Projeto e Implementação de Circuitos Classificadores Digitais com Controle da Generalização Baseado na Regra do Vizinho-mais-próximo Modificada

Wilian Soares Lacerda

Universidade Federal de Minas Gerais
Escola de Engenharia
Programa de Pós-Graduação em Engenharia Elétrica

Projeto e Implementação de Circuitos Classificadores Digitais com Controle da Generalização Baseado na Regra do Vizinho-mais-próximo Modificada

Wilian Soares Lacerda

Orientador: *Prof. Dr. Antônio de Pádua Braga*

Tese de doutorado submetida à Banca Examinadora designada pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais - PPGEE/UFMG, como requisito parcial à obtenção do título de Doutor em Engenharia da Computação.

Belo Horizonte
fevereiro/2006

À minha filha

Yasmim

e à minha esposa

Adriana

Agradecimentos

- À Deus, por tudo.
- À minha esposa Adriana e minha filha Yasmim pela paciência e compreensão.
- Ao orientador Prof. Antônio de Pádua Braga, pelas horas de dedicação e disponibilidade.
- Aos amigos do LITC e CPDEE, pela ajuda nos momentos mais necessários.
- Aos funcionários do CPDEE, pelo profissionalismo.
- À CAPES, pela ajuda financeira.
- À Universidade Federal de Lavras por minha liberação.
- À banca examinadora pelo trabalho.

Resumo

Este trabalho de tese visa à implementação de classificadores de padrões binários em circuitos digitais de forma a se obter um sistema embutido com características de portabilidade, treinamento *on-line*, funcionamento em tempo real e com capacidade de generalização.

O método proposto para projeto utiliza o processo de filtragem (ou seleção) dos dados de treinamento do classificador antes da síntese do circuito digital. Assim, é proposto um algoritmo de seleção mínima de amostras baseada na Regra do k Vizinho-mais-próximo (kNN), para que a fase de projeto do classificador necessite de menos recursos de armazenamento e processamento, incrementando a capacidade de generalização do circuito resultante.

São apresentados alguns exemplos de projetos do classificador digital gerados a partir de dados sintéticos e dados reais. Os resultados são comparados com outras técnicas de geração do circuito classificador (Rede Neural Artificial, Máquina de Vetores de Suporte) mostrando a eficácia do método proposto. Com o método de projeto proposto, obtém-se o circuito classificador com menos portas lógicas e algumas vezes com maior capacidade de generalização do que outros métodos.

É apresentada uma implementação em *hardware* do método de geração do circuito classificador proposto. Foi adotada uma solução baseada em *hardware* reconfigurável em FPGA (*Field Programmable Gate Array*) com sistema de multiprocessamento baseado no processador NIOS II. Algumas medidas de desempenho do sistema implementado em hardware são apresentadas, evidenciando a viabilidade da implementação.

Enfim, este trabalho tem como principais contribuições: um novo método de seleção de amostras baseado no kNN; duas novas métricas de distância entre padrões; um esquema de projeto de circuito digital combinatorial para operar como classificador de padrões binários com capacidade de generalização; e uma proposta de implementação em *hardware/software* de um sistema classificador digital.

Abstract

This work aims at the implementation of classifying binary patterns with digital circuits in order to get a embedded system with the following features: portability, on-line training, operating in real time and with capacity of generalization.

The proposed method makes use of training data filtering (or selection) before digital circuit synthesis. It is proposed an algorithm for minimum selection of samples that is based on the k Nearest Neighbor Rule (kNN). This results on a reduced complexity design phase, less resources of storage and processing, and yields also a degree of generalization capacity of the resulting circuit.

Some examples of designs of digital classifier circuits generated from synthetic data and real data are presented. The results are compared with others techniques such as Artificial Neural Networks and Support Vector Machines, showing the effectiveness of the proposed method. With the proposed design method, the generated circuit classifier works with less logic gates and with higher generalization capacity than some of the other methods.

An implementation in hardware of the method of generation of the proposed circuit classifier is also presented. A solution based on the reconfigurable hardware in FPGA *Field Programmable Gate Array* with multiprocessing based on the NIOS II processor was adopted. Some measures of performance of the system implemented in hardware are presented, showing the viability of the implementation.

Finally, this work has the main contributions: has proposed a new method for sample selection based on kNN; has pressented two new metrics of distance between patterns; has presented a scheme for a digital combinational circuit design working as a binary pattern classifier with generalization capacity; and has presented a proposal for the implementation of a digital classifier system in hardware/software.

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

Neste capítulo é apresentada a motivação para a realização deste trabalho de tese, bem como os seus objetivos principais. É descrito sucintamente o problema para o qual esta tese propõe uma solução. As principais contribuições deste trabalho e a organização geral do texto são mostradas em seguida.

1.1 Classificadores de Padrões Binários

Classificadores de padrões são sistemas capazes de determinar a classe (categoria ou rótulo) de uma dada amostra (ou padrão, instância, exemplo, etc.) dentre um conjunto de categorias conhecidas, vide (Duda, Hart, & Stork 2000). Por exemplo: padrões de vozes digitalizados podem ser identificados e classificados pelos fonemas; imagens digitalizadas de caracteres manuscritos podem ser identificadas em um correspondente código; códigos emitidos por sensores podem ser classificados em algum *status*; etc.

Classificadores de padrões binários podem ser implementados em circuitos digitais combinatoriais, desde que os dados de entrada e suas correspondentes saídas sejam conhecidas. Por meio destes dados conhecidos, é montada uma tabela que relaciona cada entrada binária à sua correspondente saída binária (comumente chamada de tabela verdade). Um exemplo é mostrado na Tabela 1.1 onde x, y e z são as entradas binárias e S é a saída. Na Figura 1.1 são apresentados os mesmos dados em forma gráfica, onde o símbolo "quadrado" representa a saída S em "0", e o símbolo "círculo" representa a saída S em "1".

De posse da tabela de dados binários emprega-se um método de determi-

Tabela 1.1: Exemplo de tabela verdade.

Entradas			Saída
z	y	x	S
0	0	0	1
0	0	1	0
0	1	1	1
0	1	0	1
1	1	0	1
1	1	1	X
1	0	1	0
1	0	0	0

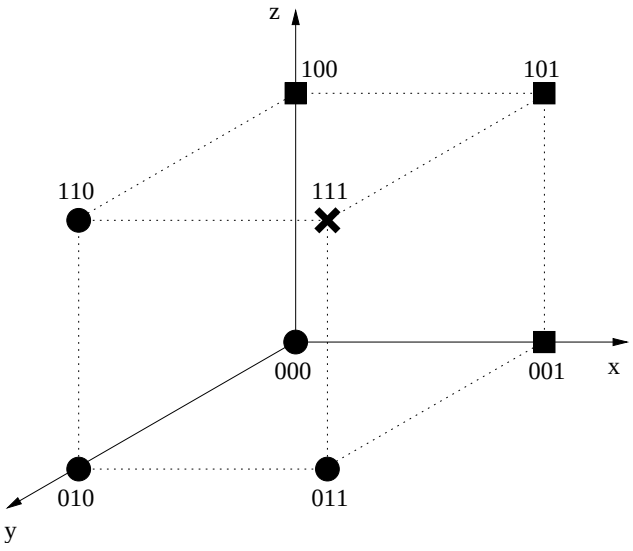


Figura 1.1: Dados binários em formato gráfico.

nação da função Booleana (algoritmo de minimização lógica) que melhor represente a função que relaciona a entrada com a saída. Alguns destes métodos são: mapa de Karnaugh, método tabular de Quine-McCluskey, e Espresso (De Micheli 1994). Por meio da equação Booleana do circuito, na forma de soma de produtos (ou produto de somas), obtém-se diretamente o circuito digital com portas lógicas (portas *AND*, *OR*, *NOT*). O circuito digital pode ser implementado utilizando circuitos integrados digitais ou mesmo gravando-o em um dispositivo lógico programável (PLD - *Programmable Logic Device*) (Ercegovac, Lang, & Moreno 2000) tipo:

- PAL - *Programmable Array Logic*;
- PLA - *Programmable Logic Array*; ou
- FPGA - *Field Programmable Gate Array.*

Em determinados problemas de classificação não são conhecidas todas as possibilidades do espaço de entrada/saída do sistema. Neste caso, a saída do circuito combinatorial é representada por uma situação de "não importa" (*don't care*) para uma entrada desconhecida (sem classificação prévia), antes da fase de projeto do circuito digital. Esta situação é simbolizada por um "X" na saída correspondente à entrada desconhecida (veja Tabela 1.1).

Um algoritmo de minimização lógica tem como objetivo principal gerar a função Booleana que melhor representa a relação das entradas com as saídas, utilizando o menor número de produtos (ou somas), de tal forma a resultar em um circuito com menos portas lógicas (De Micheli 1994). O algoritmo aproveita as situações de *don't care* da tabela de entrada para minimizar a função Booleana geradora do circuito. Assim, para o exemplo mostrado na Tabela 1.1, a função lógica Booleana encontrada é:

$$S = y + \bar{z}.\bar{x} \tag{1.1}$$

o que resulta em um menor número de portas lógicas para implementar o circuito. Entretanto, o circuito resultante associa o dado de entrada desconhecido (saída *don't care*) à saída "1" ("círculo"), o que pode não corresponder ao desejado. A Figura 1.2 apresenta a cobertura obtida pelo minimizador Booleano.

Há uma boa probabilidade de o circuito classificador combinatorial não apresentar a resposta desejada para uma entrada desconhecida, uma vez que não é objetivo da minimização lógica tratar estes dados. Para o problema exemplo dado na Tabela 1.1, talvez a melhor solução seja:

$$S = \bar{z}.\bar{x} + \bar{z}.y + y.\bar{x} \tag{1.2}$$

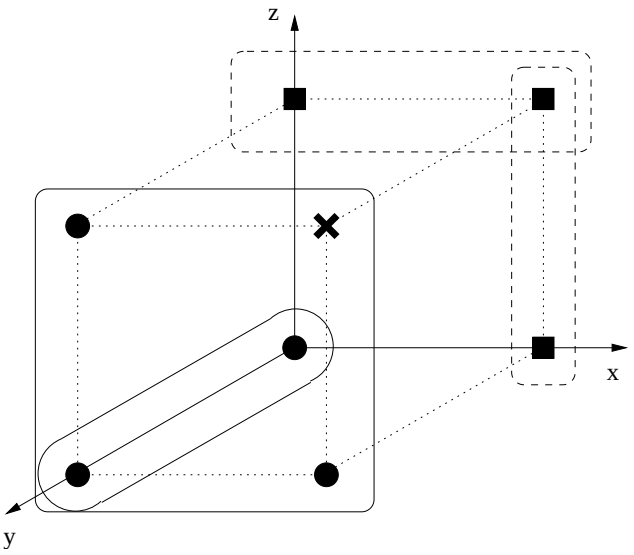


Figura 1.2: Cobertura obtida pelo minimizador Booleano.

o que garante uma resposta equilibrada (mesma quantidade) para a saída em "0" e em "1". Isto resultaria em um maior número de portas lógicas, mas possivelmente em acerto de classificação para a entrada desconhecida. A Figura 1.3 apresenta a cobertura utilizada para obter a Equação 1.2.

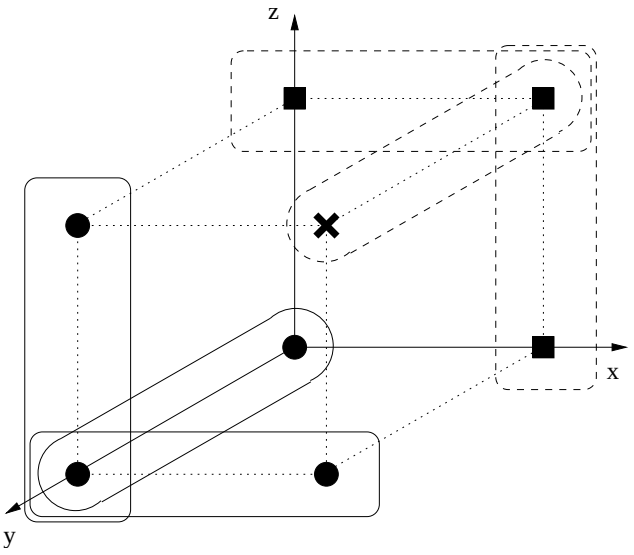


Figura 1.3: Outra opção de cobertura.

Na Figura 1.4 é apresentada uma superfície de separação possível para a resposta do circuito digital definido pela Equação 1.2, levando em conta a simetria do espaço de entrada dos dados binários mostrados na Figura 1.1. Assim, o circuito resultante associa o dado de entrada desconhecido (*don't care*) à saída "0" ("quadrado"), o que pode ser o resultado desejado para o classificador na solução deste problema.

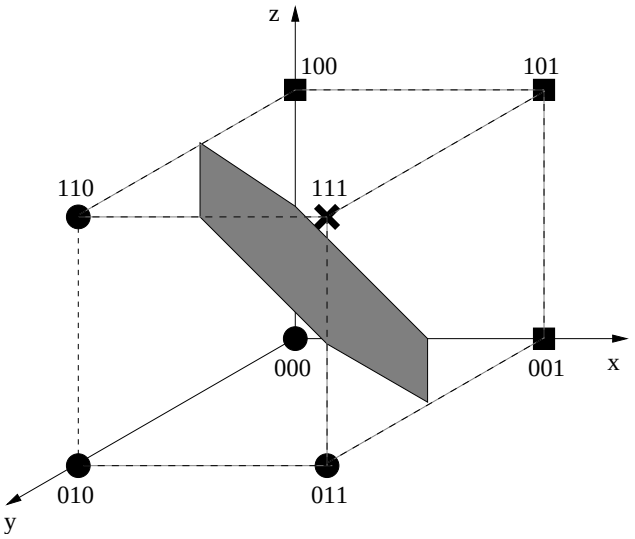


Figura 1.4: Superfície de separação entre as classes “quadrado” e “círculo”.

1.2 Controle da Generalização

Geralmente, em problemas de classificação, é mais interessante acertar a resposta para os dados desconhecidos (não disponíveis na fase de projeto) do que acertar a resposta para os dados conhecidos (dados disponíveis na fase de projeto). Assim, são tolerados erros para os dados de treinamento para incrementar os acertos dos dados desconhecidos.

Algoritmos de geração e minimização de funções Booleanas são fiéis à tabela verdade, ou seja, o circuito digital gerado pela equação lógica resultante de tais algoritmos produz 100% de acertos para os dados conhecidos. Mas a ocorrência de erro mínimo (0%) para os dados conhecidos (treinamento) não significa que o classificador terá um ótimo desempenho na classificação dos dados desconhecidos (Teixeira, Braga, Takahashi, & Saldanha 2000).

Em geral, o classificador pode errar nos dados de treinamento para acertar mais nos dados desconhecidos. Então, uma maneira de melhorar o classificador binário projetado pela sua tabela verdade é eliminar dados conhecidos da tabela verdade que poderiam prejudicar o desempenho do classificador para dados desconhecidos. A dificuldade está justamente em selecionar quais dados devem ser descartados e quais devem ser mantidos para fins de projeto do melhor classificador.

Tanto os dados desconhecidos quanto os dados eliminados da tabela verdade são tratados pelo minimizador de função Booleana como uma situação de *don’t care*. Isto possibilita ao minimizador Booleano expandir os dados conhecidos e retidos para projeto de forma a gerar uma superfície de separação entre as classes que dependerá dos dados *don’t care*. Assim, é possível controlar a forma da superfície de separação do resultado de classificação do

circuito gerado por meio da escolha seletiva dos dados de treinamento a serem descartados.

A superfície de separação não deve ser tão suave a ponto de privilegiar uma resposta (*underfitting*), e nem tão encurvada a ponto de contornar todos os dados de treinamento (*overfitting*). Este é o conhecido "dilema entre a polarização e a variância", amplamente conhecido no contexto de Redes Neurais Artificiais (Haykin 2001).

1.3 Seleção de Padrões

Em uma **Máquina de Vetores de Suporte** (*Support Vector Machine* - SVM), conforme (Vapnik 1998), o hiperplano ótimo de separação das classes é determinado em função dos vetores de suporte, ou seja, amostras que se encontram entre as margens ($\alpha_i = C$) e sobre as margens ($0 < \alpha_i < C$). Margem é definida como a menor distância entre os exemplos do conjunto de treinamento e o hiperplano de separação das classes (vide Figura 1.5), sendo que esta distância é flexível o bastante para separar amostras de classes diferentes sobrepostas espacialmente. Os $\alpha_i s$ são os coeficientes de Lagrange obtidos para cada amostra no processamento do algoritmo do SVM e que determinam, dentre os dados de treinamento, quais são os vetores de suporte. Amostras externas às margens ($\alpha_i = 0$) são ignoradas. Isto inspira a possibilidade de projeto de uma máquina de aprendizado que utilize as amostras conhecidas que estejam perto da margem de separação das classes.

No treinamento de uma máquina de aprendizado com técnica de **Boosting** (Munro 1992; Cachin 1994; Schapire 2002), os padrões com maior erro de classificação têm sua probabilidade de apresentação aumentada. Estes dados mais utilizados para treinamento, selecionados por esta técnica, são justamente aqueles próximos à fronteira de separação das classes, conforme mostrado na Figura 1.6. Novamente, os dados de maior importância são justamente aqueles próximos a fronteira de decisão das classes.

O algoritmo de seleção de amostras proposto por (Wilson 1972) possui a característica de eliminar tanto os dados ruidosos quanto os dados que estão perto da fronteira de separação das classes (ou dados entre as margens das classes). Assim os dados resultantes são aqueles que descrevem puramente cada classe, e podem ser utilizados como dados de treinamento (projeto) para um classificador, garantindo uma melhor generalização para os dados desconhecidos (Wagner 1973) do que utilizando todo o conjunto de treinamento. Na Figura 1.7 são mostrados os dados selecionados pelo algoritmo de Wilson utilizando os dados da Figura 1.5.

Nem todos os dados selecionados pelo algoritmo de Wilson são necessários

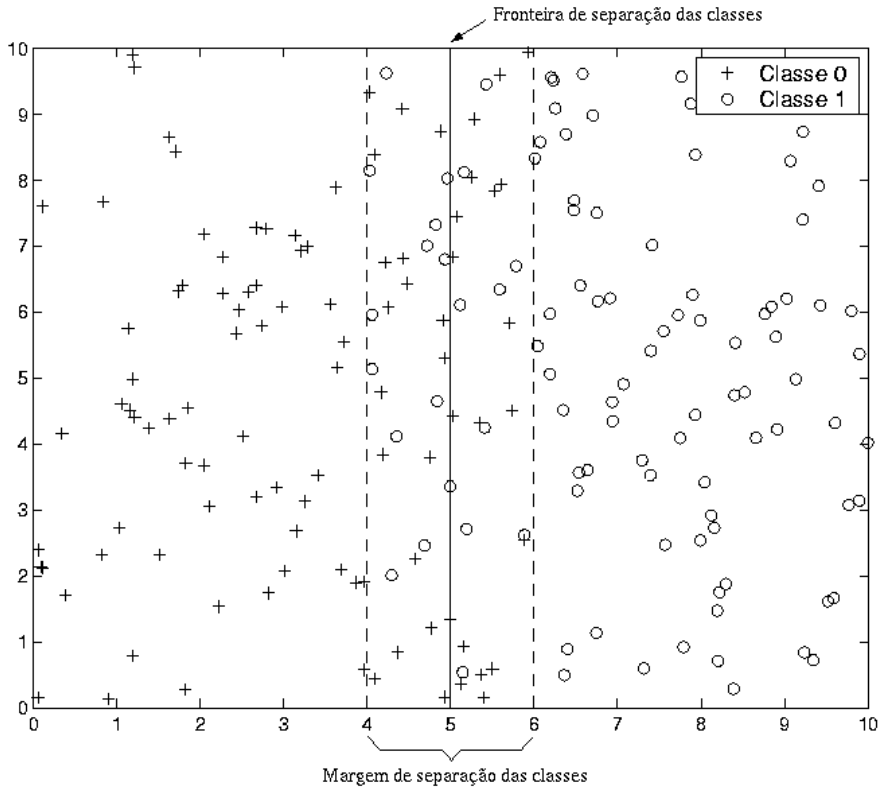


Figura 1.5: Conjunto de dados reais de duas dimensões e duas classes (“cruz” e “círculo”).

