atualização de pesos	mesmo do IB3, e: ▪ mecanismo próprio de atualização de pesos dos atributos

Tabela 2.5 Tratamento de instâncias e atributos nos algoritmos da Família IBL

Tratamento de	IB1	IB2	IB3	IB4	IB5
Atributos Ausentes	$x = (x_1 = ?, \ldots, x_n = v_{i_n})$ e $y = (y_1 = ?, \ldots, y_n = v_{j_n})$ ▪ o valor do atributo x_i é assumido como aquele mais diferente possível de v_{j_i} ▪ se ambos os valores, tanto na representação de y quanto na de x, estão ausentes, $f(x_i, y_i)=1$	mesmo do IB1	mesmo do IB1	mesmo do IB1	▪ ignora atributos com valores ausentes ▪ usa apenas atributos com valores conhecidos na classificação
Instâncias com Ruído	não tem	não tem	▪ descarta instâncias com ruídos baseado nos registros de classificação	mesmo do IB3	mesmo do IB3
Novos Atributos	não tem	não tem	não tem	não tem	▪ novos atributos são tratados, permitindo a sua incorporação na expressão do conceito
Pesos dos Atributos	não tem	não tem	não tem	$peso_{c_i} = \max\left(\frac{pesoacumulado_{c_i}}{normalizapeso_{c_i}} - 0.5, 0\right)$ ▪ o peso é aumentado quando o atributo faz predição correta e, diminuído, caso contrário.	mesmo do IB4, mas: ▪ atualiza o peso do atributo somente quando o valor do atributo é conhecido para ambas as instâncias

3 capítulo

Fundamentos de Algoritmos Genéticos

3.1 Introdução

Algoritmo genético (AG) é um método de busca e otimização inspirado nos mecanismos de evolução dos seres vivos e em conceitos de Genética. Os princípios básicos de algoritmos genéticos (AGs) foram rigorosamente estabelecidos por John Holland (HOLLAND (1975)) e popularizados por David Goldberg (GOLDBERG (1989)), um de seus alunos. Estes algoritmos seguem o princípio da seleção natural estabelecido por Charles Darwin em seu livro *The Origin of Species* em 1859; e, de acordo com este princípio, quanto melhor um indivíduo se adaptar ao seu meio ambiente, maior será sua chance de sobreviver e gerar descendentes, enquanto que indivíduos menos adaptados, tendem a desaparecer durante o processo evolutivo.

Em geral, AGs têm sido usados para solução de problemas de otimização. Os AGs diferem das técnicas de busca e otimização convencionais em vários aspectos. Esses aspectos são sumarizados em GOLDBERG (1989) como segue:

- AGs trabalham com uma codificação do conjunto de parâmetros e não com os próprios parâmetros;
- AGs trabalham com uma população de soluções potenciais e não apenas com uma;
- AGs utilizam informações de custo ou recompensa e não derivadas ou outro conhecimento auxiliar;

- AGs utilizam regras de transição probabilísticas e não determinísticas.

Vários autores buscam justificar a grande popularidade de AGs. Segundo MITCHELL (1997) essa popularidade se deve, ao fato:

- da evolução ser um método de adaptação reconhecidamente bem sucedido e robusto em sistemas biológicos;
- de poderem realizar buscas em espaços com hipóteses (também chamadas de soluções candidatas) contendo partes complexas que interagem entre si e onde o impacto de cada parte no desempenho da hipótese como um todo é de difícil modelagem;
- de serem facilmente paralelisáveis.

Além disso, segundo BEASLEY *et al* (1993), "o poder de AGs vem do fato de que a técnica é robusta e pode lidar com sucesso com um vasto número de áreas de problemas, incluindo aquelas consideradas difíceis para outros métodos resolverem. Não é garantido que AGs encontram a solução ótima do sistema mas, eles geralmente são bons em encontrar soluções do problema que são 'aceitavelmente boas', de uma maneira 'aceitavelmente rápida' ".

A grande popularidade de AGs é conseqüência das vantagens que estes apresentam. Dentre as principais cabe destacar:

- a facilidade na implementação e o fato de não exigir um profundo conhecimento matemático sobre o problema;
- a otimização de um grande número de variáveis;
- o fornecimento de uma lista de parâmetros ótimos e não apenas de uma única solução;
- o fato de terem grande flexibilidade para trabalhar com restrições e otimizar múltiplas funções com objetivos conflitantes.

Apesar de inúmeras vantagens, é importante salientar que a maior parte das publicações sobre pesquisas na área de AGs enfatizam a representação, ou apenas estabelecem diretrizes que servirão como auxílio na escolha do tamanho da população para um determinado problema. É objeto de estudo também, uma análise da diferença de desempenho entre os vários mecanismos de cruzamento ou a influência de uma taxa de mutação alta ou baixa. Segundo CASTILHO (2003), "a

única forma de proceder para a solução de um problema usando AG, é avaliar resultados obtidos na solução de problemas similares e, então, escolher uma abordagem que seja sensível ao problema em questão e que também seja viável de ser implementada".

3.2 Terminologia

Devido ao fato dos AGs serem baseados em duas áreas, a ciência da computação e a genética natural, e pelo fato de poderem ser vistos como uma tentativa de metáfora da Evolução Darwiniana, esses algoritmos possuem muitos termos originados na Biologia. A Tabela 3.1 baseada em CASTILHO (2003) apresenta os principais termos citados na literatura.

Tabela 3.1 Termos de algoritmos genéticos

Termo	Representação
Cromossomo (*string*, indivíduo)	Um elemento da população. Este elemento ou indivíduo é formado pelo cromossomo e sua função de aptidão (possível solução do problema).
Genes (*bits*)	Uma informação do cromossomo (certa característica da solução-cromossomo).
População	Conjunto de indivíduos que representam os atuais pontos que fazem parte do espaço de soluções.
Geração	Identifica cada uma das várias populações criadas durante o processo evolutivo.
Função de Aptidão	Função que "mede" a adequabilidade de um cromossomo. É utilizada pelo mecanismo de seleção para identificar quais cromossomos irão "sobreviver" e recombinar.

Segundo GEN & CHENG (1997), outra característica que faz a técnica de AGs diferir das técnicas convencionais de busca, é a de inicializar a busca da solução a partir de um conjunto de soluções potenciais geradas randomicamente e chamadas população inicial, cujo tamanho é, geralmente, definido empiricamente. Cada indivíduo na população, chamado de cromossomo, corresponde a um ponto no espaço de busca e representa uma possível solução para o problema – solução que

também pode ser chamada de hipótese. Um cromossomo é uma cadeia de símbolos; é usualmente, mas não necessariamente, uma cadeia de *bits*.

Os cromossomos evoluem através de sucessivas interações, chamadas de gerações. Para a criação de uma geração, os cromossomos são avaliados, usando uma medida de aptidão. Para criar a próxima geração, novos cromossomos, chamados descendentes são formados de uma das seguintes maneiras:

1. combinando dois cromossomos da geração atual usando um operador de cruzamento (*crossover*) (ver Seção 3.6.2);
2. modificando um cromossomo através de um operador de mutação (ver Seção 3.6.3);

Dessa forma, uma nova geração é formada:

- selecionando, de acordo com os valores de aptidão, alguns pais e descendentes, por meio de um operador de seleção;
- substituindo a geração corrente pelos descendentes.

Espera-se que após um certo número de gerações (número dependente do problema), o algoritmo convirja para o melhor cromossomo, que possivelmente representa a solução ótima (ou quase ótima) para o problema. Esta "melhor" solução geralmente é definida como aquela que otimiza um valor numérico predefinido para o problema em questão.

Um AG termina quando um critério de parada é satisfeito. Entre os principais critérios de parada cabe destacar:

- número de gerações ou um tempo-limite;
- chegada ao valor ótimo da função (se conhecido) durante o processamento;
- convergência, ou seja, quando não ocorrer melhoramento significativo no cromossomo de melhor aptidão durante um dado número de gerações;
- quando um alto percentual da população possuir o mesmo valor de função de aptidão.

3.3 Um Algoritmo Genético Simples

Um AG simples consiste de (COLEY (1999)):

- um número ou população de soluções potenciais do problema;
- uma maneira de calcular quão "boa" ou "ruim" é cada uma das soluções individuais em uma população;
- um método para compor partes das melhores soluções, de maneira que novas possam ser formadas;
- um operador de mutação para evitar perda permanente de diversidade na população.

Organizando essas características é possível obter o pseudocódigo do AG canônico, extraído de LACERDA (1999) e mostrado na Figura 3.1.

```
{S(t) representa a população de cromossomos na geração t}

t ← 0
inicializar S(t)
avaliar S(t)
enquanto o critério de parada não for satisfeito faça
        t ← t + 1
        selecionar S(t) a partir de S(t-1)
        aplicar crossover sobre S(t)
        aplicar mutação sobre S(t)
        avaliar S(t)

fim enquanto
```

Figura 3.1 Pseudocódigo do AG canônico

Uma descrição detalhada de um AG envolve o estabelecimento de duas taxas: p_c e p_m que fornecem respectivamente, a probabilidade de cruzamento e a probabilidade de mutação. Essas taxas podem ser fixadas inicialmente e permanecerem constantes durante todo o processo evolutivo. Entretanto, existe a possibilidade de variá-las, dinamicamente, à medida que o AG evolui.

Normalmente a população inicial é gerada randomicamente. Sobre uma população é aplicado um conjunto de operações para que, a partir dela, seja possível gerar uma nova população. Espera-se que, ao longo das sucessivas

gerações, a aptidão de seus elementos (soluções potenciais do problema) melhore. Em AGs existem duas operações fundamentais, categorizadas como:

- Operações de evolução: operador de seleção.
- Operações genéticas: operadores de cruzamento (*crossover*) e mutação.

Essas operações são utilizadas para assegurar, entre outros aspectos, uma diversidade na população. Particularmente, o operador de mutação garante que a probabilidade de exame de qualquer ponto no espaço de busca nunca será zero.

3.4 Possíveis Representações dos Dados

A escolha da representação das variáveis de um problema é determinante para que este problema possa ser resolvido por AGs. Entretanto, o mecanismo de codificação depende diretamente da natureza do problema a ser resolvido.

A representação binária é historicamente importante, uma vez que foi utilizada nos primeiros trabalhos de John Holland (HOLLAND (1975)). Este tipo de representação é mais fácil de utilizar e manipular e facilmente analisável teoricamente. Entretanto, nem sempre esta é a representação mais adequada para todos os problemas. Segundo GEN & CHENG (1997), dependendo da caracterização do problema esta representação pode não ser adequada uma vez que não é uma codificação natural. Geralmente, a representação binária é utilizada para a representação de problemas com variáveis discretas, já se o problema apresentar variáveis contínuas e for necessário trabalhar com cromossomos longos, então a representação real será a mais adequada.

A representação de um cromossomo utilizando números reais apresenta vantagens, tais como, o fato de ser mais facilmente compreendida pelo ser humano do que uma cadeia de *bits*, de requerer menos memória e ainda permitir a criação de novos operadores e de variações dos operadores existentes.

Além da representação real e binária, existem ainda representações que são típicas de um determinado problema, mas, que tentam refletir de alguma forma, a estrutura natural dos dados que envolvem este problema.

Ainda segundo GEN & CHENG (1997), “a escolha de uma representação apropriada para as soluções candidatas de um problema é fundamental para a aplicação de AG na resolução de problemas do mundo real, o que condiciona todos os passos subsequentes desta técnica. Para qualquer aplicação é necessário primeiro analisar cuidadosamente (o problema) para garantir uma representação da solução que seja apropriada, bem como, definir um conjunto de operadores genéticos, específicos ao problema, que sejam significativos”.

3.5 Função de Aptidão

A aptidão irá expressar o quanto uma solução codificada por um cromossomo é “boa” e, sua definição é uma das maiores dificuldades encontradas na utilização de AGs para resolução de determinados problemas. De acordo com a complexidade do problema, esta pode ser bastante complicada e demandar um alto custo computacional.

Os cromossomos que tiverem melhores valores de aptidão serão os que terão maiores chances de passarem para a geração seguinte.

3.6 Operadores Genéticos

Os operadores genéticos selecionam e transformam os cromossomos de uma população através de sucessivas gerações, repetindo o processo até chegar a um resultado satisfatório.

3.6.1 Operador de Seleção

O propósito do operador de seleção é fazer a seleção dos possíveis candidatos ao cruzamento, com o objetivo de reproduzir membros da população que tenham bons valores da função de aptidão (via de regra, a função que “descreve” o problema).

Na literatura de AGs existem numerosos esquemas de seleção, cujas descrições não fornecem indicações rigorosas para quais problemas são mais

convenientes. Dessa forma, a seguir são descritos três métodos de seleção: *rank*, roleta e torneio.

Seleção *Rank*

Neste método, os cromossomos da população são classificados de forma crescente de acordo com sua aptidão de 1 a N_{pop} (onde N_{pop} = tamanho da população). De acordo com BENETT (1997) é associado à posição 0 o elemento com o melhor valor de aptidão e a posição N_{pop} - 1 o elemento com pior valor de aptidão. Dessa forma, os melhores cromossomos possuem as melhores posições e, conseqüentemente, maiores chances de reprodução. Assim, um indivíduo α com posição r_α no *rank* é selecionado com probabilidade p_α dada pela equação:

$$p_a = \frac{2r_a}{N_{pop}(N_{pop} - 1)}$$

Seleção por Roleta (*Roulette Wheel*)

Este método foi inicialmente proposto por GOLDBERG (1989) e consiste na criação de uma roleta onde cada cromossomo possui um segmento proporcional ao valor de sua aptidão.

A probabilidade de seleção p_i de um cromossomo com aptidão F_i em uma população N_{pop} é dada pela equação:

$$p_i = \frac{F_i}{\sum_{i=1}^{N_{pop}} F_i}$$

A partir de p_i é possível calcular a probabilidade acumulada (q_i) de cada cromossomo, de acordo com a equação:

$$q_i = \sum_{j=1}^{i} p_j$$

A Figura 3.2 extraída de LACERDA (1999) mostra o pseudocódigo do algoritmo referente ao método de seleção por roleta.

$$total \leftarrow \sum_{i=1}^{N} F_i$$

```
{
  N   : tamanho da população
  Σ_{i=1}^{N} F_i : soma das aptidões de todos os cromossomos na população
  f_i  : aptidão de cada cromossomo
}
rand ← randômico(0,total)
totalparcial ← 0
i ← 0
repetir
      i ← i + 1
      totalparcial ← totalparcial + F_i
até totalparcial ≥ rand
retornar o cromossomo s_i
```

Figura 3.2 Seleção por roleta

A seguir é mostrado um exemplo baseado em DE NARDIN (2000) de como o método de seleção por roleta funciona.

Tendo uma população com quatro cromossomos, cada um deles representado por uma cadeia de símbolos como mostrado na Tabela 3.2 e utilizando os valores apresentados na terceira coluna da Tabela 3.2, é elaborada a roleta mostrada na Figura 3.3. Os dados mostrados na Tabela 3.2 foram escolhidos unicamente com o propósito de ilustrar a técnica e, não correspondem à representação de uma situação real.

Tabela 3.2 Exemplo da seleção por roleta em uma população de tamanho 4

Nº do cromossomo	**Cadeia de símbolos**	**Porcentagem do total (%)**
1	0010100	7
2	0011001	10
3	1011001	35
4	1111101	48
	Total	100

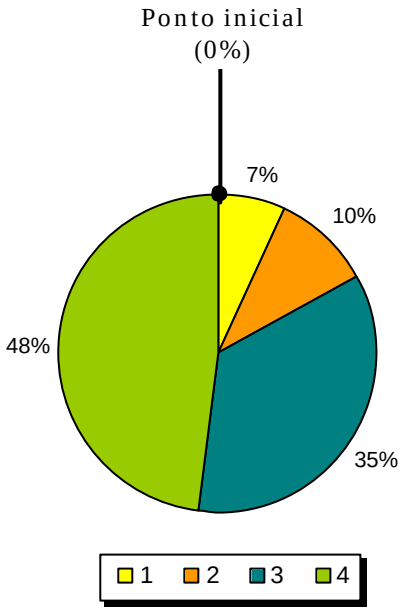

Figura 3.3 Representação gráfica da roleta de seleção

A roleta da Figura 3.3 será girada n vezes (n = 4 - tamanho da população) para efetuar a seleção da população intermediária (utilizada para alocar os cromossomos pais selecionados). Indivíduos que tiverem maior área na roleta têm maiores chances de serem selecionados para o cruzamento que indivíduos com menor área. Ainda na Figura 3.3 deve-se considerar que:

- a região pertencente ao cromossomo 1 pertence ao intervalo [0,7];
- a região pertencente ao cromossomo 2 pertence ao intervalo]7,17] e assim, sucessivamente.

Portanto, se durante o giro da roleta, o valor randômico gerado for o que pertencer a região do cromossomo 4, este será o escolhido.

Neste método de seleção é possível perceber que indivíduos que têm maior valor de aptidão associado têm maior chances de serem selecionados. Isto faz com que o método de seleção por roleta tenda a ser elitista, mas não necessariamente. O elitismo, de acordo com MITCHELL (1996), “é uma adição a muitos métodos de seleção, que forçam o AG a manter alguns dos melhores indivíduos a cada geração”. Isto é, tais indivíduos passam para a próxima geração, sem sofrer qualquer alteração.

Seleção por Torneio

Neste método de seleção são escolhidos aleatoriamente (com probabilidades iguais) n cromossomos da população (sendo que n geralmente assume valor 2), e o melhor (com melhor aptidão) entre esses dois cromossomos é selecionado. Esse processo é repetido N_{pop} vezes, sendo que N_{pop} representará o tamanho da população.

Para a seleção por torneio não é necessário realizar escalonamento de aptidão nem *ranking* (LACERDA (1999)).

Cabe ressaltar que ao utilizar uma seleção baseada em aptidão não é garantida a seleção de qualquer indivíduo em particular, mas, daquele que é o mais apto.

3.6.2 Operador de Cruzamento (*Crossover)*

O operador de cruzamento é um operador binário e é utilizado após o operador de seleção. Essa fase se caracteriza pela troca de segmentos entre "pares" de cromossomos selecionados (pais), para dar origem a novos cromossomos (filhos), que poderão ou não fazer parte da população da próxima geração.

É considerado o operador genético predominante e ocorre com uma probabilidade definida pela taxa de cruzamento p_c ($0.6 \leq p_c \leq 1$). Uma taxa alta permite uma exploração maior do espaço de solução e reduz as chances de convergência para um ótimo local, em contrapartida, se essa taxa for “muito” alta pode resultar em desperdício de tempo computacional, pois acontecerá a exploração de regiões não promissoras dentro do espaço de soluções.

A seguir estão descritas as principais formas de cruzamento considerando cada uma das representações citadas na Seção 3.4.

Representação Binária

Cruzamento de um ponto

Nesta forma é escolhido um ponto de cruzamento; a partir deste ponto, as informações genéticas dos pais são trocadas, ou seja, as informações anteriores ao ponto escolhido em um dos pais são concatenadas às informações posteriores a este ponto, no outro pai, na construção de um descendente.

Exemplo: Considere dois pais: P1 = 0 0 1 1 0 1 0 1 e P2 = 1 1 1 0 1 0 1 1. Se o ponto 4 for escolhido aleatoriamente como ponto de corte, então, os filhos terão as configurações como mostra a Figura 3.4.

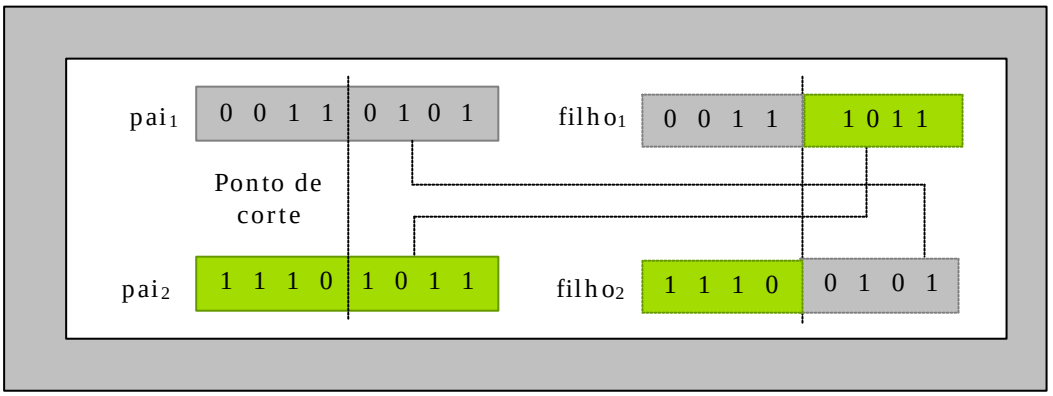

Figura 3.4 Exemplo de cruzamento de um ponto

Cruzamento multiponto

O cruzamento multiponto é semelhante ao cruzamento de um ponto. Neste cruzamento, entretanto, a troca de material genético acontece usando mais de um ponto. A Figura 3.5 mostra o exemplo do cruzamento de dois pontos. Os dois pontos de cortes são escolhidos aleatoriamente e as seções entre eles são trocadas entre os pais. Neste exemplo, os pontos escolhidos para corte são o ponto 2 e o ponto 6, e os pais são os mesmos do exemplo anterior.

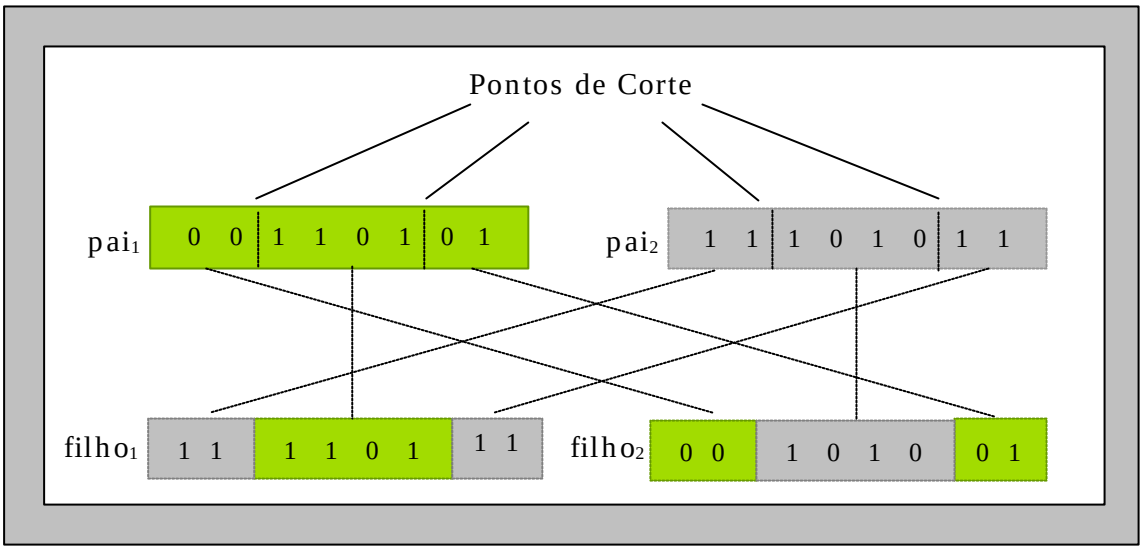

Figura 3.5 Exemplo de cruzamento de dois pontos

Cruzamento uniforme

O cruzamento uniforme não utiliza pontos de cruzamento, mas, para cada par de pais é gerada uma máscara de *bits* aleatórios e o cruzamento é função dos valores da máscara. A Figura 3.6 ilustra o pseudocódigo do cruzamento uniforme.

```
{
    n = número de bits da cadeia
    mascara, pai1, pai2, filho1 e filho2 = vetores de tamanho n
}

for i:=1 to n do
    if mascara[i]=1
        then
          begin
              filho1[i] := pai1[i]
              filho2[i] := pai2[i]
          end
        else
          begin
              filho1[i] := pai2[i]
              filho2[i] := pai1[i]
          end
```

Figura 3.6 Pseudocódigo do cruzamento uniforme

Após a execução do algoritmo da Figura 3.6, usando o pai_1 e pai_2 da seção anterior tem-se como resultado as Figuras 3.7a (geração do $filho_1$) e 3.7b (geração do $filho_2$).

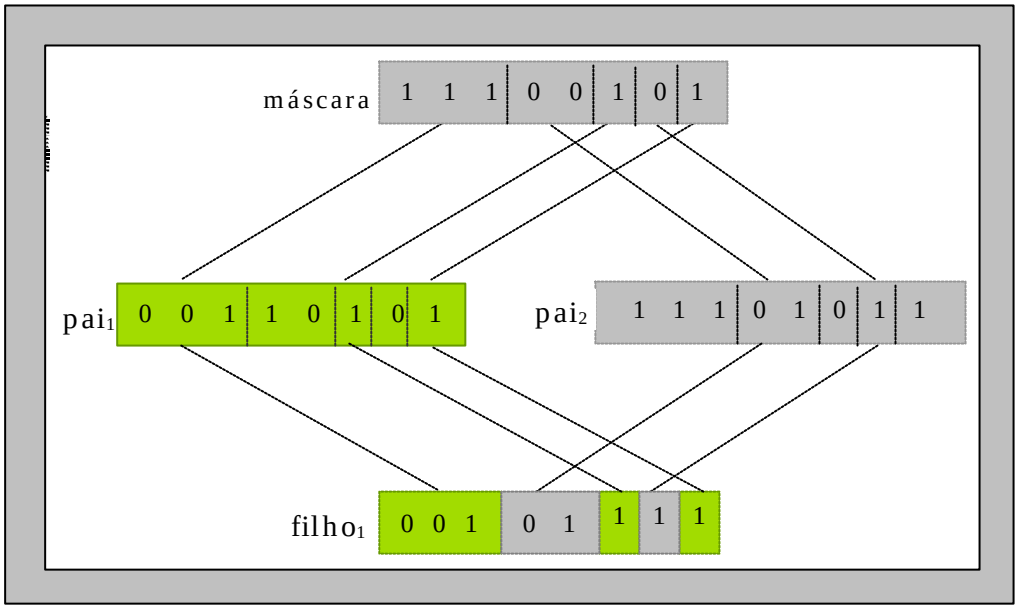

Figura 3.7a Geração do $filho_1$ por cruzamento uniforme

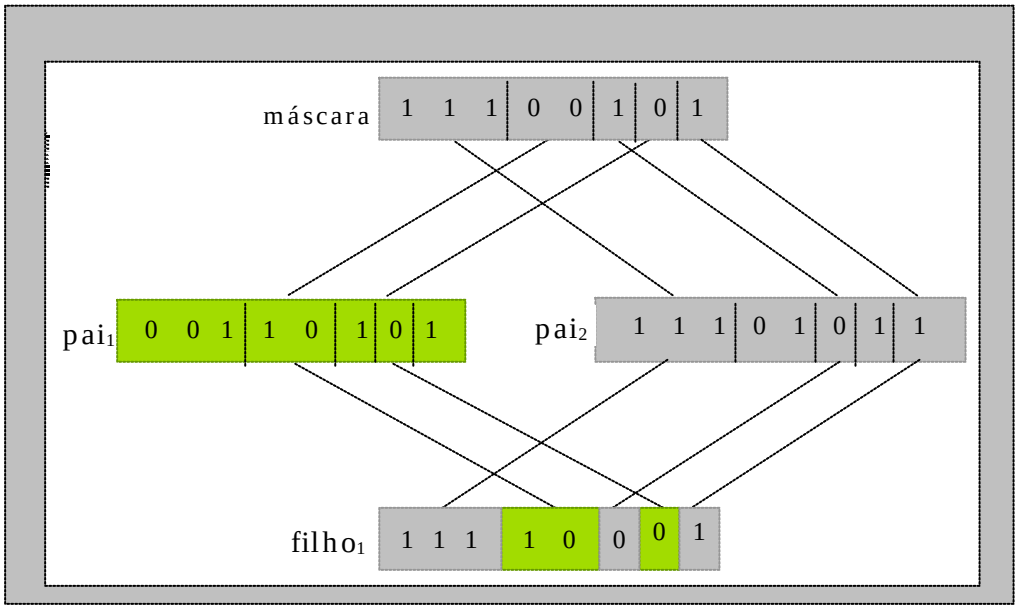

Figura 3.7b Geração do $filho_2$ por cruzamento uniforme

Representação Real

A literatura apresenta diversas propostas de cruzamento utilizando representação real. A seguir, serão descritas as principais.