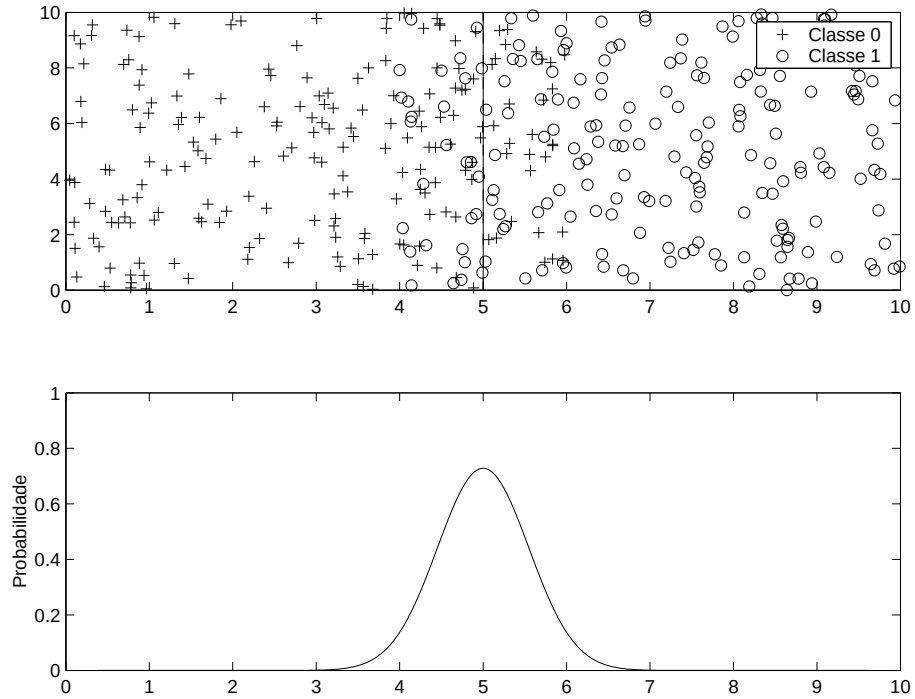


Figura 1.6: Probabilidade de apresentação das amostras durante treinamento de uma máquina de aprendizado utilizando a técnica de Boosting.

para caracterizar uma determinada classe. Dados contidos mais no interior (mais longe da margem) da classe podem também ser descartados e o restante utilizado para treinamento de um classificador sem prejudicar o seu desempenho. Na Figura 1.8 é mostrado o subconjunto de dados obtido após a aplicação da regra de Wilson e posterior eliminação das amostras que estão longe da margem de separação das classes utilizando os dados da Figura 1.5. Os dados de interesse são justamente aqueles que estão fora e próximos às margens de separação das classes, garantindo a generalização necessária ao classificador.

Para o projeto de um classificador de dados binários, o algoritmo de Wilson pode ser utilizado como pré-processamento dos dados de entrada do minimizador de função Booleana, de tal forma a gerar um circuito digital com melhor generalização. Entretanto, quando a quantidade e dimensão dos dados de entrada é muito grande, torna-se dispendioso para o minimizador de função Booleana gerar a equação lógica do circuito digital. Torna-se então necessário diminuir o tamanho dos dados de entrada, eliminando novamente os dados desnecessários.

1.4 Objetivo da Tese

O objetivo principal deste trabalho de tese é a geração de circuitos combinatoriais que atuem como classificadores de padrões binários e que apresentem um resultado de classificação dos dados de entrada desconhecidos (generalização) com um desempenho tão bom ou melhor do que para dados de entrada conhecidos (disponíveis durante a fase de projeto). Ou seja, o circuito digital gerado possui uma probabilidade baixa de classificar erroneamente dados desconhecidos, tendo os dados conhecidos como base de projeto.

O método proposto utiliza o processo de filtragem dos dados binários de entrada (dados conhecidos) de tal forma a eliminar os dados ruidosos ou que estão dentro da margem de separação das classes, e também os dados que estão bem no interior das classes. Os dados selecionados são justamente aqueles fora da margem de separação das classes, mas próximos da fronteira de decisão. Assim, um outro objetivo desta tese é a proposta de um algoritmo de seleção mínima de amostras, baseado na Regra do Vizinho-mais-próximo, para que a fase de projeto do classificador necessite de menos recursos de armazenamento e processamento, sem degradar a capacidade de generalização.

A metodologia de projeto proposta para o classificador digital é resumida na Figura 1.9. Os dados de projeto (ou treinamento) do classificador são selecionados por um algoritmo e posteriormente aplicados ao algoritmo de minimização Booleana. O minimizador Booleano fornece a expressão lógica do

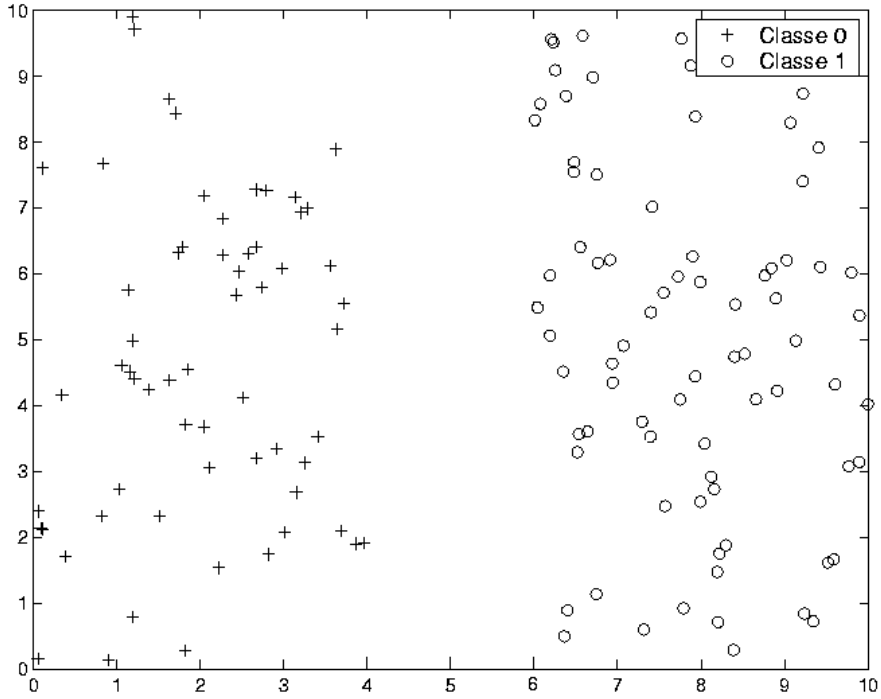


Figura 1.7: Dados selecionados fora da margem.

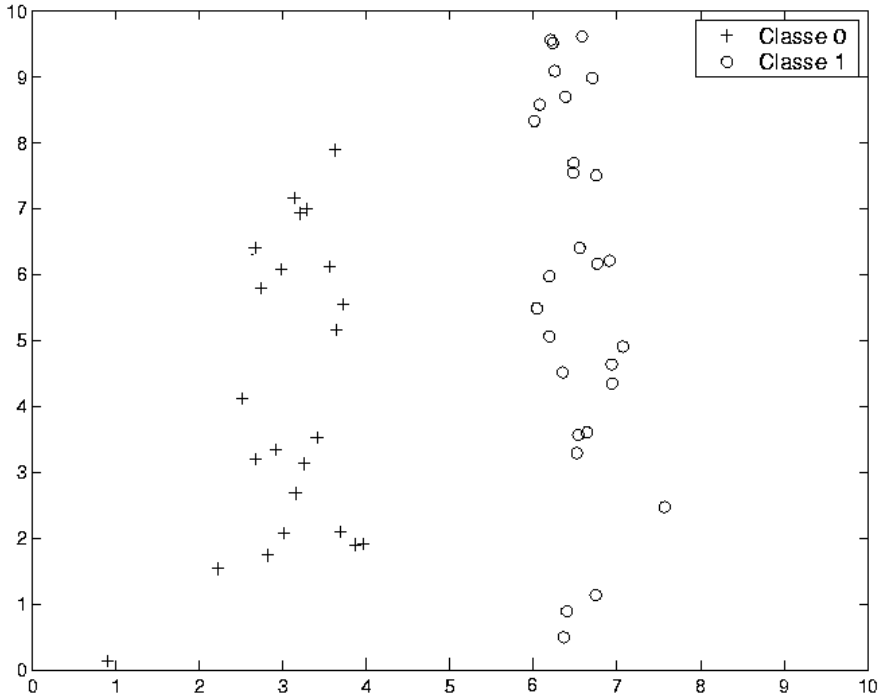


Figura 1.8: Dados selecionados pelo algoritmo de Wilson e posterior redução.

circuito classificador com capacidade de generalização.

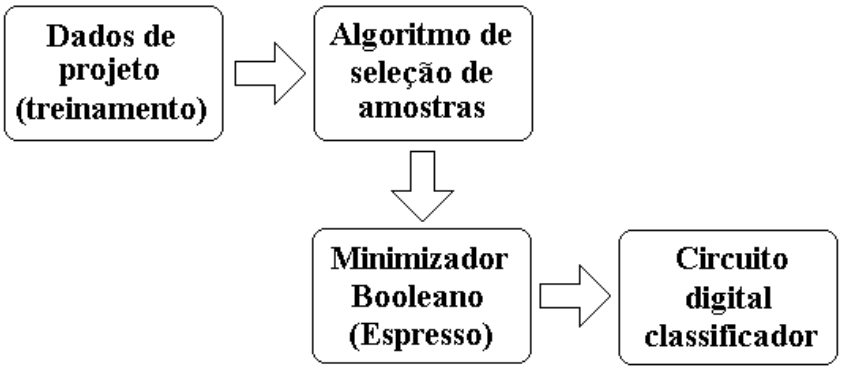


Figura 1.9: Diagrama simplificado do método de projeto proposto para o circuito digital classificador.

É realizada uma comparação do método proposto com outros métodos de classificação conhecidos que possuem um desempenho já comprovado, com o objetivo de validar o método proposto. Estes métodos são:

- a Rede Neural Artificial treinada com o algoritmo multiobjetivo (MOBJ) para controle da generalização (Teixeira 2001); e a Máquina de Vetores de Suporte (vide (Vapnik 1998)) que durante seu processo de treinamento seleciona as amostras que melhor identificam as classes.

Finalmente, objetiva-se implementar um sistema classificador de padrões binários em um circuito embutido com características de portabilidade, funcionamento rápido em tempo real e com treinamento *on-line* para atuar em problemas críticos de classificação de padrões.

1.5 Principais Contribuições

Durante o desenvolvimento da metodologia de projeto e implementação do circuito classificador com capacidade de generalização, surgiram as seguintes principais contribuições contidas neste trabalho de tese:

- O desenvolvimento de um novo método de seleção de amostras baseado no kNN para treinamento de máquinas de aprendizado;
- A definição de uma nova métrica de distância entre padrões baseada nos coeficientes da verossimilhança da Regra de Bayes;
- Uma modificação na métrica VDM para melhoramento do classificador kNN utilizando esta métrica na solução de problemas específicos;
- Uma dedução e formulação probabilística para a métrica Distância de Discriminação;

- Um esquema de projeto de circuito digital combinatorial para operar como classificador de padrões binários com capacidade de generalização;
- Uma proposta de implementação em hardware/software de um sistema classificador autônomo, portátil e versátil com capacidade de treinamento on-line e resposta com tempo de atraso mínimo;
- Uma técnica para emular uma PAL dentro de uma FPGA para implementação de circuitos combinatoriais de dois níveis.

1.6 Organização do Texto

Este trabalho de tese está organizado da seguinte maneira:

- O Capítulo 2 apresenta o Sistema de Máquina de Aprendizado. Os principais conceitos relacionados ao assunto são descritos, bem como as principais dificuldades no projeto e implementação de um Sistema de Aprendizado. Métodos de projeto derivados do risco do erro são descritos.
- Os Padrões Binários são caracterizados no Capítulo 3. A forma de codificação binária dos dados para garantir ótima generalização por um classificador é mostrada, juntamente com as principais propriedades dos dados binários. Os conceitos principais relacionados ao espaço Booleano de alta dimensão são descritos. É apresentada uma revisão de métodos de Síntese de Circuitos Digitais, com a apresentação dos conceitos utilizados. Os algoritmos de minimização de funções lógicas Quine-McCluskey e Espresso são descritos detalhadamente.
- No Capítulo 4 é apresentado o problema de Reconhecimento de Padrões. As técnicas de classificação de padrões baseadas em métodos paramétricos e não paramétricos são exemplificadas, bem como as suas características. É dada uma ênfase especial à Regra do Vizinho-mais-próximo, por ser a técnica na qual se baseou o método proposto neste trabalho. Assim, é apresentada a evolução desta técnica encontrada na literatura. Algumas métricas de similaridade são descritas e algumas são desenvolvidas. Experimentos com a Regra do Vizinho-mais-próximo são realizados para validar as métricas.
- O Capítulo 5 apresenta alguns métodos de seleção das amostras baseados na Regra do Vizinho-mais-próximo, os quais diminuem a quantidade de amostras de treinamento para armazenamento e processamento, além de melhorar a generalização dos classificadores projetados pelas

amostras selecionadas. É proposto um algoritmo de seleção de amostras derivado dos métodos conhecidos, e alguns experimentos são realizados para validar a proposta.

- No Capítulo 6 é apresentada a metodologia de projeto proposta para a geração do circuito digital classificador com ótima generalização. Um dos métodos de minimização de função Booleana é utilizado em conjunto com o algoritmo de seleção de amostras descrito no capítulo anterior para garantir a capacidade de generalização do classificador. Resultados são obtidos e comparados com outras técnicas para geração do circuito digital com capacidade de generalização.

- A implementação em hardware do circuito classificador proposto é apresentada no Capítulo 7. Foi adotada uma solução baseada em hardware reconfigurável em FPGA com sistema de multiprocessamento baseado no processador NIOS II. São descritas as ferramentas utilizadas para o projeto e implementação do sistema classificador. Algumas medidas de desempenho do sistema implementado em hardware são também apresentadas.

- No Capítulo 8 são apresentadas as Conclusões e Propostas de Continuidade deste trabalho, o qual acredita-se ser possível a sua exploração futura em vários aspectos.

CAPÍTULO
2

Aprendizagem de Máquina

Ciência moderna e engenharia são baseadas em modelos analíticos para descrever sistemas físicos, biológicos e sociais. Entretanto, em muitas aplicações estes modelos analíticos são desconhecidos ou o sistema sob estudo é muito complexo para ser analiticamente descrito.

Com o crescente uso de computadores e sensores de baixo custo para coleta de dados, há um grande volume de dados sendo gerados por tais sistemas. Na ausência de modelos analíticos, tais dados prontamente disponíveis podem ser usados para derivar modelos pela estimação de relações úteis entre variáveis do sistema, isto é, dependências entrada-saídas desconhecidas.

Um método de aprendizagem é um algoritmo, usualmente implementado em software, que estima um mapeamento desconhecido (dependência) entre entradas e saídas do sistema baseado nos dados disponíveis, isto é, nas amostras (entradas, saídas) conhecidas, segundo (Cherkassky & Mulier 1998). Uma vez que a dependência tenha sido estimada, ela pode ser usada para a predição das saídas futuras do sistema.

Nas seções que se seguem são detalhados o problema de aprendizado e suas características.

2.1 Sistemas de Aprendizado

Um sistema de aprendizado pode realizar as seguintes tarefas específicas:

- Classificação: reconhecimento de padrões ou estimação de fronteiras (limites) de decisão de classe.

- Regressão: estimação de funções contínuas desconhecidas de dados ruidosos.
- Estimação: da densidade de probabilidade das amostras.

Para a realização das tarefas, o sistema de aprendizado utiliza os seguintes estágios de operação em seqüência:

1. Aprendizagem/estimação baseada nas amostras de treinamento; e
2. Operação/predição quando predições são feitas com as amostras de teste.

Basicamente, o aprendizado de um sistema pode ser supervisionado ou não supervisionado. Aprendizado supervisionado é usado para estimar um mapeamento (entrada/saída) desconhecido baseado em amostras (entrada/saída) conhecidas. Classificação e regressão são exemplos de tarefas deste tipo. O termo supervisionado indica o fato de que valores de saída para amostras de treinamento são conhecidos.

Em aprendizado não supervisionado, apenas amostras de entrada são apresentadas ao sistema de aprendizado, e não há noção da saída durante o aprendizado. O objetivo do aprendizado não supervisionado pode ser estimar a distribuição de probabilidade das entradas ou descobrir uma estrutura natural, isto é, agrupamentos nos dados de entrada.

A distinção entre aprendizado supervisionado e não supervisionado está apenas no nível da declaração do problema (vide (Cherkassky & Mulier 1998)). Isto não implica que métodos originalmente desenvolvidos para aprendizado supervisionado não possam ser utilizados (com pequenas modificações) para tarefas de aprendizado não supervisionado, e vice-versa.

O problema de aprendizagem/estimação da dependência dos dados é apenas parte do procedimento experimental geral para tirar conclusões dos dados. O procedimento experimental geral adotado em estatística clássica envolve as seguintes etapas:

1. Declaração do problema;
2. Formulação de hipóteses: especifica uma dependência desconhecida, a qual deve ser estimada a partir dos dados experimentais;
3. Geração de dados/Projeto do experimento: artificial ou natural, observado e/ou controlado;
4. Coleta de dados e pré-processamento: escalonamento, codificação;
5. Estimação do modelo: previsão precisa, capacidade de generalização;
6. Interpretação do modelo e esboço das conclusões.

Mesmo que as etapas do procedimento experimental sejam precisas e confiáveis, o modelo resultante não poderá ser válido se os dados não são informativos ou a formulação do problema não é estatisticamente significativa. Na seção seguinte, os dados (ou variáveis) são caracterizados.

2.2 Caracterização das variáveis

As variáveis (ou características, atributos) de entrada e saída podem ser basicamente de dois tipos: numérico e categórico. Tipo numérico inclui valores reais ou inteiros (ex.: idade, velocidade, comprimento, etc). Uma característica numérica tem duas importantes propriedades: seus valores têm uma relação de ordem e uma relação de distância entre dois valores de característica. Saídas numéricas (valores reais) correspondem a problemas de regressão ou estimação de funções contínuas.

Variável categórica (ou simbólica) não tem ordem ou relação de distância definida. Dois valores de uma variável categórica podem ser iguais ou diferentes. Exemplos: cor, sexo, país. Saídas categóricas representam uma classe de problemas conhecidos como reconhecimento de padrões, classificação ou análise de discriminação.

Uma variável categórica com dois valores pode ser convertida, a princípio, a uma variável numérica binária com dois valores (0 e 1). Uma variável categórica com J valores pode ser convertida em J variáveis numéricas binárias, uma variável binária para cada valor categórico. Este método é conhecido como código 1-de-J, indicando que cada uma das J variáveis binárias codifica um valor característico. Exemplo para $J = 8$:

- Variável 1: “00000001”
- Variável 2: “00000010”
- Variável 8: “10000000”

Há dois outros tipos de variáveis menos comuns: periódico e ordinal. Uma variável periódica é uma variável numérica para a qual a relação de distância existe, mas não há relação de ordem (ex.: dias da semana). Uma variável ordinal é uma variável categórica para a qual uma relação de ordem existe, mas não há relação de distância (ex.: *ranking* de melhor jogador).

A quantidade de características para cada entrada de um sistema determina a dimensão do espaço de entrada. O aumento da dimensão dos dados de entrada provoca um crescimento exponencial da complexidade do problema de aprendizado, efeito conhecido como “maldição da dimensionalidade” (vide (Haykin 2001)), tratado na próxima seção.

2.3 Maldição da Dimensionalidade

No problema de aprendizado, o objetivo é estimar uma função usando um número finito de amostras de treinamento. O número finito de amostras de treinamento implica que qualquer estimativa de uma função desconhecida é sempre imprecisa. Estimação significativa é possível apenas para funções suficientemente suaves, onde a suavidade da função é medida com respeito à densidade de amostras dos dados de treinamento. Para funções de alta dimensão torna-se difícil coletar amostras suficientes para se conseguir esta alta densidade. Este problema é comumente referido como a "maldição da dimensionalidade" (*curse of dimensionality*) conforme (Cherkassky & Mulier 1998).

A "maldição da dimensionalidade" é devida à geometria do espaço de alta dimensão. As propriedades do espaço de alta dimensão geralmente aparecem como contra senso porque a experiência humana com o mundo físico é em espaço de baixa dimensão. Conceitualmente, objetos em espaço de alta dimensão têm uma maior área de superfície para um dado volume do que objetos em espaço de baixa dimensão. A distribuição em alta dimensão possui as seguintes propriedades:

- Tamanho das amostras que produzem a mesma densidade incrementam exponencialmente com a dimensão. Ex.: um conjunto de amostras de n dados em R^1 terá a mesma densidade de pontos em R^d se o número de pontos for n^d;

- Um raio maior é necessário para encobrir uma fração dos pontos de dados em um espaço de alta dimensão;

- Todo ponto fica mais próximo a uma borda (margem) do que a outro ponto, o que requer extrapolação pela máquina de aprendizado.

A "maldição da dimensionalidade" tem sérias conseqüências quando se está tratando com número finito de amostras em espaço de alta dimensão. Há dificuldade em predizer uma resposta em um dado ponto, uma vez que qualquer ponto estará na média mais próximo a uma borda do que a um ponto do conjunto de treinamento.

2.4 O Problema de Aprendizado

O objetivo do aprendizado preditivo é estimar dependências não conhecidas entre as variáveis de entrada (x) e a saída (w) de um conjunto de observações

passadas de valores $(\mathrm{x}_i, \mathrm{w}_i)$, sendo $i = 1, ..., n$, como mostra a Figura 2.1 segundo (Cherkassky & Mulier 1998). O outro conjunto de variáveis rotuladas z denota todos os outros fatores que afetam a saída mas cujos valores não são observados ou controlados. Portanto, o conhecimento de valores de entrada observáveis (x) não especifica unicamente as saídas (w). Esta incerteza nas saídas reflete a falta de conhecimento dos fatores não observados (z), e isto resulta em dependência estatística entre os dados observados e saídas. O efeito de entradas não observáveis (z) pode ser caracterizado pela distribuição de probabilidade condicional $p(\mathrm{w}|\mathrm{x})$, o qual indica a probabilidade de que w ocorrerá dada a entrada x.

Figura 2.1: Um sistema com entradas-saída.

O cenário geral de aprendizado envolve três componentes: um **gerador** de vetores de entrada aleatório, um **sistema** que retorna uma saída para cada vetor de entrada, e a **máquina de aprendizado** a qual estima um mapeamento (entrada, saída) desconhecido do sistema baseado nas amostras (entrada, saída) observadas (veja Figura 2.2).

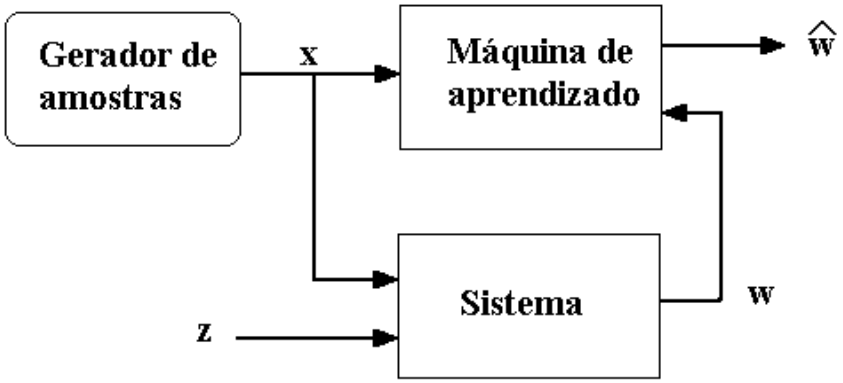


Figura 2.2: Um sistema de aprendizado com entradas-saída.

O gerador produz vetores aleatórios $\mathrm{x} \in R^d$ tirados independentemente de uma densidade de probabilidade fixa $p(\mathrm{x})$, que é desconhecida. Em geral, o modelador (máquina de aprendizado) não tem controle sobre quais vetores de entrada são fornecidos ao sistema.

O sistema produz um valor de saída w para todo vetor de entrada x de acordo com uma densidade de probabilidade condicional $p(\mathrm{w}|\mathrm{x})$, a qual é também desconhecida. Esta descrição inclui o caso específico de um sistema determinístico onde $\mathrm{w} = f(\mathrm{x})$, e também o caso da formulação de regressão de

$\mathbf{w} = f(\mathbf{x}) + \epsilon$, onde ϵ é um ruído aleatório com média zero. Sistemas reais raramente têm saídas verdadeiramente aleatórias; entretanto eles geralmente têm entradas não medidas (**z**). Estatisticamente, o efeito da variação das entradas não observáveis (**z**) na saída do sistema pode ser caracterizado como aleatório e representado como uma distribuição de probabilidade.

No caso mais geral, a máquina de aprendizado é capaz de implementar um conjunto de funções $f(\mathbf{x}, \omega)$, $\omega \in \Omega$, onde Ω é um conjunto de parâmetros abstratos usados apenas para indexar o conjunto de funções. Na formulação, o conjunto de funções implementadas pela máquina de aprendizado pode ser qualquer conjunto de funções, escolhido *a priori*, antes do processo de aprendizado ser iniciado.

O problema encontrado pela máquina de aprendizado é selecionar uma função, do conjunto de funções que ele suporta, que melhor aproxima a resposta do sistema. A máquina de aprendizado é limitada a observar um número finito de exemplos (n) em seqüência para produzir esta seleção. Estes dados de treinamento produzidos pelo gerador e pelo sistema serão independentes e identicamente distribuídos (iid) de acordo com a função de densidade de probabilidade conjunta (pdf - probability density function) indicada na Equação 2.1.

$$p(\mathbf{x}, \mathbf{w}) = p(\mathbf{x})p(\mathbf{w}|\mathbf{x}) \tag{2.1}$$

A qualidade da aproximação produzida pela máquina de aprendizado é medida pela perda $L(\mathbf{w}, f(\mathbf{x}, \omega))$ ou discrepância entre a saída produzida pelo sistema e pela máquina de aprendizado para um dado ponto **x**. Por convenção, a perda recebe valores não-negativos, de tal forma que grandes valores positivos correspondem à aproximação pobre. O valor esperado da perda é chamado risco funcional, dado pela Equação 2.2.

$$R_{func}(\omega) = \int L(\mathbf{w}, f(\mathbf{x}, \omega))p(\mathbf{x}, \mathbf{w})d\mathbf{x}d\mathbf{w} \tag{2.2}$$

Aprendizado é o processo de estimar a função $f(\mathbf{x}, \omega_0)$, a qual minimiza o risco funcional sobre o conjunto de funções suportadas pela máquina de aprendizado, usando apenas os dados de treinamento ($p(\mathbf{x}, \mathbf{w})$ não é conhecido). Com dados finitos, não se pode esperar encontrar $f(\mathbf{x}, \omega_0)$ exatamente; então é indicado $f(\mathbf{x}, \omega^*)$ como a estimativa da solução ótima obtida com dados de treinamento finitos usando algum procedimento de aprendizado. É claro que qualquer tarefa de aprendizado pode ser solucionada pela minimização de $R_{func}(\omega)$ se a densidade $p(\mathbf{x}, \mathbf{w})$ é conhecida.

Em um problema de classificação binária (duas classes), a saída do sistema toma apenas dois valores simbólicos ($\mathbf{w} = \{0, 1\}$) correspondendo às duas classes. Conseqüentemente, a saída da máquina de aprendizado necessita

apenas de escolher bem um entre dois valores. Então o conjunto de funções $f(\mathbf{x}, \omega)$, $\omega \in \Omega$ torna-se um conjunto de funções indicadoras. Uma função de perda comumente usada para este problema é mostrada na Equação 2.3, a qual mede o erro de classificação.

$$L(\mathbf{w}, f(\mathbf{x}, \omega)) = \begin{cases} 0 & \text{se } \mathbf{w} = f(\mathbf{x}, \omega) \\ 1 & \text{se } \mathbf{w} \neq f(\mathbf{x}, \omega). \end{cases} \tag{2.3}$$

Usando esta função de perda, o risco funcional (Equação 2.2) quantifica a probabilidade de erro na classificação. O aprendizado então torna-se o problema de encontrar a função indicadora $f(\mathbf{x}, \omega_0)$ (classificador) a qual minimiza a probabilidade de erro na classificação usando apenas os dados de treinamento.

2.5 Aprendizado Adaptativo

Com amostras finitas, é sempre melhor resolver diretamente um exemplo particular do problema de aprendizado do que tentar resolver um problema mais geral (e muito mais difícil) de estimação de densidade conjunta entre entrada e saída (vide (Cherkassky & Mulier 1998)). Os métodos clássicos podem não ser apropriados para muitas aplicações porque modelagem paramétrica (com amostras finitas) impõem muitas suposições rígidas sobre a dependência desconhecida, o que especifica sua forma paramétrica. Isto tende a introduzir grande polarização na modelagem, isto é, a discrepância do modelo paramétrico assumido e a (desconhecida) verdade.

Igualmente, métodos não paramétricos clássicos funcionam apenas em casos assintóticos (tamanho das amostras muito grande).

As limitações da abordagem clássica motivam o uso de métodos adaptativos (ou flexíveis). Métodos adaptativos conseguem maior flexibilidade especificando uma larga classe de funções de aproximação. O modelo de predição é então selecionado desta larga classe de funções. O principal problema torna-se escolher o modelo de complexidade ótima (flexibilidade) para os dados finitos à disposição.

Dois dos princípios indutivos que são mais comumente utilizados em processos de aprendizado adaptativos são a minimização do risco empírico (*Empirical Risk Minimization* - ERM) e a minimização do risco estrutural (*Structural Risk Minimization* - SRM) segundo (Cherkassky & Mulier 1998), assuntos das próximas subseções.

2.5.1 Princípio da Minimização do Risco Empírico

O objetivo do aprendizado preditivo é estimar dependências desconhecidas em uma classe de funções de aproximação usando dados disponíveis. A estimativa ótima corresponde ao mínimo do risco funcional esperado. O problema é que o risco funcional depende da cdf (*cumulative probability distribution function*) $F(\mathbf{x}, \mathbf{w})$ desconhecida, ou pdf $f(\mathbf{x}, \mathbf{w})$ também desconhecida. A única informação disponível sobre esta distribuição está nas amostras de treinamento disponíveis.

Uma solução para o problema de aprendizado é estimar a cdf $F(\mathbf{x}, \mathbf{w})$ desconhecida, ou pdf $f(\mathbf{x}, \mathbf{w})$ desconhecida, dos dados disponíveis e então encontrar uma estimativa ótima $f(\mathbf{x}, \omega_0)$. Outra solução é procurar uma estimativa que garanta o mínimo do risco empírico conhecido, como um substituto para o risco funcional desconhecido. Esta abordagem é chamada de minimização do risco empírico (ERM - *Empirical Risk Minimization*). Com amostras finitas, a abordagem ERM é sempre preferível à estimação da densidade conforme (Cherkassky & Mulier 1998).

O princípio indutivo ERM é tipicamente usado em problemas clássicos (paramétricos) onde o modelo é especificado primeiro e então seus parâmetros são estimados dos dados. Esta abordagem trabalha bem apenas quando o número de dados de treinamento é grande relativamente à complexidade do modelo pré-especificado (ou o número de parâmetros livres).

Usando o princípio indutivo da minimização do risco empírico, empiricamente é estimada a função de risco usando os dados de treinamento. O risco empírico é o risco médio para os dados de treinamento, e pode ser minimizado escolhendo os parâmetros apropriados.

Sob o princípio indutivo da minimização do risco empírico (ERM), é procurada uma solução $f(\mathbf{x_i}, \omega^*)$ que minimize o risco empírico (erro de treinamento) como um substituto para o risco funcional esperado. Para estimação do risco funcional esperado para classificação usando a média do risco empírico sobre os dados utiliza-se a Equação 2.4:

$$R_{emp}(\omega) = \frac{1}{n} \sum_{i=1}^{n} I(\mathbf{w}_i \neq f(\mathbf{x_i}, \omega)) \tag{2.4}$$

onde $I(\cdot)$ é a função indicadora que toma o valor 1 se seu argumento é verdadeiro e 0 se falso, e $f(\mathbf{x_i}, \omega)$ é a regra de decisão de classificação.

Uma propriedade geral necessária para qualquer princípio indutivo é a assintoticidade, ou seja, a estimação obtida pelo ERM deve convergir para o valor verdadeiro (ou melhor valor possível) com o número de amostras de treinamento grande. Um objetivo inicial da teoria de aprendizado é formular as

condições sobre as quais o princípio ERM é consistente (assintótico).

Considerando a aplicação do princípio ERM ao problema de aprendizado preditivo, suponha que $R_{emp}(\omega^*|n)$ indique o valor ótimo do risco empírico fornecido pela função de perda $L(\mathbf{x}, \mathbf{w}, \omega^*|n)$ minimizando o risco empírico para n amostras iid, e $R_{func}(\omega^*|n)$ indique o valor desconhecido do risco funcional para a mesma função $L(\mathbf{x}, \mathbf{w}, \omega^*|n)$. Então o princípio ERM é consistente se o risco funcional (desconhecido) $R_{func}(\omega^*|n)$ e o risco empírico $R_{emp}(\omega^*|n)$ convergem para o mesmo limite de risco mínimo $R(\omega_0)$ quando o número de amostras cresce para infinito (vide (Cherkassky & Mulier 1998)). Isto é resumido pelas Equações 2.5 e 2.6, e a noção de consistência é mostrada pelo gráfico da Figura 2.3.

$$R_{func}(\omega^*|n) \rightarrow R(\omega_0) \text{ quando } n \rightarrow \infty \tag{2.5}$$

$$R_{emp}(\omega^*|n) \rightarrow R(\omega_0) \text{ quando } n \rightarrow \infty \tag{2.6}$$

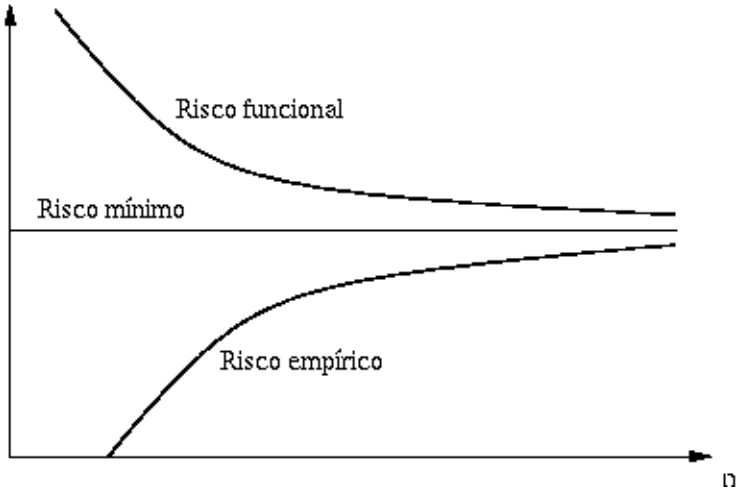


Figura 2.3: Risco empírico e risco funcional.

Assumindo um problema de classificação, o risco empírico corresponde à probabilidade de erro de classificação para os dados de treinamento (erro de treinamento), e o risco funcional esperado é a probabilidade de erro médio de classificação sobre a distribuição $p(\mathbf{x}, \mathbf{w})$ desconhecida, onde $(\mathbf{x}_i, \mathbf{w}_i)$ são as amostras de treinamento. Para um dado conjunto de amostras, espera-se $R_{emp}(\omega^*|n) < R_{func}(\omega^*|n)$ porque a máquina de aprendizado sempre escolhe uma função (estimada) que minimiza o risco empírico mas não necessariamente o risco funcional. Em outras palavras, funções $L(\mathbf{x}, \mathbf{w}, \omega^*|n)$ produzidas pelo princípio ERM para um conjunto de amostras de tamanho n são sempre estimativas polarizadas da melhor função de minimização do risco funcional.

2.5.2 Princípio da Minimização do Risco Estrutural

Com métodos de modelagem adaptativos ou flexíveis, o modelo não é conhecido, e ele é estimado usando um grande número de modelos candidatos (isto é, funções de aproximação de máquina de aprendizado) para descrever os dados avaliados. O principal objetivo é a escolha do modelo candidato de complexidade certa para descrever os dados de treinamento.

O objetivo do aprendizado é escolher um elemento ótimo da estrutura e estimar seus coeficientes por meio das amostras de treinamento. A escolha do modelo da máquina de aprendizado de complexidade ótima resulta no mínimo do risco funcional esperado. Assim, a estimativa do risco funcional esperado pode ser usada para a seleção do modelo.

O princípio indutivo chamado minimização do risco estrutural (*Structural Risk Minimization* – SRM) providencia um mecanismo formal para a escolha de um modelo com complexidade ótima para amostras finitas. Sob SRM, o conjunto S de funções de perda $L(\mathbf{x}, \mathbf{w}, \omega), \omega \in \Omega$ possui uma estrutura, a qual consiste dos elementos $S_k = \{L(\mathbf{x}, \mathbf{w}, \omega), \omega \in \Omega_k\}$ tal que:

$$S_1 \subset S_2 \subset \ldots S_k \subset \ldots$$

Para um conjunto de dados de treinamento $\{(\mathbf{x}_1, \mathbf{w}_1), (\mathbf{x}_2, \mathbf{w}_2), \ldots, (\mathbf{x}_n, \mathbf{w}_n)\}$, o princípio SRM seleciona a função $L_k(\mathbf{x}, \mathbf{w}, \omega|n)$ minimizando o risco empírico para as funções de S_k. Então, para cada elemento da estrutura S_k, o risco garantido é encontrado. No final é escolhido um elemento da estrutura ótima $S_{\text{ótimo}}$ que garanta risco funcional mínimo. Este subconjunto $S_{\text{ótimo}}$ é um conjunto de funções tendo complexidade ótima para um determinado conjunto de dados.

O SRM providencia caracterização quantitativa do compromisso entre a complexidade das funções de aproximação e a qualidade da adaptação (*fitting*) dos dados de treinamento (vide (Cherkassky & Mulier 1998)). Enquanto a complexidade (subconjunto de índice k) aumenta, o mínimo do risco empírico decresce, ou seja, a qualidade dos ajustes dos dados aumenta. O SRM escolhe um elemento ótimo da estrutura que produz o limite mínimo do risco funcional.

O princípio SRM não especifica uma estrutura particular. Entretanto, aplicação de sucesso do SRM na prática depende da estrutura escolhida.

2.6 Conclusão

Uma máquina de aprendizado é capaz de predizer as saídas futuras do sistema baseado em um conjunto de dados de treinamento. Várias caracte-

rísticas dos dados de treinamento devem ser levadas em consideração para garantir que a máquina de aprendizado seja capaz de realizar sua tarefa com precisão: quantidade de amostras, dimensão das amostras, tipos de atributos, etc.

O objetivo básico do aprendizado de máquina é estimar as dependências não conhecidas entre as entradas de dados e a saída do sistema, por meio das entradas/saídas conhecidas. Entretanto, nem todas as entradas do sistema em estudo podem ser observadas, o que aumenta a complexidade da máquina para realizar a sua tarefa de predição. O risco funcional indica o valor esperado para o erro de predição produzido pela máquina de aprendizado. Sendo assim, idealmente a máquina de aprendizado deve ter risco funcional mínimo.

O risco funcional só é conhecido se for conhecida a distribuição conjunta dos dados de treinamento. Como na maioria dos problemas isto não é possível, então pode-se substituir o risco funcional pelo risco empírico (função de risco estimada pelos dados de treinamento). Assim, o princípio da minimização do risco empírico é um método de modelagem da máquina de aprendizado pelo ajuste de seus parâmetros, quando a quantidade de dados disponíveis é grande. Outro método de aprendizado adaptativo é baseado no princípio da minimização do risco estrutural. Este método seleciona a melhor complexidade do modelo da máquina de aprendizado a partir dos dados de treinamento, e que ainda garanta risco funcional mínimo.

CAPÍTULO
3

Padrões Binários e Síntese de Circuitos Digitais

Como o objetivo principal deste trabalho é a geração de circuitos digitais classificadores capazes de classificar dados binários de entrada, então são apresentados neste capítulo as principais características inerentes a conjunto de dados binários e os principais métodos de minimização Booleana para projeto de circuitos digitais a partir destes conjuntos de dados.

3.1 Padrões Binários

Padrões binários são basicamente vetores de 0 e 1's, também chamados de *strings* binárias. São utilizados para representar ou codificar alguma informação, seja numérica ou nominal, por exemplo: valores de temperatura, velocidade, uma imagem ou som digitalizado, presença ou ausência de atributos, etc.

Os números de 0 a 10, por exemplo, podem ser representados por padrões binários. Isto poderia ser feito por 4 dígitos binários codificados pelo modo usual (0000 representando o decimal 0, 0001 representando o decimal 1, 0010 representando o decimal 2, 0011 representando o decimal 3, e assim por diante até o decimal 10 representado como 1010). Isto é chamado de representação distribuída porque os 1's no padrão aparecem em todas as 4 posições binárias dependendo da escolha do código. Alternativamente, dez bits binários poderiam ser escolhidos para apresentar os mesmos números colocando o 1 no enésimo lugar a direita para o enésimo número. Assim, o

decimal 0 é representado como 0000000000, o decimal 1 como 0000000001, o decimal 2 como 0000000010, o decimal 3 como 00000000100 e assim por diante até o decimal 10 representado como 1000000000. Isto é chamado de representação localizada porque o código usa a localização do 1 para representar o dado conforme (Aleksander 1995). Outras formas de representação em código binário podem ser encontradas na literatura, cada uma com uma característica diferente.