Considerando que, os cromossomos pai, mãe[1] e filhos sejam notados por:

$$pai = (p_1, p_2, \ldots, p_n)$$

$$mãe = (m_1, m_2, \ldots, m_n)$$

$$filho_1 = (a_1, a_2, \ldots, a_n)$$

$$filho_2 = (b_1, b_2, \ldots, b_n)$$

Os dois filhos de p_i e m_i são calculados como:

$$a_i = \gamma_1 m_i + \gamma_2 p_i$$

$$b_i = \gamma_1 p_i + \gamma_2 m_i$$

onde: $\gamma_1 + \gamma_2 = 1;$

$\gamma_1 > 0$ e $\gamma_2 > 0$

$1 \leq i \leq n$

Alguns cruzamentos recebem nomes especiais de acordo com o valor de γ_1 e γ_2, como mostra a Tabela 3.3 extraída de GEN & CHENG (1997).

Tabela 3.3 Tipos de cruzamento de acordo com o valor γ

Cruzamento	g_1	g_2
média	0.5	0.5
affine	1.5	-0.5
Linear - $\gamma_1 + \gamma_2 \leq 2$	> 0	> 0

O cruzamento média possui uma variação denominada média geométrica que gera apenas um filho. Essa média geométrica é dada pela equação:

$$a_i = \sqrt{m_i p_i}$$

[1] Nesta seção serão utilizados os termos pai e mãe em substituição ao termo pais utilizado anteriormente para denotar o par de cromossomos que será cruzado. Isto foi empregado com o objetivo de facilitar a compreensão do leitor na exemplificação do cruzamento.

Outra proposta é apresentada em MICHALEWICZ (1996), onde os filhos são obtidos por meio das duas equações abaixo:

$$a_i = \beta m_i + (1 - \beta) p_i$$

$$b_i = (\beta - 1) m_i + \beta p_i$$

onde β é um número aleatório escolhido de uma distribuição uniforme no intervalo [0,1].

Abaixo estão descritos dois outros tipos de cruzamento encontrados na literatura.

Cruzamento Uniforme

Neste cruzamento é gerado apenas um filho cujas componentes são escolhidas de forma aleatória (uniforme) no intervalo $[p_i, m_i]$: $p_i < a_i < m_i$.

Cruzamento Simples

Esse cruzamento é como o cruzamento de um ponto da representação binária e realiza a troca de informações entre cromossomos (pais) a partir de um determinado ponto escolhido.

Na implementação proposta no Capítulo 4 foi utilizado esse modelo de cruzamento.

3.6.3 Operador de Mutação

Após a aplicação de cruzamento, os cromossomos estão sujeitos a mutação. A mutação tem como objetivo a introdução e manutenção da diversidade genética na população.

O operador de mutação é aplicado aos indivíduos com uma probabilidade dada pela taxa de mutação p_m. Uma baixa taxa de mutação previne que uma dada posição fique estagnada em um valor, além de possibilitar que se chegue a qualquer ponto no espaço de busca. Com uma taxa muito alta a busca se torna essencialmente aleatória além de aumentar muito a possibilidade de que uma boa

solução seja destruída. Dessa forma, normalmente são utilizadas baixas taxas de probabilidade de mutação.

Em uma cadeira binária, a mutação de um bit significa mudá-lo de um para zero ou vice-versa. Os *bits* de uma cadeia são mutados independentemente, de forma que a mutação de um bit não afeta a probabilidade de mutação de outros *bits*. A Figura 3.8 mostra um exemplo de como ocorre um processo de mutação em um determinado indivíduo.

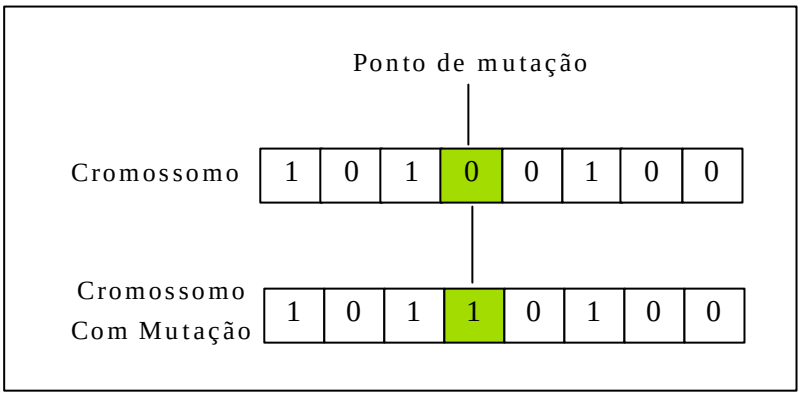

Figura 3.8 Esquema gráfico da ocorrência de mutação

Se a representação utilizada for a real, existem vários operadores. Os principais são:

- mutação randômica ou aleatória:. Consiste apenas na substituição de um gene por um número escolhido aleatoriamente no intervalo permitido pelo problema;
- mutação *creep*: adiciona ao gene um pequeno número aleatório obtido de uma distribuição normal (com média zero e desvio padrão pequeno) ou de uma distribuição uniforme. Outra alternativa possível, é multiplicar o gene por um número aleatório próximo de 1. A idéia por trás deste operador é a de que, se o cromossomo está perto do ponto máximo, uma pequena perturbação pode movê-lo rapidamente para esse ponto. Uma vez que esse operador é usado apenas para explorar um espaço de busca localmente, sua taxa pode ser relativamente alta (LACERDA (1999)).

Um algoritmo genético simples como o proposto na Figura 3.1 trata a mutação apenas como um operador secundário com a função de recuperar a perda de material genético. Essa recuperação de material genético é exemplificada por

SRINIVAS & PATNAIK (1994) da seguinte forma: suponha que todas as cadeias em uma população convergiram para o valor 0 em uma dada posição e a solução ótima tem valor 1 naquela posição. O operador de cruzamento não pode restaurar o valor 1 naquela posição, enquanto que a mutação pode.

3.7 Principais Parâmetros Genéticos

Alguns parâmetros influem no comportamento do AG, dessa forma, é conveniente analisá-los e estabelecê-los de acordo com as necessidades do problema e dos recursos disponíveis. A escolha ideal desses parâmetros é essencial para que o AG possa obter uma solução ótima ou quase ótima para o problema. Alguns desses parâmetros estão descritos abaixo:

- Tamanho da População: o tamanho da população afeta diretamente o desempenho global e a eficiência do AG. Essa influência pode ser de duas maneiras: se a população for pequena, esta oferece uma pequena cobertura do espaço de busca o que faz com que caia o desempenho; se a população for grande, há uma maior cobertura do domínio do problema, o que previne convergência prematura do problema para soluções locais do problema ao invés de globais, porém, um maior esforço computacional é requerido ou pode ser necessário que o algoritmo trabalhe por mais tempo.
- Taxa de Cruzamento: a velocidade com que novas estruturas serão introduzidas na população é diretamente dependente dessa taxa (quanto maior taxa, maior a velocidade). Se o valor da taxa for baixo, o algoritmo pode se tornar lento. Se for muito alto, a maior parte da população poderá ser substituída, ocorrendo perda de estruturas com alta aptidão.
- Taxa de Mutação: um baixo valor para esta taxa previne que uma dada posição fique estagnada em um valor, além de possibilitar a chegada em qualquer ponto no espaço de busca. Se esta taxa for muito alta, a busca se tornará essencialmente aleatória.

Neste trabalho, são analisadas as influências do tamanho da população e do número de gerações.

3.8 Um Exemplo do Uso de AG

Um exemplo simples da criação de uma nova população usando um AG, extraído de SRINIVAS & PATNAIK (1994) é apresentado a seguir. São utilizados os seguintes parâmetros:

- tamanho da população: 4 cromossomos;
- representação e tamanho do cromossomo: binária, 10 *bits*;
- função objetivo: assume valores no intervalo de 0 a 10, de acordo com o número de caracteres 1 na cadeia. Uma cadeia que apresente 10 caracteres 1 terá o valor máximo para a função objetivo, isto é, 10;
- função de avaliação: realiza uma operação de "divisão por 10" para normalizar o valor da função objetivo no intervalo de 0 a 1;

Suponha que as quatro cadeias geradas randomicamente, que formam a população inicial são aquelas mostradas na Figura 3.9a. Na mesma figura, pode ser verificado que os respectivos valores de aptidão são respectivamente: 0.3, 0.6, 0.6 e 0.9.

População Inicial

Cadeia	Valor de Aptidão
1 - 0000011100	0.3
2 - 1000011111	0.6
3 - 0110101011	0.6
4 - 1111111011	0.9

Figura 3.9a Cadeias que formam a população inicial e valores de aptidão

A seguir, assumindo que, por um determinado mecanismo de seleção, as cadeias eleitas para compor a população intermediária são as mostradas na Figura 3.9b.

Cadeia	Valor de Aptidão
1 – 1000011111	0.6
2 - 0110101011	0.6
3 - 1111111011	0.6
4 - 1111111011	0.6

Ponto de cruzamento

Figura 3.9b Cromossomos selecionados da população inicial para sofrerem cruzamento

Se a taxa de cruzamento for, por exemplo, 0.5, apenas dois cromossomos irão cruzar; por exemplo, o par (1,3). Se o ponto de cruzamento for estabelecido como aquele entre o quinto e o sexto *bits* da cadeia, após o cruzamento do par (1,3) a população intermediária terá a configuração mostrada na Figura 3.9c.

Cadeia	Valor de Aptidão
1 – 1000011111	0.5
2 - 0110101011	0.6
3 - 1111111011	0.6
4 – 1111111011	1.0

Figura 3.9c Cromossomos após cruzamento

A Figura 3.9d mostra a nova população, obtida a partir da população inicial, após a aplicação do operador de mutação na população exibida na Figura 3.9c. Note que o sexto bit da cadeia 2 e o primeiro bit da cadeia 4 sofreram mutação. Pode-se perceber que de um total de 40 *bits* apenas dois foram mutados, o que representa uma taxa de mutação efetiva de 0.05.

População Final	
Cadeia	Valor de Aptidão
1 – 1000011011	0.5
2 – 01101**1**1011	0.7
3 - 1111111011	0.9
4 – **0**111111111	0.9

Figura 3.9d Cromossomos após mutação

Para o término da execução do AG é necessário especificar um critério de parada. Este critério pode ser determinado em função de um número fixo de gerações, do valor da função de aptidão ou quando todas as cadeias da população atingirem um grau alto de similaridade.

Neste capítulo foram descritas as principais características da técnica de busca e otimização denominada Algoritmos Genéticos. Essa técnica será incorporada a alguns algoritmos descritos no Capítulo 2, visando melhoria de desempenho. O próximo capítulo descreve a cooperação entre AG e algoritmos baseada em instâncias, bem como os resultados obtidos com experimentos realizados utilizando essa cooperação.

4 capítulo

Descrição dos Experimentos e Análise dos Resultados

4.1 Introdução

Um dos objetivos deste trabalho foi investigar o uso de AGs na busca de vetores de pesos que produzam um melhoramento de desempenho de métodos de aprendizado baseado em instâncias, particularmente do *k* vizinho mais próximo. Um vetor de pesos associado a um conjunto de atributos busca ponderar a representatividade dos atributos na expressão do conceito.

Como visto no Capítulo 2, o algoritmo NN é baseado na idéia que, dado um conjunto de exemplos classificados, um exemplo não classificado deve pertencer à mesma classe que seu vizinho mais próximo, do conjunto de exemplos. Uma extensão desse algoritmo é o *k*-NN, que classifica uma nova instância considerando seus *k* vizinhos ($k > 1$). A nova instância é classificada na classe que for a majoritária entre os *k* vizinhos. Existem várias maneiras de gerenciar os conflitos entre os vizinhos de uma instância; a mais simples é a contagem entre os *k* vizinhos mais próximos.

O algoritmo *k*-NN é simples, rápido e bastante eficiente e trata todos os atributos que descrevem as instâncias, da mesma maneira, i.e., todos são igualmente significativos. Entretanto, existem situações em que o número de atributos que são significativos (ao processo de classificação) é pequeno, quando comparado com o número de atributos que são irrelevantes. A atribuição de pesos a atributos é, então, uma possível maneira de priorizar aqueles atributos que

efetivamente contribuem para a classificação. Um algoritmo *k*-NN que implementa uma política de pesos de atributos geralmente é referenciado como *Wk*-NN.

É importante lembrar que, como existe uma ordenação (baseada nas distâncias) entre os *k* vizinhos mais próximos (ver Seção 2.2.2) os vizinhos não necessariamente têm a mesma influência na decisão de classificação – é mais provável que a influência do vizinho mais próximo, na classificação da nova instância, seja maior do que a do *k*-ésimo vizinho, por exemplo. Assim sendo, além da associação de pesos a atributos, usualmente associam-se pesos a cada um dos *k* vizinhos mais próximos. Embora, a tendência seja que o desempenho de um algoritmo *Wk*-NN exceda o desempenho de um algoritmo *k*-NN, dado um vetor de pesos de atributos que seja ótimo, é muito difícil encontrar esse vetor de pesos que seja ótimo.

O vetor de pesos associados a atributos pode ser criado manualmente; um dos inconvenientes desse método é o fato de ser tendencioso, dado que, quem o define, o define convenientemente. Uma outra possibilidade é a de fazer uma abordagem exaustiva, varrendo o espaço de busca de todos os possíveis vetores de pesos, com aquela dimensão. Dependendo da dimensão do espaço, essa opção pode se tornar computacionalmente intratável. Uma terceira opção é usar uma outra ferramenta matemática que permita a obtenção do vetor de pesos – se não do vetor ótimo, pelo menos de um vetor que permita que o desempenho do *Wk*-NN seja superior à do *k*-NN. É sobre a pesquisa no uso de AG na determinação desse vetor de pesos que esse projeto de pesquisa investiu.

A determinação do vetor de pesos pode ser equacionada como um problema de otimização, que pode ser classificado como relativamente difícil, devido à possibilidade do espaço de busca poder ser bem vasto – em alguns domínios de conhecimento, instâncias possuem mais de 50 atributos que as descrevem. O problema, nessa situação, corresponderia à uma busca por um vetor num espaço 50-dimensional, onde o peso associado a cada um dos atributos pode assumir um valor real.

A próxima seção apresenta e descreve os experimentos realizados para avaliar o desempenho dos algoritmos, que foram implementados e estão disponibilizados como parte de um único sistema computacional. São analisados os resultados obtidos em experimentos utilizando cada um deles. A principal

contribuição do sistema é a de viabilizar uma plataforma computacional para experimentar tanto o aprendizado baseado em instâncias quanto com o aprendizado híbrido genético-baseado em instâncias, no qual algoritmos genéticos são usados para determinar o vetor de pesos associados a atributos, a ser incorporado a um algoritmo baseado em instância. O capítulo termina com a apresentação das conclusões e das possíveis linhas para continuidade do trabalho.

4.2 Descrição dos Experimentos e Análise dos Resultados

Geralmente, em aprendizado de máquina, a abordagem utilizada para comparar a eficiência relativa de dois algoritmos de aprendizado é executar os procedimentos que os implementam em diferentes subconjuntos do conjunto de dados disponível, testar as hipóteses aprendidas no restante dos dados e, então, fazer a média dos resultados desses experimentos. Esse processo é muitas vezes referenciado na literatura como validação cruzada (*cross validation*) (MITCHELL (1997)). No que segue, são descritos os experimentos realizados em vários domínios de conhecimento de maneira a avaliar a colaboração genética-baseada em instâncias. Para tanto, inicialmente, são descritos os domínios utilizados.

4.2.1 Descrição dos Domínios de Conhecimento

Para a avaliação da colaboração proposta foram realizados testes utilizando dados disponibilizados no repositório UCI *Repository of Machine Learning Databases* (*http://www.ics.uci.edu/~mlearn/MLRepository.html*) que, tradicionalmente, são usados para avaliações empíricas de sistemas de aprendizado de máquina. Além dos domínios do UCI *Repository*, foram utilizados dois outros domínios sendo, um domínio intitulado domínio A para controle e, um domínio denominado Sistema Vestibular.

O domínio Íris foi escolhido porque é um domínio tradicionalmente utilizado para avaliações da maioria dos métodos e técnicas de AM. O sistema vestibular é um domínio novo, explorado até o momento apenas usando redes neurais (VOLPINI et al (2002)), (PALMA NETO et al (2003)); a razão para a sua escolha foi a de poder experimentar esse domínio utilizando um outro modelo de aprendizado. O domínio

Wine foi escolhido porque os valores de atributos são valores reais e, além disso, sabe-se que alguns atributos não colaboram na expressão do conceito.

A métrica usada para avaliações é, geralmente, o percentual de classificações corretas nos conjuntos de teste. Os conjuntos de dados foram divididos em grupos: conjunto de treinamento e conjunto de testes. Segue uma descrição detalhada de cada um dos domínios de dados utilizados:

- Domínio A: conjunto com 100 instâncias, com quatro atributos numéricos e duas classes associadas (classe 1 e 2). Os dados foram gerados aleatoriamente com a seguinte característica: o segundo atributo é determinante na caracterização da classe 1 e o quarto atributo na classe 2. Os atributos 1 e 3 são irrelevantes na caracterização de quaisquer das duas classes.

  O conjunto de dados não apresenta ausência de atributos e a distribuição de classe é de 50% para cada uma das classes.

- Íris: este conjunto de dados contém 150 instâncias, sendo 50 instâncias em cada uma das três classes e cada classe faz referência a um tipo de planta Íris. As classes que representam esse conjunto são denominadas: Íris Setosa, Íris Versicolor e Íris Virgínica. Cada instância possui quatro atributos, todos com valores numéricos.

  O conjunto de dados não apresenta ausência de atributos e a distribuição de classe é de 33,3% para cada uma das três classes.

- Sistema Vestibular: este conjunto de dados contém 198 instâncias, cada uma delas descrita por seis atributos (três relativos ao olho esquerdo e três relativos ao olho direito) e uma classe associada. Cada instância descreve dados de um paciente e foram obtidas por meio do teste sacádico fixo realizado pela equipe do Prof. Dr. José F. Colafemina do Serviço de Otoneurologia do Hospital das Clínicas da Escola de Medicina da Universidade de São Paulo em Ribeirão Preto. Movimentos sacádicos são produzidos quando o paciente tem que olhar, sem mover a cabeça, para um ponto de luz que se alterna, com uma freqüência constante, entre as extremidades de uma barra eletrônica colocada horizontalmente na frente do mesmo (VOLPINI et al (2002)). Para analisar esta situação, alguns eletrodos são posicionados próximos aos olhos (direito e esquerdo)

do paciente com o intuito de medir os potenciais elétricos produzidos pelos movimentos sacádicos. Os possíveis valores de classe são: anormal (valor 1) e normal (valor 0).

O conjunto de dados não apresenta ausência de atributos e a distribuição de classe é de 51,01% para a classe anormal e 48,99% para a classe normal.

- *Wine:* esse conjunto de dados é o resultado de uma análise química de cultivos de vinhos localizados em três regiões diferentes da Itália. A análise determina a quantidade de 13 constituintes encontrados em cada um dos vinhos de cada uma das três regiões. Apresenta um total de 178 instâncias classificadas em três classes da seguinte forma: classe 1 – 59 instâncias; classe 2 - 71 instâncias e classe 3 – 48 instâncias. Cada instância tem um total de treze atributos, todos contínuos e não apresenta ausência de nenhum atributo.

  A distribuição de classe é de 33,15% para a classe 1, 39,89% para a classe 2 e 26,96% para a classe 3.

4.2.2 Função de Avaliação e Parâmetros Genéticos Adotados

Para todos os experimentos descritos neste capítulo, foram adotados:

- para a implementação da função de avaliação de cromossomo do AG, foi utilizada a técnica de 10-validação cruzada. Portanto, a avaliação de cada cromossomo leva em consideração os valores obtidos nos dez pares de arquivos Treina_i-Teste_i, sendo i=1,...,10, como mostra a Figura 4.1.

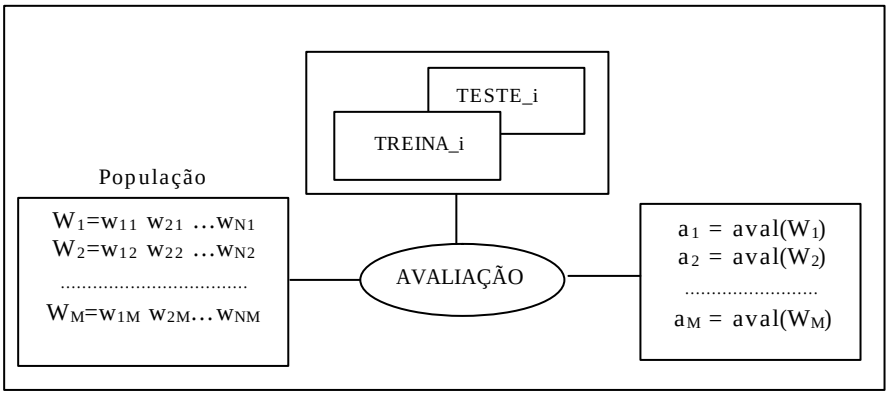

Figura 4.1 Diagrama da função de avaliação

- o critério de parada utilizado foi o número de gerações;
- o valor *default* de k foi pré-estabelecido como 5 e pode ser alterado pelo usuário.

No que segue são identificados por Experimento determinadas combinações de características genéticas que estão sendo investigadas, com o objetivo de evidenciar a combinação que favorece a determinação do "melhor" vetor de pesos.

4.2.3 Experimentos com *W5*-NN utilizando AG

Domínio de Conhecimento: Domínio A

O objetivo do experimento descrito a seguir foi apenas o de validar a geração de vetores de pesos em um domínio controlado, no caso o domínio artificial A, usando AG. Para tanto o *W5*-NN com AG foi utilizado com validação cruzada de 10 e o melhor vetor de pesos na última geração foi escolhido, como mostra a Tabela 4.1. Foram realizados quatro experimentos, variando o tamanho da população e o número de gerações. Em ambos, a taxa de cruzamento ficou fixa em 80% e taxa de mutação em 1%. A Tabela 4.1 mostra o vetor de peso obtido em cada experimento. Note que em todos os vetores, os pesos associados ao primeiro e ao terceiro atributos são bem menores que aos associados ao segundo e quarto.

Tabela 4.1 Resultados dos experimentos para o Domínio A

Tamanho da População	Número de Gerações	Melhor Vetor de Pesos da Última Geração
		(Peso_Atrib$_1$, Peso_Atrib$_2$, Peso_Atrib$_3$, Peso_Atrib$_4$)
50	20	(0.0315, 0.9599, 0.0569, 0.8997)
50	50	(0.0247, 0.9714, 0.1899, 0.8669)
100	20	(0.1089, 0.9975, 0.0591, 0.8589)
100	50	(0.1084, 0.9797, 0.2344, 0.9984)

Domínio de Conhecimento: Íris

Os experimentos numerados de 1 a 4 investigam a relevância do número de gerações na obtenção dos resultados. Para tanto, experimenta os mesmos valores genéticos descritos na Tabela 4.2, variando apenas o número de gerações.

Tabela 4.2 Características Genéticas dos Experimentos 1-4

Experimento	Tamanho da População	Método de Seleção	Taxa de Cruzamento	Taxa de Mutação	Número de Gerações
1	50	Elitismo e roleta	80%	1%	20
2					50
3					100
4					500