É geralmente assumido que dois padrões binários são similares se eles diferem apenas em uns poucos bits de seus valores de entrada. Mas isto não pode ser assumido para o caso geral. Alguns códigos numéricos binários convencionais, os quais são numericamente adjacentes, têm códigos muitos diferentes (por exemplo: o código 15 é 01111 e o 16 é 10000). Em outras aplicações, dois padrões são ditos similares se eles contém o mesmo número de 1's ou se eles representam o mesmo padrão em diferentes posições deslocadas em um registrador.

A generalização de atributos numéricos pode ser relacionada a diferentes métricas de similaridade (ou distância) entre padrões. Em redes neurais artificiais sem peso (*weightless artificial neural networks*) implementadas em memória tipo RAM (*Random Access Memory*) (Braga 1995), por exemplo, a generalização é geralmente relacionada a distância Hamming (quantidade de bits diferentes). Neste caso é importante transformar números em *string* de bits de tal modo que a proximidade numérica seja transformada em proximidade de Hamming (Rohwer & Morciniec 1995).

Nas seções que se seguem, são apresentados o problema de classificação de dados binários e a capacidade de generalização do classificador de acordo com a forma de representação dos dados binários. As características dos padrões binários são também levantadas bem como o do espaço Booleano.

3.1.1 Classificação Binária

Em um processamento convencional, problemas de classificação binária podem ser facilmente implementados programando um algoritmo simples tal como:

1. Armazene os padrões conhecidos (amostras de treinamento);
2. Quando um padrão desconhecido é encontrado, calcule a sua distância de Hamming aos padrões conhecidos e classifique-o de acordo com a menor distância.

A dificuldade com esta técnica ocorre quando a quantidade de padrões é grande e quando a definição das classes é vaga. Não há garantia, neste caso,

que a medida da distância de Hamming forneça a resposta correta. Logo, outras métricas devem ser empregadas, como será apresentado no capítulo 4.

3.1.2 Generalização Binária

Propriedades de generalização dependem fortemente da forma de representação da entrada que está sendo usada no sistema de classificação. O código binário convencional permite uma representação natural de uma variável inteira, mas possui propriedade não-linear e não monotonicidade. Especificamente, valores inteiros diferentes apenas por um nível podem ter sua distância de Hamming entre um e o máximo.

Uma máquina de aprendizado assume que todo bit do padrão de entrada tem a mesma importância relativa. Este não é certamente o caso da notação binária natural, onde os bits são ordenados em nível incremental de importância. Um método eficiente para generalizar distância Hamming usa uma combinação de técnicas de codificação CMAC e Gray (Rohwer & Morciniec 1995).

O código Gray de um inteiro i pode ser obtido com o "ou-exclusivo" de i (expresso como um número ordinário de base 2) com $i/2$ (arredondado para baixo). O código Gray não incrementa a faixa dinâmica da distância do espaço de padrões, mas resulta em um mapeamento de distância mais regular que o código binário convencional. Por causa de suas propriedades, variáveis inteiras diferentes por um nível têm exatamente uma posição do bit trocado na sua representação em código Gray. Assim, em uma vizinhança muito pequena, o mapeamento da distância é melhorado monotonicamente.

CMAC emprega um tipo especial de mapeamento não-linear na codificação da variável de entrada. O resultado produzido pelo CMAC é computado como uma soma de pesos selecionados. O mapeamento resulta na propriedade de que pontos muito próximos um do outro no espaço de entrada compartilham alguns pesos. O código é gerado transformando um vetor espacial de dimensão D em um vetor no espaço K dimensional, onde K é um coeficiente de associação arbitrariamente escolhido, sendo o principal parâmetro responsável pela sua propriedade de generalização (Kolcz & Allinson 1994).

A prescrição para codificar um inteiro x em um código CMAC/Gray é concatenar K *strings* de bits, sendo o j-ésimo (contado a partir de 1) calculado pela Equação 3.1, arredondado para baixo e expresso usando o código Gray.

$$k_j = \frac{x + j - 1}{K} \tag{3.1}$$

Isto garante uma representação em aK bits dos inteiros entre 0 e $(2^a - 1)K$ (inclusive), de tal forma que se inteiros x e y diferem aritmeticamente por K ou menos, seus códigos diferem pela distância de Hamming $|x - y|$; e se sua

distância aritmética é K ou mais, sua correspondente distância de Hamming é ao menos K.

3.1.3 Hipercubos binários

Vetores binários podem ser representados graficamente na forma de hipercubos, onde cada vértice do hipercubo corresponde a um vetor de determinada categoria (conforme já mencionado no Capítulo 1). Na Figura 3.1 é mostrado graficamente um conjunto de dados binários de dimensão três, classificados em duas categorias distintas ("círculo" e "quadrado"). Um vértice (vetor binário) que não pertence a alguma categoria é denominado *don't care*, representado no gráfico por um "X". Para vetores binários de dimensão maior que 3, é difícil realizar a sua representação gráfica, mas o princípio é o mesmo.

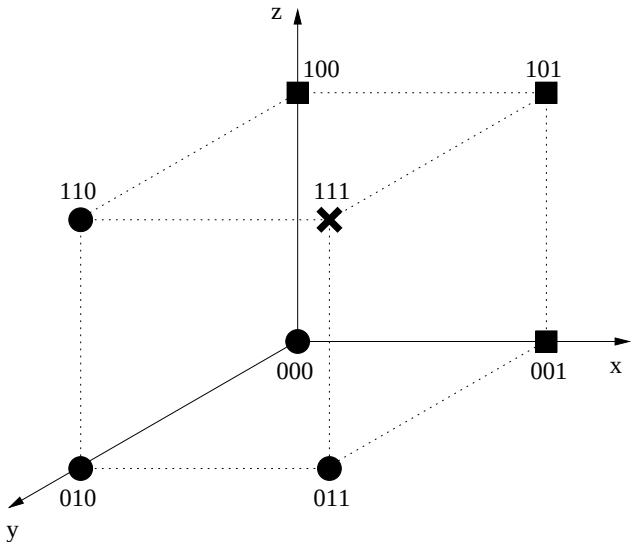


Figura 3.1: Hipercubo de dimensão 3.

3.1.4 Espaço Booleano

Suponha que n é o número de dimensões do espaço Booleano. O número de pontos possíveis é então $2^n = N$. Pontos de N são naturalmente representados por n-tuplas de "zeros" e "uns" e podem ser escritos como n-bits em representação binária (por exemplo: 10010101 para $n = 8$). Assim, os objetos de estudo do espaço $\{0,1\}^n$ são vetores n-dimensionais com componentes binários segundo (Kanerva 1988). Não se deve pensar neles como números inteiros, uma vez que os inteiros são como conjunto ordenado de uma dimensão.

As propriedades úteis do espaço $\{0,1\}^n$ resultam de seu grande número de dimensões, e são relatadas a seguir.

Conceitos

Alguns conceitos relacionados ao espaço $\{0,1\}^n$, segundo (Kanerva 1988), são apresentados:

- Origem, O: o ponto com todas coordenadas 0 é chamado de origem.
- Complemento, x': é a n-tupla que tem 1 onde x tem 0 e vice-versa.
- Norma, $|x|$: é o número de 1's na representação binária de x.
- Diferença, $x - y$: a diferença de dois pontos x e y é a n-tupla que tem 1 onde x e y diferem e zeros caso contrário. É o "ou-exclusivo" de x e y tomado bit a bit: $x - y = x \bigoplus y$. A diferença é comutativa: $x - y = y - x$.
- Distância, $d(x, y)$: a distância entre dois pontos x e y é o número de dimensões na qual x e y diferem. É chamada distância de Hamming (sua raiz quadrada é a distância Euclidiana) e é expressa em bits. Distância é a norma da diferença: $d(x, y) = |x - y|$.

  Dois pontos de N que estão perto um do outro são ditos serem similares.
- Intermediariedade, $(x : y : z)$: o ponto y está entre os pontos x e z se e somente se a distância de x a z é a soma das distâncias de x a y e de y a z: $d(x, z) = d(x, y) + d(y, z)$.
- Ortogonalidade, $x \perp y$: ponto x é ortogonal ao ponto y, ou os dois são perpendiculares ou indiferentes, se e apenas se a distância entre os dois é metade do valor de suas dimensões: $d(x, y) = n/2$.

  A distância $n/2$ é chamada a distância indiferença de $\{0,1\}^n$. Se x é ortogonal a y, ele é também ortogonal ao seu complemento y' (x é o meio entre y e y'): $x \perp y$ implica $x \perp y'$.
- Círculo, $O(r, x)$: um círculo com raio r e centro x é o conjunto dos pontos que estão no máximo r bits de x: $O(r, x) = \{y | d(x, y) \leq r\}$

  Qualquer círculo com raio n é o espaço N completo ($O(n, x) = N$).

O espaço N pode ser representado pelos vértices da formação de cubo no espaço Euclidiano n-dimensional. A distância entre dois pontos de N é o comprimento do caminho mais curto ao longo das bordas do cubo juntando os dois pontos. Quando dois pontos estão mais distantes do que um bit um do outro, há múltiplos caminhos mais curtos entre eles.

Analogia Esférica

Os vértices de uma formação cúbica n-dimensional ficam na superfície de uma esfera n-dimensional com (métrica Euclidiana) raio $\sqrt{n/2}$. Isto faz parte da analogia esférica para o espaço $\{0,1\}^n$ conforme (Kanerva 1988).

Como N é esférico, pode-se pensar como a superfície de uma esfera com circunferência $2n$ (utilizando distância de Hamming). Assim, N pode ser rotacionado para mover qualquer de seus pontos para a origem. Todos pontos de N podem ser igualmente qualificados como pontos da origem.

Um ponto de N e seu complemento são como dois pólos com distância n um do outro, com todo espaço entre eles. Os pontos do meio entre os pólos e perpendicular a eles são como o equador. Um conjunto de pontos dentro de uma dada distância de um ponto é chamado um círculo em N.

Os pontos de N entre 2 pontos x e y formam um subespaço de dimensão $d(x,y)$, sendo o correspondente conjunto representado como uma linha reta na esfera (um segmento de um grande círculo). Caso y seja o oposto de x, então o subespaço entre x e y é a esfera toda. Os segmentos (isto é, o caminho mínimo entre 2 pontos) de N não são únicos, e também os círculos em N, via de regra, não são convexos.

Distribuição do Espaço N

O número de pontos que estão exatamente d bits de um arbitrário ponto x é o número de modos para escolher d coordenadas de um total de n coordenadas, e portanto dado pelo coeficiente binomial mostrado na Equação 3.2 (vide (Kanerva 1988)).

$$(n:d) = \left(\begin{array}{c} n \\ d \end{array} \right) = \frac{n!}{d!(n-d)!} \tag{3.2}$$

Assim, a probabilidade de se ter um ponto x a uma distância d da origem é mostrada pela Equação 3.3, sendo p a probabilidade de uma coordenada (bit) ser 0 ou 1 (por exemplo: $p = 1/2$).

$$P(x) = \left(\begin{array}{c} n \\ d \end{array} \right) .p^d(1-p)^{n-d} \tag{3.3}$$

A distribuição de N então é a distribuição binomial com parâmetros n e p, mostrada na Equação 3.4 (Peebles 1993), sendo u a função degrau. A média do binômio é $n/2$, e a variância é $n/4$. A correspondente função de densidade de probabilidade é mostrada na Equação 3.5, sendo δ a função impulso.

$$F_x(x) = \sum_{d=0}^{n} \left(\begin{array}{c} n \\ d \end{array} \right) .p^d(1-p)^{n-d}u(x-d) \tag{3.4}$$

$$f_x(x) = \sum_{d=0}^{n} \left(\begin{array}{c} n \\ d \end{array} \right) .p^d(1-p)^{n-d}\delta(x-d) \tag{3.5}$$

Tendência à Ortogonalidade

Uma notável propriedade de N é que a maioria dos seus pontos fica aproximadamente a uma distância média $n/2$ de um ponto (e seu complemento). Em outras palavras, a maioria do espaço é aproximamente ortogonal a qualquer ponto dado, e quanto maior for n, mais pronunciado é o efeito. Isto pode ser visualizado pelo gráfico mostrado na Figura 3.2. Esta propriedade é a base do trabalho desenvolvido por Kanerva (vide (Kanerva 1988)) no projeto do SDM (*Sparse Distributed Memory*).

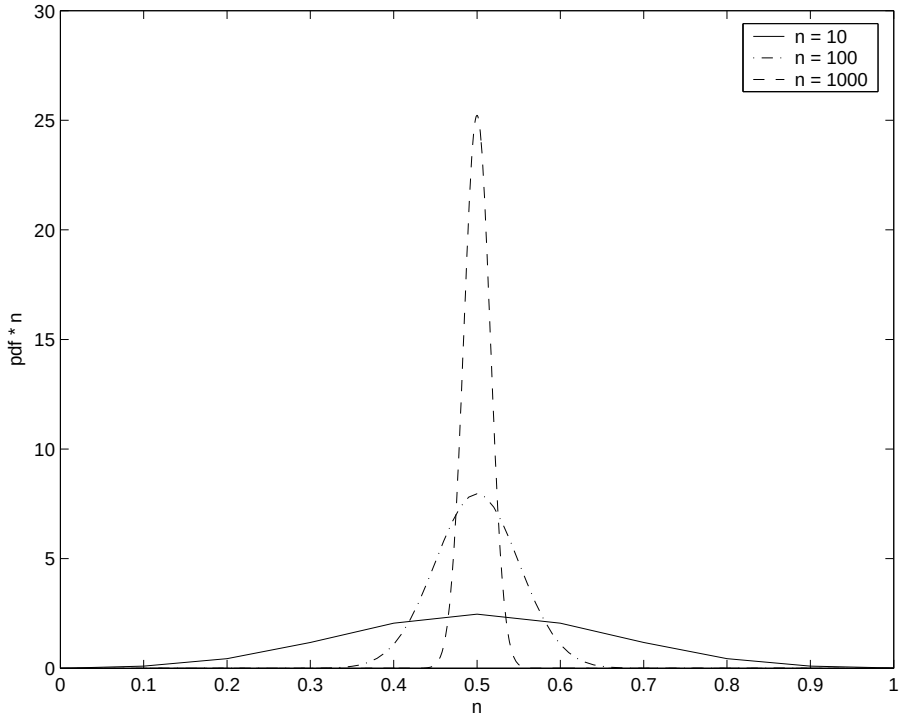


Figura 3.2: Distribuição da distância de Hamming em N em função de n.

Distribuição do Círculo

Algumas questões podem ser levantadas a respeito da distribuição dos pontos em um círculo $O(r, x)$:

- Como o espaço N é distribuído ao redor de um ponto?
- Qual é a área (número de pontos) do círculo $O(r, x)$?
- Como os pontos do círculo são distribuídos?

A área do círculo $O(r, x)$ é o número de pontos que são $0, 1, 2, ..., r$ bits distantes de x; portanto é a soma dos primeiros $r + 1$ coeficientes binomiais:

$$|O(r, x)| = (n : 0) + (n : 1) + ... + (n : r)$$

Um ponto x de N tem n vizinhos que estão a 1 bit distante dele, $n.(n-1)/2$ que estão a dois bits distante, $n.(n-1).(n-2)/3!$ que estão a 3 bits distante, e em geral, $(n : r)$ que estão a r bits distante. Para valores de r que são pequenos comparados com n, o número cresce quase exponencialmente com r, segundo (Kanerva 1988). A distribuição do círculo que inscreve menos do que a metade do espaço N ($r < n/2$) é aproximadamente binomial.

3.2 Síntese de Circuitos Digitais

Uma das aplicações da teoria de chaveamento é resolver o problema de minimização do número de portas lógicas digitais e o número de conexões entre portas digitais em circuitos VLSI (*Very Large Scale Integration*). Minimizar o número de entradas por porta *AND* e o número total de portas *AND* em um circuito VLSI é análogo a minimizar o número de antecedentes por regra e o número total de regras em uma base de regras em inteligência artificial (Jaskolski 1992).

A otimização de circuitos lógicos combinacionais passa pela modelagem da função lógica na forma de expressão em soma de produtos (dois níveis), ou produto de somas. Esta otimização é importante pelos seguintes aspectos:

- permite a translação direta da forma de tabela;
- reduz a informação necessária para expressar qualquer função lógica;
- é um modo formal de processar a representação de sistemas.

O objetivo do algoritmo de minimização lógica é minimizar uma cobertura F de dois níveis de uma função f. São representados por F^{ON}, F^{OFF} e F^{DC} os conjuntos *on set* (saída em 1), *off set* (saída em 0) e *dc set* (saída em X - *don´t care*) da função f, respectivamente. É assumido que ao menos dois destes conjuntos são especificados como entradas para o problema de minimização. O terceiro conjunto pode ser derivado complementando a união dos outros, ou seja, f pode ser especificada incompletamente. Para funções completamente especificadas, F^{DC} é vazio e F^{OFF} e F^{ON} são usados intercambiavelmente.

Em síntese de circuitos VLSI, é suposto que apenas os vetores de entrada que o circuito recebe são os mesmos na descrição dos dados do circuito. Vetores de entrada que não estão nos dados de descrição do circuito são considerados condições *don't care* (*dc set*). Assim o circuito VLSI gera uma saída

definida para vetores de entrada nos dados de descrição do circuito mas não para todos. Em classificação, se um ponto de entrada está fora do conjunto conhecido, é ainda desejável providenciar uma estimativa de qual categoria o ponto pertence. Este problema é resolvido classificando o ponto como sendo do hipercubo mais próximo (e correspondente categoria) em termos de alguma métrica de distância como a de Hamming (Jaskolski 1992). Isto corresponde a uma forma de generalização providenciada pelo circuito digital.

A seguir é apresentada uma explanação sobre definições e alguns métodos de minimização de funções Booleanas.

3.2.1 Definições

Um **literal** é uma variável de entrada complementada ou não complementada para uma função Booleana segundo (Ercegovac, Lang, & Moreno 2000). A parte de entrada de uma função Booleana possui valores no conjunto $\{0, 1, X\}$ e representa um produto de literais. O valor "X" representa a condição *don't care*. A parte de saída da função Booleana tem valores no conjunto $\{0, 1\}$.

Um **mintermo** de n variáveis é um termo de produto de n literais, em que cada variável aparece exatamente uma vez ou na forma não complementada ou na complementada. Há 2^n mintermos possíveis de n variáveis. Cada mintermo tem o valor 1 para uma única atribuição das variáveis, porque o produto tem o valor 1 somente quando todos os literais tem valor 1, e isto acontece para uma atribuição, uma vez que todas as outras atribuições produzem pelo menos um literal com o valor 0.

Uma **soma de mintermos** é uma soma de produtos em que todos os produtos são mintermos e nenhum mintermo é repetido.

Qualquer **função de chaveamento** (ou tabela verdade) pode ser representada por uma soma de mintermos. Esta expressão é a soma de mintermos correspondente aos elementos da representação do conjunto-um (*on set*) da função.

A soma de mintermos que representa uma função Booleana é única, de forma que é uma **representação canônica**.

Um termo de produto p é um **implicante** de uma função f se, para qualquer atribuição para a qual p tenha o valor 1, a função também tem o valor 1. Qualquer termo de produto, que aparece em uma soma de produtos que represente uma função, é um implicante dessa função.

Implicante primo (*prime implicant*) de uma função é um implicante que não é sobreposto por qualquer outro implicante da mesma função. Em outras palavras, um implicante é primo se ele não é contido por qualquer implicante da função.

Toda **soma de produtos mínima** consiste em uma soma de implicantes primos, mas não necessariamente todos os implicantes primos da função.

Um **implicante primo essencial** é um implicante primo que cobre pelo menos uma atribuição que não é coberta por outro implicante primo. Para formar uma expressão mínima, todos os implicantes primos essenciais, e possivelmente alguns não essenciais são necessários; estes últimos devem ser selecionados para que o custo seja mínimo. A escolha de implicantes primos não essenciais não é única e, portanto, pode haver mais do que uma soma de produtos mínima. Um implicante primo é essencial se existe um mintermo da função coberto por apenas aquele implicante dentre todos os implicantes primos da função.

Funções especificadas completamente são o caso especial de funções com nenhuma condição *don't care*. **Funções de chaveamento incompletamente especificadas** podem ser representadas por duas expressões: uma correspondente aos 1's (ou 0's) da função e a outra aos *dc* (valores *don't care*).

Uma **cobertura** de uma função Booleana é um conjunto (lista) de implicantes que cobre todos seus mintermos.

O tamanho, ou **cardinalidade**, de uma cobertura é o número de seus implicantes. *Don't care* podem ser efetivamente usados para reduzir o tamanho de uma cobertura de uma função incompletamente especificada. Na Figura 3.3 é mostrado graficamente um exemplo de cobertura referente ao conjunto de dados binários mostrado na Figura 3.1. Neste exemplo, a amostra *Don't care* forma uma cobertura com 3 outras amostras da classe "círculo".