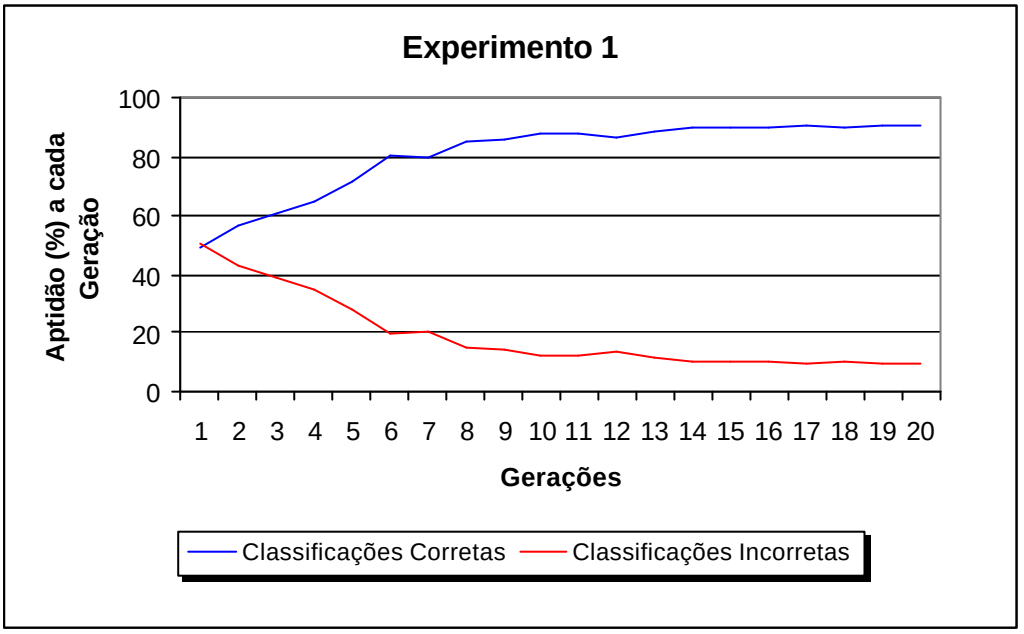

Figura 4.2 Experimento 1 – Número de Gerações = 20

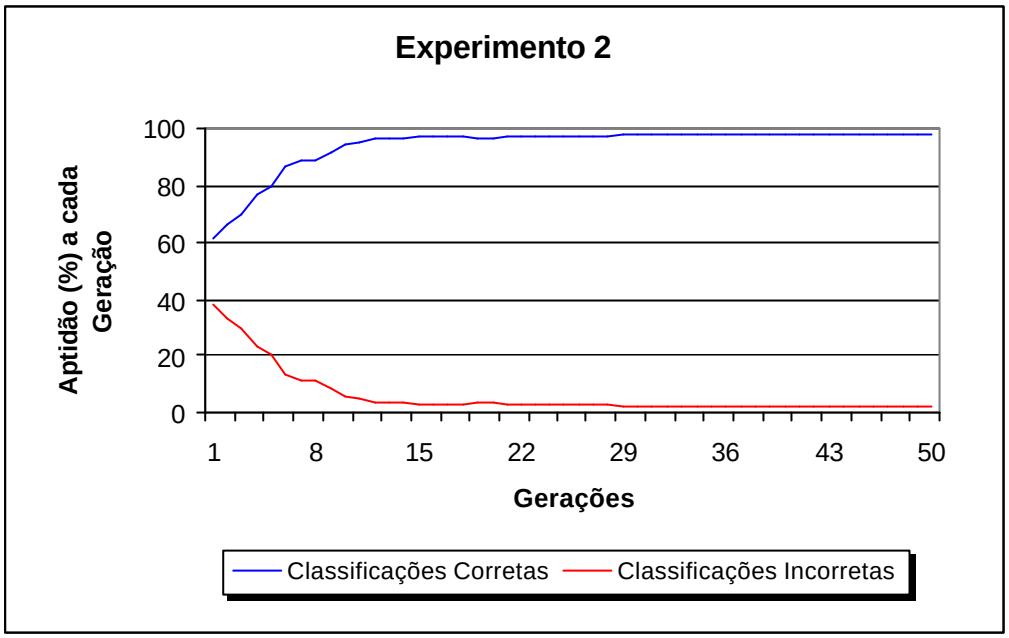

Figura 4.3 Experimento 2 – Número de Gerações = 50

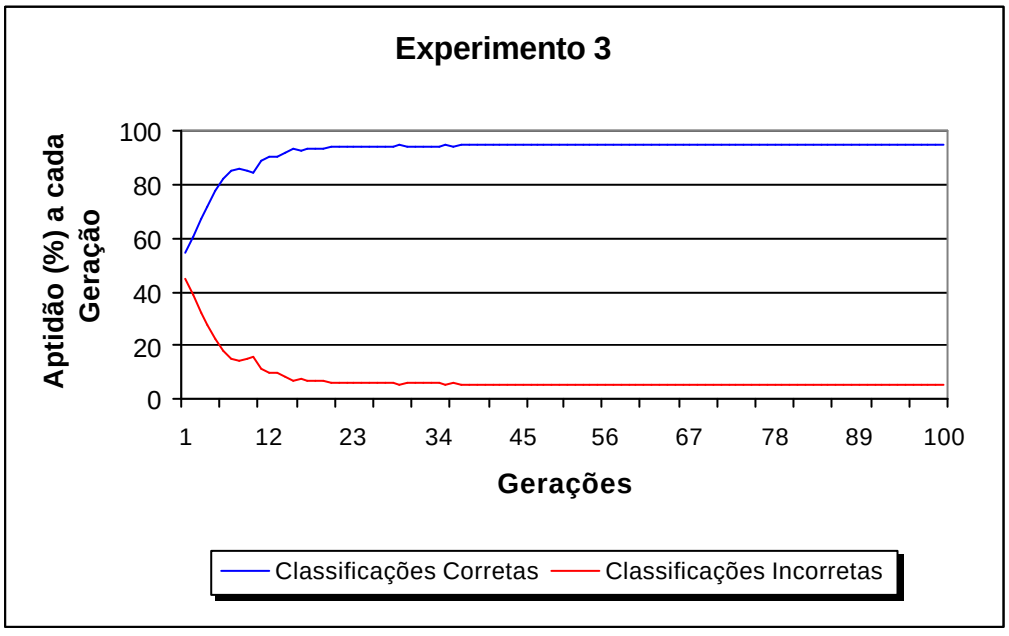

Figura 4.4 Experimento 3 – Número de Gerações = 100

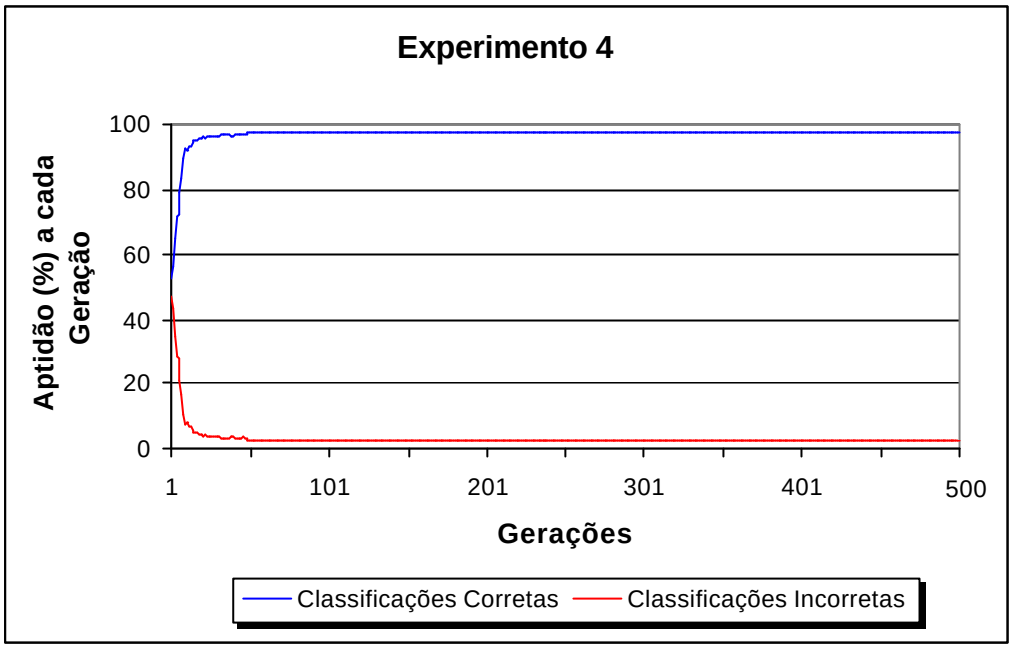

Figura 4.5 Experimento 4 – Número de Gerações = 500

Como pode ser constatado nas Figuras 4.2 - 4.5, o número de gerações não teve grande interferência nos resultados, uma vez que o processo evolutivo atinge seus melhores resultados logo nas gerações iniciais. Se o número de gerações escolhido fosse por volta de trinta, os resultados não seriam tão satisfatórios.

Os experimentos numerados de 5 a 8 investigam se um aumento na população influencia o desempenho do sistema. Para tanto, os experimentos realizados anteriormente são repetidos, para uma população de 100 indivíduos, como descreve a Tabela 4.3.

Tabela 4.3 Características Genéticas dos Experimentos 5-8

Experimento	Tamanho da População	Método de Seleção	Taxa de Cruzamento	Taxa de Mutação	Número de Gerações
5	100	Elitismo e roleta	80%	1%	20
6					50
7					100
8					500

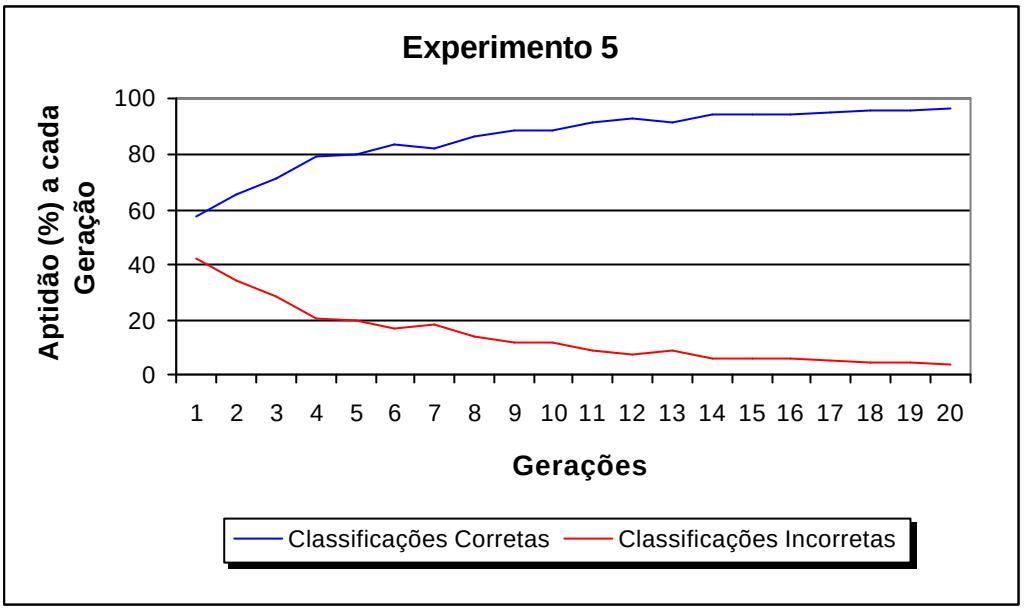

Figura 4.6 Experimento 5 – Número de Gerações = 20

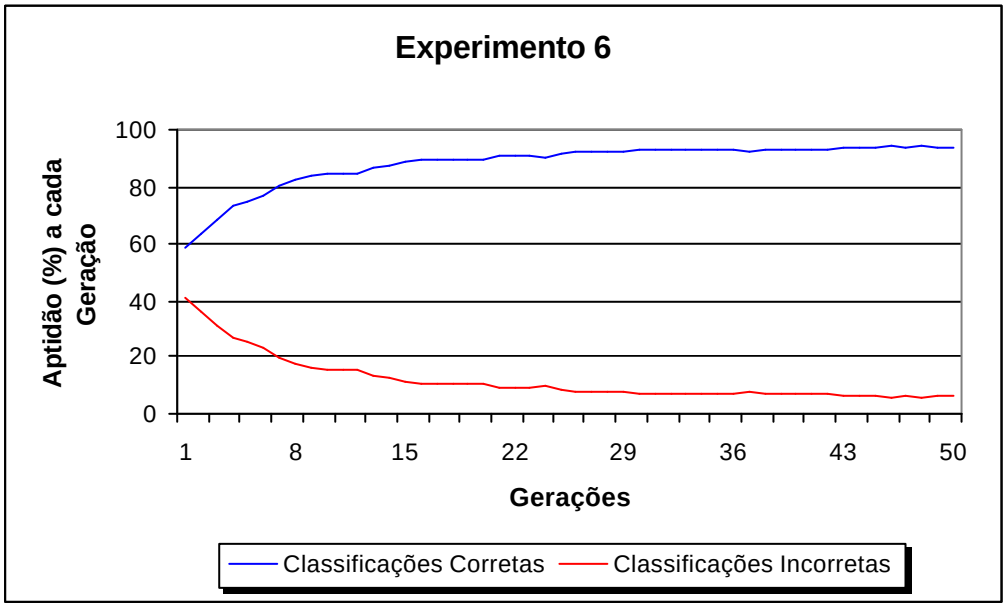

Figura 4.7 Experimento 6 – Número de Gerações = 50

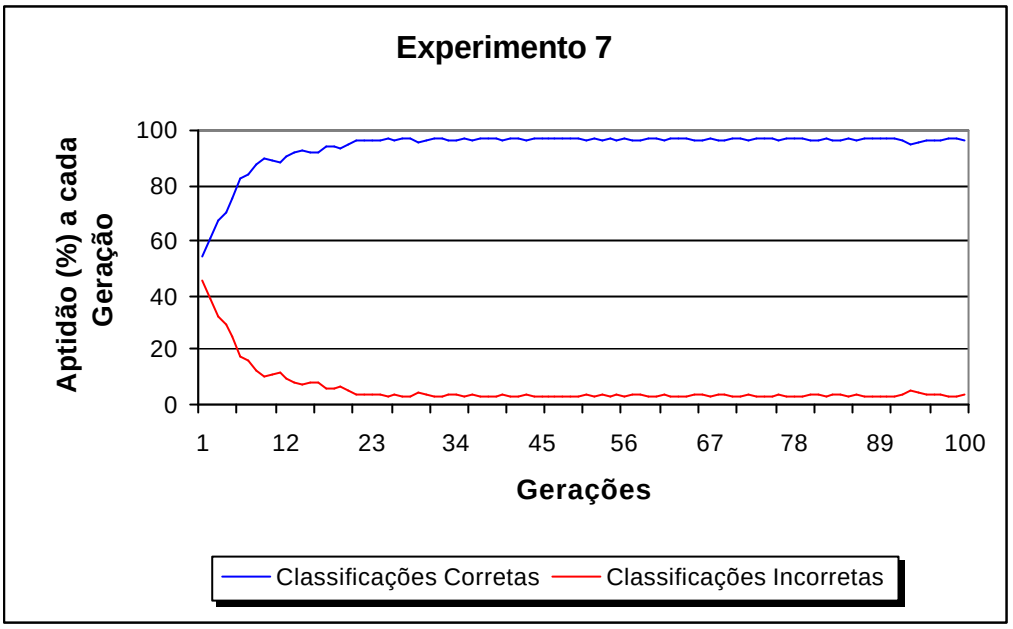

Figura 4.8 Experimento 7 – Número de Gerações = 100

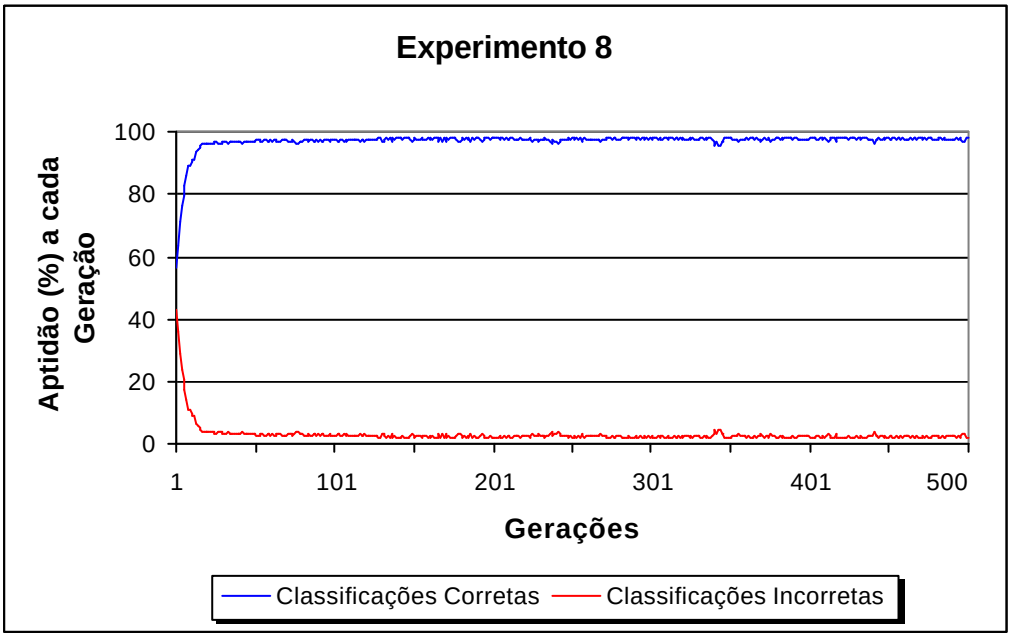

Figura 4.9 Experimento 8 – Número de Gerações = 500

Como pode ser constatado nos gráficos relativos aos experimentos de 5 a 8 (Figura 4.6 - 4.9), o aumento da população não interferiu nos resultados obtidos neste domínio; apenas o tempo de processamento sofreu um aumento considerável.

Com o objetivo de obter uma avaliação comparativa entre os métodos *5*-NN e *W5*-NN utilizando AG, a Tabela 4.4 mostra dados relativos a 10-validação cruzada.

Tabela 4.4 Valor comparativo de classificação entre *5*-NN e *W5*-NN utilizando AG

	***5*-NN**	***W5*-NN utilizando AG**
Classificações Corretas (%)	95,94	98
Desvio Padrão	0,8498364548	0,0005900055

A Tabela 4.5 descreve o melhor e o pior indivíduo, o *fitness* médio, o desvio padrão e a porcentagem de classificações corretas e incorretas feita pelo melhor indivíduo em cada um dos experimentos utilizando o domínio Íris.

Em alguns dos experimentos realizados não houve a ocorrência de mutação (Experimento 1, Experimento 2, Experimento 3 e Experimento 4), devido ao fato de sua taxa ser extremamente baixa. Foi possível perceber que quando não acontece mutação os resultados se estabilizam e o melhor vetor de pesos encontrado até então é retornado. Por outro lado, quando a mutação ocorre, notou-se que a melhor solução ao longo do processo evolutivo nem sempre é preservada. Analisando os valores dos vetores de pesos que ponderam atributos, pode-se concluir que o terceiro atributo é o mais relevante na caracterização dos conceitos e que o segundo e o quarto atributos participam de maneira equilibrada na definição dos conceitos tendo, aproximadamente, a mesma relevância. A importância do primeiro atributo é questionável, dado que no melhor resultado ele é bem ponderado e no segundo melhor, não.

Tabela 4.5 Tabela-Resumo dos resultados obtidos para o módulo *W5*-NN utilizando AG com o domínio Íris

	Melhor Indivíduo	**Pior Indivíduo**	***Fitness* Médio**	**Desvio Padrão**	**Classificações Corretas (%)**	**Classificações Incorretas (%)**
Experimento 1	(0.0049999999, 0.2773999870, 0.9858000278, 0.3889999985)	(0.0658000037, 0.6011000276, 0.9858000278, 0.3889999985)	0,9070663452	0,0025467030	90,71	9,29
Experimento 2	(0.8342999815, 0.8463000060, 0.9829000235, 0.9113000035)	(0.8342999815, 0.8463000060, 0.9829000235, 0.9113000035)	0,9800000191	0,0005900055	98	2
Experimento 3	(0.0331000015, 0.1518000066, 0.9653999805, 0.0540999994)	(0.0331000015, 0.1518000066, 0.9653999805, 0.0540999994)	0,9466666580	0	94,67	5,33
Experimento 4	(0.0171000007, 0.8269000053, 0.9703000188, 0.9416000247)	(0.0171000007, 0.8269000053, 0.9703000188, 0.9416000247)	0,9733340740	0,0000010115	97,34	2,66
Experimento 5	(0.8648999930, 0.9250000119, 0.9779999852, 0.8450000286)	(0.4587000012, 0.6610000133, 0.9779999852, 0.8450000286)	0,9615333676	0,0015602142	96,15	3,85
Experimento 6	(0.0410000011, 0.3524000049, 0.9477999806, 0.0395000018)	(0.0410000011, 0.2116000056, 0.6169999838, 0.0395000018)	0,9397327900	0,0027388886	93,97	6,03
Experimento 7	(0.9023000002, 0.8514000177, 0.9965000152, 0.7700999975)	(0.1196999997, 0.8514000177, 0.3138999939, 0.7700999975)	0,9651343822	0,0068571837	96,51	3,49
Experimento 8	(0.0249000005, 0.8590999842, 0.9933000207, 0.8072000146)	(0.2723000049, 0.8590999842, 0.9933000207, 0.8072000146)	0,9796008468	0,0003375734	97,96	2,04

Domínio de Conhecimento: Sistema Vestibular

Para este domínio foi realizado apenas o experimento com as características que apresentaram melhor desempenho para o domínio Íris. Tal escolha foi feita porque, em testes preliminares, o tamanho da população (maior que 50) e o número de gerações (maior do que 50) não foram determinantes nos resultados obtidos. A Tabela 4.6 descreve os valores dessas características e a Figura 4.21 ilustra o desempenho dessas características para o domínio Sistema Vestibular.

Tabela 4.6 Características genéticas do melhor experimento

Tamanho da População	Método de Seleção	Taxa de Cruzamento	Taxa de Mutação	Número de Gerações
50	Elitismo e roleta	80%	1%	50

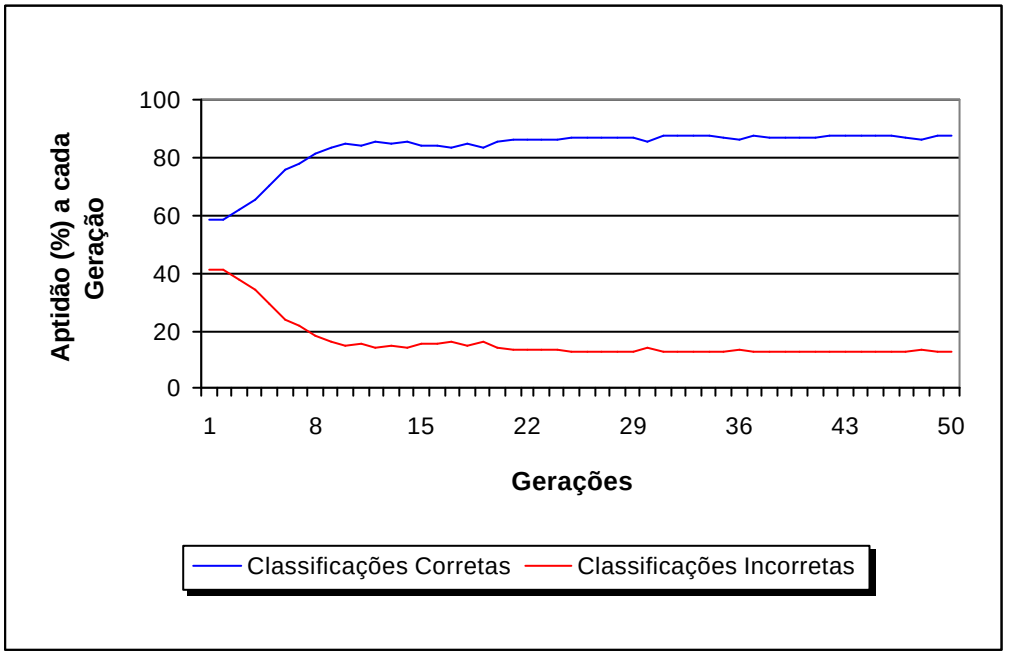

Figura 4.10 Tamanho da População = 50 e Número de Gerações = 50

A Tabela 4.7 mostra um comparativo de desempenho entre o *5*-NN e o *W5*-NN com o domínio Sistema Vestibular. Enquanto que, a Tabela 4.8 mostra o vetor de pesos encontrado na última geração e a porcentagem de classificações corretas e incorretas.

Tabela 4.7 Valor comparativo de classificação entre *5*-NN e *W5*-NN utilizando AG

	***5*-NN**	***W5*-NN utilizando AG**
Classificações Corretas (%)	87	87,35
Desvio Padrão	1,5491933384	0,0001399646

Tabela 4.8 Informações sobre o vetor de pesos na última geração (k = 5)

Vetor de pesos na última geração	**Classificações Corretas (%)**	**Classificações Incorretas (%)**
(0.1242000013, 0.0480999983, 0.5910000205, 0.4192000031, 0.1242000013, 0.7404999732)	87,35	12,65

Domínio de Conhecimento: *Wine*

Para o domínio *Wine* foi realizado o mesmo experimento do domínio Sistema Vestibular. Dados preliminares mostraram que o número de gerações (maior que 50) e o tamanho da população (maior que 50) não interferiam nos resultados finais obtidos, de forma relevante. A Tabela 4.9 descreve a configuração genética deste experimento e a Figura 4.11 mostra os resultados obtidos.

Tabela 4.9 Características genéticas do melhor experimento

Tamanho da População	**Método de Seleção**	**Taxa de Cruzamento**	**Taxa de Mutação**	**Número de Gerações**
50	Elitismo e roleta	80%	1%	50

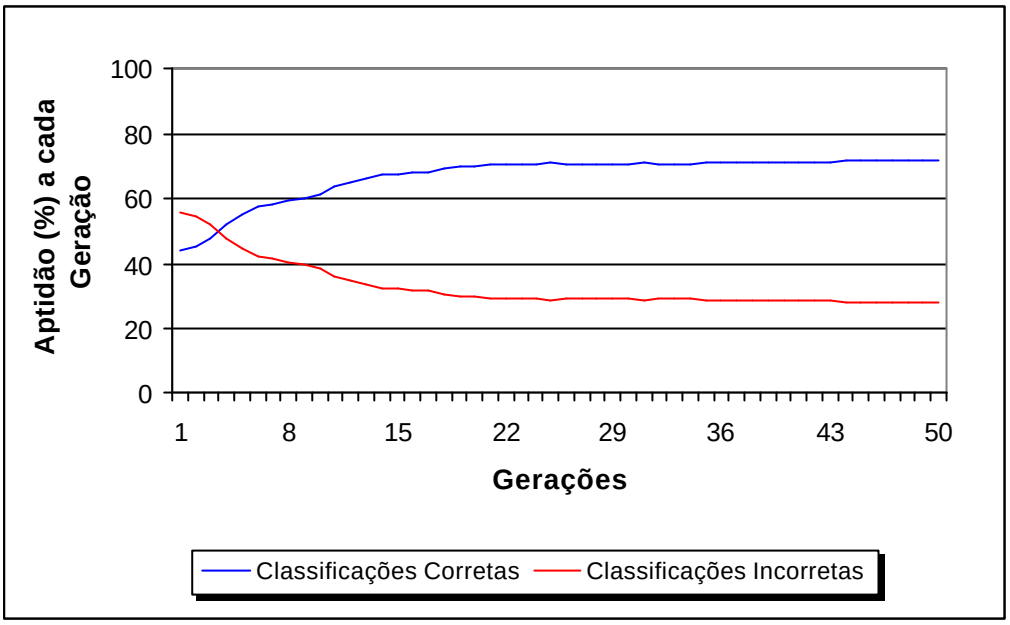

Figura 4.11 Tamanho da População = 50 e Número de Gerações = 50

A Tabela 4.10 mostra dados relativos ao desempenho do *5*-NN e *W5*-NN utilizando AG, relativos a 10-validação cruzada. A Tabela 4.11 mostra o vetor de pesos obtido na última geração e a porcentagem de classificações corretas e incorretas.

Tabela 4.10 Valor comparativo de classificação entre *5*-NN e *W5*-NN utilizando AG

	5-NN	***W5-NN* utilizando AG**
Classificações Corretas (%)	78,88	71,77
Desvio Padrão	1,3165610506	0,0032039615

Tabela 4.11 Informações sobre o vetor de pesos na última geração (k = 5)

Vetor de pesos na última geração	**Classificações Corretas (%)**	**Classificações Incorretas (%)**
(0.8213999867, 0.6686999798, 0.7336000204, 0.4837999940, 0.9865000248, 0.8235999942, 0.9053999781, 0.0502000004, 0.6205000281, 0.8295000195, 0.5246000290, 0.0544999987, 0.9545000195)	71,77	28,23

Como pode ser evidenciado nas Tabelas 4.4, 4.7 e 4.10 onde são feitas as comparações entre o *5*-NN e o *W5-NN* usando AGs, o único domínio onde a ponderação de atributos teve um desempenho inferior com relação ao algoritmo onde os atributos são não ponderados foi no domínio *Wine*. Como o número de atributos envolvidos na definição do domínio é razoável, a análise dos valores obtidos é praticamente impossível.

Devido a esses resultados, com o objetivo de explorar um pouco mais o uso do HIGEBI neste domínio, decidiu-se por uma eliminação dos atributos menos relevantes de maneira a tornar esse domínio mais facilmente "interpretável". De acordo com os dados dos experimentos utilizando um sistema de aprendizado simbólico (CN2 - CLARK & NIBLETT) descritos em RAMER & NICOLETTI (2000), o quinto, sexto, oitavo e nono atributos têm pouca participação na caracterização das três classes, de acordo com métodos de aprendizado simbólico (CN2). Por essa razão, o domínio foi reduzido a nove atributos e o resultado do experimento com esse domínio reduzido está mostrado na Figura 4.12. Na referência citada (RAMER & NICOLETTI (2000)) é comentado que, com base nos resultados obtidos usando o CN2, o segundo, terceiro e quarto atributos não são relevantes para a representação da classe 3. A Tabela 4.12 mostra o vetor de pesos encontrado na última geração e sua porcentagem de classificações corretas e incorretas.

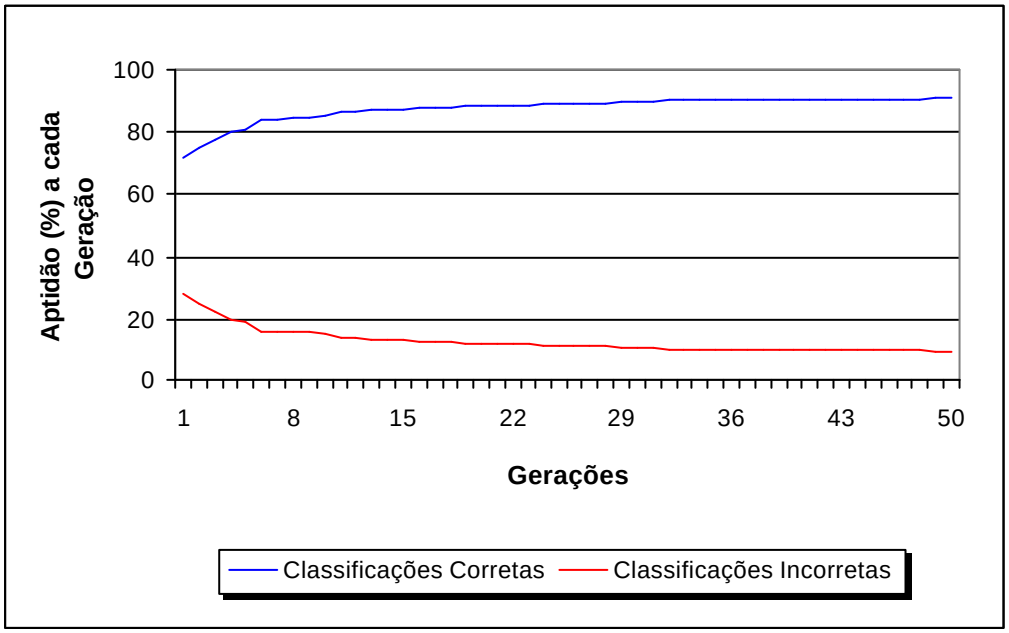

Figura 4.12 Tamanho da População = 50 e Número de Gerações = 50

Tabela 4.12 Informações sobre o vetor de pesos na última geração (k = 5)

Vetor de pesos na última geração	Classificações Corretas (%)	Classificações Incorretas (%)
(0.0943000018, 0.9247000217, 0.9218000173, 0.4498000144, 0.6529999971, 0.7454000115, 0.9848999977, 0.6236000061, 0.9922000169)	90,35	9,65

É possível perceber que, com a eliminação dos três atributos proposta por RAMER & NICOLETTI (2000), o desempenho do algoritmo *W5*-NN usando AG melhorou significativamente, quando comparadas as Tabelas 4.11 e 4.12. Quando comparado ao *5*-NN é possível perceber uma pequena melhora como mostra a Tabela 4.13.

Tabela 4.13 Valor comparativo de classificação entre *5*-NN e *W5*-NN utilizando AG

	5-NN	*W5*-NN utilizando AG
Classificações Corretas (%)	90	90,35
Desvio Padrão	1,2292725491	0,0012612338

4.2.4 Experimentos para o *W*-IB1 utilizando AG

Domínio de Conhecimento: Íris

Nos experimentos numerados de 1 a 4 foram utilizadas as mesmas características genéticas dos experimentos com o domínio Íris descritas na Tabela 4.14. As Figuras 4.13 – 4.16 mostram o ilustram o resultado gráfico destes experimentos.

Tabela 4.14 Características genéticas dos Experimentos 1-4

Experimento	Tamanho da População	Método de Seleção	Taxa de Cruzamento	Taxa de Mutação	Número de Gerações
1	50	Elitismo e roleta	80%	1%	20
2					50
3					100
4					500

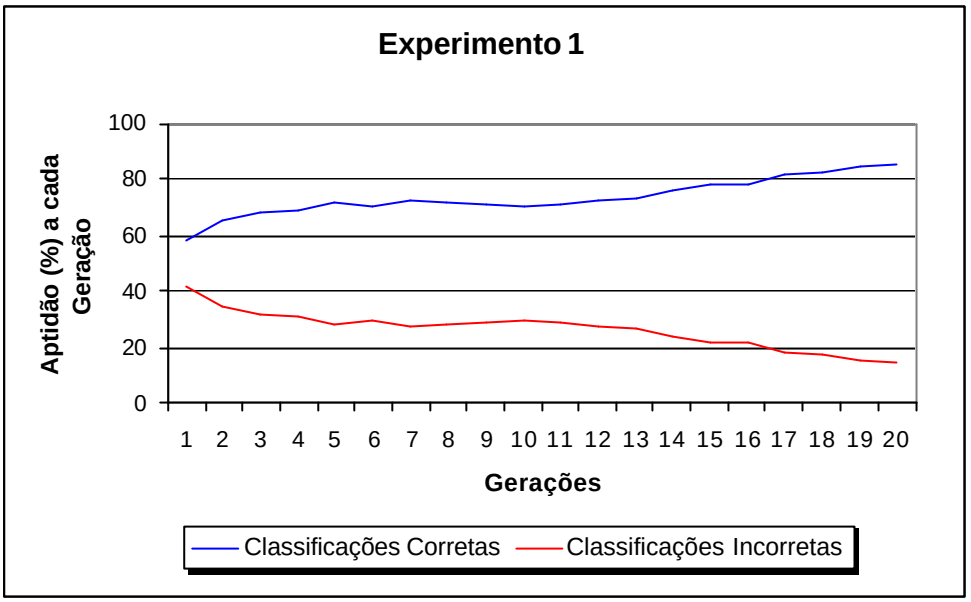

Figura 4.13 Experimento 1 – Número de Gerações = 20

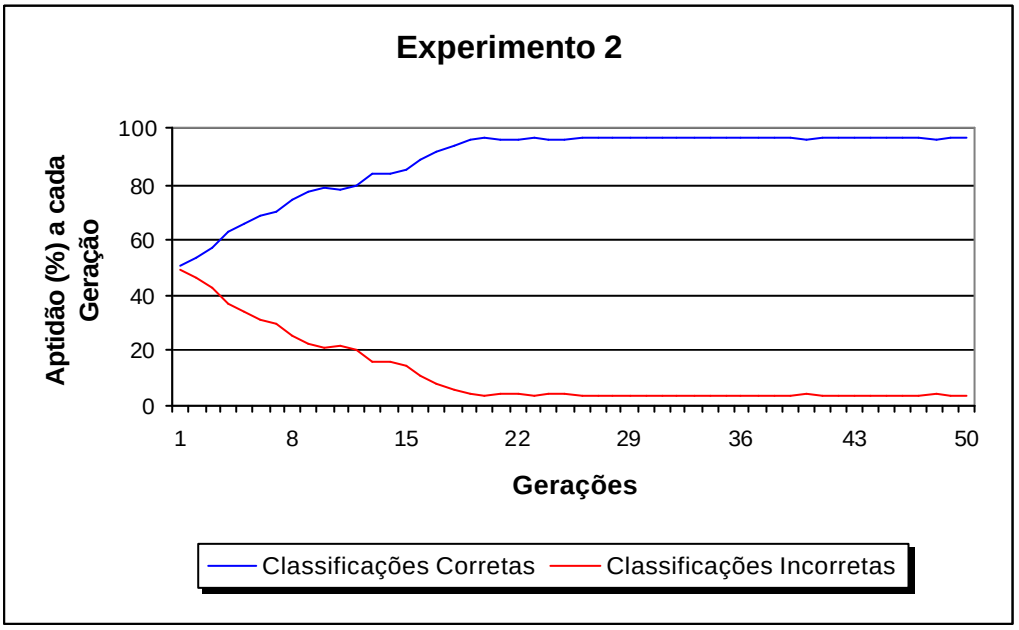

Figura 4.14 Experimento 2 – Número de Gerações = 50

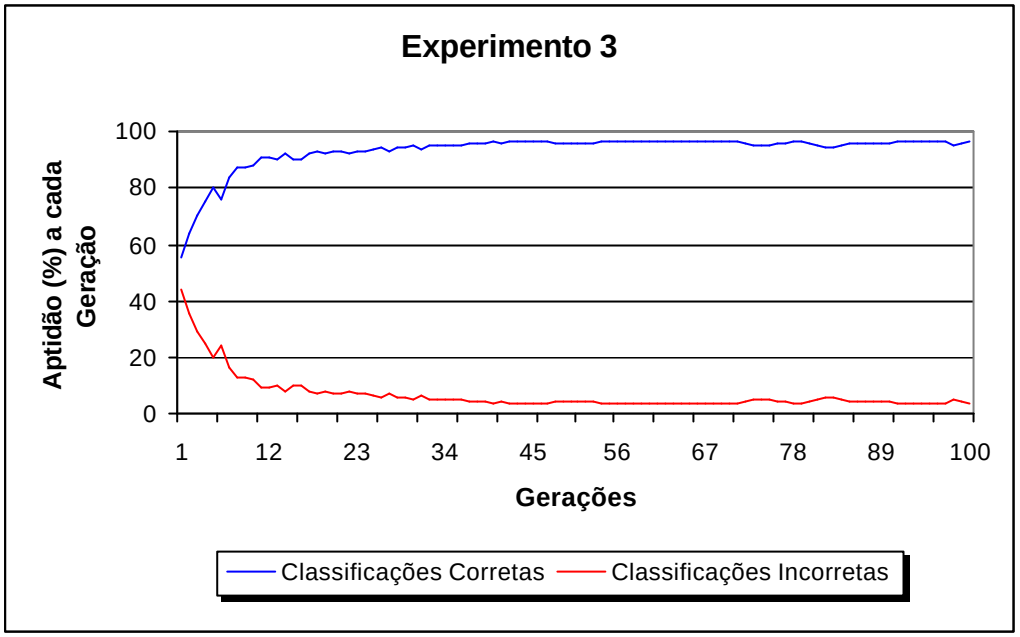

Figura 4.15 Experimento 3 – Número de Gerações = 100

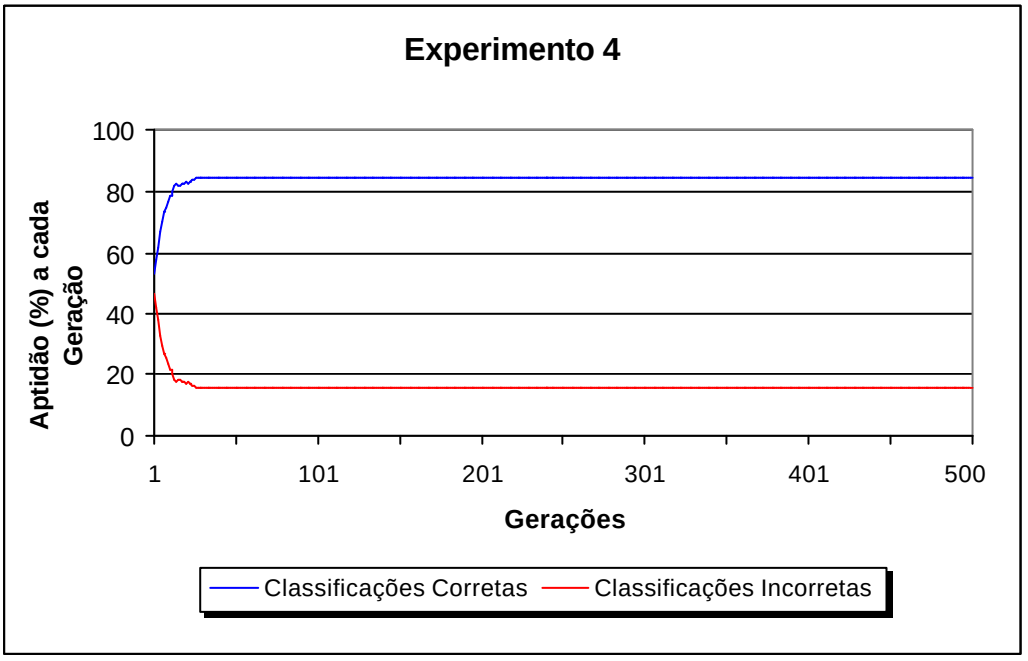

Figura 4.16 Experimento 4 – Número de Gerações = 500

É possível perceber nas Figuras 4.13 a 4.16 que o número de gerações não interfere muito nos resultados. Se o número de gerações for muito pequeno, entretanto, os resultados não são satisfatórios.

Os experimentos numerados de 5 a 8 investigam se um aumento na população influencia o desempenho do sistema. Para tanto, os experimentos realizados anteriormente são repetidos, para uma população de 100 indivíduos, como descreve a Tabela 4.15 e como ilustram as Figuras 4.17 a 4.20.

Tabela 4.15 Características Genéticas dos Experimentos 5-8

Experimento	Tamanho da População	Método de Seleção	Taxa de Cruzamento	Taxa de Mutação	Número de Gerações
5	100	Elitismo e roleta	80%	1%	20
6					50
7					100
8					500

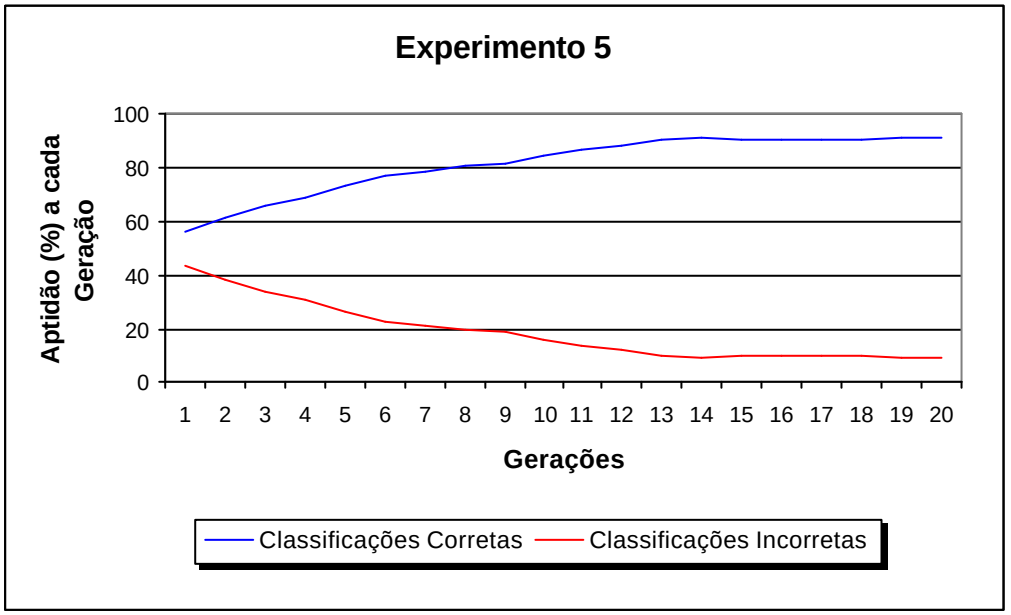

Figura 4.17 Experimento 5 – Número de Gerações = 20

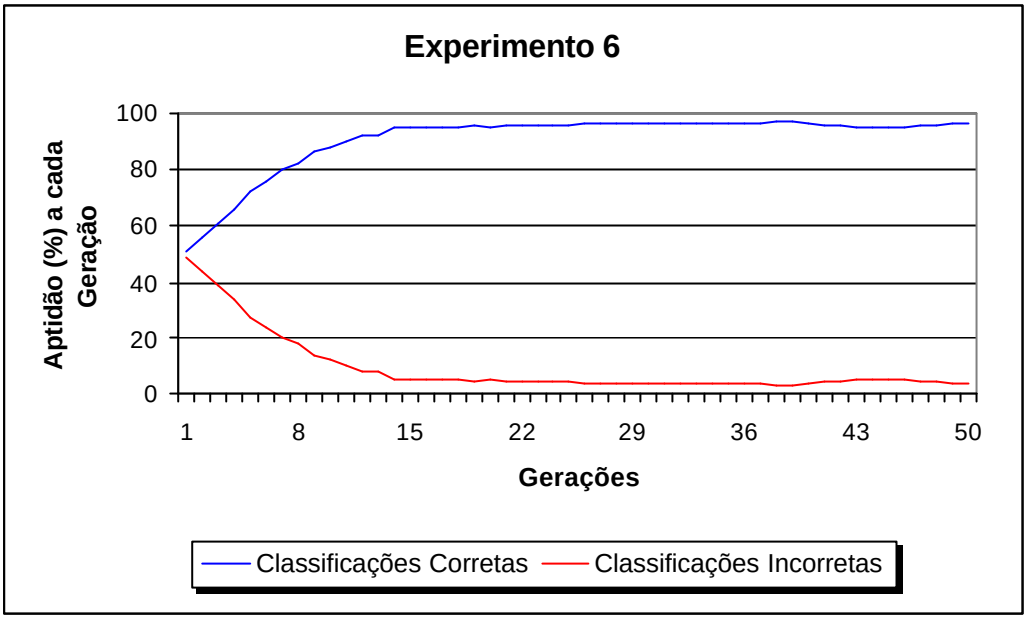

Figura 4.18 Experimento 6 – Número de Gerações = 50

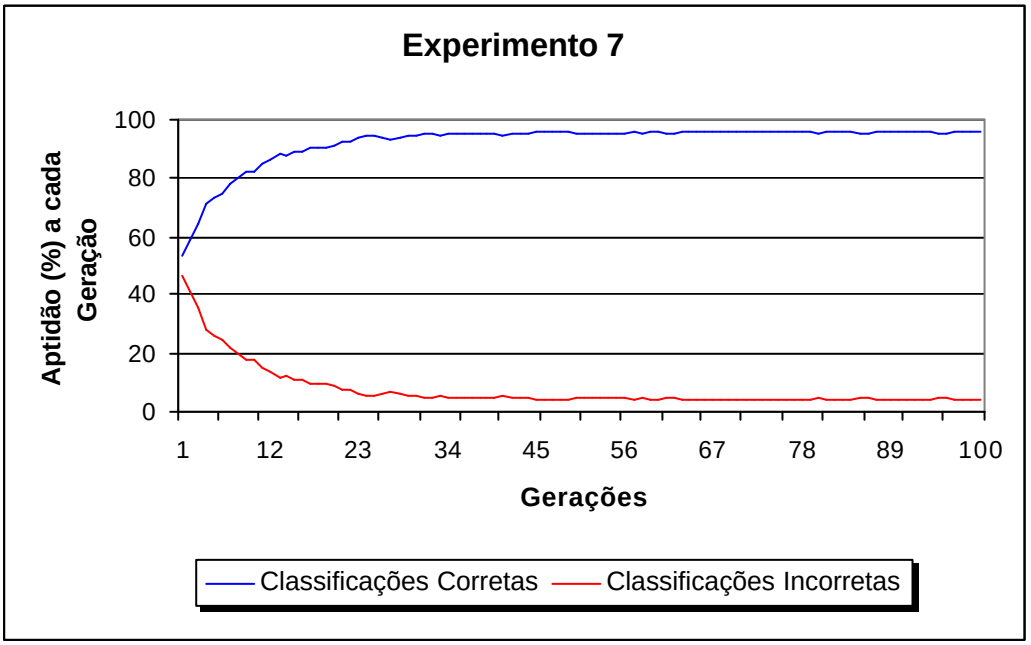

Figura 4.19 Experimento 7 – Número de Gerações = 100

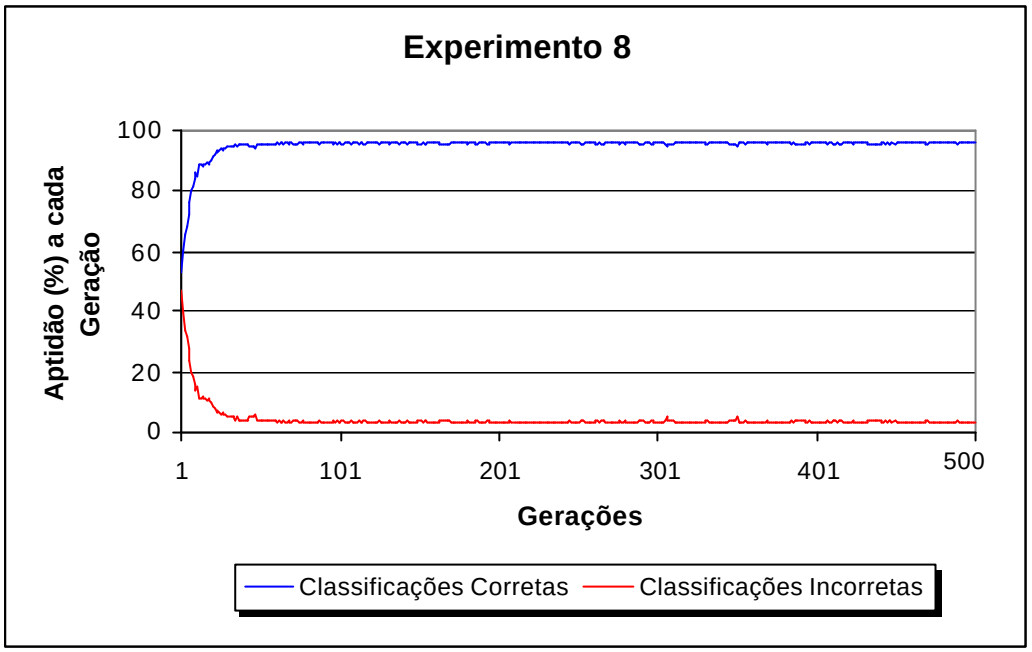

Figura 4.20 Experimento 8 – Número de Gerações = 500

Os resultados mostram que um aumento na população não interfere numa melhora significativa dos resultados obtidos.

Analisando os valores dos vetores de pesos que ponderam atributos, pode-se concluir que o primeiro e terceiro atributos são os que, aparentemente, contribuem mais para a caracterização das classes neste domínio, usando o IB1. Isso, entretanto, não é categórico, dado que o IB1 "puro" tem um desempenho próximo do IB1 ponderado usando AG, como pode ser constatado na Tabela 4.16. O melhor resultado é obtido por um vetor de pesos que não enfatiza muito a participação do atributo 2, como já foi identificado anteriormente.

Tabela 4.16 Valor comparativo de classificação entre IB1 e *W*-IB1 utilizando AG (k = 5)

	IB1	***W*-IB1 utilizando AG**
Classificações Corretas (%)	94,67	96,99
Desvio Padrão	0,7888104968	0,0030514008

A Tabela 4.17 mostra o melhor e o pior vetor de pesos encontrado em cada um dos experimentos, a porcentagem de classificações corretas e incorretas do melhor indivíduo, bem como o *fitness* médio e o desvio padrão na população.

Tabela 4.17 Tabela-Resumo dos resultados obtidos com o domínio Íris para o módulo *W*-IB1 utilizando AG

	Melhor Indivíduo	**Pior Indivíduo**	***Fitness* Médio**	**Desvio Padrão**	**Classificações Corretas (%)**	**Classificações Incorretas (%)**
Experimento 1	(0.9663000106, 0.7972999811, 0.0098000001, 0.8291000127)	(0.0841000005, 0.7972999811, 0.0098000001, 0.8291000127)	0,8563998341	0,0102575039	85,64	14,36
Experimento 2	(0.9570000171, 0.8550999760, 0.9645000100, 0.8640999794)	(0.0100999996, 0.8550999760, 0.9645000100, 0.0729999989)	0,9653335809	0,0013199320	96,53	3,47
Experimento 3	(0.9666666388, 0.1996999979, 0.9695000052, 0.7322000265)	(0.9666666388, 0.1996999979, 0.9695000052, 0.0423999987)	0,9656001925	0,0006892175	96,56	3,44
Experimento 4	(0.8550999760, 0.0348000004, 0.0076499998, 0.6550999879)	(0.0939000025, 0.4049000144, 0.5066999793, 0.3287000060)	0,8466668725	0,0185943506	84,67	15,33
Experimento 5	(0.1770000010, 0.2545999884, 0.9891999959, 0.5756000280)	(0.3801000118, 0.2545999884, 0.9891999959, 0.5756000280)	0,9089995622	0,0010214387	90,9	9,1
Experimento 6	(0.9386000037, 0.8274999856, 0.9293000102, 0.9215999841)	(0.9386000037, 0.6765000224, 0.1850000023, 0.9215999841)	0,9628003835	0,0030813852	96,28	3,72
Experimento 7	(0.9868999719, 0.5313000082, 0.9758999943, 0.9172000288)	(0.9868999719, 0.5313000082, 0.4045000076 0.1080999970)	0,9788008928	0,0030514008	96,99	3,01
Experimento 8	(0.9977999925, 0.1342999935, 0.9599999785, 0.8786000013)	(0.3923999965, 0.1641000062, 0.8927000164, 0.1342999935)	0,9699345231	0,0004620733	96,65	3,35

Os experimentos realizados para os domínios Sistema Vestibular e *Wine* para o *W*-IB1 seguiram o mesmo padrão adotado na seção anterior, ou seja, experimentar apenas a configuração que atingiu melhor resultado para o *W*-IB1 utilizando AG com o domínio Iris. A justificativa para tal escolha é a mesma citada anteriormente: dados preliminares com um número variado de gerações e com um número variado de tamanho de população não produziram mudanças significativas nos resultados obtidos. As características desse experimento estão listadas na Tabela 4.18.

Tabela 4.18 Características genéticas do melhor experimento

Tamanho da População	Método de Seleção	Taxa de Cruzamento	Taxa de Mutação	Número de Gerações
100	Elitismo e roleta	80%	1%	100

Domínio de Conhecimento: Sistema Vestibular

A Figura 4.21 mostra o resultado deste experimento sob o domínio de conhecimento Sistema Vestibular.

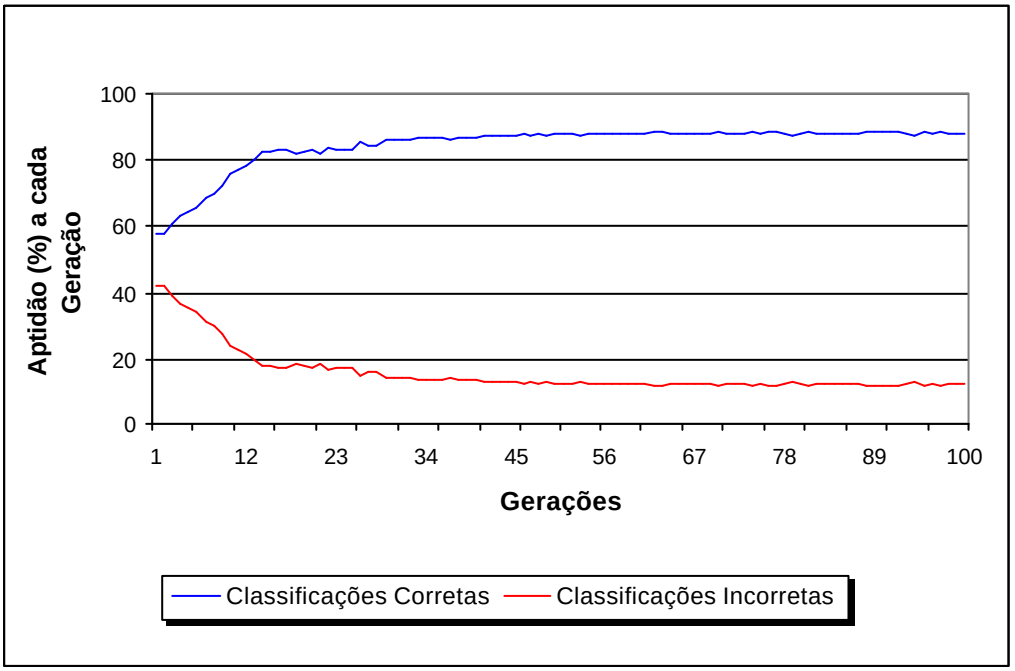

Figura 4.21 Tamanho da População = 100 e Número de Gerações = 100

A Tabela 4.19 mostra dados comparativos de desempenho entre o IB1 e o *W*-IB1 no domínio Sistema Vestibular.

Tabela 4.19 Valor comparativo de classificação entre IB1 e *W*-IB1 utilizando AG (k = 5)

	IB1	***W*-IB1 utilizando AG**
Classificações Corretas (%)	87,87	88,03
Desvio Padrão	1,5491933384	0,0034714990

A Tabela 4.20 mostra o vetor de pesos encontrado na última geração e a porcentagem de classificações corretas e incorretas que este obteve.

Tabela 4.20 Informações sobre o vetor de pesos na última geração (k = 5)

Vetor de pesos na última geração	**Classificações Corretas (%)**	**Classificações Incorretas (%)**
(0.1458999961, 0.0337000004, 0.4684999883, 0.4379000067, 0.1791000068, 0.9987000226)	88,03	11,97

Domínio de Conhecimento: *Wine*

A Figura 4.22 mostra o resultado do experimento com as configurações genéticas descritas na Tabela 4.18 para o domínio *Wine*.