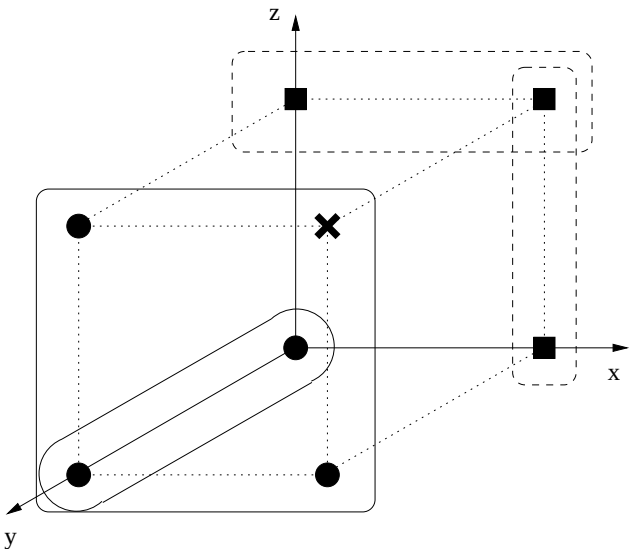


Figura 3.3: Exemplo de cobertura.

O objetivo da **minimização lógica de dois níveis** é determinar uma cobertura mínima de uma função. O objetivo da heurística de minimização lógica é determinar coberturas mínimas. Minimização heurística é motivada pela

necessidade de reduzir o tamanho das formas de 2 níveis com tempo de computação e armazenamento limitados. Uma **cobertura canônica mínima** é um conjunto de cubos de máxima assinatura que identifica unicamente o conjunto de primos cobrindo cada mintermo. Uma cobertura canônica mínima pode ser computada e usada como um ponto de partida para minimização heurística.

Uma **cobertura prima** é aquela em que todos seus implicantes são primos.

3.2.2 Métodos de minimização

O objetivo primário da minimização lógica de dois níveis é a redução do tamanho da função Booleana, ou pela forma de soma de produtos ou pela forma de produto de somas. O objetivo secundário é a redução dos literais, segundo (De Micheli 1994). Minimização lógica de saída única e funções de múltiplas saídas seguem o mesmo princípio, mas o último é mais complexo. O procedimento para obter uma soma de produtos mínima de uma função de chaveamento é:

1. Determine todos os implicantes primos.
2. Obtenha os implicantes primos essenciais.
3. Se nem todos as saídas em 1 estiverem cobertas pelos implicantes primos essenciais, escolha dentre os implicantes primos restantes um conjunto que cubra as saídas em 1 não cobertas e que obtenha o custo mínimo.

Existem alguns métodos de minimização lógica. Os mais comuns são:

- Exato: Quine & McCluskey, McBoole, ESPRESSO-EXACT, ESPRESSO-SIGNATURES. Minimização lógica exata ataca o problema de computar uma cobertura mínima. É um problema clássico em teoria de chaveamento. A solução do problema depende do Teorema de Quine, o qual delimita o espaço de procura para uma solução ótima.

  Teorema de Quine: Há uma cobertura mínima que é prima.

  A dificuldade de abordagem é causada pelo tamanho da tabela de implicantes. Com n entradas, uma função pode ter tantos quantos $\frac{3^n}{n}$ implicantes primos e 2^n mintermos. Então um algoritmo exponencial com um problema de tamanho exponencial requer provavelmente tamanho de memória e tempo de processamento proibitivos.

- Heurístico (estratégia de melhoria iterativa): MINI, PRESTO, ESPRESSO (ferramenta padrão), BOOM (Hlavicka & Fiser 2001; Fiser & Hlavicka 2001). Alguns minimizadores heurísticos produzem coberturas primas e

mínimas as quais a cardinalidade é geralmente perto do mínimo. Minimizadores heurísticos computam todos os primos e utilizam algoritmos heurísticos para cobrir a tabela de implicantes primos.

Muitos minimizadores heurísticos não computam todos implicantes primos, evitando então o potencial gargalo do armazenamento de um grande conjunto. Em vez disto, eles computam uma cobertura prima iniciando da especificação original da função. Esta cobertura é então manipulada modificando ou deletando implicantes até que uma cobertura com uma propriedade conveniente mínima seja encontrada. Estes minimizadores heurísticos usam uma estratégia de melhoria iterativa.

Minimização lógica heurística pode ser vista como a aplicação de um conjunto de operadores à cobertura lógica, a qual é inicialmente fornecida ao minimizador com o conjunto *don't care* (DC). A cobertura é declarada ser o resultado final da minimização quando os operadores não podem reduzir mais o tamanho da cobertura. Os operadores mais comuns são:

- Expande
- Reduz
- Irredundante
- Essencial

A seguir é apresentada uma explanação sobre os métodos Quine-McCluskey e ESPRESSO, por serem os mais comumente utilizados.

Quine-McCluskey

Quine-McCluskey é um método tabular para obter a soma de produtos mínima. Neste método, utilizam-se duas fases (vide (Ercegovac, Lang, & Moreno 2000)):

1. determinação dos implicantes primos da função, e
2. escolha de um conjunto de implicantes primos que cubra a função dada e tenha um custo mínimo.

Duas tabelas são construídas. A primeira tabela, a qual fornece os implicantes primos usando a identidade $xy + xy' = x$, é organizada da seguinte maneira:

1. A primeira coluna corresponde a uma lista de mintermos da função, os quais são obtidos diretamente do conjunto *on set*. A notação atribui um

1 a uma variável não complementada, e 0 a uma variável complementada. Os mintermos são agrupados de acordo com o número de 1's na representação binária.

2. A segunda coluna mostra os implicantes com $n-1$ literais. Estes implicantes são obtidos agrupando-se em pares os elementos da primeira coluna que diferem somente no valor de uma variável. Um elemento da primeira coluna pode ser usado na formação de mais de um par. Para produzir o emparelhamento, basta examinar grupos adjacentes.

   A entrada na segunda coluna, correspondente a determinado par, relaciona os valores comuns dos elementos que formam o par, e um "-" para a variável cujo valor difere. Os elementos da primeira coluna que participam da formação de pares são marcados com um "N", o qual indica que eles não são implicantes primos.

3. A terceira coluna mostra os implicantes de $n-2$ literais. Isto é obtido formando-se pares de elementos da segunda coluna. Para formar um par, dois elementos devem ter seus "-" na mesma posição, e o restante deve diferir somente em uma variável. Como antes, os elementos da segunda que participam na formação de pares são marcados com um "N", o qual indica que eles não são implicantes primos.

4. O processo é continuado da mesma maneira até que mais nenhum elemento possa ser emparelhado.

5. Os implicantes primos são todos aqueles elementos que não são marcados com um "N".

O segundo passo do método consiste em obter uma cobertura mínima, ou seja, o número mínimo de implicantes primos que cobre o conjunto-um. Uma segunda tabela é construída para este propósito, na qual as linhas correspondem aos implicantes primos obtidos da primeira tabela, e as colunas aos elementos do conjunto-um. Para cada linha (implicante primo), marca-se com "X" as entradas correspondentes aos elementos do conjunto-um cobertos pelo implicante primo.

Em seguida, identifica-se os implicantes primos essenciais como aqueles que têm um "X" em uma coluna que não tem nenhum outro "X". Estes implicantes primos essenciais fazem parte de qualquer cobertura, de forma que os elementos do conjunto-um que eles cobrem são eliminados (como já cobertos). Finalmente, os elementos do conjunto-um restantes são cobertos com tanto menos implicantes primos quanto possível.

Como exemplo de utilização do método Quine-McCluskey, são apresentadas as Tabelas 3.1 e 3.2 para minimização lógica da seguinte função de quatro entradas:

$$f(x_3, x_2, x_1, x_0) = \text{conjunto-um}(0, 1, 3, 5, 7, 11, 12, 13, 14)$$

Tabela 3.1: Etapa 1 do método Quine-McCluskey para determinação dos implicantes primos.

Mintermos	Prod. de 3 literais	Prod. de 2 literais	Prod. de 1 literal
0 0 0 0 N 0 0 0 1 N	0 0 0 – 0 0 – 1 N 0 – 0 1 N	0 – – 1	
0 0 1 1 N 0 1 0 1 N 1 1 0 0 N	 0 – 1 1 N – 0 1 1 0 1 – 1 N		
0 1 1 1 N 1 0 1 1 N 1 1 0 1 N 1 1 1 0 N	– 1 0 1 1 1 0 – 1 1 – 0		

Tabela 3.2: Etapa 2 do método Quine-McCluskey para escolha dos implicantes primos.

	0	1	3	5	7	11	12	13	14
0 0 0 –	X	X							
– 0 1 1			X			X			
– 1 0 1				X				X	
1 1 0 –							X	X	
1 1 – 0							X		X
0 – – 1		X	X	X	X				

A soma de produtos mínima resultante é:

$$f(x_3, x_2, x_1, x_0) = x_3' x_2' x_1' + x_2' x_1 x_0 + x_2 x_1' x_0 + x_3 x_2 x_0' + x_3' x_0$$

ou:

$$f(x_3, x_2, x_1, x_0) = x_3' x_2' x_1' + x_2' x_1 x_0 + x_3 x_2 x_1' + x_3 x_2 x_0' + x_3' x_0$$

O método é prontamente estendido para o caso em que há *don't care*. Estes *don't care* são incluídos no primeiro passo (para obter os implicantes primos), mas não são usados no segundo passo (para obter a cobertura mínima).

O método Quine McCluskey é bastante direto de ser programado, porém, para problemas práticos, o número de implicantes primos normalmente é muito grande, de forma que a obtenção de uma cobertura mínima é algo que

consome muito tempo. Além disto, este método é aplicável somente a funções de saída única, enquanto que, na prática, é necessário considerar o caso de múltiplas saídas.

Espresso

Espresso (University of Texas at Austin – College of Engineering 1990) é um minimizador de dois níveis eficiente e rápido. Ele explora os operadores enquanto efetua um refinamento iterativo de uma cobertura. A saída do Espresso é uma cobertura irredudante, geralmente mínima em cardinalidade.

O algoritmo do Espresso se divide em algumas etapas (ou operadores) para atingir o objetivo final: determinação da cobertura mínima. Estas etapas são:

1. **Expande:** Este operador é utilizado por muitos minimizadores heurísticos. Seu objetivo é incrementar o tamanho de cada implicante de uma dada cobertura, de tal forma que implicantes de tamanho pequeno podem ser cobertos e deletados. Implicantes maximamente expandidos são primos da função. A expansão de um implicante é feito elevando um (ou mais) de seus 0's para 1. Isto corresponde a incrementar seu tamanho, e portanto cobrindo mais mintermos.

   Uma rápida análise da posição do implicante no espaço Booleano com respeito ao conjunto OFF permite determinar alguma possível ou impossível direção para expansão, mas não todas. Então o próximo passo é pesquisar por outros cubos na cobertura que poderiam ser cobertos por uma expansão específica possível. Quando esta pesquisa é finalizada, observa-se por direções de expansão que permitem ao implicante expandido sobrepor um máximo número de outros cubos. No fim, já se terá o implicante primo tornando ele tão grande quanto possível.

   A questão fundamental do processo de expansão é se o cubo expandido é ainda válido, isto é, se ele é ainda um implicante da função f. Há dois métodos principais para este teste:

   - Verificando pela interseção do implicante expandido com o subconjunto OFF;
   - Verificando pela cobertura do implicante expandido por $F^{ON} \cup F^{DC}$. F^{OFF} não é necessário.

   Muitos primos diferentes podem ser derivados pela expansão dos cubos em direções diferentes. Cada primo poderia cobrir diferentes subconjuntos de F. O principal objetivo é cobrir o maior subconjunto de implicantes de F.

2. **Reduz:** O operador Reduz objetiva decrementar o tamanho de cada implicante de uma dada cobertura F de tal forma que uma sucessiva expansão poderia levar a uma outra cobertura de menor cardinalidade. Um implicante reduzido é válido quando, com os implicantes restantes, ele cobre a função.

   Entretanto, coberturas reduzidas não são primas e o operador Reduza não garante a irredundância da cobertura.

3. **Irredundante:** O operador irredundante deleta um número máximo de implicantes redundantes, tornando a cobertura irredundante. Um mínimo subconjunto de implicantes é selecionado tal que nenhum implicante naquele subconjunto é coberto pelo restante.

4. **Essencial:** A deteção de implicantes primos essenciais é importante para ambos métodos (exato e heurístico) de minimização. Primos essenciais devem fazer parte de qualquer cobertura, por isto, eles podem ser extraídos e armazenados enquanto o minimizador manipula os primos não essenciais. Implicantes primos essenciais são acrescentados de volta à cobertura no final do método de minimização.

Espresso computa primeiro o complemento de uma cobertura para ser usado pelo operador Expande. Então ele aplica os operadores Expande e Irredundante, os quais tornam a cobertura prima e irredundante. Depois que os essenciais são extraídos, Espresso itera os operadores Reduza, Expande e Irredundante na pesquisa por coberturas irredundantes de cardinalidade decrescente. Quando nenhuma melhora pode ser alcançada, Espresso tenta reduzir e expandir a cobertura usando uma heurística diferente. Estes passos são mostrados no Algoritmo 1.

O Algoritmo 1 é uma descrição simplificada do Espresso. É assumido que Espresso recebe como entradas os conjuntos ON e DC, e gera o conjunto OFF por complementação. Na prática, o programa pode aceitar quaisquer dois conjuntos entre ON, OFF e DC.

A função "custo" é uma soma ponderada da cobertura e cardinalidade literal. A rotina "torne_esparso" modifica o número de 1/0s no vetor sem afetar a cardinalidade da cobertura, para implementação em circuitos lógicos, como PLA (*Programmable Logic Array*). A função "último" aplica diferentes heurísticas em último caso.

Espresso fornece uma cobertura mínima (conjunto mínimo dos maiores hipercubos possíveis) que cobre todos os dados de treinamento de uma classe sem cobrir qualquer ponto pertencente a outra classe. É assumido que todos os outros pontos são condições *don't care* os quais podem ser cobertos se

Algoritmo 1 ESPRESSO(F^{ON}, F^{DC})

$F^{OFF} = complemento(F^{ON} \cup F^{DC})$
$F = expande(F^{ON}, F^{OFF})$
$F = irredundante(F, F^{DC})$
$E = essencial(F, F^{DC})$
$F = F - E$
$F^{DC} = F^{DC} \cup E$
repita
  $\phi_2 = custo(F)$
  repita
    $\phi_1 = |F|$
    $F = reduza(F, F^{DC})$
    $F = expande(F, F^{OFF})$
    $F = irredundante(F, F^{DC})$
  até $|F| \leq \phi_1$
  $F = último(F, F^{DC}, F^{OFF})$
até $custo(F) \leq \phi_2$
$F = F \cup E$
$F^{DC} = F^{DC} - E$
$F = torne\_esparso(F, F^{DC}, F^{OFF})$

necessário para resolver o problema de minimização. Isto, de fato, é uma generalização (Jaskolski 1992).

3.3 Conclusão

Padrões binários representam dados em forma de *strings* binárias. Para que uma máquina de aprendizado possa aprender alguma informação a partir de um conjunto de dados binários, é necessário que os dados sejam codificados em um formato apropriado, de modo que a máquina possa generalizar dados desconhecidos. O método de codificação CMAC/Gray é apresentado como uma alternativa de codificação de padrões binários, de forma que padrões semelhantes terão distância de Hamming pequena.

As características e propriedades do espaço Booleano foram apresentadas, evidenciando a complexidade da análise em um espaço de alta dimensão. Algumas propriedades são difíceis de assimilar devido à impossibilidade de representação gráfica, uma vez que estas propriedades não aparecem em duas ou três dimensões (espaço de baixa dimensão). A distribuição dos pontos no espaço $n-$dimensional foi quantificada, bem como mostrada a tendência dos pontos em se concentrarem a uma distância de Hamming com valor $n/2$.

Métodos de minimização de funções Booleanas objetivam obter a função lógica com o menor número de produtos e literais a partir dos subconjuntos F^{ON}, F^{OFF} e F^{DC}. Dois métodos foram revistos: Quine-McCluskey (exato) e

Espresso (heurístico). O método exato possui um custo computacional elevado para um grande conjunto de dados de entrada, enquanto o método heurístico tenta minimizar esta deficiência, porém sem a garantia de que se obterá a função lógica mínima (mas pelo menos próxima da mínima). Ambos métodos podem aproveitar o subconjunto DC (*don't care*) para aumentar a cobertura dos implicantes da função, diminuindo o número de produtos e literais necessários para especificação da função Booleana. Isto corresponde a um tipo de generalização para dados de entrada não conhecidos, ou seja, dados do subconjunto F^{DC}.

CAPÍTULO 4

Reconhecimento de Padrões

Padrões são representados convencionalmente como vetores multidimensionais, onde cada dimensão é uma única característica (ou atributo) (Jain, Murty, & Flynn 1999). Padrões podem ser representados como valores de números reais ou listas ordenadas de atributos, ou descrição de partes e suas relações. Assim, as características dos padrões são divididas nos seguintes tipos:

- Características quantitativas: valores contínuos, valores discretos, intervalos de valores.
- Características qualitativas: nominais ou não ordenadas (ex.: cor), ordinais (ex.: frio ou quente).

Dentre as aplicações das máquinas que reconhecem padrões (também denominado classificador), pode-se citar:

- reconhecimento automático da fala;
- identificação de manuscritos (Hastie & Simard 1997);
- reconhecimento de caracter óptico;
- identificação de seqüência de DNA (Lorena, Batista, & de Carvalho 2002);
- reconhecimento de face humana (Partridge 2000; Mahamud & Hebert 2003);
- identificação para radar.

A tarefa de um classificador é usar o vetor de características fornecido pelo extrator de características para associar o objetivo a uma categoria. Pelo fato de a classificação ser, em base, a tarefa de recuperar o modelo que gerou os padrões, diferentes técnicas de classificação são úteis dependendo do tipo de modelos candidatos, a saber:

- reconhecimento estatístico: foca as propriedades estatísticas dos padrões (geralmente expressa em densidades de probabilidade);
- reconhecimento neural: descendente de reconhecimento estatístico;
- reconhecimento sintático: regras ou gramáticas que descrevem a decisão.

É importante compreender a diferença entre *clustering* (classificação não supervisionada) e análise discriminatória (classificação supervisionada). Em classificação supervisionada, os padrões (treinamento) rotulados são usados para aprender as descrições das classes e assim rotular um novo padrão. Enquanto no caso de *clustering*, o problema é agrupar uma dada coleção de padrões não rotulados em *clusters* significativos, ou seja, os rótulos são obtidos unicamente dos dados (Jain, Murty, & Flynn 1999). Intuitivamente, padrões dentro de um *cluster* válido são mais similares entre si do que os padrões pertencentes a outros *clusters*.

Clustering é um problema combinatorial difícil. A saída do *clustering* pode ser *hard* (uma partição dos dados em cada grupo) ou *fuzzy* (onde cada padrão possui um grau de pertinência em cada cluster de saída).

Em problemas de classificação, há duas abordagens distintas: uma usa o conhecimento estatístico da distribuição conjunta das amostras e de suas categorias quando é possível obtê-las (método paramétrico). Outra abordagem não possui nenhum conhecimento da distribuição das amostras, exceto o que pode ser inferido das amostras (método não-paramétrico).

A análise padrão de Bayes (método paramétrico), segundo (Duda, Hart, & Stork 2000), produz um ótimo procedimento de decisão e uma correspondente probabilidade de erro mínimo de classificação.

Talvez o procedimento de decisão não paramétrico mais simples seja a forma da Regra do Vizinho-mais-próximo. No caso de grandes amostras, esta regra simples tem uma probabilidade de erro que é menor do que o dobro da probabilidade de erro da regra de Bayes (Cover & Hart 1967; Cover 1968).

A seguir, são apresentados as etapas para o projeto de um classificador e de algumas técnicas de classificação paramétricas e não paramétricas. Em seguida, alguns experimentos são realizados para classificação de dados binários.

4.1 Projeto de um Classificador

Para o projeto de um classificador, as seguintes fases e problemas são normalmente encontrados:

1. Coleta de dados: Qual o tamanho adequado e representativo do conjunto de exemplos para treinamento e teste do sistema?
2. Escolha das características: Como combinar conhecimento prévio e dados empíricos para encontrar características relevantes e efetivas?
3. Escolha do modelo: Quando rejeitar um modelo e procurar outro? Qual a expectativa do desempenho obtido por um modelo?
4. Treinamento: Qual técnica de treinamento é mais apropriada para o conjunto de dados?
5. Avaliação: É importante para medir o desempenho do sistema e para identificar a necessidade de melhoramento de seus componentes.