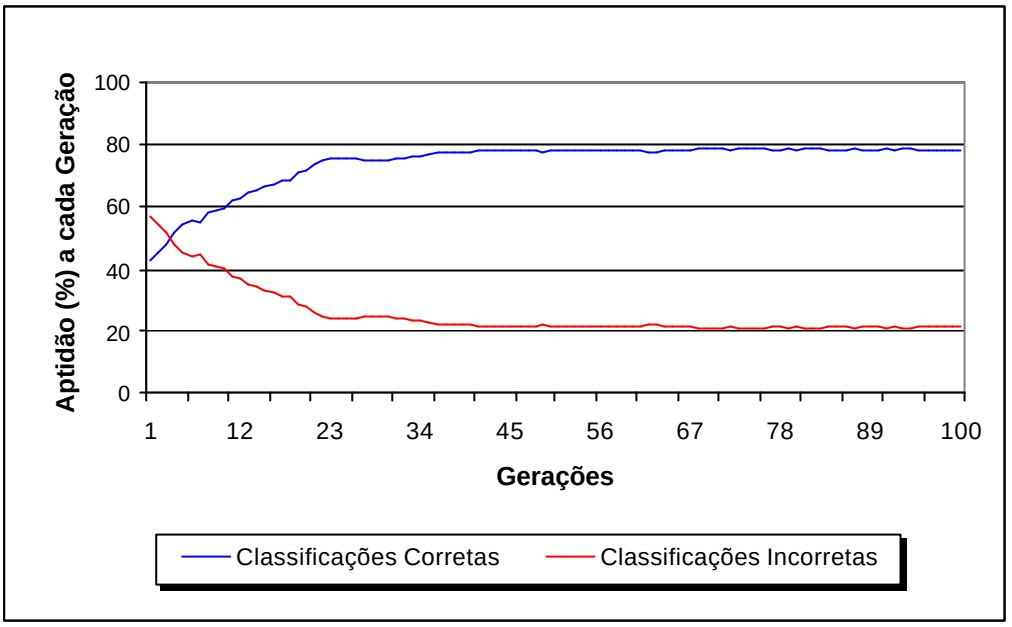

Figura 4.22 Tamanho da População = 100 e Número de Gerações = 100

A Tabela 4.21 mostra um comparativo de desempenho entre o IB1 e o *W*-IB1 sob o domínio *Wine*.

Tabela 4.21 Valor comparativo de classificação entre IB1 e *W*-IB1 utilizando AG (k = 5)

	IB1	***W*-IB1 utilizando AG**
Classificações Corretas (%)	89	78,40
Desvio Padrão	1,5491933384	0,0014417954

A Tabela 4.22 mostra o vetor de pesos encontrado na última geração e a porcentagem de classificações corretas e incorretas que este obteve.

Tabela 4.22 Informações sobre o vetor de pesos na última geração (k = 5)

Vetor de pesos na última geração	**Classificações Corretas (%)**	**Classificações Incorretas (%)**
(0.8306000232, 0.6511999964, 0.6603999733, 0.5088000297, 0.9438999891, 0.8646000027, 0.9894999861, 0.2919000089, 0.5978000164, 0.9656999707, 0.6488999724, 0.1990000009, 0.9973999857)	78,04	21,60

Como descrito na seção 4.2.3 é possível perceber que o desempenho para o *W*-IB1 com o domínio *Wine* também foi inferior do que para o IB1 "puro". Dessa forma, utilizando a sugestão proposta nos experimentos com o CN2 (CLARK & NIBLETT) e descrita em RAMER & NICOLETTI (2000), o quinto, sexto, oitavo e nono atributos serão removidos. A Figura 4.23 mostra o resultado deste experimento e a Tabela 4.23 mostra o vetor de pesos encontrado na última geração e a porcentagem de classificações corretas e incorretas.

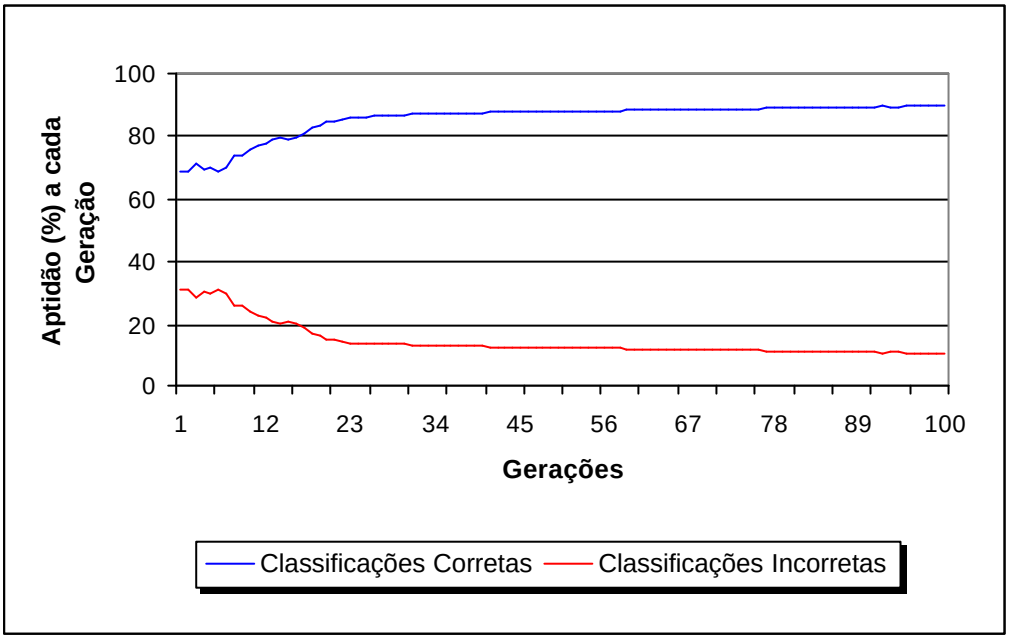

Figura 4.23 Tamanho da População = 100 e Número de Gerações = 100

Tabela 4.23 Informações sobre o vetor de pesos na última geração (k = 5)

Vetor de pesos na última geração	Classificações Corretas (%)	Classificações Incorretas (%)
(0.0936999991, 0.9650999903, 0.8862000107, 0.4503999948, 0.7351999878, 0.7427999973, 0.9751999974, 0.3609000146, 0.9650999903)	90,05	9,95

Estabelecendo a mesma comparação da seção 4.2.3 é possível perceber que o *W*-IB1 apresentou resultado bem similar ao IB1, como mostra a Tabela 4.24.

Tabela 4.24 Valor comparativo de classificação entre IB1 e *W*-IB1 utilizando AG (*k* = 5)

	IB1	***W*-IB1 utilizando AG**
Classificações Corretas (%)	90	90,05
Desvio Padrão	1,2292722549	0,0006985067

4.2.5 Experimentos para o *W*-IB2 utilizando AG

As características genéticas utilizadas nos experimentos descritos nesta seção são as mesmas descritas anteriormente na Seção 4.2.3. Dessa forma, serão demonstradas nas próximas subseções, os resultados dos experimentos em cada domínio e as conclusões possíveis com relação aos resultados obtidos.

Domínio de Conhecimento: Íris

As Figuras 4.24 a 4.31 apresentam os resultados dos oito experimentos realizados para o domínio Íris utilizando o algoritmo *W*-IB2 cujas configurações genéticas estão descritas na Tabela 4.25.

Tabela 4.25 Características genéticas dos Experimentos 1-8

Experimento	**Tamanho da População**	**Método de Seleção**	**Taxa de Cruzamento**	**Taxa de Mutação**	**Número de Gerações**
1	50	Elitismo e roleta	80%	1%	20
2					50
3					100
4					500
5	100				20
6					50
7					100
8					500

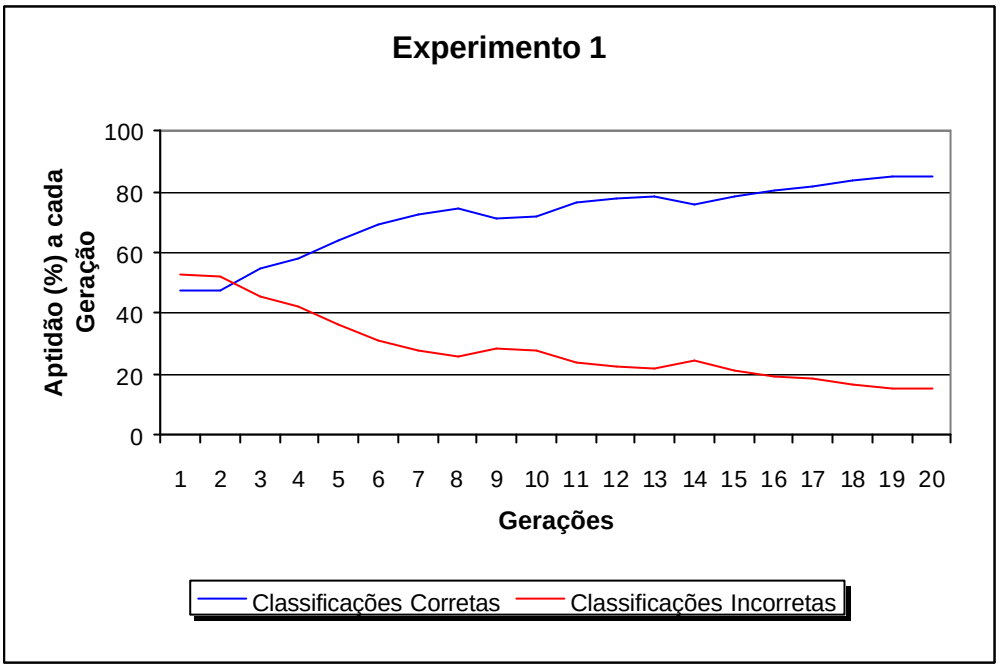

Figura 4.24 Experimento 1 – Número de Gerações = 20

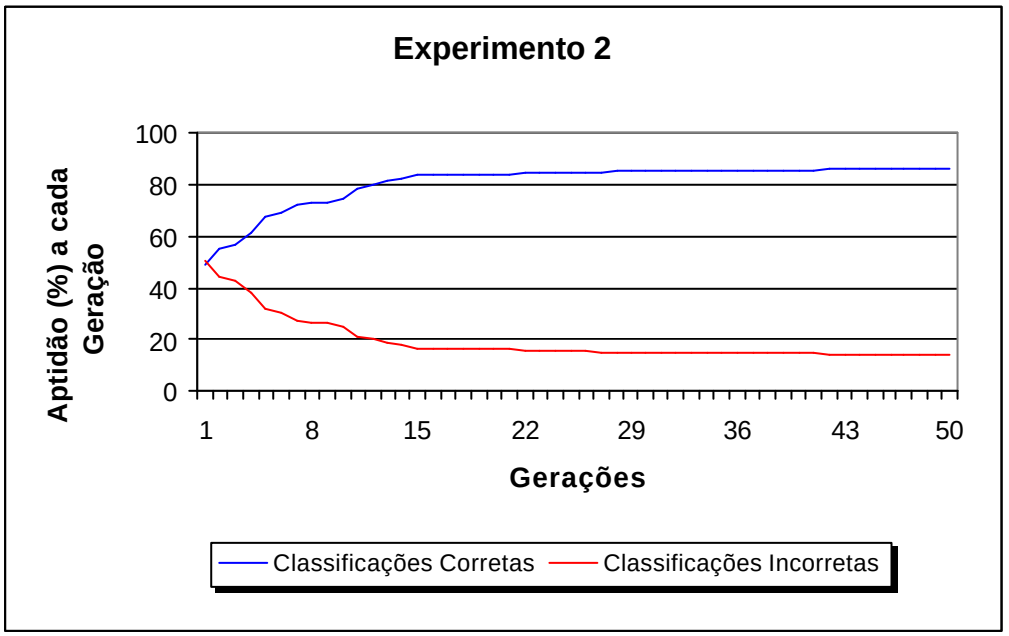

Figura 4.25 Experimento 2 – Número de Gerações = 50

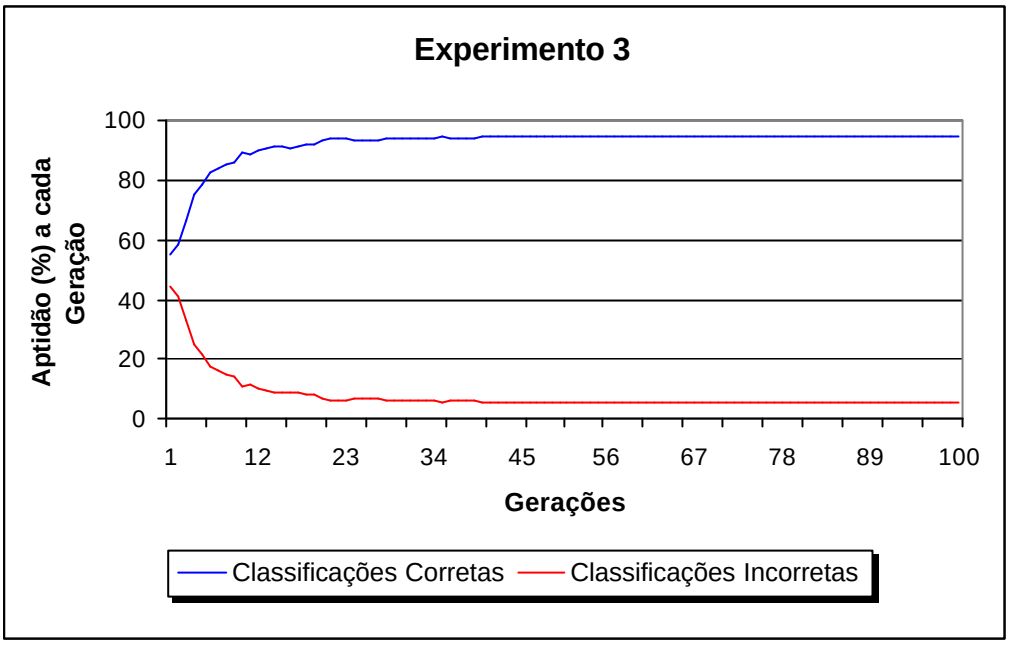

Figura 4.26 Experimento 3 – Número de Gerações = 100

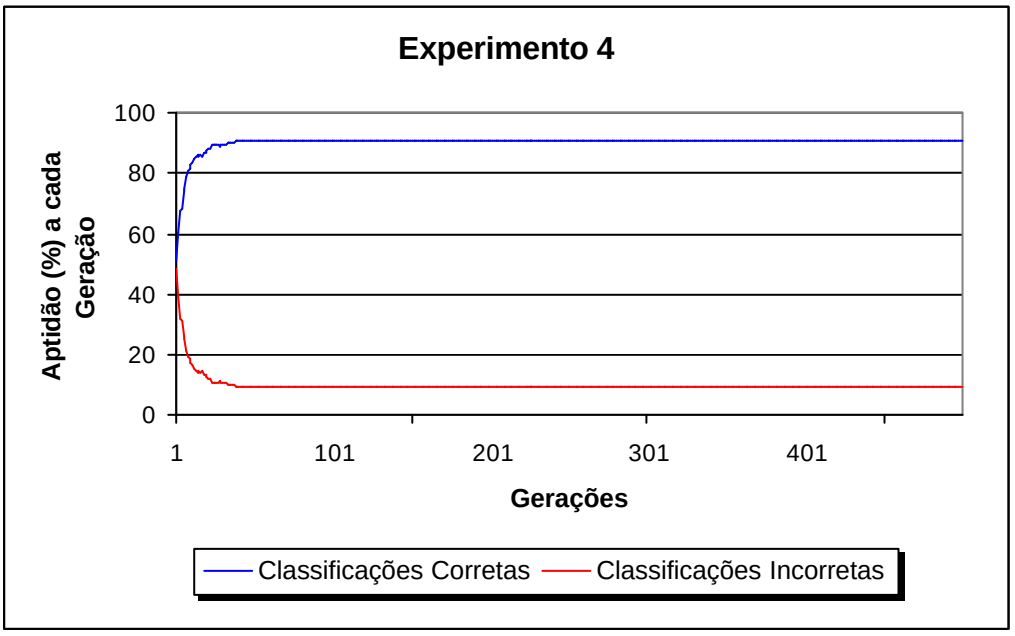

Figura 4.27 Experimento 4 – Número de Gerações = 500

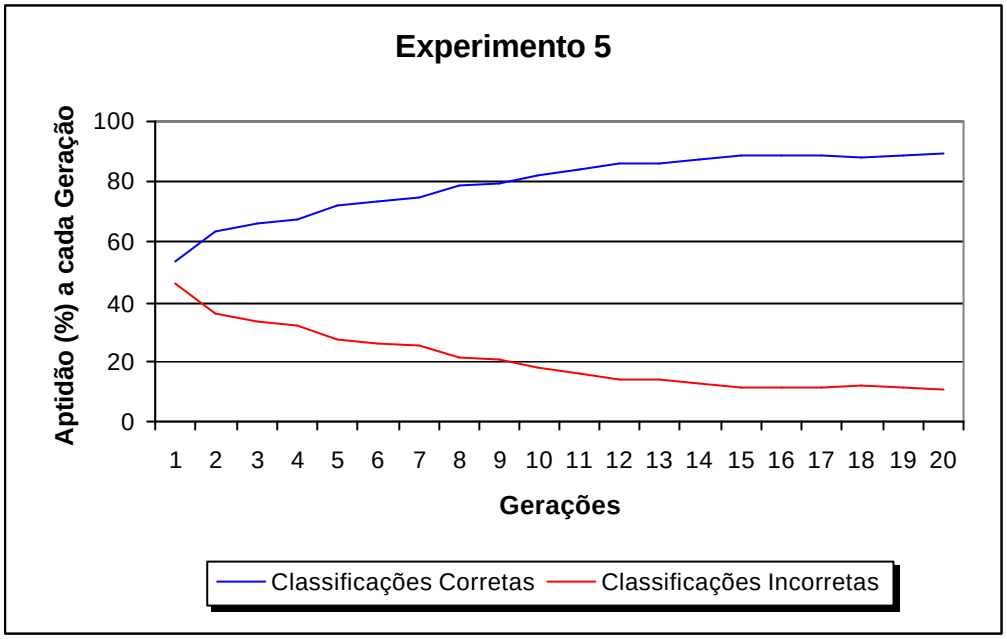

Figura 4.28 Experimento 5 – Número de Gerações = 20

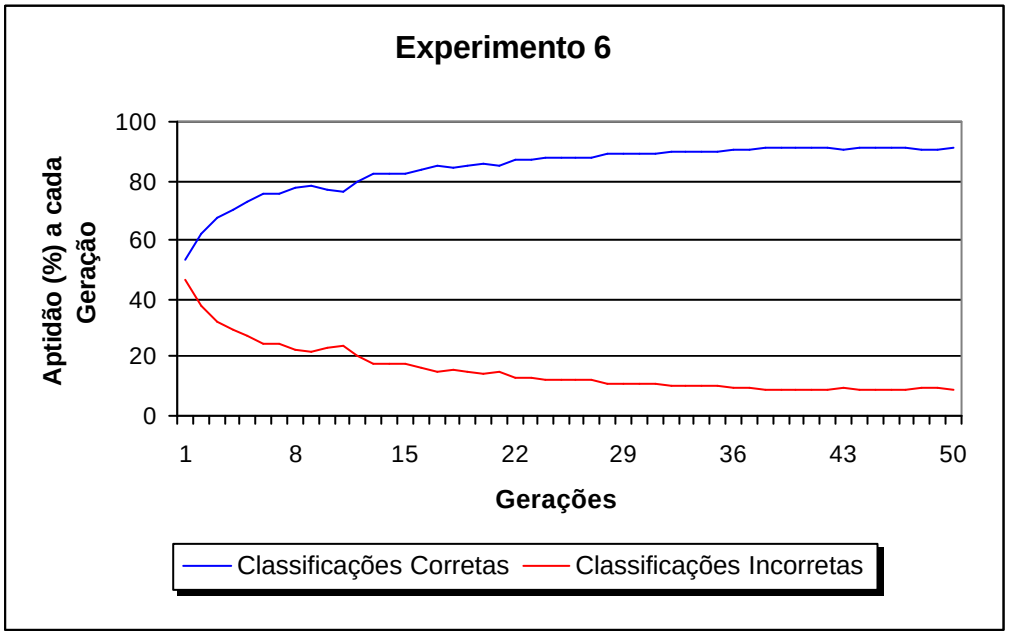

Figura 4.29 Experimento 6 – Número de Gerações = 50

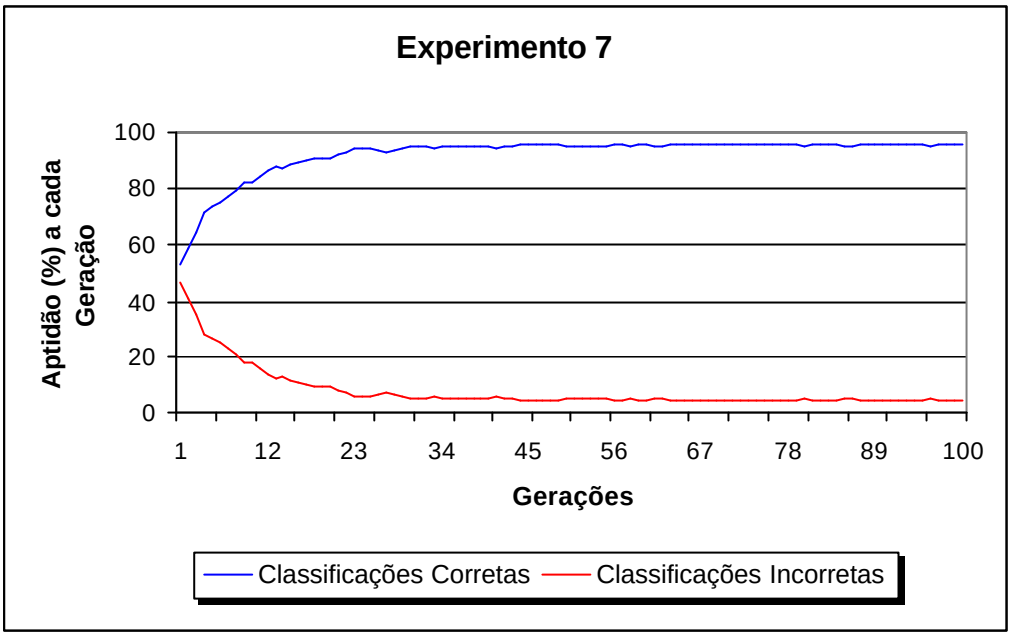

Figura 4.30 Experimento 7 – Número de Gerações = 100

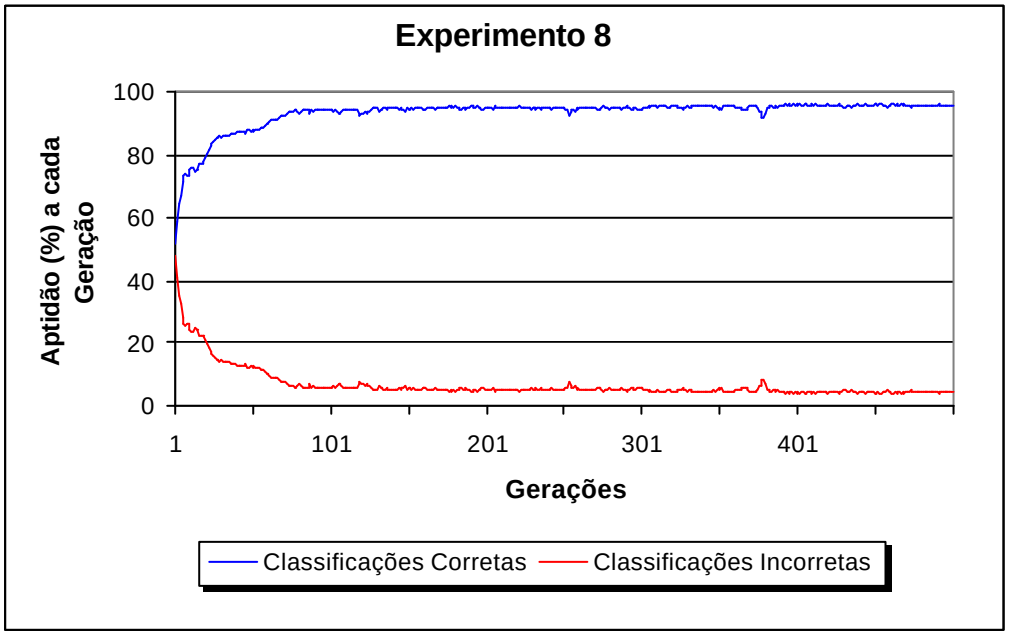

Figura 4.31 Experimento 8 – Número de Gerações = 500

A Tabela 4.26 sumariza os resultados destes experimentos e exibe o melhor e o pior vetor de pesos, o *fitness* médio e o desvio padrão da população, além do número de classificações corretas e incorretas para o melhor indivíduo. Analisando esta tabela é possível perceber, que de acordo com os valores dos vetores de pesos que ponderam atributos, o melhor resultado foi obtido por aquele vetor de pesos cujos valores estão todos bastante próximos de 1. Quando os seguintes melhores resultados são analisados, pode ser evidenciado que o peso associado ao terceiro atributo sempre se mantém alto. Vale lembrar que a estratégia utilizada pelo IB2 difere completamente daquela usada pelo IB1, dado que, durante o aprendizado, o IB2 armazena como parte do conceito apenas aquelas instâncias que fizeram classificação incorreta.

A Tabela 4.27 mostra dados comparativos de desempenho entre o IB2 e o *W*-IB2 com o domínio Íris.

Tabela 4.27 Valor comparativo de classificação entre IB2 e *W*-IB2 utilizando AG (k = 5)

	IB2	***W*-IB2 utilizando AG**
Classificações Corretas (%)	92,67	95,74
Desvio Padrão	0,9944289818	0,0019296422

Tabela 4.26 Tabela-Resumo dos resultados obtidos com o domínio Íris para o módulo *W*-IB2 utilizando AG

	Melhor Indivíduo	**Pior Indivíduo**	***Fitness* Médio**	**Desvio Padrão**	**Classificações Corretas (%)**	**Classificações Incorretas (%)**
Experimento 1	(0.0063999998, 0.2536999881, 0.9397000074, 0.9512000083)	(0.3779999911, 0.2536999881, 0.9020000100, 0.3759999871)	0,8481329083	0,0121286259	84,81	15,19
Experimento 2	(0.9940000176, 0.7588000297, 0.8848000168, 0.7325000166)	(0.9940000176, 0.7588000297, 0.8848000168, 0.7325000166)	0,8600003123	0,0000421468	86	14
Experimento 3	(0.9562000036, 0.3546000123, 0.9024999737, 0.9562000036)	(0.9562000036, 0.3546000123, 0.9024999737, 0.9562000036)	0,9466666579	0	94,67	5,33
Experimento 4	(0.9760000109, 0.9470000267, 0.9805999994, 0.0627999976)	(0.9760000109, 0.9470000267, 0.9805999994, 0.0627999976)	0,9066661000	0,0084293695	90,67	9,33
Experimento 5	(0.9506000280, 0.8091999888, 0.9921000003, 0.9348000288)	(0.7773000001, 0.3646999895, 0.4395999908, 0.9143999814)	0,8933329582	0,0066272155	89,33	10,67
Experimento 6	(0.9854999780, 0.9355999827, 0.9513000249, 0.7242000102)	(0.3844000101, 0.0527999997, 0.9348000288, 0.0089999996)	0,9131332635	0,0028572792	91,31	8,69
Experimento 7	(0.9957000017, 0.9106000065, 0.9879000186, 0.9599999785)	(0.3643000125, 0.4684000015, 0.9106000065, 0.9879000186)	0,9573993086	0,0019296422	95,74	4,26
Experimento 8	(0.9118999838, 0.8386999964, 0.9977999925, 0.9010000228)	(0.3919000029, 0.8386999964, 0.9865000247, 0.6794999837)	0,9565326313	0,0010937392	95,49	4,51

Os experimentos realizados para os domínios Sistema Vestibular e *Wine* para o *W*-IB2 seguiram o mesmo padrão adotado na seção anterior, ou seja, experimentar apenas a configuração que atingiu melhor resultado para o *W*-IB1 utilizando AG com o domínio Iris. Como comentado anteriormente, variações no número de gerações e tamanho da população não produziram variações substanciais nos resultados. As características desse experimento estão listadas na Tabela 4.28.

Tabela 4.28 Características genéticas do melhor experimento

Tamanho da População	Método de Seleção	Taxa de Cruzamento	Taxa de Mutação	Número de Gerações
100	Elitismo e roleta	80%	1%	100

Domínio de Conhecimento: Sistema Vestibular

A Figura 4.32 apresenta o resultado do experimento realizado com a melhor configuração de características genéticas mostrada na Tabela 4.28.