Uma vez definidas as características e parâmetros do classificador, ainda permanecem pendências do tipo: Como ajustar a complexidade do modelo? (não tão simples que ele não possa explicar as diferenças entre as categorias, e nem tão complexo para resultar em classificação pobre em padrões novos.) A propriedade geral em reconhecimento de padrão é o compromisso (negociação) da complexidade da pesquisa contra a precisão. Este é o dilema do projetista e que motiva o desenvolvimento de diferentes metodologias de classificação. Em geral, alta precisão de classificação e baixa taxa de retenção de dados (número de amostras selecionadas) implicam em melhor desempenho (Lam, Keung, & Liu 2002). Isto pode ser quantificado pelas fórmulas:

$$\text{precisão} = \frac{\text{número de classificações corretas nas amostras de teste}}{\text{número de amostras de teste}}$$

$$\text{taxa de retenção} = \frac{\text{número de protótipos aprendidos (amostras selecionadas)}}{\text{número de amostras de treinamento}}$$

4.2 Técnicas de Classificação Paramétricas

Uma das técnicas de classificação paramétricas mais difundidas é a Regra de Bayes, apresentada na próxima subseção. Uma função de discriminação pode ser derivada facilmente por esta técnica. A Regra *Naive* de Bayes também é apresentada com um exemplo simples de aplicação.

4.2.1 Regra de Bayes

Suponha que se conheça a probabilidade prévia (*a priori*) $P(w_j)$ e a densidade condicional $p(x|w_j)$ para $j = 1, 2$, onde x representa as variáveis de entrada e w a saída (categoria) de um sistema. A densidade de probabilidade conjunta de se encontrar um padrão que é da categoria w_j e possui característica de valor x (ou seja: $p(w_j, x)$), pode ser escrita de duas maneiras, mostradas na Equação 4.1 (vide (Duda, Hart, & Stork 2000)).

$$p(w_j, x) = P(w_j|x).p(x) = p(x|w_j).P(w_j) \tag{4.1}$$

Rearranjando a Equação 4.1, obtém-se a chamada fórmula de Bayes (Equação 4.2).

$$P(w_j|x) = \frac{p(x|w_j).P(w_j)}{p(x)} \tag{4.2}$$

onde para o caso de duas categorias, $p(x)$ é determinado pela Equação 4.3.

$$p(x) = \sum_{j=1}^{2} p(x|w_j).P(w_j) \tag{4.3}$$

A fórmula de Bayes pode ser expressa informalmente em palavras como mostrado pela Equação 4.4.

$$\textit{a posteriori} = \frac{\textit{verossimilhança} \times \textit{a priori}}{\textit{evidência}} \tag{4.4}$$

A fórmula de Bayes mostra que observando o valor de x pode-se converter a probabilidade "*a priori*" $P(w_j)$ para a probabilidade "*a posteriori*" $P(w_j|x)$ - a probabilidade do estado natural de w_j, dado que o valor x da característica tenha sido medido. Chama-se $p(x|w_j)$ a verossimilhança de w_j com respeito a x, um termo escolhido para indicar a categoria w_j para a qual $p(x, w_j)$ é maior e mais parecida para ser a categoria verdadeira. O fator evidência, $p(x)$, pode ser visto meramente como fator de escala que garante que a probabilidade posterior soma 1.

Tendo-se uma observação x para a qual $P(w_1|x)$ é maior que $P(w_2|x)$, poderia-se naturalmente ser inclinado a decidir que o estado natural real é w_1. Quando observa-se um x particular, a probabilidade do erro de classificação é dada pela Equação 4.5.

$$P(erro|x) = \begin{cases} P(w_1|x) & \text{se decidir por } w_2 \\ P(w_2|x) & \text{se decidir por } w_1 \end{cases} \tag{4.5}$$

Para minimizar a probabilidade do erro, a regra de decisão de Bayes torna-

se:

"Decida por w_1 se $P(w_1|x) > P(w_2|x)$, senão decida por w_2."

Ou a seguinte regra de decisão equivalente:

"Decida por w_1 se $p(x|w_1).P(w_1) > p(x|w_2).P(w_2)$, caso contrário decida por w_2."

Então, a probabilidade do erro de classificação para a Regra de Bayes é dada pela Equação 4.6.

$$P(erro|x) = min[P(w_1|x), P(w_2|x)] \tag{4.6}$$

4.2.2 Função de Discriminação

Para o caso de taxa de erro mínima, a função de discriminação corresponde à máxima probabilidade "*a posteriori*" mostrada na Equação 4.7 ou 4.8.

$$g_i(x) = P(w_i|x) \tag{4.7}$$

$$g_i(x) = p(x|w_i).P(w_i) \tag{4.8}$$

Remanejando todo $g_i(x)$ por $f(g_i(x))$, onde $f(\cdot)$ é uma função monotônica crescente, o resultado da classificação não se altera. Assim, outra forma de apresentação da função de discriminação é mostrada na Equação 4.9.

$$g_i(x) = \ln p(x|w_i) + \ln P(w_i) \tag{4.9}$$

Ou seja, para classificação por taxa mínima de erro qualquer destas funções tem resultados idênticos, mas algumas podem ser mais simples para compreender ou para computar do que outras.

Se as probabilidades *a priori* $P(w_i)$ são as mesmas para todas as classes, então o termo $\ln P(w_i)$ torna-se uma constante aditiva sem importância que pode ser ignorada.

4.2.3 Regra Naive de Bayes

Quando as relações de dependência entre as características usadas por um classificador são desconhecidas, geralmente faz-se a suposição simples de que as características são condicionalmente independentes dada a categoria. Isto simplifica o cálculo da *verossimilhança*, como mostrado pela Equação 4.10.

$$p(x|w_j) = \prod_{i=1}^{d} p(x_i|w_j) \tag{4.10}$$

onde d é a dimensão dos dados de entrada x. Esta fórmula aplicada à Equação 4.2 é a chamada Regra *Naive* de Bayes. Na prática, geralmente apresenta bons resultados, como é exemplificado na subseção seguinte.

Considerando ainda o problema de classificação de duas categorias nas quais os componentes do valor de características são valores binários e condicionalmente independentes, ou seja:

$$x = (x_1, ..., x_d)^T$$

onde os componentes x_i são 0 ou 1, com probabilidades dadas pela Equações 4.11 e 4.12.

$$p_i = P_r[x_i = 1|w_1] \tag{4.11}$$

$$q_i = P_r[x_i = 1|w_2] \tag{4.12}$$

Assumindo independência condicional, pode-se escrever $P(x|w_i)$ como o produto das probabilidades para os componentes de x, mostradas nas Equações 4.13 e 4.14.

$$p(x|w_1) = \prod_{i=1}^{d} p_i^{x_i}.(1 - p_i)^{1-x_i} \tag{4.13}$$

$$p(x|w_2) = \prod_{i=1}^{d} q_i^{x_i}.(1 - q_i)^{1-x_i} \tag{4.14}$$

A função de discriminação para o caso de duas classes é dada pela Equação 4.15, segundo (Schalkoff 1992).

$$g(x) = g_1(x) - g_2(x) \tag{4.15}$$

Ou de outro modo:

$$g(x) = P(w_1|x) - P(w_2|x) \tag{4.16}$$

Assim:

$$g(x) = p(x|w_1).P(w_1) - p(x|w_2).P(w_2) \tag{4.17}$$

Substituindo 4.13 e 4.14 em 4.17, e aplicando a função logarítmica em cada termo do lado direito da equação, obtém-se a função de discriminação mostrada na Equação 4.18.

$$g(x) = \sum_{i=1}^{d} \left[x_i \ln \frac{p_i}{q_i} + (1 - x_i) \ln \frac{1 - p_i}{1 - q_i} \right] + \ln \frac{P(w_1)}{P(w_2)} \tag{4.18}$$

Assim, a regra de decisão pode ser escrita:

"Decida w_1 se $g(x) > 0$ e w_2 se $g(x) \leq 0$."

Então, a condição de característica independente leva a um classificador linear simples (vide (Schalkoff 1992)). Geometricamente, os possíveis valores para x aparecem como os vértices de um hipercubo d-dimensional; a superfície de decisão definida por $g(x) = 0$ é um hiperplano que separa vértices w_1 dos vértices w_2.

4.2.4 Exemplo da Regra Naive de Bayes

A Tabela 4.1 mostra distribuições de freqüência para as relações entre as características e a classe no conjunto de dados do Jogo de Tênis (Hall 1999). Desta tabela, é fácil calcular as probabilidades necessárias para aplicar as Equações 4.2 e 4.10.

Tabela 4.1: Dados do Jogo de Tênis.

	Características				
Exemplo	Tempo	Temperatura	Umidade	Vento	Classe
1	ensolarado	quente	alta	fraco	não jogar
2	ensolarado	quente	alta	forte	não jogar
3	nublado	quente	alta	fraco	jogar
4	chuva	média	alta	fraco	jogar
5	chuva	frio	normal	fraco	jogar
6	chuva	frio	normal	forte	não jogar
7	nublado	frio	normal	forte	jogar
8	ensolarado	média	alta	fraco	não jogar
9	ensolarado	frio	normal	fraco	jogar
10	chuva	média	normal	fraco	jogar
11	ensolarado	média	normal	forte	jogar
12	nublado	média	alta	forte	jogar
13	nublado	quente	normal	fraco	jogar
14	chuva	média	alta	forte	não jogar

As probabilidades para características nominais são estimadas usando contagem de freqüência calculadas dos dados de treinamento. As probabilidades para características numéricas são assumidas para serem de uma distribuição normal; outra vez os parâmetros necessários são estimados dos dados de treinamento. Qualquer freqüência zero é recolocada por $\frac{0.5}{m}$ como a probabilidade, onde m é o número de exemplos de treinamento. Assim são montadas as Tabelas 4.2 à 4.5 tendo como referência a Tabela 4.1.

Imagine acordar de manhã e desejar determinar se o dia é apropriado para um jogo de tênis. Notando que o clima é ensolarado, a temperatura é quente, a umidade é normal e o vento é fraco, aplica-se as Equações 4.2, 4.3 e 4.10 e calcula-se a probabilidade posterior de cada classe, usando probabilidades derivadas das Tabelas 4.2 à 4.5:

Tabela 4.2: Ocorrências da característica Tempo.

	Jogar	Não Jogar	
ensolarado	2	3	5
nublado	4	0	4
chuva	3	2	5
	9	5	14

Tabela 4.3: Ocorrências da característica Temperatura.

	Jogar	Não Jogar	
quente	2	2	4
média	4	2	6
frio	3	1	4
	9	5	14

Tabela 4.4: Ocorrências da característica Umidade.

	Jogar	Não Jogar	
alta	3	4	7
normal	6	1	7
	9	5	14

Tabela 4.5: Ocorrências da característica Vento.

	Jogar	Não Jogar	
forte	3	3	6
fraco	6	2	8
	9	5	14

$$p(nãojogar) = \frac{5}{14} = 0,357$$

$$p(jogar) = \frac{9}{14} = 0,643$$

$$\begin{aligned} p(ensolarado, quente, normal, fraco|nãojogar) &= p(ensolarado|nãojogar) \times \\ &\quad p(quente|nãojogar) \times \\ &\quad p(normal|nãojogar) \times \\ &\quad p(fraco|nãojogar) \\ &= \frac{3}{5} \times \frac{2}{5} \times \frac{1}{5} \times \frac{2}{5} \\ &= 0,0192 \end{aligned}$$

$$\begin{aligned} p(ensolarado, quente, normal, fraco|jogar) &= p(ensolarado|jogar) \times \\ &\quad p(quente|jogar) \times p(normal|jogar) \times \\ &\quad p(fraco|jogar) \\ &= \frac{2}{9} \times \frac{2}{9} \times \frac{6}{9} \times \frac{6}{9} \\ &= 0,0219 \end{aligned}$$

$$p(ensolarado, quente, normal, fraco) = 0,0192 \times 0,357 + 0,0219 \times 0,643 = 0,0209$$

$$p(nãojogar|ensolarado, quente, normal, fraco) = \frac{0,0192 \times 0,357}{0,0209} = \frac{0,00685}{0,0209} = 0,328$$

$$p(jogar|ensolarado, quente, normal, fraco) = \frac{0,0219 \times 0,643}{0,0209} = \frac{0,0141}{0,0209} = 0,675$$

Assim, neste dia é recomendável jogar tênis, pois a probabilidade de jogar é maior do que não jogar, dado as condições climáticas do dia.

Devido à suposição de que os valores das características são independentes dentro da classe, o classificador *Naive* de Bayes apresenta um desempenho de

predição desfavoravelmente afetado pela presença de atributos redundantes nos dados de treinamento. Por exemplo, se há uma característica X que é perfeitamente correlacionada com uma segunda característica Y, então tratando-as com significados diferentes provoca o dobro do efeito na Equação 4.2 do que deveria ter. Assim, dependências moderadas entre as características resultarão em imprecisão na estimação da probabilidade, mas as probabilidades não são tão fortes na prática para resultar no incremento do erro de classificação (Zhang & Ling 2003). Assim sendo, o classificador *Naive* de Bayes ainda é um ótimo classificador.

4.3 Técnicas de Classificação Não Paramétricas

Todas as densidades paramétricas clássicas são unimodais (possuem um único máximo local), enquanto muitos problemas práticos envolvem densidades multimodais. Entretanto, procedimentos não paramétricos podem ser usados sem a suposição de que as formas das densidades são conhecidas.

Há diversos tipos de métodos não paramétricos em reconhecimento de padrões. Alguns consistem de procedimentos para estimação das funções de densidade dos padrões de amostras. Se estas estimativas são satisfatórias, elas podem substituir as densidades verdadeiras para o projeto do classificador. Outras consistem de procedimentos para estimar diretamente as probabilidades "*a posteriori*" $P(w_j|x)$. Isto é relativo a procedimentos de projeto não paramétricos, tais como a regra do vizinho-mais-próximo, a qual ultrapassa a estimação da probabilidade e vai diretamente às funções de decisão.

Em geral, procedimentos não paramétricos utilizam alguma forma de medida de similaridade entre os dados de treinamento e os dados desconhecidos para a tomada de decisões. Idealmente, não se pode computar a distância entre dois padrões até que se tenha transformado-os para serem tão similares quanto possível. Entretanto, a transformação é geralmente de grande complexidade. A seguir são apresentadas as métricas de similaridade mais comuns e a regra do vizinho-mais-próximo utilizada para classificação de padrões.

4.3.1 Métricas de Distância

Uma métrica $D(\cdot,\cdot)$ é meramente uma função que fornece uma distância escalar entre dois padrões. É útil pré-computar os valores de todas as $n(n-1)/2$ distâncias tomadas aos pares para os n padrões e armazená-las em uma matriz simétrica, para então utilizá-las na classificação dos padrões.

Supondo três vetores a, b e c, tem-se as seguintes propriedades para qualquer métrica de distância entre eles:

- não negatividade: $D(a, b) \geq 0$;
- reflexibilidade: $D(a, b) = 0$ se e somente se $a = b$;
- simetria: $D(a, b) = D(b, a)$;
- desigualdade triangular: $D(a, b) + D(b, c) \geq D(a, c)$.

Basicamente, uma métrica pode ser local ou global. Métricas locais são aquelas que variam dependendo da posição dos pontos no espaço de entrada. Métricas globais assumem que a avaliação da similaridade é independente da área do espaço de entrada (Blanzieri & Ricci 1999b). Há prós e contras no uso de métricas locais. De um lado, métricas locais geram classificadores que são sensíveis a mudanças locais dos dados. Por outro lado, métricas globais tem poucos parâmetros e conseqüentemente os classificadores são computacionalmente mais leves e menos suceptíveis a efeitos de ruído de dados.

Classificadores baseados em métricas globais têm uma dominância na polarização da componente de erro, enquanto aqueles baseados em métricas locais tendem a ter uma maior componente da variância. O ponto crítico parece ser a grade de localidade da métrica: escolha da localidade correta em diferentes áreas do espaço de entrada pode levar a uma descrição melhor das superfícies de separação.

Um inconveniente encontrado em base de dados reais é que características contínuas e nominais podem ser encontradas juntas no mesmo vetor de características de uma amostra. Isto complica o cálculo da distância entre diferentes padrões. O problema pode ser atacado de diferentes maneiras (Blanzieri & Ricci 1999a):

- Ordenação: ordenando e numerando os valores das características nominais e aplicando uma métrica contínua como a Euclidiana. Em geral esta abordagem introduz vizinhos fictícios.
- Discretização: discretizando os valores numéricos e aplicando uma métrica nominal, por exemplo *overlap* ou VDM (descritas adiante). Com a discretização alguma informação é inevitavelmente perdida e parâmetros da discretização podem ser críticos.
- Combinação: combinando duas métricas, uma nominal e uma numérica, obtendo uma métrica heterogênea.

A seguir é dada uma descrição de algumas métricas mais comuns e é apresentado um desenvolvimento de uma nova métrica.

Distância Euclidiana

A distância Euclidiana é a métrica mais comumente empregada. A fórmula Euclidiana, para distância entre duas amostras em d dimensões, é mostrada na Equação 4.19.

$$D(a,b) = \left(\sum_{k=1}^{d} (a_k - b_k)^2 \right)^{\frac{1}{2}} \tag{4.19}$$

Distância Manhattan

A distância Manhattan é simplesmente definida pela Equação 4.20. Ela é caracterizada por usar apenas a distância horizontal e vertical para dados de duas dimensões (quando $d = 2$).

$$D(a,b) = \sum_{k=1}^{d} |a_k - b_k| \tag{4.20}$$

Distância Minkowsk

Métrica Minkowsk (ou norma D_w) para padrões d-dimensionais é definida pela Equação 4.21.

$$D_w(a,b) = \left(\sum_{k=1}^{d} |a_k - b_k|^w \right)^{\frac{1}{w}} \tag{4.21}$$

A distância Euclidiana é a norma D_2, enquanto a distância Manhattan é a norma D_1.

A desvantagem do uso direto da métrica Minkowsk é a tendência da característica de escala mais alta dominar as outras. Soluções para este problema incluem a normalização de características contínuas ou outros esquemas de peso.

Distância overlap ou Hamming

Quando todos os atributos são simbólicos, o modo tradicional de utilizar o algoritmo do vizinho-mais-próximo é aplicar a métrica *overlap* a qual simplesmente conta o número de atributos que tem valores diferentes para dois exemplos.

A distância entre dois padrões binários em termos de número de elementos nos quais eles diferenciam é chamada distância de Hamming, depois que o engenheiro americano R. W. Hamming, em meados de 1950, usou esta medida para desenvolver um código de correção de erro (Aleksander 1995).

A métrica *overlap* calcula a distância entre dois padrões considerando igual a zero quando os atributos são iguais, e igual a um quando os atributos são diferentes. Isto é mostrado pelas Equações 4.22 e 4.23.

$$D(a,b) = \sum_{k=1}^{d} \delta(a_k, b_k) \tag{4.22}$$

$$\delta(a_k, b_k) = \begin{cases} 0 & \text{se } a_k = b_k \\ 1 & \text{se } a_k \neq b_k \end{cases} \tag{4.23}$$

Esta métrica pode, geralmente, falhar na captura da complexidade do domínio do problema, podendo levar a um desempenho pobre de classificação (Liu & White 1997).

Distância HEOM

Heterogeneous Euclidian-overlap Metric (HEOM) é uma combinação da métrica Euclidiana e *overlap* (Wilson & Martinez 1997; Wilson & Martinez 2000). Basicamente HEOM é uma função de distância heterogênea que usa diferentes funções de distância em diferentes tipos de atributos. Ela é expressa pelas Equações 4.24 e 4.25.

$$Dheom(a,b) = \sqrt{\sum_{k=1}^{d} d_k(a_k, b_k)^2} \tag{4.24}$$

onde:

$$d_k(a_k, b_k) = \begin{cases} \text{distância } \textit{overlap} & \text{se o } k_{\text{ésimo}} \text{ atributo é nominal} \\ \text{distância Euclidiana} & \text{se o } k_{\text{ésimo}} \text{ atributo é numérico} \end{cases} \tag{4.25}$$

Distância VDM

A métrica valor diferença (*Value Difference Metric* - VDM) foi proposta como uma alternativa para a determinação da distância entre dois valores simbólicos (Stanfill & Waltz 1986). Supondo dois padrões a e b com um conjunto de atributos d, esta distância é determinada pela comparação das distribuições de probabilidades condicionais para os valores a_k e b_k para cada atributo $k \in d$ (Payne & Edwards 1998). Isto é representado pelas Equações 4.26, 4.27 e 4.28.

$$D_{VDM}(a,b) = \sum_{k=1}^{d} \delta(a_k, b_k).\omega(a_k) \tag{4.26}$$

$$\delta(a_k, b_k) = \sum_{c \in C} |P(c|a_k) - P(c|b_k)|^2 \tag{4.27}$$

$$\omega(a_k) = \left[\sum_{c \in C} P(c|a_k)^2 \right] \tag{4.28}$$

onde C é o conjunto de todos os rótulos das classes presentes no conjunto de dados, e $P(c|a_k)$ é a probabilidade condicional da classe de a_k, isto é, a probabilidade de o valor a_k ocorrer no conjunto de dados para atributo k em amostras da classe c. Esta probabilidade é determinada diretamente dos dados de treinamento contando o número de ocorrências contendo o valor a_k para o atributo k, e determinando a proporção sobre a classe c, ou seja:

$$P(c|a_k) = \frac{\text{padrões contendo } a_k \text{ para classe = } c}{\text{padrões contendo } a_k} \tag{4.29}$$

O peso (ω) componente de VDM fornece uma indicação de como um valor de atributo discrimina entre diferentes classes. O seu valor mínimo representa uma distribuição de classe uniforme onde um valor de atributo aparece com probabilidade igual em amostras de todas as classes. O valor máximo do peso igual a 1 indica um valor de atributo que aparece em apenas uma classe. O peso é usado para controlar a influência do atributo para cada amostra de treinamento quando determinando o vizinho-mais-próximo final.