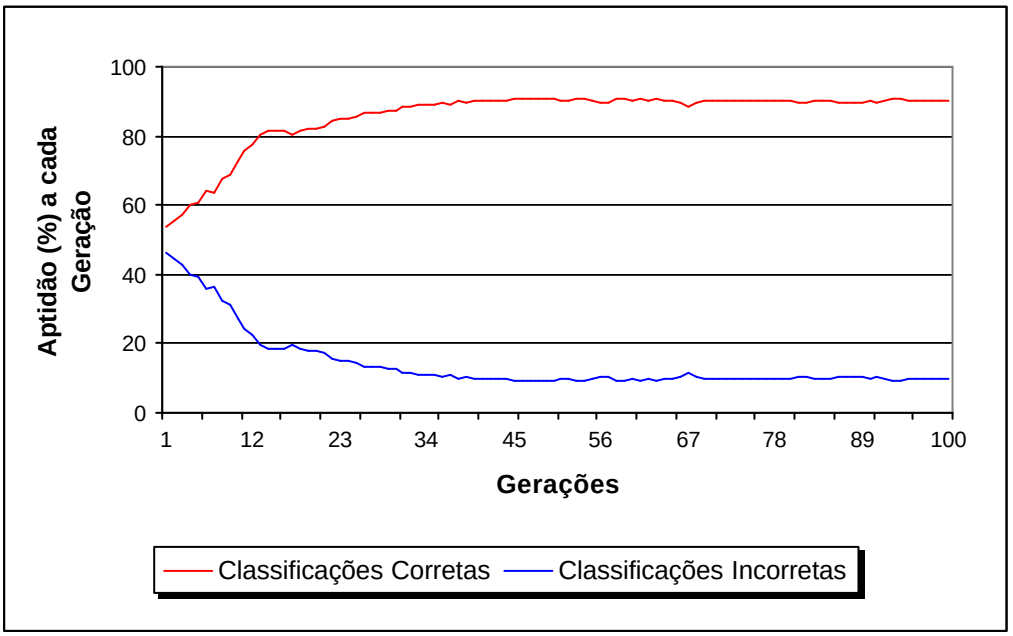

Figura 4.32 Tamanho da População = 100 e Número de Gerações = 100

A Tabela 4.29 mostra o vetor de pesos encontrado na última geração e a porcentagem de classificações corretas e incorretas que este obteve.

Tabela 4.29 Informações sobre o vetor de pesos na última geração (k = 5)

Vetor de pesos na última geração	Classificações Corretas (%)	Classificações Incorretas (%)
(0.9886000156, 0.0610000006, 0.9006999731, 0.0653000026, 0.0586999990, 0.9714999794)	89,98	10,02

A Tabela 4.30 traça um comparativo entre o IB2 e *W*-IB2 proposto sob o domínio Sistema Vestibular.

Tabela 4.30 Valor comparativo de classificação entre IB2 e *W*-IB2 utilizando AG (k = 5)

	IB2	*W*-IB2 utilizando AG
Classificações Corretas (%)	77,5	89,98
Desvio Padrão	1,7795131356	0,0037664172

Domínio de Conhecimento: *Wine*

A Figura 4.33 mostra o resultado do experimento para o domínio *Wine*.

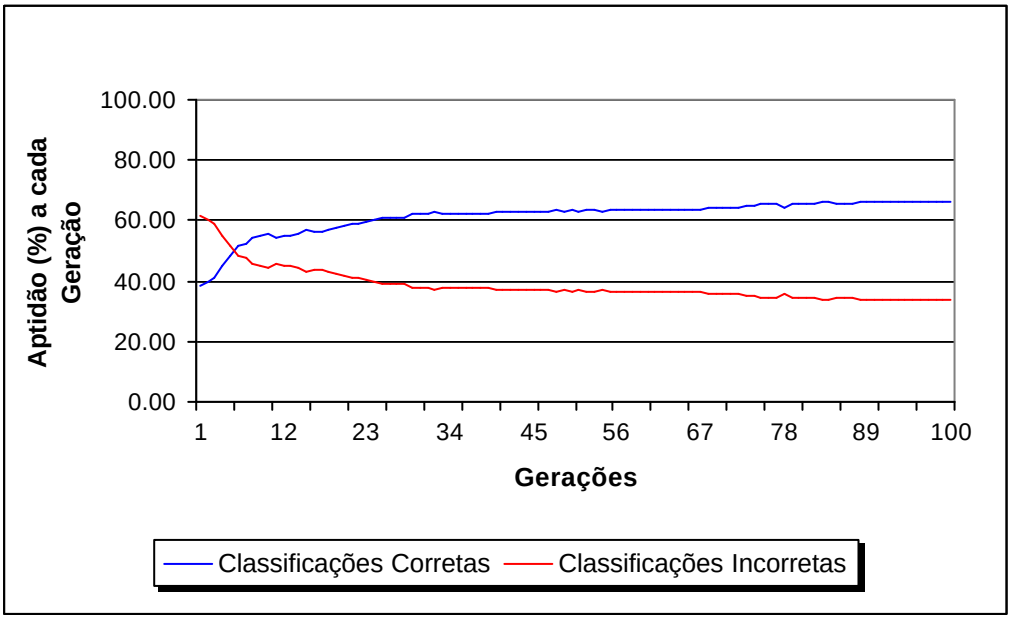

Figura 4.33 Tamanho da População = 100 e Número de Gerações = 100

A Tabela 4.31 mostra o vetor de pesos encontrado na última geração e a porcentagem de classificações corretas e incorretas que este obteve.

Tabela 4.31 Informações sobre o vetor de pesos na última geração (k = 5)

Vetor de pesos na última geração	**Classificações Corretas (%)**	**Classificações Incorretas (%)**
(0.8985999822, 0.9667000174, 0.3339000046, 0.0441999994, 0.9866999983, 0.8792999982, 0.8464000225, 0.8259000182, 0.9189000129, 0.5544000267, 0.5816000103, 0.0632999986, 0.9998999834)	66,29	33,71

A Tabela 4.32 traça um comparativo entre o IB2 e *W*-IB2 proposto com o domínio *Wine*.

Tabela 4.32 Valor comparativo de classificação entre IB2 e *W*-IB2 utilizando AG (k = 5)

	IB2	***W*-IB2 utilizando AG**
Classificações Corretas (%)	71,66	66,29
Desvio Padrão	2,1317701564	0,0003683525

A Figura 4.34 mostra o desempenho do *W*-IB2 utilizando AG com o domínio *Wine* de acordo com a proposta descrita na Seção 4.2.3.

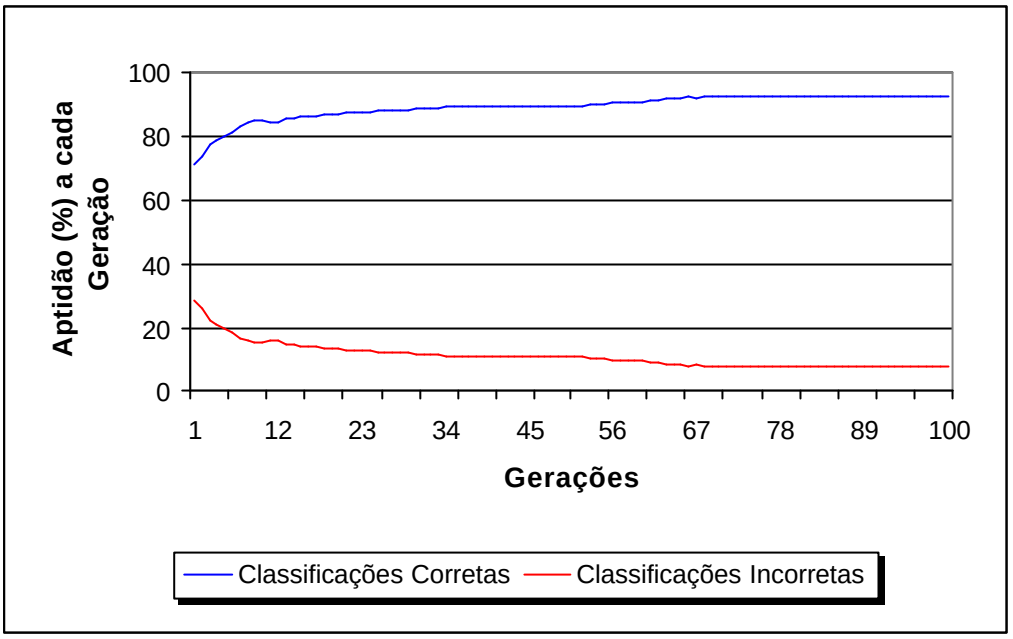

Figura 4.34 Tamanho da População = 100 e Número de Gerações = 100

A Tabela 4.33 ilustra o vetor de pesos encontrado na última geração e a porcentagem de classificações corretas e incorretas.

Tabela 4.33 Informações sobre o vetor de pesos na última geração (k = 5)

Vetor de pesos na última geração	Classificações Corretas (%)	Classificações Incorretas (%)
(0.1402000039, 0.9269663095, 0.7986999750, 0.5070000290, 0.7206000089, 0.7860000133, 0.9688000082, 0.4948999881, 0.9728999733)	92,70	7,30

Pelas Tabelas 4.31 e 4.33 é possível perceber uma melhora de desempenho na classificação de novas instâncias. Quando comparado ao IB2 também é possível perceber uma sensível melhora como mostra a Tabela 4.34.

Tabela 4.34 Valor comparativo de classificação entre IB2 e *W*-IB2 utilizando AG (k = 5)

	IB2	*W*-IB2 utilizando AG
Classificações Corretas (%)	90	92,70
Desvio Padrão	1,2292722549	0,0000035762

4.3 Conclusões e Possíveis Desdobramentos

Este trabalho investigou a viabilidade de uma colaboração entre aprendizado de máquina baseado em instâncias e uma técnica de otimização e busca, com o objetivo de obter uma melhor representação do conceito aprendido.

Com esse objetivo, algoritmos genéticos foram implementados para realizar uma busca no espaço de possíveis vetores de pesos de atributos, procurando vetores que traduzissem a relevância dos atributos na representação das classes.

Como comentado nessa dissertação, o problema de busca tratado é computacionalmente bastante complexo. Buscou-se, com o uso de AG, torná-lo mais facilmente representável e tratável. Os resultados obtidos nos experimentos, muito embora não exaustivos, limitados a alguns domínios de conhecimento e limitados a determinadas características, permitem concluir que os algoritmos baseados em instâncias têm um melhor desempenho quando estão articulados a AGs para a determinação do vetor de pesos.

O trabalho não é exaustivo no que diz respeito à exploração de todas as possibilidades de combinações de características genéticas, mesmo porque o seu número é considerável e dinâmico. Foram experimentados AGs com configurações básicas e apenas um tipo de representação (real, a mais adequada para o tipo de dado dos domínios tratados). A literatura dispõe de um volume razoável de possíveis alternativas para operadores de seleção, cruzamento e mutação. Isso sem falar das taxas de cruzamento e mutação que, também, são variáveis e podem, eventualmente, interferir nos resultados obtidos.

Os experimentos descritos variaram apenas o tamanho da população e o número de gerações e os resultados mostrados são aqueles usando a seleção da roleta com elitismo de um indivíduo, cruzamento simples e mutação randômica. É importante comentar que a seleção por torneio e a mutação *creep* também foram implementadas; em virtude dos resultados não terem sido tão satisfatórios, foram omitidos.

Esse trabalho de pesquisa focalizou apenas relevância de atributos na caracterização de todas as classes do domínio e não relevância de instâncias,

quando da classificação. Um possível desdobramento do trabalho descrito é o de atribuir pesos às instâncias, com base no seu poder classificatório. Outro o de criar pesos associados a atributos, particulares a cada uma das classes representadas.

Referências Bibliográficas

AHA, D. W. (1990). A study of instance-based algorithms for supervised learning tasks. Dissertação (Ph. D.). *University of California at Irvine*.

AHA, D. W. (1991). Analyses of instance-based learning algorithms. *Proceedings of the 9th National Conference on Artificial Intelligence*. AAAI Press - The MIT Press, v.2.

AHA, D. W. (1992). Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. *International Journal Man-Machine Studies*, v. 36, n. 2, p. 267-287.

AHA, D. W. (1997). Lazy Learning. USA: *Kluwer Academic Publisher*, Norwell.

AHA, D. W. (1998) Feature weighting for Lazy Learning Algorithms. In: LIU, H.; MOTODA, H. (Eds). *Feature extraction, construction and selection: a Data Mining Perspective*. Norwell MA: Kluwer, p.13-32.

AHA, D. W.; KIBLER, D; ALBERT, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, v.6, n.1, p. 37-66.

AKKUS, A.; GÜVENIR, A. (1996). Weighting features in k nearest neighbor classification on feature projections. *Scientific and Technical Research Council of Turkey*. p.1-9.

BEASLEY, D.; RALPH, R. M.; DAVID, R. B. (1993). *An overview of genetic algorithms: Part 2, Research Topics*. *University Computing,* v.15, n.4, p.170-181.

BOULOS, P.; CAMARGO, I. (1987). *Geometria Analítica: um tratamento vetorial.* São Paulo*:* McGrawHill.

BOOKER, L. B.; GOLDBERG, D. E.; HOLLAND, J. H. (1989). Classifier systems and genetic algorithms*. Artificial Intelligence*, v. 40, n.1-3, p. 235-282.

BREIMAN, L.; FRIEDMAN, J. H.; OLSHEN, R. A.; STONE, C. J. (1984). Classification and regression trees*. Machine Learning*, v. 10, p. 57-78.

CARBONELL, J. G. (1989). Paradigms for machine learning - introduction*. Artificial Intelligence*, v. 40, p. 1-9.

CASTILHO, V. C. (2003). *Otimização de componentes de Concreto Pré-moldado Protendidos mediante Algoritmos Genéticos*. Tese (Doutorado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

CLARK P.; NIBLETT T. (1999). The CN2 Induction Algorithm. Machine Learning 3, 261-283

COLEY, D. A. (1999). *An Introduction to genetic algorithms for scientists and engineers*. Singapore, World Scientific.

COVER, T.; HART, P. (1967). Nearest Neighbor pattern classification. *IEEE Transactions on Information Theory*, v. IT 13, p. 21-27.

DE NARDIN, L. (2000). *Estudo da aplicação de algoritmos genéticos para resolução do problema de geração de horários*. Monografia (Graduação) – Departamento de Informática – Universidade Estadual do Oeste do Paraná.

FIGUEIRA, L. B.; COLAFEMINA, J. F.; ROQUE, A. C. (2003). Diagnóstico de Patologias do Sistema Vestibular utilizando Redes Neurais na Análise dos Movimentos Sacádicos. Encontro Nacional de Inteligência Artificial. Campinas, SP.

GATES, G. W. (1972). The reduced nearest neighbor rule. *IEEE Transactions on Information Theory* v. IT 18, n. 3, p. 431-433.

GEN, M.; CHENG, R. (1997). *Genetic Algorithms and Engineering Design*. New York, John Wiley.

GOLDBERG, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading, U.S.A., Addison Wesley Publishing.

HART, P. E. (1968). The Condensed Nearest Neighbor Rule. *IEEE Transactions on Information Theory*, v. IT 14, n. 3, p. 515-516.

HOGG, R. V.; TANIS, E. A. (1983). *Probability and statistical inference*. New York: Macmillan.

HOLLAND, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, University of Michigan Press.

LACERDA, E. G. M.; CARVALHO, A. C. P. L. F. (1999). Introdução aos Algoritmos Genéticos. In: GALVÃO, C. O.; VALENÇA, M. J. S. (org.) (1999). *Sistemas inteligentes: aplicações a recursos hídricos e ciências ambientais*. Porto Alegre, Ed. Universidade/UFRG/ABRH. p.99-150.

MICHALEWICZ, Z. (1996). *Genetic algorithms + data structures = evolution programs*. Berlin, Springer-Verlag.

MITCHELL, M. (1996). *An introduction to genetic algorithms*. University of Michigan Press.

MITCHELL, T. M. (1997). *Machine Learning*. New York, McGraw-Hill.

MUGGLETON, S. (1992). *Inductive logic programming*. Academic Press Limited.

NICOLETTI, M. C. (1994). *Ampliando os limites do aprendizado indutivo de máquina através das abordagens construtiva e relacional*. Tese (Doutorado). Instituto de Física de São Carlos - Universidade de São Paulo.

NICOLETTI, M.C.; SANTOS, F.O. (1996). *O modelo de aprendizado PAC*. Relatório Técnico RT-DC 004/96. Departamento de Computação - Universidade Federal de São Carlos.

PALMA NETO, L.G.; FIGUEIRA, L. B.; NICOLETTI, M.C. (2003). Using a family of perceptron-based neural networks for detecting central vestibular system problems. *Procedings of the International Conference on Machine Learning and Aplications – ICMLA'03,* Los Angeles, CA, 2003, p.193-199

QUINLAN, J. R. (1987). Simplifying decision trees. *International Journal Man-Machine Studies*, v.27, n.3, p.221-234.

RAMER, A.; NICOLETTI, M.C. (2000). The symbolic side of a neuro-fuzzy system, *Studies in Fuzziness and Soft Computing*, v. 54, P. SINCAK, J. VASCAK (eds.), Physica-Verlag, Heidelberg, p.447-452 .

RAYMER, M. L.; PUNCH, W. F.; KUHN, E. D. G. L.; JAIN, A. K. (2000). Dimensionality reduction using genetic algorithms. *IEEE Transactions on Evolutionary Computation*, v.4. n-2, p.164-171.

SALZBERG, S.; DELCHER, A.; HEATH, D.; KASIF, S. (1995). Best-case results for nearest-neighbor learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v.17, n.6, p.599-608.

SANTOS, F. O. (1997). *O uso de exemplares generalizados fuzzy no aprendizado indutivo de máquina*. Dissertação (Mestrado). Departamento de Computação - Universidade Federal de São Carlos.

SANTOS, F.O.; NICOLETTI, M.C. (1996). *O modelo de aprendizado baseado em instâncias – Algoritmo IB1.* Relatório Técnico RT-DC 008/96. Departamento de Computação - Universidade Federal de São Carlos.

SANTOS, F.O.; NICOLETTI, M.C. (1997). *A família de algoritmos Instance-Based Learning (IBL).* Relatório Técnico RT-DC 006/97. Departamento de Computação - Universidade Federal de São Carlos.

SRINIVAS, M.; PATNAIK, L. M. (1994). Genetic algorithms: a survey. *IEEE Transactions on Information Theory*, v.27, n.6, p.17-26.

TING, K. M. (1997). Discretisation in lazy learning algorithms. *Artificial Intelligence Review*, v.11, n.1-5, p.157-174.

VOLPINI, P.; FIGUEIRA, L. B.; COLAFEMINA, J. F.; ROQUE, A.C (2002). A neural network-based system for the diagnosis of central vestibular lesion. In: Valafar, F. (editor). *Proceedings of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences - METMBS'02*, CSREA Press, p.29-33.

WETTSCHERECK, D.; AHA, D. W.; MOHRI, T. (1997). A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, v.11, p.273-314.

WETTSCHERECK, D.; AHA, D. W. (1995). Weighting features. In: *International Conference on Case-Based Reasoning*.

APÊNDICE A

O Sistema HIGEBI – Uma Descrição

O Sistema HIGEBI – Uma Descrição

O sistema híbrido Genético–Baseado em Instâncias denominado HIGEBI foi desenvolvido na linguagem *Borland* C++ *Builder* 6 para a plataforma *Windows* e implementa seis diferentes métodos de aprendizado baseado em instâncias, por meio de seis módulos principais, como mostra a Figura A.1. Cada método é acionado por meio da escolha, via *mouse*, de seu botão correspondente, na tela inicial do sistema.

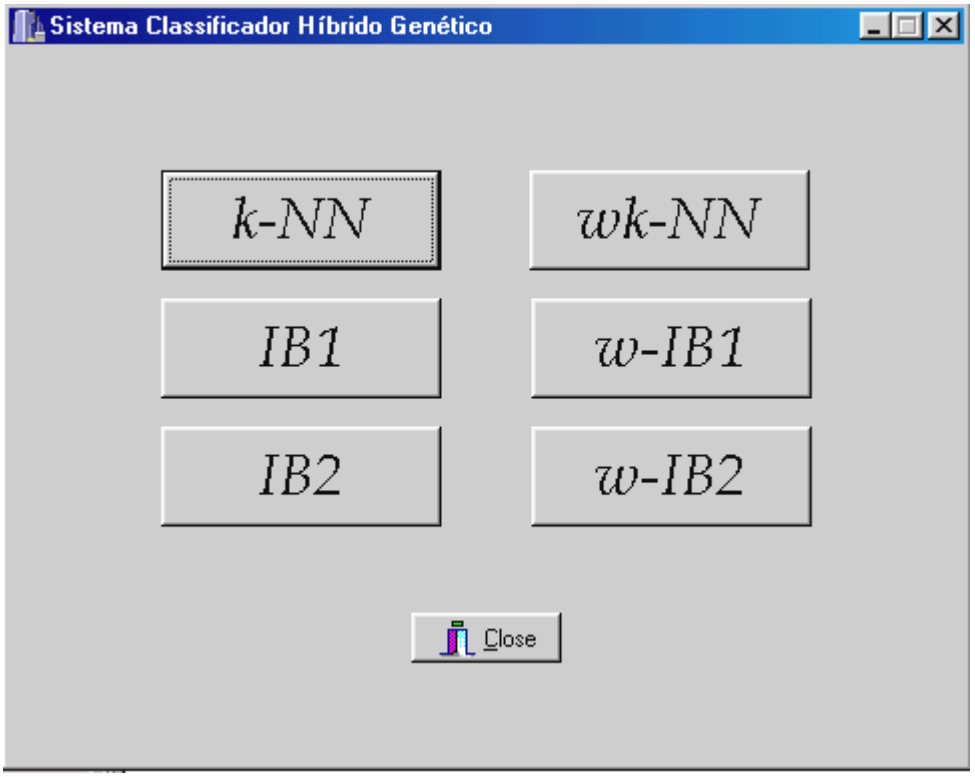

Figura A.1 Tela principal do HIGEBI

✓ Módulo 1: *k*-NN

O módulo 1 (*k*-NN) implementa o método *k*-NN. Como descrito, anteriormente, no Capítulo 2, quando da classificação de uma nova instância, este método considera os *k* vizinhos mais próximos dela, sendo que o valor de *k* é definido pelo usuário. Ao clicar no botão *k*-NN, a janela mostrada na Figura A.2 será aberta.

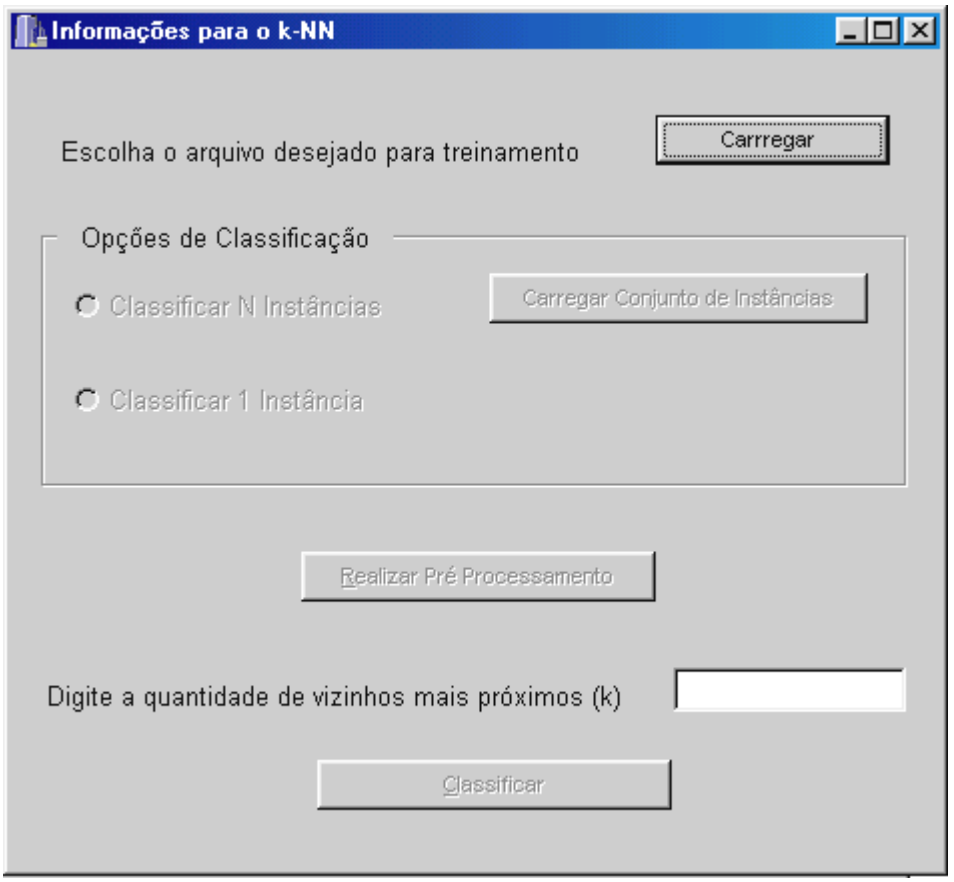

Figura A.2 Tela de informações para o *k*-NN

Uma vez feito isso é necessário escolher o arquivo que será utilizado pelo *k*-NN para treinamento. É fornecida ao usuário a opção de classificar apenas uma instância ou um conjunto de instâncias. Se o usuário deseja classificar apenas uma instância (Classificar 1 Instância), serão habilitados os campos para que esta possa ser digitada. Se a opção desejada for um conjunto de instâncias (Classificar N Instâncias), o usuário deverá fornecer o arquivo que contém esse conjunto. Essa opção pode ser usada tanto para avaliação da expressão do conceito num arquivo de teste quanto para a classificação das instâncias de teste. O sistema necessita também o número de vizinhos que serão levados em consideração no processo de classificação. Se o número de vizinhos definido for maior do que o número de instâncias presentes no arquivo de treinamento, uma mensagem de aviso será exibida informando o número máximo de vizinhos que poderão ser levados em consideração.

Vale lembrar que para todos os módulos implementados pelo HIGEBI (*k*-NN, *Wk*-NN, IB1, *W*-IB1, IB2 e *W*-IB2), similaridade é descrita como proximidade, ou

seja, as instâncias mais similares são as instâncias que estão mais próximas. Portanto, o cálculo da distância é feito através da distância Euclidiana já descrita na Figura 2.7 do Capítulo 2.

Ao clicar no botão Realizar Pré Processamento automaticamente o botão Classificar é habilitado; quando o Classificar é clicado, a classificação será efetivamente realizada. O resultado da classificação (classe das instâncias) será retornado ao usuário.

✓ Módulo 2: *Wk*-NN

O módulo 2 (*Wk*-NN) utiliza o algoritmo *Wk*-NN para a classificação de novas instâncias e está dividido em dois submódulos: o primeiro que não utiliza AGs (botão Sem Utilizar AG) e o segundo utilizando AGs (botão Utilizando AG), como mostra a Figura A.3.

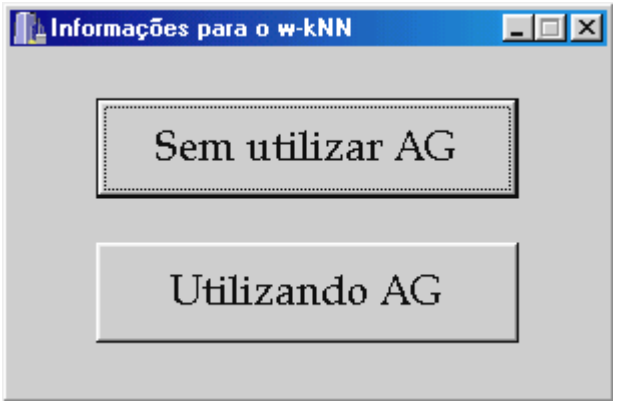

Figura A.3 Tela de informações iniciais para o *Wk*-NN

Sem utilizar AG: este módulo implementa o *k*-NN ponderado "puro" ou seja, o *k*-NN sem qualquer colaboração de outro método. Cabe ao usuário definir o vetor de pesos de atributos que julgar mais conveniente a ser utilizado na classificação. Ao clicar no botão Sem utilizar AG a tela mostrada pela Figura A.4 será aberta e outras informações serão requeridas para que a classificação possa ser realizada.

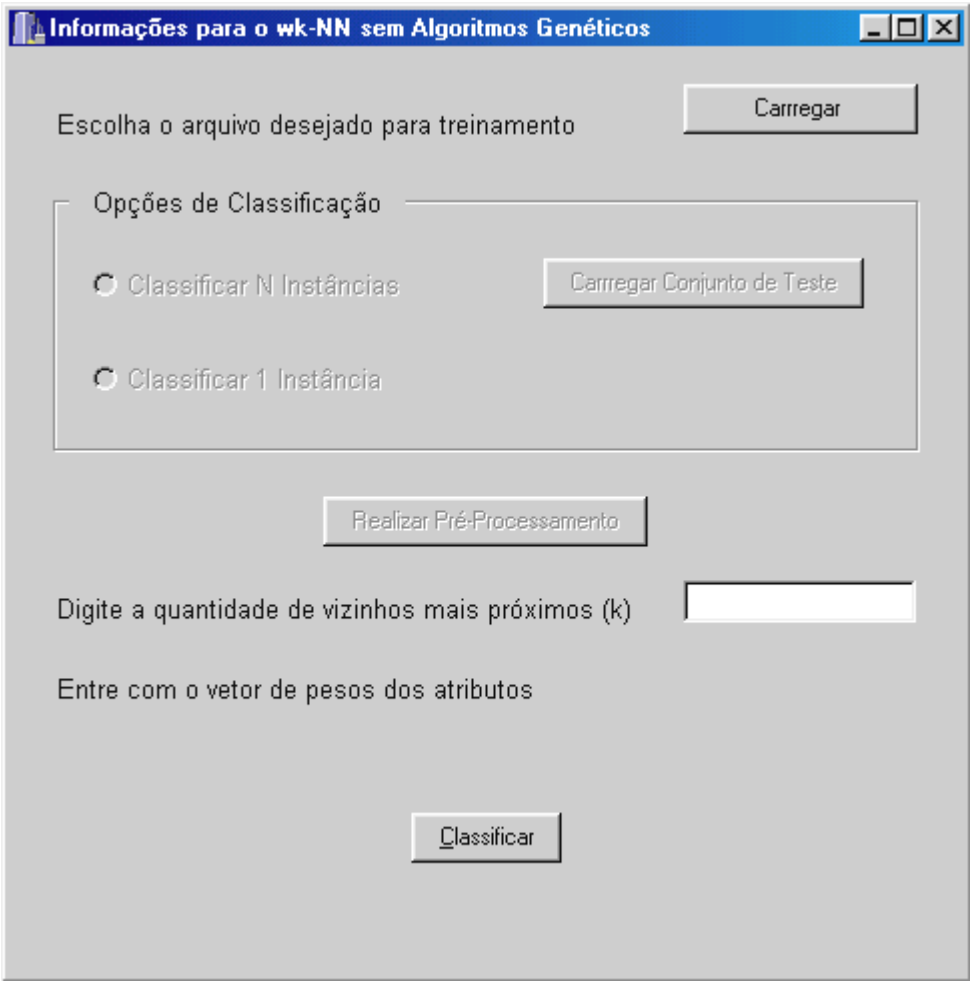

Figura A.4 Tela de informações para o *Wk*-NN sem utilizar AGs

Nesta etapa será necessário definir o arquivo de treinamento e de teste a ser utilizado na classificação. Se houver apenas uma instância a ser classificada, os campos para que seja feita a entrada dos atributos dessa instância serão habilitados. Ao clicar no botão Realizar Pré-Processamento, serão habilitados os campos onde deverá ser digitado o vetor de pesos; o número de campos habilitados corresponde ao número de atributos que descrevem as instâncias de treinamento, como mostrado na Figura A.5.

Neste vetor serão definidos os pesos de cada atributo que descreve as instâncias do conjunto de treinamento de acordo com sua relevância a caracterização do conceito e o valor deverá variar no intervalo [0, 1]. Ao clicar no botão Classificar, a classificação será efetivamente realizada e o resultado da classificação (classe das instâncias) será retornado ao usuário como mostra a Figura A.5.

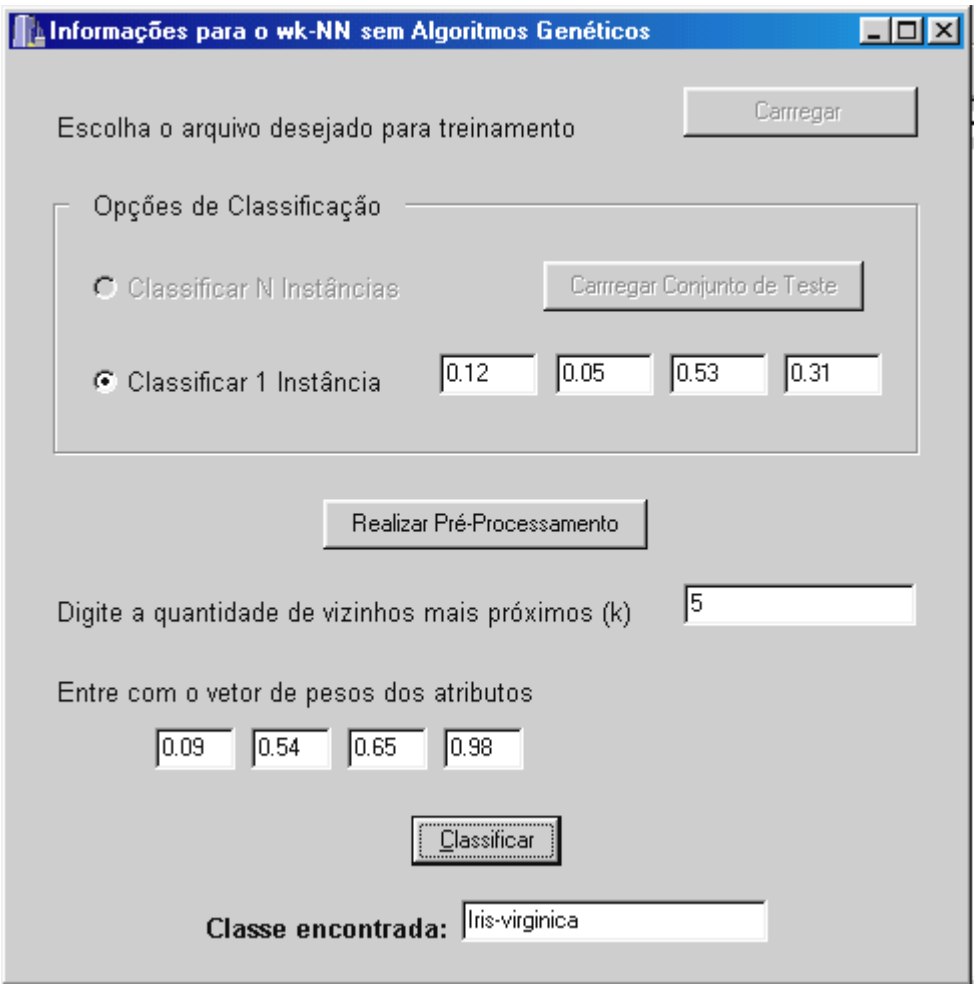

Figura A.5 Resultado da classificação de uma instância pelo *Wk*-NN sem utilizar AGs

Obviamente, uma "boa" definição de pesos de atributos, deixada a cargo do usuário, depende muito da sensibilidade do usuário ao domínio de conhecimento do problema que está sendo representado. Um usuário que desconhece a relevância de cada um dos atributos na caracterização do conceito será incapaz de fornecer um vetor de pesos que caracteriza essa relevância. Por outro lado, um usuário familiar com o problema e com o domínio de conhecimento, na maioria das vezes terá dificuldades em fornecer um vetor de pesos que efetivamente represente a relevância de cada um dos atributos envolvidos. Devido ao fato da determinação de um vetor de pesos representativo ser de extrema importância para a caracterização refinada do conceito, outras formas para a definição deste vetor têm sido cogitadas. Uma delas é aquela feita por meio do uso de um método de otimização no caso, AGs que é implementada no submódulo descrito a seguir.

Utilizando AG: esse submódulo implementa a colaboração genética-baseada em instâncias, cuja investigação é o objetivo principal deste trabalho. Ao clicar o botão Utilizando AG uma janela é aberta onde deve ser informado o número de arquivos de treinamento e teste a serem utilizados no aprendizado e na classificação, bem como, especificados quais são os arquivos. Após a escolha destes arquivos a tela resultante é mostrada como na Figura A.6.

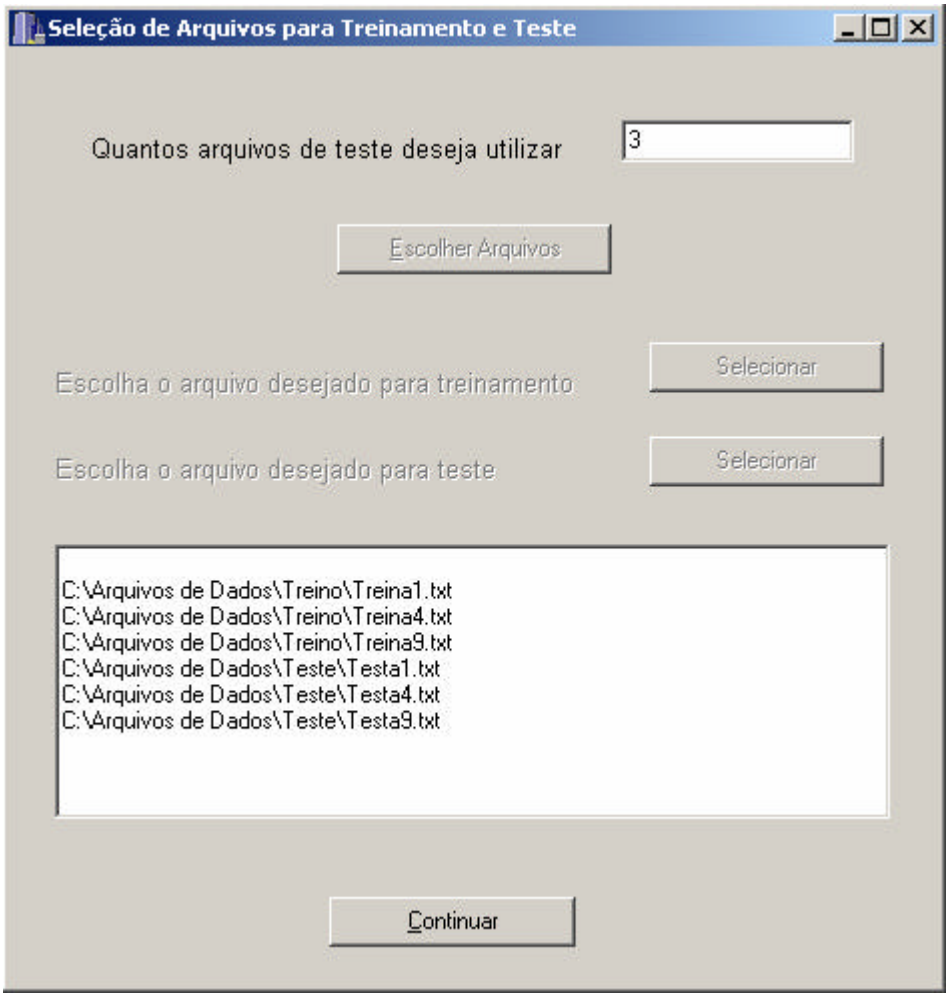

Figura A.6 Tela com os arquivos selecionados para treinamento e teste

Ao clicar no botão Continuar, a janela mostrada na Figura A.7 será aberta.

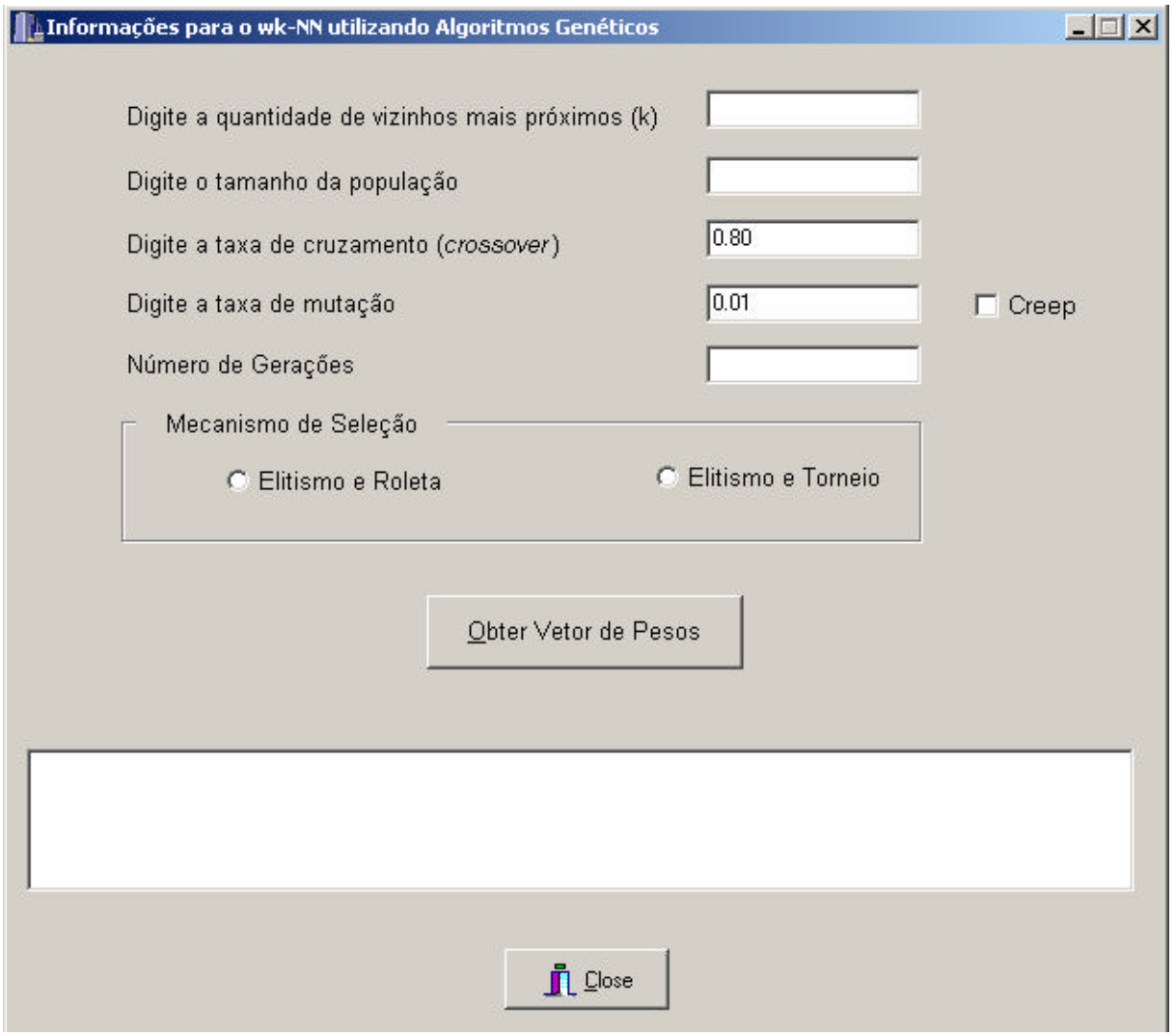

Figura A.7 Tela de informações para o *Wk*-NN usando AG

Na tela mostrada na Figura A.7 devem ser passadas as informações necessárias para a execução do AG. Além disso, o sistema espera como entrada o número de vizinhos mais próximos que serão levados em consideração (isto é, qual *k*-NN será usado). As informações necessárias para o funcionamento do AG são:

- tamanho da população;
- taxa de cruzamento (normalmente utilizada em 80%);
- taxa de mutação (normalmente utilizada 1%);
- tipo de mutação:
  - mutação randômica;
  - mutação *creep*;

- número de gerações;
- mecanismo de seleção:

  - elitismo e roleta;

  - elitismo e torneio (para maiores detalhes sobre esses operadores rever Seção 3.6).

Após a definição dessas informações, ao clicar o botão Obter Vetor de Pesos o código relativo à implementação do AG é executado e, ao final da execução, o vetor de pesos com o melhor valor de função de aptidão é mostrado, como na Figura A.8.

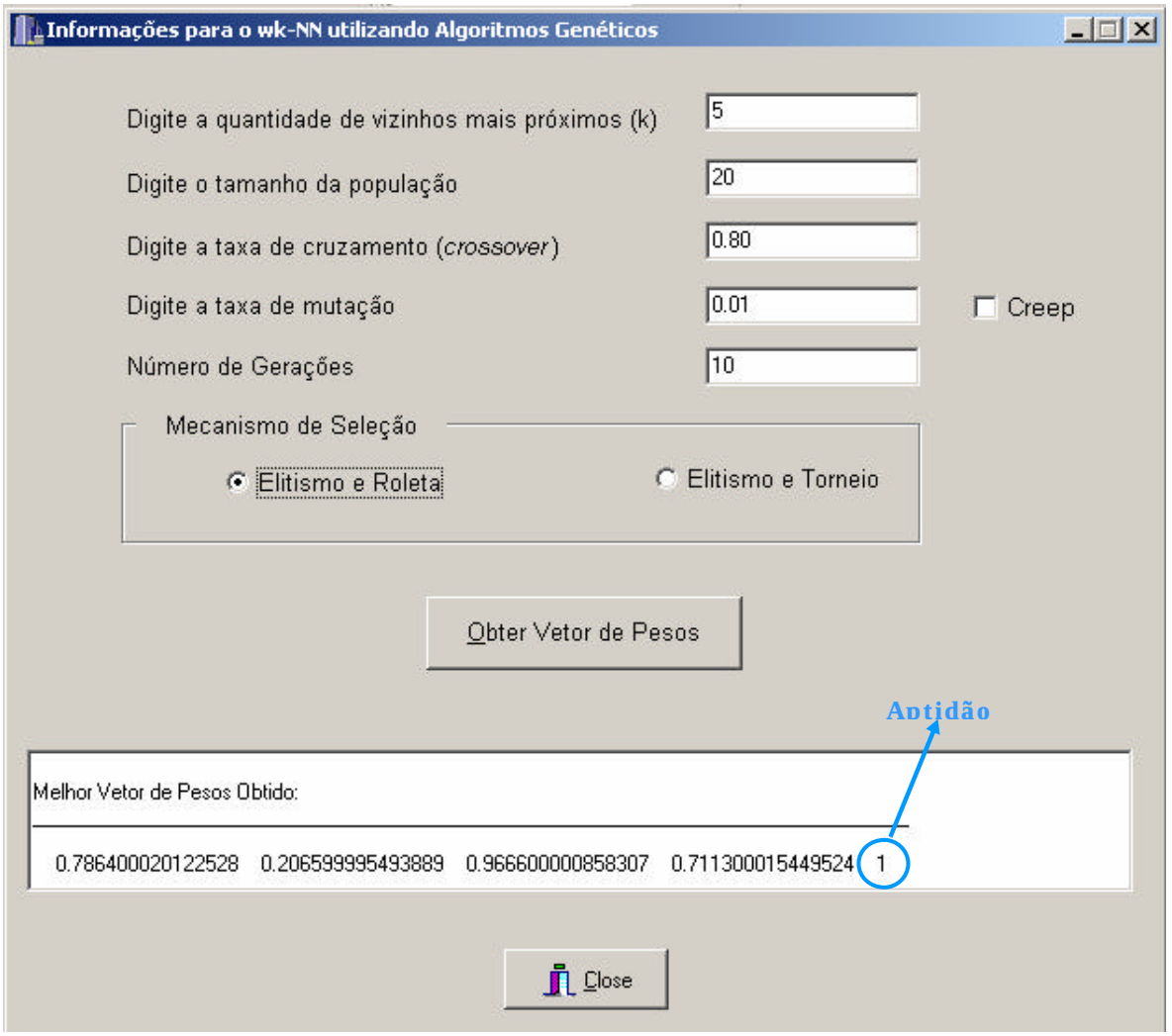

Figura A.8 Tela com o melhor vetor de pesos obtido e sua aptidão

Como a taxa de mutação geralmente é pequena, pode que não aconteça mutação em nenhum dos genes. Caso isso aconteça, o usuário será informado por meio de uma caixa de diálogo.

O algoritmo genético utilizado para classificar através do *Wk*-NN é o descrito na Figura A.9.

```
procedure AG()
begin
   gera população inicial;
   while critério de parada não for satisfeito do
   begin
       avalia();
       seleciona();
       aplica cruzamento();
       aplica mutação();
   end
end


procedure avalia(P,C_K,T_K);
{
  P – população a ser avaliada onde o indivíduo W_I é notado por:
       W_1I W_2I … W_NI
  C_K –  expressão do conceito obtida com o conjunto de treinamento treina_k
  T_K –  conjunto de pontos (conjunto teste_k) a ser classificado por C_K, levando
  em consideração cada W_I ∈ P. Cada t_p ∈ T_K é da forma: t_1p, t_2p, t_3p … t_Np, classe(t_p)
}


begin
     for_all W_I ∈ P do
     begin
          correta(W_I) ← 0
          for_all t_p = (t_1p  t_2p t_3p … t_Np) ∈ T_K do
          begin
               ponderar(W_I,t_p,W_It_p)
               classificar(C_K,W_It_p,R)
               if R then correta(W_I) ← correta(W_I) + 1
          end
          aval(W_I) ← correta(W_I)/|T_K|
     end
end

ponderar(W_I,t_p,W_It_p)
begin
     for q ← 1 to N do
          W_It_p[q] ← W_I[q]* t_p[q]
end

classificar(C_K,W_It_p,R)
begin
     R ← false
     k_NN(C_K,W_It_p,Classe)
     if Classe = classe(W_It_p) then R ← true
end
```

Figura A.9 Pseudocódigo do processo de avaliação de uma população

✓ Módulo 3: IB1

O módulo 3 (IB1) implementa o algoritmo IB1, que é o algoritmo mais simples da família IBL. Como descrito no Capítulo 2, o IB1 armazena todas as instâncias de treinamento, que são processadas incrementalmente; a descrição do conceito é o próprio conjunto de treinamento. Neste módulo, uma funcionalidade comentada, mas não presente na descrição do algoritmo foi adicionada; a cada instância é associado um registro de desempenho, que armazena quantas vezes essa instância foi utilizada na classificação de outra, e se a classificação foi correta ou incorreta. Ao clicar no botão IB1, a janela mostrada na Figura A.10 será aberta.

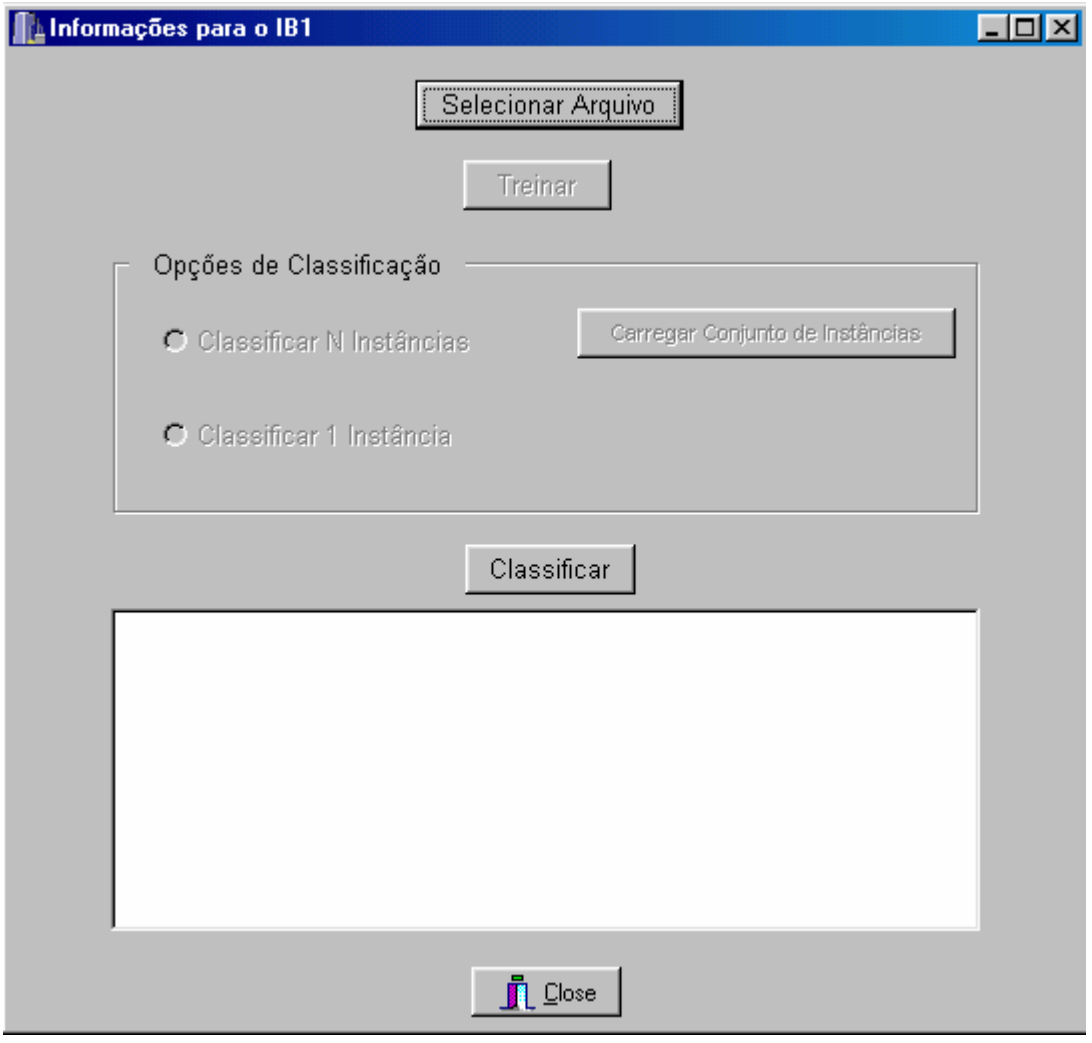

Figura A.10 Tela de informações para o IB1

Nesta tela é necessário especificar o arquivo para treinamento e o tipo de classificação: se de uma instância apenas ou de um conjunto de instâncias. Se a opção desejada, for um conjunto, o usuário deverá informar o arquivo que corresponde a esse conjunto. Ao selecionar o conjunto de instâncias e clicar no botão Classificar, a classificação será efetivamente realizada e as instâncias com suas respectivas classes associadas serão exibidas, como mostra a Figura A.11.