A métrica VDM foi originalmente desenvolvida para aplicação em problemas onde todos os atributos são simbólicos. VDM é, estatisticamente falando, uma medida de associação entre classes e um par de valores de atributos, porque a medida é aplicada apenas a 2 valores de um atributo em qualquer instante de tempo. Isto é explicitado na equação abaixo como outra forma de formulação da métrica:

$$D_{VDM}(a, b) = \sum_{k=1}^{d} \sum_{i=1}^{C} |p(c_i|a_k) - p(c_i|b_k)| \tag{4.30}$$

Uma proposta de modificação da métrica VDM é apresentada por (Cost & Salzberg 1993), denominada MVDM. O princípio do VDM é estendido para incluir pesos (ω_x e ω_y):

$$D_{MVDM}(a, b) = \omega_a \omega_b \sum_{k=1}^{d} d(a_k, b_k)^r \tag{4.31}$$

A inclusão de pesos no MVDM é baseado na idéia que algumas instâncias no conjunto de treinamento são mais confiáveis na classificação que outras.

Outra versões do VDM:

- DVDM: também trata instâncias com atributos numéricos. É feita uma

discretização dos atributos numéricos.

- HVDM (Heterogeneous VDM): combina métrica Euclidiana para características numéricas com um VDM.

Distância VDM modificada

Em determinados problemas com dados binários, o valor do atributo igual a zero em uma dada amostra não indica nenhuma informação útil. Neste caso, considera-se apenas informação útil se o valor do atributo é "um" (*bit* ativo em "1"). Para estes tipos de dados, propõe-se neste trabalho de tese que a métrica VDM possa ser simplificada. Assim, a probabilidade da amostra ser de uma determinada classe, dado que o atributo possui valor "0", pode ser definida como $0,5$ ($p(c_i|a_k = 0) = 0,5$), considerando problemas de duas classes. Para problemas de mais classes, a probabilidade $p(c_i|a_k = 0)$ é dividida igualmente pelo número total de classes. Esta proposta de modificação na métrica VDM é testada e comparada em seção posterior.

Distância Mahalanobis

Correlação linear entre características podem também distorcer medidas de distâncias. Esta distorção pode ser aliviada pela aplicação de uma transformação dos dados ou usando a distância Mahalanobis, dada pela Equação 4.32.

$$D(a, b) = (a - b)\Sigma^{-1}(a - b)^T \tag{4.32}$$

onde os padrões a e b são vetores linha, e Σ é a matriz de covariância das amostras dos padrões ou a matriz de covariância conhecida do processo de geração dos padrões.

A distância Mahalanobis associa diferentes pesos para diferentes características baseados nas suas variâncias e correlações aos pares.

Distância de Discriminação (DD)

A distância de discriminação foi proposta por (Aleksander, Clarke, & Braga 1994) como alternativa à distância de Hamming para o projeto de neurônios binários. A distância é explicada a seguir utilizando a mesma nomenclatura do trabalho original. Em seguida, é desenvolvida originalmente uma formulação estatística para a métrica.

Supondo um conjunto de treinamento armazenado como pares de entrada-saída:

$$T = \{(I_1, z_1), (I_2, z_2), ..., (I_t, z_t)\}$$

onde cada I_j é um vetor binário de dimensão n, e cada z_j é 0 ou 1 para identificar a classe pertencente à cada amostra. Os vetores binários do conjunto de entrada podem ser particionados em dois conjuntos disjuntos: pares t_1 que tem $z_j = 1$ e pares t_0 que possuem $z_j = 0$ onde $t_1 + t_0 = t$.

É associado um coeficiente de discriminação π_k a cada variável (bit) de entrada i_k o qual é a medida da capacidade de discriminação daquela entrada como definida pelo conjunto de treinamento (Aleksander 1995). Para o cálculo de π_k, é necessário definir os índices a_{k1} e a_{k0} como sendo:

- a_{k1} é o número de vezes que $i_k = 1$ no subconjunto de treinamento t_1 (quando $z_j = 1$); e
- a_{k0} é o número de vezes que $i_k = 1$ no subconjunto de treinamento t_0 (quando $z_j = 0$).

Assim, π_k é dada pela Equação 4.33.

$$\pi_k = mag\left[\frac{a_{k1}}{t_1} - \frac{a_{k0}}{t_0}\right] \tag{4.33}$$

onde $mag[x]$ é a magnitude de x.

É definido $\pi_k = 1$ se t_1 ou $t_0 = 0$, notando que se ambos são 0, significa que o sistema não foi treinado.

Uma entrada é mais discriminatória quando a ocorrência de $i_k = 1$ coincide totalmente com a ocorrência de $z_j = 1$ e não com a ocorrência de $z_j = 0$. Isto significa que $a_{k1} = t_1$ e $a_{k0} = 0$ o qual resulta $\pi_k = 1$. Em contraste, há muitos modos nos quais uma entrada pode ser má discriminatória. No caso, por exemplo quando $i_k = 1$ para exatamente a metade das instâncias de $z_j = 0$ e $z_j = 1$, então $\frac{a_{k1}}{t_1} = \frac{a_{k0}}{t_0} = 0.5$ e $\pi_k = 0$. Similarmente, se $i_k = 1$ em todas amostras de treinamento, então $\frac{a_{k1}}{t_1} = \frac{a_{k0}}{t_0} = 1$ e $\pi_k = 0$. O mesmo é válido se $i_k = 0$ em todas amostras de treinamento.

O valor de a_{k1} pode ser calculado por produto interno mostrado na Equação 4.34.

$$a_{k1} = I^k \cdot z = \sum_{j=1}^{t} I_j^k z_j \tag{4.34}$$

onde I^k é um vetor coluna dos bits k de todas as entradas. Similarmente, a_{k0} pode ser também calculado como mostrado na Equação 4.35.

$$a_{k0} = I^k \cdot \bar{z} = \sum_{j=1}^{t} I_j^k (1 - z_j) \tag{4.35}$$

De posse do coeficiente de discriminação π_k, é definida uma alternativa para distância de Hamming a qual é chamada distância de discriminação (DD). Dado dois vetores de entrada x e y representados como: $x = (x_1, x_2, ..., x_n)$ e $y = (y_1, y_2, ..., y_n)$, a distância de discriminação é definida como na Equação 4.36.

$$DD(x, y) = \sum_{k=1}^{n} \pi_k.mag[x_k - y_k] \tag{4.36}$$

Ou de outra forma mostrada na Equação 4.37.

$$DD(x, y) = \sum_{k=1}^{n} \pi_k \qquad \forall x_k \neq y_k \tag{4.37}$$

Aplicando a Equação 4.37 para calcular a distância de um ponto x à origem, obtém-se a Equação 4.38.

$$DD(x, 0) = \sum_{k=1}^{n} \pi_k \qquad \forall x_k = 1 \tag{4.38}$$

Assim, é fácil provar que para dois pontos diferentes x e y, a distância de discriminação entre eles é dada pela Equação 4.39.

$$DD(x, y) = |DD(x, 0) - DD(y, 0)| \tag{4.39}$$

Assim é possível calcular e armazenar a distância de cada amostra à origem. De posse destes valores, é possível calcular facilmente a distância entre quaisquer pontos tomados dois a dois.

Fazendo uma nova formulação, obtém-se a Equação 4.40 a partir de $\frac{a_{k1}}{t_1}$.

$$\frac{a_{k1}}{t_1} = \frac{\text{número de amostras com } i_k = 1 \text{ e } z = 1}{\text{número de amostras com } z = 1} \tag{4.40}$$

Dividindo o numerador e denominador da Equação 4.40 por t (total de amostras), obtém-se a Equação 4.41.

$$\frac{a_{k1}}{t_1} = \frac{\frac{\text{número de amostras com } i_k = 1 \text{ e } z = 1}{\text{número total de amostras } t}}{\frac{\text{número de amostras com } z = 1}{\text{número total de amostras } t}} \tag{4.41}$$

Colocando a Equação 4.41 em forma de probabilidade, uma vez que é possível estimar a probabilidade pela freqüência, obtém-se a Equação 4.42.

$$\frac{a_{k1}}{t_1} = \frac{P[(z = 1) \cap (i_k = 1)]}{P(z = 1)} = P(i_k = 1 | z = 1) \tag{4.42}$$

Isto indica a probabilidade do bit i_k ser 1, dado a classe $z = 1$.

Fazendo o mesmo desenvolvimento para $\frac{a_{k0}}{t_0}$, partindo da Equação 4.43:

$$\frac{a_{k0}}{t_0} = \frac{\text{número de amostras com } i_k = 1 \text{ e } z = 0}{\text{número de amostras com } z = 0} \tag{4.43}$$

obtém-se a Equação 4.44.

$$\frac{a_{k0}}{t_0} = \frac{P[(z = 0) \cap (i_k = 1)]}{P(z = 0)} = P(i_k = 1|z = 0) \tag{4.44}$$

Assim, π_k pode ser escrito como indicado na Equação 4.45, simplesmente substituindo na Equação 4.33 as Equações 4.44 e 4.42.

$$\pi_k = |P(i_k = 1|z = 1) - P(i_k = 1|z = 0)| \tag{4.45}$$

Isto indica qual a probabilidade (chance) do bit k poder contribuir na diferenciação (discriminação) de uma classe da outra.

Desenvolvendo $\frac{a_{k0}}{t_0}$ de outra maneira, partindo da Equação 4.43 e dividindo o numerador e denominador por t (total de amostras), resulta na Equação 4.46.

$$\frac{a_{k0}}{t_0} = \frac{\frac{\text{número de amostras com } i_k = 1 \text{ e } z = 0}{\text{número total de amostras } t}}{\frac{\text{número de amostras com } z = 0}{\text{número total de amostras } t}} \tag{4.46}$$

Como $t_0 = t - t_1$, então obtém-se a Equação 4.47.

$$\frac{a_{k0}}{t_0} = \frac{\frac{\text{número de amostras com } i_k = 1 \text{ e } z = 0}{\text{número total de amostras } t}}{1 - \frac{\text{número de amostras com } z = 1}{\text{número total de amostras } t}} \tag{4.47}$$

Colocando em forma de probabilidade, obtém-se a Equação 4.48.

$$\frac{a_{k0}}{t_0} = \frac{P[(z = 0) \cap (i_k = 1)]}{1 - P(z = 1)} \tag{4.48}$$

Recalculando π_k pela substituição das Equações 4.48 e 4.42 na Equação 4.33, obtém-se a Equação 4.49,

$$\pi_k = mag\left[\frac{P[(z = 1) \cap P(i_k = 1)]}{P(z = 1)} - \frac{P[(z = 0) \cap P(i_k = 1)]}{1 - P(z = 1)}\right] \tag{4.49}$$

a qual após algumas deduções, chega-se na Equação 4.50.

$$\pi_k = mag\left[\frac{P[(i_k = 1)|(z = 1)] - P(i_k = 1)}{P(z = 0)}\right] \tag{4.50}$$

Assim, os coeficientes de discriminação π_k podem ser calculados pelas probabilidades de ocorrência dos k bits como mostrado pela Equação 4.45 ou 4.50. Estas duas equações são uma contribuição deste trabalho de tese para

a distância de discriminação calculada pela Equação 4.36.

Distância de Bayes

Considere um problema de reconhecimento de padrões de duas classes. Suponha dois exemplos (amostras) em $[0,1]^n$:

$$x = (x_1, ..., x_n)$$

$$y = (y_1, ..., y_n)$$

Suponha que $p(c_1|x)$ seja a probabilidade que o exemplo x seja da classe c_1. Então:

$$r(x,y) = p(c_1|x)p(c_2|y) + p(c_2|x)p(c_1|y) \tag{4.51}$$

é a probabilidade de erro de classificação de x pela regra do vizinho-mais-próximo dado que o vizinho-mais-próximo de x usando uma métrica particular é y (Blanzieri & Ricci 1999b). Também:

$$r^*(x) = 2p(c_1|x)p(c_2|x) = 2p(c_1|x)(1 - p(c_1|x)) \tag{4.52}$$

é a probabilidade de erro de classificação de x pela regra do vizinho-mais-próximo, dado um conjunto hipotético infinito.

Short e Fukunaga (Short & Fukunaga 1981) mostraram que minimizando a expectativa:

$$E[(r(x,y) - r^*(x))^2] \tag{4.53}$$

é equivalente a minimizar $E[(p(c_1|x) - p(c_1|y))^2]$, então a melhor métrica é:

$$D(x,y) = |p(c_1|x) - p(c_1|y)| \tag{4.54}$$

Para problemas de múltiplas classes, o cálculo da métrica se torna:

$$D(x,y) = \sum_{i=1}^{C} |p(c_i|x) - p(c_i|y)| \tag{4.55}$$

A estimação de $p(c_i|x)$ e $p(c_i|y)$ é a dificuldade desta métrica. Ela pode ser estimada utilizando o teorema de Bayes:

$$p(c_i|x) = \frac{p(x|c_i)p(c_i)}{p(x)} = \frac{p(x|c_i)p(c_i)}{\sum_{k=1}^{C} p(x|c_k)p(c_k)} \tag{4.56}$$

Assim, o problema se reduz a estimar $p(x|c_k)$ e $p(c_k)$.

A estimativa de probabilidade mais simples é baseada em contagem de

freqüência. Deste modo, é possível estimar $p(c_i)$ com:

$$p^*(c_i) = \frac{N(c_i)}{N} \tag{4.57}$$

onde $N(c_i)$ é o número de casos que são da classe c_i e N é o tamanho do conjunto das amostras. Infelizmente, estimativa de probabilidade baseada em freqüência resulta em desempenho pobre se o tamanho das amostras é pequeno. Entretanto, o desempenho desta métrica é comparável ao classificador de Bayes (Blanzieri & Ricci 1999a). Além disto, a métrica baseada em estimação de probabilidade garante um esquema natural de unificação para tratar com tipos de características nominais e numéricas.

Distância MRM (Minimum Risk Metric)

É uma métrica muito simples que minimiza diretamente o risco de erro de classificação (Blanzieri & Ricci 1999b). Dado um exemplo x da classe c_i e um vizinho-mais-próximo y, o risco finito de erro de classificação de x é dado por:

$$r(x, y) = p(c_i|x)(1 - p(c_i|y)) \tag{4.58}$$

O risco finito total é a soma dos riscos estendidos a todas as diferentes classes e é dado por:

$$r(x, y) = \sum_{i=1}^{C} p(c_i|x)(1 - p(c_i|y)) \tag{4.59}$$

Minimizando diretamente o risco, obtém-se a métrica:

$$D_{MRM}(x, y) = r(x, y) = \sum_{i=1}^{C} p(c_i|x)(1 - p(c_i|y)) \tag{4.60}$$

Novamente, a estimação de $p(c_i|x)$ e $p(c_i|y)$ é a dificuldade desta métrica. Ela pode ser estimada utilizando o teorema de Bayes, como na métrica anterior, e assim o problema se reduz a estimar $p(x|c_i)$ e $p(c_i)$.

Nova métrica

Propõe-se neste trabalho de tese a definição de uma nova métrica baseada nos coeficientes usados para cálculo da verossimilhança na equação de Bayes: $p(x_i|c_j)$. A idéia parte da suposição de que a diferença absoluta entre os valores dos coeficientes relativos ao mesmo atributo para duas amostras, resulta numa medida de distância entre as amostras devido ao mesmo atributo. Considerando este cálculo para todos os atributos e somando a contribuição de todos, resulta na medida de distância total entre as duas amostras.

Assim, esta nova métrica é definida como:

$$D(x,y) = \sum_{i=1}^{d} \sum_{j=1}^{C} |p(x_i|c_j) - p(y_i|c_j)| \tag{4.61}$$

Novamente a estimação de $p(x_i|c_j)$ e $p(y_i|c_j)$ é feita por contagem de freqüência. Alguns experimentos são apresentados posteriormente com o intuito de validar a métrica e comparar o seu desempenho com outras métricas.

4.3.2 Regra do Vizinho-mais-próximo

Uma das mais atrativas regras de decisão não paramétricas é a regra do vizinho-mais-próximo (*nearest neighbor decision rule* - NN) (Cover & Hart 1967). A regra do vizinho-mais-próximo é bastante utilizada, principalmente por causa de sua simplicidade conceitual que leva a uma implementação direta. Este método é explicado a seguir com o histórico de sua evolução.

Suponha que $D^n = \{x_1, ..., x_n\}$ seja um conjunto de amostras rotuladas e que $x' \in D^n$ é a amostra mais próxima a um ponto de teste x. Então a regra do vizinho-mais-próximo para classificação de x é associar a ele o rótulo associado com x'.

Esta regra pode ser entendida mais facilmente utilizando recursos gráficos. Supondo um conjunto de dados de duas dimensões em $\Re(\mathrm{x} \in \Re^2)$, separados em duas classes: círculo e cruz. A Figura 4.1 exemplifica um conjunto de dados hipotético, onde o dado identificado como "asterisco" representa a amostra desconhecida que deve ser classificada.

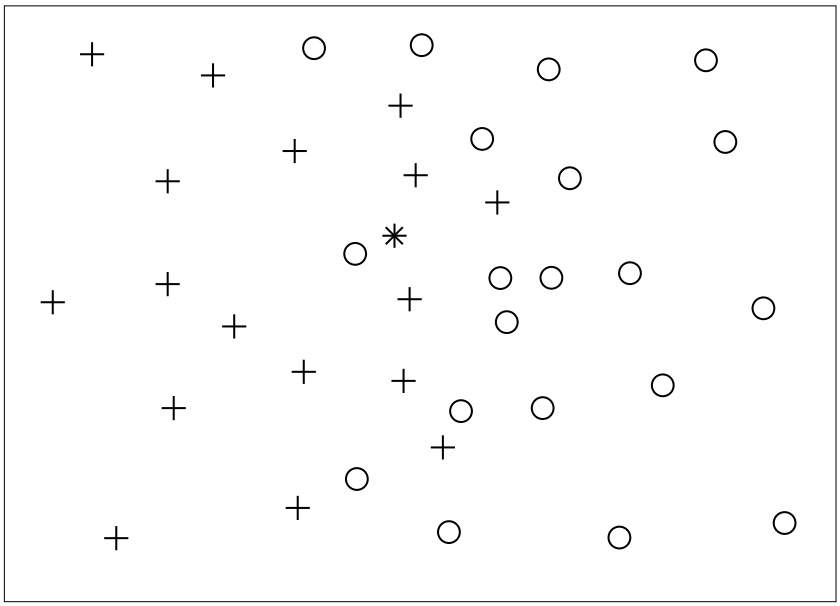


Figura 4.1: Exemplo de conjunto de dados de duas classes.

Pela regra do vizinho-mais-próximo, a amostra desconhecida será associada a mesma classe da amostra conhecida mais próxima, ou seja, círculo. A Figura 4.2 mostra a regra.