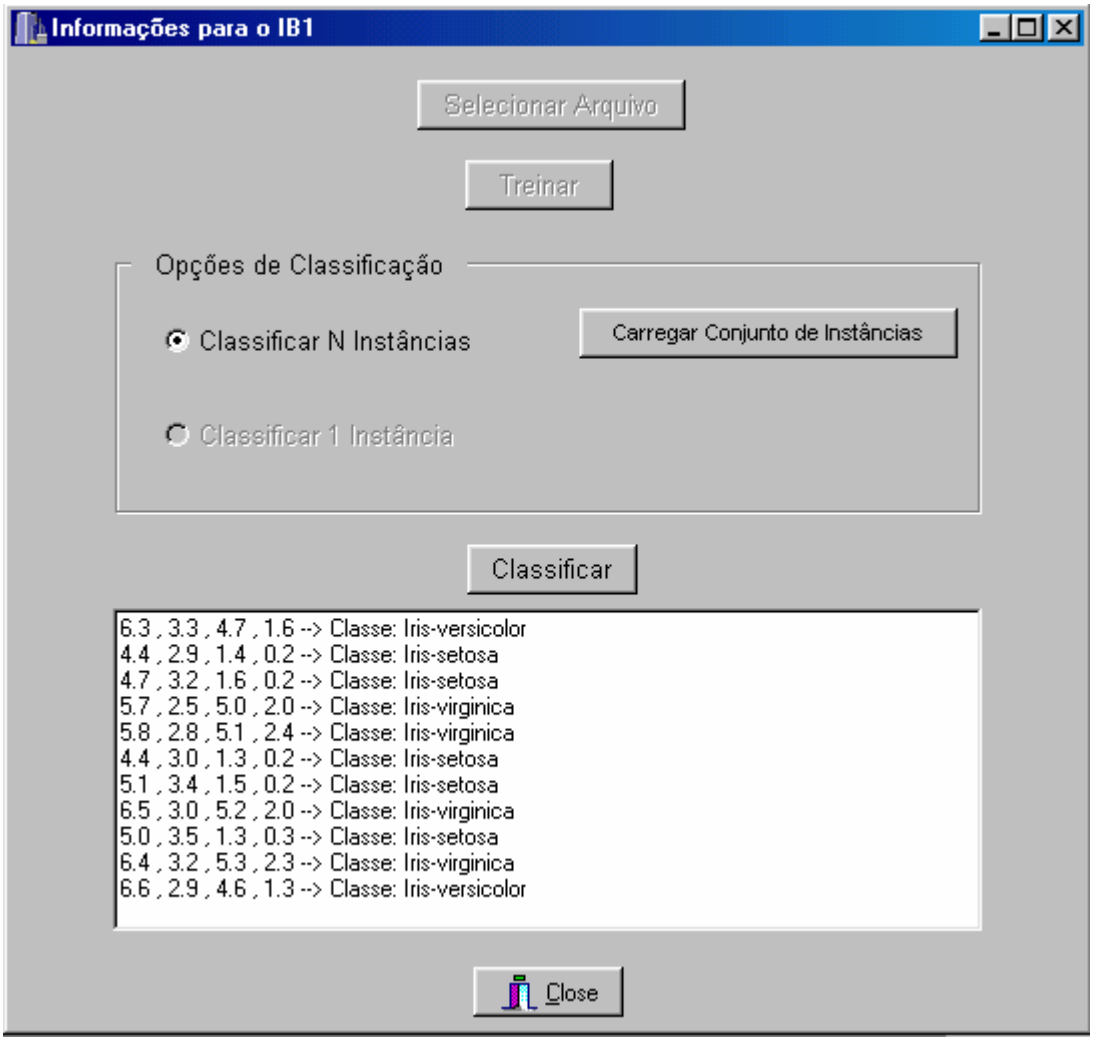

Figura A.11 Tela com resultados de classificação pelo IB1

✓ Módulo 4: *W*-IB1

O módulo 4 (*W*-IB1) implementa o algoritmo IB1 descrito na Seção 2.3.1 sem e com a colaboração genética para identificação de um vetor de pesos e, para tanto, disponibiliza a opção sem usar AGs (botão Sem Utilizar AG) e usando AGs (botão Utilizando AG).

Sem utilizar AG: Se o usuário não deseja utilizar AG, o funcionamento deste módulo é semelhante ao módulo descrito no item W*k*-NN Sem utilizar AG, diferindo apenas com relação ao número de vizinhos mais próximos que não é utilizado e, portanto, o usuário não fornecerá como entrada.

Após a entrada dos parâmetros necessários, a classificação será realizada e o resultado desta será retornado ao usuário da mesma forma como foi mostrado na Figura A.11.

Utilizando AG: este submódulo também funciona de maneira semelhante ao descrito no item W*k*-NN Utilizando AG e também difere no mesmo ponto citado anteriormente para o submódulo W-IB1 Sem Utilizar AG.

✓ Módulo 5: IB2

Este módulo funciona da mesma maneira que o descrito no item *W*-IB1, diferindo apenas no algoritmo que o implementa. O algoritmo utilizado neste módulo é o IB2, que difere do IB1, no sentido de que durante o aprendizado seu atualizador da descrição do conceito armazena apenas as instâncias que foram classificadas incorretamente. Além disso, neste módulo, para cada instância da descrição do conceito é armazenado o número de vezes que esta instância foi utilizada para classificar outra instância, identificando assim, o número de classificações corretas e incorretas em todas as participações de cada uma das instâncias que fazem parte da descrição do conceito. As informações necessárias para este módulo são como as descritas no módulo IB1.

✓ Módulo 6: *W*-IB2

Este módulo tem o mesmo objetivo do módulo 4, que é o de investigar a colaboração híbrida genética-baseada em instâncias. As informações a serem fornecidas são as mesmas. Este módulo também disponibiliza os dois submódulos descritos no módulo 4.

APÊNDICE B

Um Exemplo pelo HIGEBI

Introdução

Considere um espaço tridimensional (definido pelos atributos a_1, a_2 e a_3) onde a classe A está representada por três instâncias, a classe B por duas instâncias e a classe C por duas instâncias. A Tabela B.1 descreve os valores de atributos dessas sete instâncias que representam o conjunto de treinamento C1. Cada instância é notada por tr_i, i = 1,...,7. A Tabela B.2 descreve cada uma das instâncias do conjunto de teste T1, notadas por te_i, i=1,2,3.

Tabela B.1 Conjunto de Treinamento

	A_1	a_2	a_3	Classe
tr_1	1.20	0.01	1.80	A
tr_2	0.1	2.13	8.10	A
tr_3	1.30	1.50	1.80	A
tr_4	2.01	1.74	1.02	B
tr_5	1.01	1.15	0.15	B
tr_6	5.10	0.02	1.70	C
tr_7	1.15	0.21	0.73	C

Tabela B.2 Conjunto de Teste

	a_1	a_2	a_3	Classe
te_1	1.00	1.13	0.80	B
te_2	0.10	0.11	1.16	A
te_3	1.01	1.53	0.87	C

Considere também uma população inicial de tamanho quatro inicializada randomicamente onde cada cromossomo, notado por w_i, i=1, 2, 3, 4 representa um conjunto de pesos de atributos. Nesta população, cada cromossomo é representado por um vetor numérico real, como mostrado na Tabela B.3.

Tabela B.3 População inicial gerada randomicamente

	Peso de a_1	Peso de a_2	Peso de a_3
w_1	0.8342	0.8463	0.9829
w_2	0.2127	0.0386	0.9113
w_3	0.1164	0.5123	0.7111
w_4	0.3749	0.9413	0.6883

Cada cromossomo será avaliado de acordo com o algoritmo descrito na Figura A.9 do Apêndice A.

Segue uma descrição do funcionamento do algoritmo mostrado na Figura A.9 do Apêndice A, usando o 3-NN e tendo como entradas o conjunto de treinamento e teste descritos nas Tabelas B.1 e B.2, respectivamente.

❑ Avaliando o cromossomo w_1

	Peso de a_1	Peso de a_2	Peso de a_3
w_1	0.8342	0.8463	0.9829

O conjunto de treinamento ponderado pelo vetor de pesos representado pelo cromossomo w_1 é:

$tr'_1=w_1tr_1$	1.00104	0.008463	1.76922	A
$tr'_2=w_1tr_2$	0.08342	1.802619	7.96149	A
$tr'_3=w_1tr_3$	1.08446	1.26945	1.76922	A
$tr'_4=w_1tr_4$	1.676742	1.472562	1.002558	B
$tr'_5=w_1tr_5$	0.842542	0.973245	0.147435	B
$tr'_6=w_1tr_6$	4.25442	0.016926	1.67093	C
$tr'_7=w_1tr_7$	0.95932	0.177723	0.717517	C

Uma vez obtido o novo conjunto de treinamento (C'_1), que consiste do conjunto de treinamento original ponderado pelo vetor de pesos w_1, o algoritmo prossegue determinando o valor da função de avaliação de w_1, que é dado pela precisão de classificação do conceito representado por C'_1, no conjunto de teste, usando o 3-NN.

✓ Classificando te_1

Como o *k*-NN usa distância entre instâncias para identificar as *k* instâncias mais próximas, as distâncias entre a instância de teste te_1 a cada uma das instâncias tr'_i (i=1,..., 7) estão mostradas a seguir.

Distância	Classe
1.482306	A
7.251170	A
0.982836	A
0.785084	B
0.689352	B
3.548757	C
0.956707	C

Ordenando as linhas em ordem crescente de distância, tem-se:

Distância	Classe
0.689352	**B**
0.785084	**B**
0.956707	C
0.982836	A
1.482306	A
3.548757	C
7.251170	A

Como foi estabelecido k=3, serão considerados os três vizinhos mais próximos na determinação da classe da instância de teste te_1. Como a classe B é a classe que comparece com maior freqüência, à instância de teste te_1 seria atribuída a classe B pelo 3-NN. Como te_1 tem classe B, no conjunto de teste, a classificação feita pelo 3-NN usando o C'_1 foi correta.

✓ **Classificando te_2**

Como o *k*-NN usa distância entre instâncias para identificar as *k* instâncias mais próximas, as distâncias entre a instância de teste te_2 a cada uma das instâncias tr'_i (i=1,...,7) estão mostradas a seguir.

Distância	Classe
1.092397	A
7.008958	A
1.523761	A
2.089851	B
1.638485	B
4.186755	C
0.968921	C

Ordenando as linhas em ordem crescente de distância, tem-se:

Distância	Classe
0.968921	C
1.092397	**A**
1.523761	**A**
1.638485	B
2.089851	B
4.186755	C
7.008958	A

Como foi estabelecido k=3, serão considerados os três vizinhos mais próximos na determinação da classe da instância de teste te_2. Como a classe A é a classe que comparece com maior freqüência, à instância de teste te_2 seria atribuída a classe A pelo 3-NN. Como te_2 tem classe A, no conjunto de teste, a classificação feita pelo 3-NN usando o C'_1 foi correta.

✓ **Classificando te_3**

Como o *k*-NN usa distância entre instâncias para identificar as *k* instâncias mais próximas, as distâncias entre a instância de teste te_3 a cada uma das instâncias tr'_i (i=1,...,7) estão mostradas a seguir.

Distância	Classe
1.767413	A
7.156961	A
0.939163	A
0.682213	B
0.927425	B
3.668397	C
1.361790	C

Ordenando as linhas em ordem crescente de distância, tem-se:

Distância	Classe
0.682213	**B**
0.927425	**B**
0.939163	A
1.361790	C
1.767413	A
3.668397	C
7.156961	A

Como foi estabelecido k=3, serão considerados os três vizinhos mais próximos na determinação da classe da instância de teste te_3. Como a classe B é a classe que comparece com maior freqüência, à instância de teste te_3 seria atribuída a classe A pelo 3-NN. Como te_3 tem classe C, no conjunto de teste, a classificação feita pelo 3-NN usando o C'_1 foi incorreta.

Terminada a classificação das instâncias de testes para o vetor de pesos w_1, as informações encontradas são acrescentadas ao vetor como mostra a Tabela A.4:

Tabela B.4 Avaliação do vetor de pesos w_1 onde C e I significam números de instâncias classificadas correta e incorretamente, respectivamente

	Peso de a_1	Peso de a_2	Peso de a_3	Resultado da classificação	
				C	I
w_1	0.8342	0.8463	0.9829	2	1

- **Avaliando o cromossomo w_2**

	Peso de a_1	Peso de a_2	Peso de a_3
w_2	0.2127	0.0386	0.9113

O conjunto de treinamento ponderado pelo vetor de pesos representado pelo cromossomo w_1 é:

$tr'_1=w_2tr_1$	0.25524	0.000386	1.64034	A
$tr'_2=w_2tr_2$	0.02127	0.082218	7.38153	A
$tr'_3=w_2tr_3$	0.27651	0.0579	1.64034	A
$tr'_4=w_2tr_4$	0.427527	0.067164	0.929526	B
$tr'_5=w_2tr_5$	0.214827	0.04439	0.136695	B
$tr'_6=w_2tr_6$	1.08477	0.000772	1.54921	C
$tr'_7=w_2tr_7$	0.244605	0.008106	0.665249	C

Uma vez obtido o novo conjunto de treinamento (C'_2), que consiste do conjunto de treinamento original ponderado pelo vetor de pesos w_2, o algoritmo prossegue determinando o valor da função de avaliação de w_2, que é dado pela precisão de classificação do conceito representado por C'_2, no conjunto de teste, usando o 3-NN.

✓ Classificando te_1

Como o *k*-NN usa distância entre instâncias para identificar as *k* instâncias mais próximas, as distâncias entre a instância de teste te_1 a cada uma das instâncias tr'_i (i=1,..., 7) estão mostradas a seguir.

Distância	Classe
1.592754	A
6.653904	A
1.656237	A
1.214134	B
1.494998	B
1.357813	C
1.359200	C

Ordenando as linhas em ordem crescente de distância, tem-se:

Distância	Classe
1.214134	B
1.357813	**C**
1.359200	**C**
1.494998	B
1.592754	A
1.656237	A
6.653904	A

Como foi estabelecido k=3, serão considerados os três vizinhos mais próximos na determinação da classe da instância de teste te_1. Como a classe C é a classe que comparece com maior freqüência, à instância de teste te_1 seria atribuída a classe C pelo 3-NN. Como te_1 tem classe B, no conjunto de teste, a classificação feita pelo 3-NN usando o C'_2 foi incorreta.

✓ Classificando te_2

Como o *k*-NN usa distância entre instâncias para identificar as *k* instâncias mais próximas, as distâncias entre a instância de teste te_2 a cada uma das instâncias tr'_i (i=1,...,7) estão mostradas a seguir.

Distância	Classe
0.516566	A
6.222090	A
0.514389	A
0.402774	B
1.031815	B
1.064512	C
0.525425	C

Ordenando as linhas em ordem crescente de distância, tem-se:

Distância	Classe
0.402774	B
0.514389	**A**
0.516566	**A**
0.525425	C
1.031815	B
1.064512	C
6.222090	A

Como foi estabelecido k=3, serão considerados os três vizinhos mais próximos na determinação da classe da instância de teste te_2. Como a classe A é a classe que comparece com maior freqüência, à instância de teste te_2 seria atribuída a classe A pelo 3-NN. Como te_2 tem classe A, no conjunto de teste, a classificação feita pelo 3-NN usando o C'_2 foi correta.

✓ **Classificando te_3**

Como o *k*-NN usa distância entre instâncias para identificar as *k* instâncias mais próximas, as distâncias entre a instância de teste te_3 a cada uma das instâncias tr'_i (i=1,...,7) estão mostradas a seguir.

Distância	Classe
1.871578	A
6.743417	A
1.816179	A
1.575660	B
1.837681	B
1.674949	C
1.715783	C

Ordenando as linhas em ordem crescente de distância, tem-se:

Distância	Classe
1.575660	B
1.674949	**C**
1.715783	**C**
1.816179	A
1.837681	B
1.871578	A
6.743417	A

Como foi estabelecido k=3, serão considerados os três vizinhos mais próximos na determinação da classe da instância de teste te_3. Como a classe C é a classe que comparece com maior freqüência, à instância de teste te_3 seria atribuída a classe C pelo 3-NN. Como te_3 tem classe C, no conjunto de teste, a classificação feita pelo 3-NN usando o C'_2 foi incorreta.

Terminada a classificação das instâncias de testes para o vetor de pesos w_2, as informações encontradas são acrescentadas ao vetor como mostra a Tabela B.5:

Tabela B.5 Avaliação do vetor de pesos w_2 onde C e I significam números de instâncias classificadas correta e incorretamente, respectivamente

	Peso de a_1	Peso de a_2	Peso de a_3	Resultado da classificação	
				C	I
w_1	0.8342	0.8463	0.9829	2	1
w_2	0.2127	0.0386	0.9113	2	1

- **Avaliando o cromossomo w_3**

	Peso de a_1	Peso de a_2	Peso de a_3
w_3	0.1164	0.5123	0.7111

O conjunto de treinamento ponderado pelo vetor de pesos representado pelo cromossomo w_1 é:

$tr'_1=w_3tr_1$	0.13968	0.005123	1.27998	A
$tr'_2=w_3tr_2$	0.01164	1.091199	5.75991	A
$tr'_3=w_3tr_3$	0.15132	0.76845	1.27998	A
$tr'_4=w_3tr_4$	0.233964	0.891402	0.725322	B
$tr'_5=w_3tr_5$	0.117564	0.589145	0.106665	B
$tr'_6=w_3tr_6$	0.59364	0.010246	1.20887	C
$tr'_7=w_3tr_7$	0.13386	0.107583	0.519103	C

Uma vez obtido o novo conjunto de treinamento (C'_3), que consiste do conjunto de treinamento original ponderado pelo vetor de pesos w_3, o algoritmo prossegue determinando o valor da função de avaliação de w_3, que é dado pela precisão de classificação do conceito representado por C'_3, no conjunto de teste, usando o 3-NN.

✓ Classificando te_1

Como o *k*-NN usa distância entre instâncias para identificar as *k* instâncias mais próximas, as distâncias entre a instância de teste te_1 a cada uma das instâncias tr'_i (i=1,..., 7) estão mostradas a seguir.

Distância	Classe
1.495285	A
4.960061	A
1.039883	A
0.805802	B
1.245765	B
1.259425	C
1.369101	C

Ordenando as linhas em ordem crescente de distância, tem-se:

Distância	Classe
0.805802	**B**
1.039883	A
1.245765	**B**
1.259425	C
1.369101	C
1.495285	A
4.960061	A

Como foi estabelecido k=3, serão considerados os três vizinhos mais próximos na determinação da classe da instância de teste te_1. Como a classe B é a classe que comparece com maior freqüência, à instância de teste te_1 seria atribuída a classe B pelo 3-NN. Como te_1 tem classe B, no conjunto de teste, a classificação feita pelo 3-NN usando o C'_3 foi correta.

✓ **Classificando te_2**

Como o *k*-NN usa distância entre instâncias para identificar as *k* instâncias mais próximas, as distâncias entre a instância de teste te_2 a cada uma das instâncias tr'_i (i=1,...,7) estão mostradas a seguir.

Distância	Classe
0.164222	A
4.704224	A
0.505983	A
0.904146	B
1.157325	B
0.671256	C
0.641795	C

Ordenando as linhas em ordem crescente de distância, tem-se:

Distância	Classe
0.164222	**A**
0.505983	**A**
0.641795	C
0.671256	C
0.904146	B
1.157325	B
4.704224	A

Como foi estabelecido k=3, serão considerados os três vizinhos mais próximos na determinação da classe da instância de teste te_2. Como a classe A é a classe que comparece com maior freqüência, à instância de teste te_2 seria atribuída a classe A pelo 3-NN. Como te_2 tem classe A, no conjunto de teste, a classificação feita pelo 3-NN usando o C'_3 foi correta.

✓ **Classificando te_3**

Como o *k*-NN usa distância entre instâncias para identificar as *k* instâncias mais próximas, as distâncias entre a instância de teste te_3 a cada uma das instâncias tr'_i (i=1,...,7) estão mostradas a seguir.

Distância	Classe
1.802994	A
5.010038	A
1.611781	A
1.015367	B
1.707050	B
1.288758	C
1.504769	C

Ordenando as linhas em ordem crescente de distância, tem-se:

Distância	Classe
1.015367	B
1.288758	**C**
1.504769	**C**
1.611781	A
1.707050	B
1.802994	A
5.010038	A

Como foi estabelecido k=3, serão considerados os três vizinhos mais próximos na determinação da classe da instância de teste te_3. Como a classe C é a classe que comparece com maior freqüência, à instância de teste te_3 seria atribuída a classe C pelo 3-NN. Como te_3 tem classe C, no conjunto de teste, a classificação feita pelo 3-NN usando o C'_3 foi correta.

Terminada a classificação das instâncias de testes para o vetor de pesos w_3, as informações encontradas são acrescentadas ao vetor como mostra a Tabela B.6:

Tabela B.6 Avaliação do vetor de pesos w_3 onde C e I significam números de instâncias classificadas correta e incorretamente, respectivamente

	Peso de a_1	Peso de a_2	Peso de a_3	Resultado da classificação	
				C	I
w_1	0.8342	0.8463	0.9829	2	1
w_2	0.2127	0.0386	0.9113	2	1
w_3	0.1164	0.5123	0.7111	3	0

❑ Avaliando o cromossomo w_3

	Peso de a_1	Peso de a_2	Peso de a_3
w_4	0.3749	0.9413	0.6883

O conjunto de treinamento ponderado pelo vetor de pesos representado pelo cromossomo w_1 é:

$tr'_1=w_3tr_1$	0.44988	0.009413	1.23894	A
$tr'_2=w_3tr_2$	0.03749	2.004969	5.57523	A
$tr'_3=w_3tr_3$	0.48737	1.41195	1.23894	A
$tr'_4=w_3tr_4$	0.753549	1.637862	0.702066	B
$tr'_5=w_3tr_5$	0.378649	1.082495	0.103245	B
$tr'_6=w_3tr_6$	1.91199	0.018826	1.17011	C
$tr'_7=w_3tr_7$	0.431135	0.197673	0.502459	C

Uma vez obtido o novo conjunto de treinamento (C'_4), que consiste do conjunto de treinamento original ponderado pelo vetor de pesos w_4, o algoritmo prossegue determinando o valor da função de avaliação de w_4, que é dado pela precisão de classificação do conceito representado por C'_4, no conjunto de teste, usando o 3-NN.

✓ Classificando te_1

Como o *k*-NN usa distância entre instâncias para identificar as *k* instâncias mais próximas, as distâncias entre a instância de teste te_1 a cada uma das instâncias tr'_i (i=1,..., 7) estão mostradas a seguir.

Distância	Classe
1.323259	A
4.949223	A
0.731405	A
0.572933	B
0.934773	B
1.484390	C
1.131976	C

Ordenando as linhas em ordem crescente de distância, tem-se:

Distância	Classe
0.572933	**B**
0.731405	A
0.934773	**B**
1.131976	C
1.323259	A
1.484390	C
4.949223	A

Como foi estabelecido k=3, serão considerados os três vizinhos mais próximos na determinação da classe da instância de teste te_1. Como a classe B é a classe que comparece com maior freqüência, à instância de teste te_1 seria atribuída a classe B pelo 3-NN. Como te_1 tem classe B, no conjunto de teste, a classificação feita pelo 3-NN usando o C'_4 foi correta.

✓ **Classificando te_2**

Como o *k*-NN usa distância entre instâncias para identificar as *k* instâncias mais próximas, as distâncias entre a instância de teste te_2 a cada uma das instâncias tr'_i (i=1,...,7) estão mostradas a seguir.

Distância	Classe
0.372512	A
4.805108	A
1.360647	A
1.723714	B
2.260821	B
1.814310	C
0.736213	C

Ordenando as linhas em ordem crescente de distância, tem-se:

Distância	Classe
0.372512	**A**
0.736213	C
1.360647	**A**
1.723714	B
1.814310	C
2.260821	B
4.805108	A

Como foi estabelecido k=3, serão considerados os três vizinhos mais próximos na determinação da classe da instância de teste te_2. Como a classe A é a classe que comparece com maior freqüência, à instância de teste te_2 seria atribuída a classe A pelo 3-NN. Como te_2 tem classe A, no conjunto de teste, a classificação feita pelo 3-NN usando o C'_4 foi correta.

✓ **Classificando te_3**

Como o *k*-NN usa distância entre instâncias para identificar as *k* instâncias mais próximas, as distâncias entre a instância de teste te_3 a cada uma das instâncias tr'_i (i=1,...,7) estão mostradas a seguir.

Distância	Classe
1.661937	A
4.828101	A
0.650534	A
0.324966	B
1.136829	B
1.785300	C
1.498421	C

Ordenando as linhas em ordem crescente de distância, tem-se:

Distância	Classe
0.324966	**C**
0.650534	A
1.136829	**C**
1.498421	B
1.661937	A
1.785300	B
4.828101	A

próximos na determinação da classe da instância de teste te_3. Como a classe C é a classe que comparece com maior freqüência, à instância de teste te_3 seria atribuída a classe C pelo 3-NN. Como te_3 tem classe C, no conjunto de teste, a classificação feita pelo 3-NN usando o C'_4 foi correta.

Terminada a classificação das instâncias de testes para o vetor de pesos w_4, as informações encontradas são acrescentadas ao vetor como mostra a Tabela B.7.

Tabela B.7 Avaliação do vetor de pesos w_4 onde C e I significam números de instâncias classificadas correta e incorretamente, respectivamente

	Peso de a_1	Peso de a_2	Peso de a_3	Resultado da classificação	
				C	I
w_1	0.8342	0.8463	0.9829	2	1
w_2	0.2127	0.0386	0.9113	2	1
w_3	0.1164	0.5123	0.7111	3	0
w_4	0.3749	0.9413	0.6883	3	0

O próximo passo, terminada a classificação das instâncias de teste, é calculada a aptidão equivalente a cada um dos vetores de pesos de acordo com o número de classificações corretas e incorretas.

A aptidão de cada cromossomo é representada pelo número de classificações corretas dividido pelo total de testes realizados. Para o exemplo, a Tabela B.8 descreve as aptidões de cada cromossomo.

Tabela B.8 Aptidão referente a cada cromossomo

	Nº de Classificações Corretas	Aptidão
w_1	2	0.666667
w_2	2	0.666667
w_3	3	1
w_4	3	1

O cromossomo w_3, que tem maior aptidão, é automaticamente selecionado para participar da população auxiliar (1-elitismo). Essa população auxiliar será formada pelo melhor indivíduo da população inicial e os outros membros serão selecionados através da seleção por roleta.

A seleção por roleta foi implementada de forma que cada cromossomo tenha uma porcentagem correspondente a sua aptidão. Depois disso, é gerado um número randômico de 0 a 100, e posteriormente, selecionado o indivíduo que o número selecionado pertença ao intervalo. A Tabela B.9 ilustra essa situação.

Tabela B.9 Seleção por Roleta

	Nº de Classificações Corretas	Aptidão	% Absoluta	Distribuição de Freqüência
w_1	2	0.666667	20%	0 |—| 20%
w_2	2	0.666667	20%	20 —| 40%
w_3	3	1	30%	40 —| 70%
w_4	3	1	30%	70 —|100%

Supondo que os números gerados randomicamente sejam: 17, 68, 72. Os cromossomos que formarão a população auxiliar junto com o melhor indivíduo selecionado automaticamente serão: cromossomo 1 (w_1), cromossomo 3 (w_3) e cromossomo 4 (w_4). A Tabela B.10 ilustra essa população auxiliar gerada.

Tabela B.10 População Auxiliar selecionada por Elitismo-1 e Roleta

w_3	0.1164	0.5123	0.7111
w_1	0.8342	0.8463	0.9829
w_3	0.1164	0.5123	0.7111
w_4	0.3749	0.9413	0.6883

Sobre essa população serão aplicados os operadores genéticos, sendo *crossover* com taxa de 80% e mutação com taxa de 1%. Considerando a taxa de *crossover* de 80%, significa que 3 (4 * 80% = 3.2) cromossomos serão cruzados e nenhum gene sofrerá mutação (4 * 3 * 1% = 0.12). Os três cromossomos que sofrerão *crossover* são selecionados randomicamente e colocados em uma nova população. Supondo que os cromossomos selecionados foram: w_3, w_1 e w_3 e o ponto de *crossover* (também determinado randomicamente no intervalo de]0,3]), tenha sido 2. A Tabela B.11 ilustra a nova população depois do cruzamento.

Tabela B.11 Nova população após a aplicação de cruzamento

w_3	0.1164	0.5123	0.9829
w_1	0.8342	0.8463	0.7111
w_3	0.1164	0.5123	0.7111

Ponto de cruzamento

Os cromossomos desta população substituirão os cromossomos na população auxiliar de forma que a nova população auxiliar será a mostrada na Tabela B.12.

Tabela B.12 População auxiliar após a aplicação de cruzamento

w_3	0.1164	0.5123	0.7111
w_1	0.8342	0.8463	0.7111
w_3	0.1164	0.5123	0.7111
w_4	0.3749	0.9413	0.6883

A população auxiliar substituirá a população inicial e o processo continuará até que o critério de parada estabelecido seja satisfeito. Na implementação sugerida, o critério de parada será sempre o número de gerações.