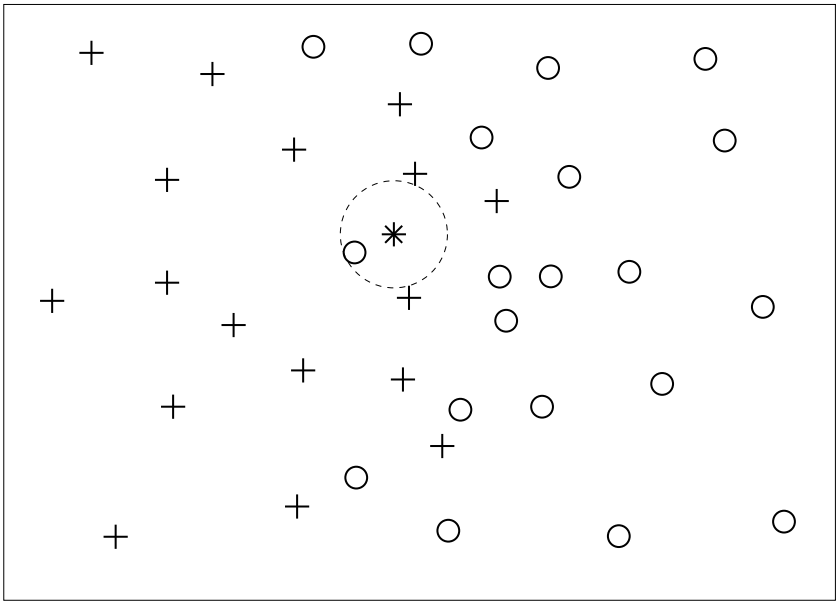


Figura 4.2: Regra do vizinho-mais-próximo ($k = 1$).

Pode-se dizer que esta regra de decisão foi difundida com o trabalho de Cover (Cover & Hart 1967; Cover 1968), quando foi provado que esta regra tem uma probabilidade de erro que é sempre menor do que o dobro da probabilidade do erro de Bayes. Com o advento da revolução do computador nos anos 90, em termos de memórias baratas e alta velocidade de processamento, tem ocorrido um ressurgimento das técnicas de vizinho-mais-próximo, e tem trazido a regra *NN* para a dianteira como uma técnica de classificação não paramétrica popular (Dasarathy, Sánchez, & Townsend 2000).

Uma variação da regra do vizinho-mais-próximo, classifica x pela associação ao rótulo mais freqüentemente representado entre os k exemplos mais próximos, ou seja, uma classificação é feita examinando os rótulos dos k vizinhos-mais-próximos e decidindo pela maioria (regra do k-vizinho-mais-próximo - kNN). A Figura 4.3 mostra o mesmo exemplo anterior usando o método para $k = 3$. Assim a amostra desconhecida é associada à classe cruz.

A regra do vizinho-mais-próximo simples ($k = 1$) seleciona w_m com probabilidade $P(w_m|x)$. A regra dos k vizinhos mais próximos seleciona w_m se uma maioria dos k vizinhos mais próximos são rotulados w_m, um evento de probabilidade:

$$\sum_{i=\frac{k+1}{2}}^{k} \left(\begin{array}{c} k \\ i \end{array} \right) P(w_m|x)^i [1 - P(w_m|x)]^{k-i} \tag{4.62}$$

Em geral, quanto maior o valor de k, maior a probabilidade que w_m seja selecionado.

A regra dos k vizinhos mais próximos pode ser vista como uma tentativa para estimar a probabilidade *a posteriori* $P(w_i|x)$ das amostras. É re-

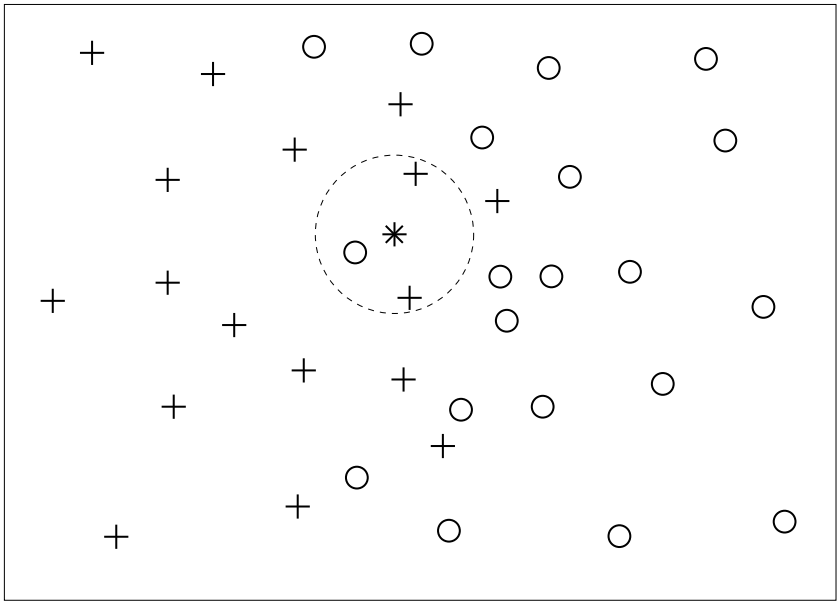


Figura 4.3: Regra do k vizinho-mais-próximo ($k = 3$).

comendável usar um alto valor de k para obter uma estimativa confiável. Por outro lado, é desejável também que todos os k vizinhos mais próximos x' estejam muito mais próximos de x para ter certeza de que $P(w_i|x')$ é aproximadamente o mesmo que $P(w_i|x)$. Isto força a escolha de k como uma pequena fração das amostras. É apenas no limite de n quando tende para infinito que pode-se assegurar o comportamento da regra do k-vizinho-mais-próximo perto do ótimo. Ou seja, é no caso assintótico, quando o número de amostras torna-se arbitrariamente grande, os k vizinhos mais próximos estarão perto da amostra sendo classificada (Koplowitz & Brown 1981).

Uma generalização da regra do vizinho-mais-próximo (*k1-NN*) é proposta por Tomek (Tomek 1976b), onde *k1* vizinhos dentre os *k* vizinhos mais próximos (sendo $0 < k1 < k$) são utilizados para decidir a classe da amostra desconhecida. Outras derivações do kNN são propostas na literatura com objetivo de incrementar o desempenho do método (Luk & Macleod 1986; Hattori & Takahashi 1999).

Um dos principais inconvenientes da regra do vizinho-mais-próximo é o alto custo computacional, tanto na parte de armazenamento das amostras de treinamento, quanto no cálculo das distâncias entre as amostras de treinamento e a amostra desconhecida. Desde que a regra do vizinho-mais-próximo seja um procedimento de pesquisa no arquivo de dados conhecidos, todas as amostras devem ser examinadas para classificar cada amostra desconhecida. Então quando o arquivo de dados torna-se grande, armazenamento e requerimento computacional podem tornar a aplicação proibitivamente cara.

Para minimizar estes problemas, Hart propôs a regra de decisão baseada no vizinho-mais-próximo condensado (*condensed nearest neighbor - CNN*) (Hart

1968). Nesta técnica, o conjunto de amostras é reduzido levando-se em conta a classificação incorreta das amostras de treinamento pelas próprias amostras de treinamento vizinhas. As amostras selecionadas formam o que ele denominou **"subconjunto consistente"**, ou seja, um subconjunto de amostras que quando usado como conjunto de referência para a regra do vizinho-mais-próximo, classifica todas as amostras de treinamento corretamente.

Não é garantido que o subconjunto consistente gerado pela regra de Hart seja mínimo, inclusive, a determinação do subconjunto consistente mínimo é um problema de difícil solução. Gates (Gates 1972) propõe um método iterativo (*reduced nearest neighbor - RNN*) para reduzir o subconjunto consistente, testando por eliminação cada amostra do subconjunto consistente.

Tomek propôs em outro trabalho (Tomek 1976c) duas modificações na regra do vizinho-mais-próximo condensado baseado na suposição de que as amostras selecionadas são justamente aquelas próximas da fronteira de decisão da classe. Entretanto, Toussaint (Toussaint 1994) provou que um destes métodos de Tomek não é válido.

Gowda propôs em seu trabalho (Gowda & Krishna 1979) um algoritmo para determinação do subconjunto consistente selecionando pares de amostras mais próximas que são de classes diferentes, ou seja, amostras perto da fronteira de decisão, e posteriormente a aplicação da técnica de redução de Gates. Entretanto, o enfoque principal do seu trabalho era justamente a métrica de distância utilizada para cálculo dos vizinhos mais próximos. Hamza (Hamza, Krim, & Karacali 2003) propõe um algoritmo semelhante ao de Gowda que também diminui os dados de treinamento, mas usando o princípio da minimização do risco estrutural como principal objetivo, semelhante ao método utilizado em máquinas de vetores de suporte (Lorena & de Carvalho 2003).

Wilson (Wilson 1972) em seu trabalho propõe um outro método que além de reduzir a quantidade de amostras, melhora o risco de erro de classificação quando comparado à regra do vizinho-mais-próximo pura (Tomek 1976a), ficando o risco próximo ao risco de Bayes (risco ótimo). Resumidamente, este método utiliza as amostras restantes do conjunto de treinamento quando retiradas as amostras pertencentes ao subconjunto consistente proposto por Hart. A convergência (desempenho assintótico) da regra de Wilson é provada por Wagner (Wagner 1973), ou seja, na medida que a quantidade de amostras aumenta, o risco de erro de classificação se aproxima ao erro de Bayes.

Várias técnicas para reduzir a quantidade de amostras de treinamento foram propostas na literatura desde então (Ritter, Woodruff, Lowry, & Isenhour 1975; Kovács & Guerrieri 1991; Aha, Kibler, & Albert 1991; Dasarathy 1994; Kuncheva & Bezdek 1998; Hattori & Takahashi 2000). Algumas destas técnicas propõem a substituição por amostras artificiais geradas a partir das

amostras de treinamento para aplicação da regra do vizinho-mais-próximo (Chang 1974; Geva & Sitte 1991; Chaudhuri, Murthy, & Chaudhuri 1994). Outras técnicas misturam as propostas de seleção de amostras (filtragem) e abstração (substituição) (Lam, Keung, & Liu 2002). Ainda outros métodos de seleção utilizam técnica de quantização vetorial (Xie, Laszlo, & Ward 1993; Yen, Young, & Nagurka 2004).

Algumas propostas de melhoramento das técnicas de condensação e classificação pela regra do vizinho-mais-próximo baseadas em métodos heurísticos aparecem na literatura (Sánchez, Barandela, Alejo, & Marqués 2001; Wu, Ianakiev, & Govindaraju 2002). Comparações entre os métodos baseados na regra do vizinho-mais-próximo são realizados (Dasarathy, Sánchez, & Townsend 2000; Wilson & Martinez 2000), além de comparações com máquinas de vetores de suporte (Vincent & Bengio 2001).

O desempenho da regra do vizinho-mais-próximo também depende diretamente da métrica de distância utilizada. Um estudo mais aprofundado da influência da métrica de distância para o desempenho da regra do vizinho-mais-próximo foi realizado por Short (Short & Fukunaga 1981). O melhoramento do desempenho em classificação baseada na regra do vizinho-mais-próximo pode ser obtido pela escolha apropriada da medida de distância. Vários tipos de métricas são propostas na literatura para utilização com a regra do vizinho-mais-próximo com o intuito de melhorar o desempenho do método (Gowda & Krishna 1979; Fukunaga & Flick 1984; Payne & Edwards 1998; Weinshall, Jacobs, & Gdalyahu 1999; Avesani, Blanzieri, & Ricci 1999; Peng, Heisterkamp, & Dai 2003; Xing, Ng, Jordan, & Russell 2003). Algumas técnicas utilizam métricas adaptativas que dependem diretamente das amostras de treinamento para cálculo das distâncias (Dudani 1976; Aleksander, Clarke, & Braga 1994; Hastie & Tibshirani 1996; Ricci & Avesani 1999; Paredes & Vidal 1998; Paredes & Vidal 2000; Domeniconi, Peng, & Gunopulo 2000; Peng, Heisterkamp, & Dai 2002; Domeniconi & Gunopulos 2002).

Outro problema que interfere no desempenho da regra do vizinho-mais-próximo é a dimensionalidade das amostras de treinamento. A taxa de convergência do classificador pela regra do vizinho-mais-próximo diminui dramaticamente na medida em que a dimensionalidade do espaço de características incrementa (Snapp, Psaltis, & Venkatesh 1991). Algumas técnicas são propostas na literatura na tentativa de minimizar este problema (Nene & Nayar 1997; Liu, Moore, & Gray 2004).

A seguir é descrita a influência dos atributos para a regra do vizinho-mais-próximo. No capítulo seguinte, são detalhados os métodos presentes na literatura para diminuir a quantidade de amostras de treinamento e conseqüentemente o custo computacional da regra de decisão pelo vizinho-mais-próximo.

Influência dos Atributos

Um atributo (ou característica) de uma amostra é irrelevante se ele não contribui em nada para a tarefa de classificação do padrão. A regra do vizinho-mais-próximo é sensível a inclusão de atributos irrelevantes no conjunto de dados. Geralmente, o desempenho da classificação degrada com o incremento de atributos irrelevantes.

A seleção de atributos é um processo de identificação do subconjunto de atributos relevantes dentre os atributos presentes no conjunto de dados. Algumas técnicas de implementação da regra do vizinho-mais-próximo utilizam pesos para identificar atributos irrelevantes, de tal forma que atributos irrelevantes contribuem pouco para a tarefa de classificação.

Pesos dos atributos são determinados avaliando o algoritmo do vizinho-mais-próximo nos dados de treinamento. Um vetor de pesos de atributo é gerado inicialmente com valores iguais para todos os atributos. A cada padrão de treinamento que é avaliado, os pesos são ajustados de tal forma a penalizar os pesos dos atributos que produzem classificação errada, e incrementar os pesos dos atributos que produzem classificação correta. O resultado dos pesos pode ser usado para determinar quais atributos devem ser mantidos no subconjunto de atributos, e quais devem ser descartados.

4.3.3 Experimentos com dados binários

Foram realizados alguns experimentos com o classificador baseado na regra dos k vizinhos mais próximos (kNN) utilizando dados binários como entrada, com o objetivo de testar tanto o método kNN quanto algumas das métricas já descritas. Inicialmente são mostrados os detalhes e resultados obtidos com dados gerados artificialmente, e em seguida com dados de problemas reais. Na próxima seção do texto é apresentada uma síntese dos resultados obtidos.

Dados sintéticos

Foram geradas artificialmente 2974 amostras de valores binários com 24 bits cada, divididas em duas classes ("0" e "1"). Na Figura 4.4 é apresentado o histograma dos dados gerados para cada classe de dados, levando em conta a freqüência com que cada bit está ativado (nível lógico "1"). Observa-se que há uma tendência na preferência de ativação de alguns bits, diferente para cada classe. Para os dados gerados para a classe "0", os bits próximos de 5 e 18 tem uma freqüência maior de ativação. Para os dados gerados para a classe "1", os bits próximos de 12 tem uma freqüência maior de ativação.

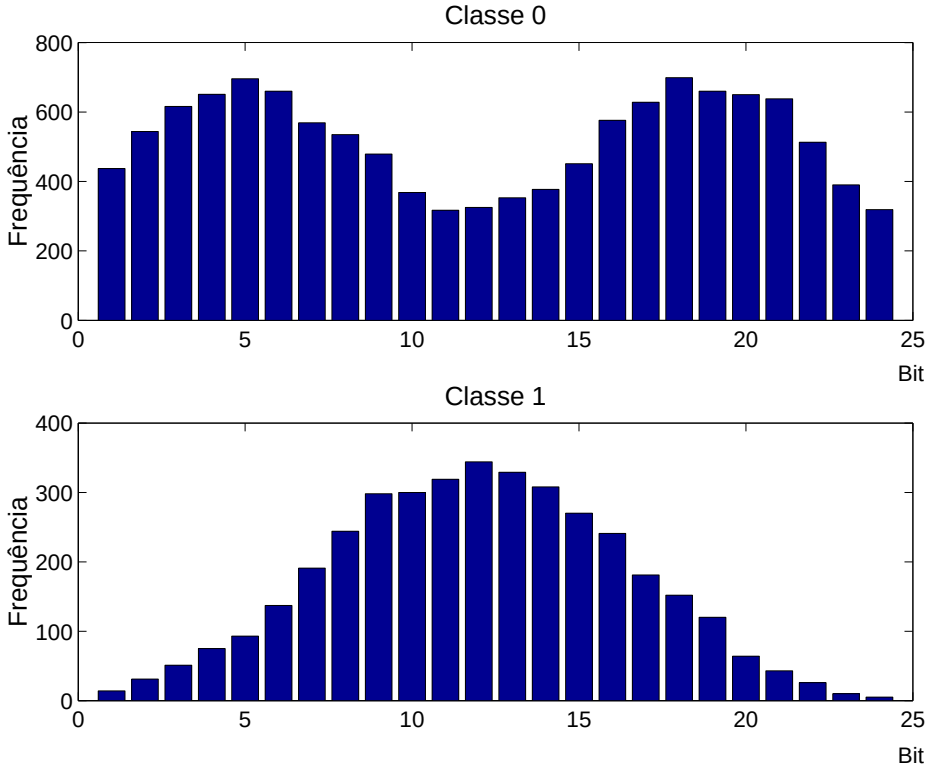


Figura 4.4: Histograma dos dados sintéticos.

A ordem de armazenamento das amostras é aleatória, podendo ser de qualquer classe. Do total de amostras, 2380 (80%) foram utilizadas como referên-

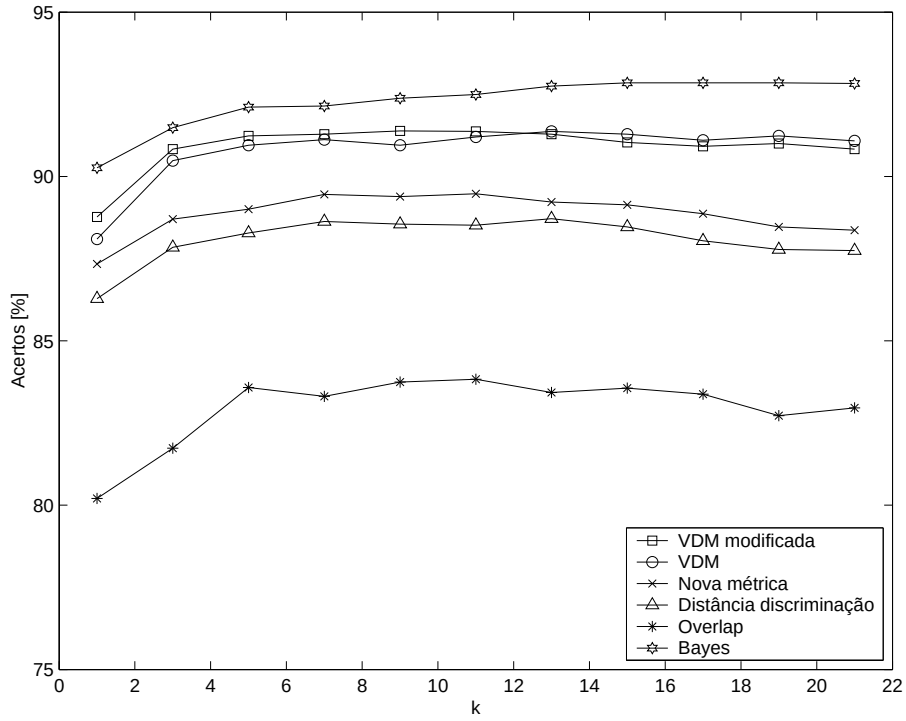


Figura 4.5: Número médio de acertos do classificador kNN em função de k, utilizando diferentes métricas na solução do problema de dados sintéticos.

cia para o classificador kNN (treinamento), e 594 (20%) foram reservadas para teste. A geração destas amostras foi repetida 10 vezes, formando 10 conjuntos de amostras artificiais de treinamento e teste.

O gráfico 4.5 apresenta os resultados obtidos pelo classificador kNN para os dados de teste em cada métrica, em função do parâmetro k (foram utilizados valores ímpares para k entre 1 e 21. Conforme mostrado no gráfico, o desempenho do classificador kNN foi melhor utilizando a métrica de distância de Bayes, enquanto o pior desempenho foi obtido com a métrica *overlap*. A nova métrica proposta neste trabalho de tese obteve um desempenho médio entre a pior e melhor métrica. A métrica VDM modificada obteve resultados melhores que a VDM original para baixos valores de k ($k < 12$), enquanto que o inverso é observado para valores mais altos de k ($k > 12$).

Dados caracteres manuscritos

Neste problema foi utilizado um conjunto de amostras de uma base de dados de domínio público (UCI Repository of Machine Learning Databases and Domain Theories) da Universidade da Califórnia (Blake & Merz 1998), referente a imagens digitalizadas de caracteres numéricos manuscritos (*Optical Recognition of Handwritten Digit*). Na Figura 4.6 são mostradas algumas amostras de caracteres manuscritos do banco de dados. Esta base de dados possui as seguintes características:

- Cada amostra compreende uma matriz 32x32 de bits {0 ou 1}, ou 1024 bits em forma de vetor.
- Cada amostra é classificada como sendo uma dentre 10 classes (correspondente à imagem dos caracteres de “0” a “9”).
- O conjunto total de dados possui 1934 amostras, das quais 189 correspondem ao caracter manuscrito “0”. A ordem das amostras é aleatória.

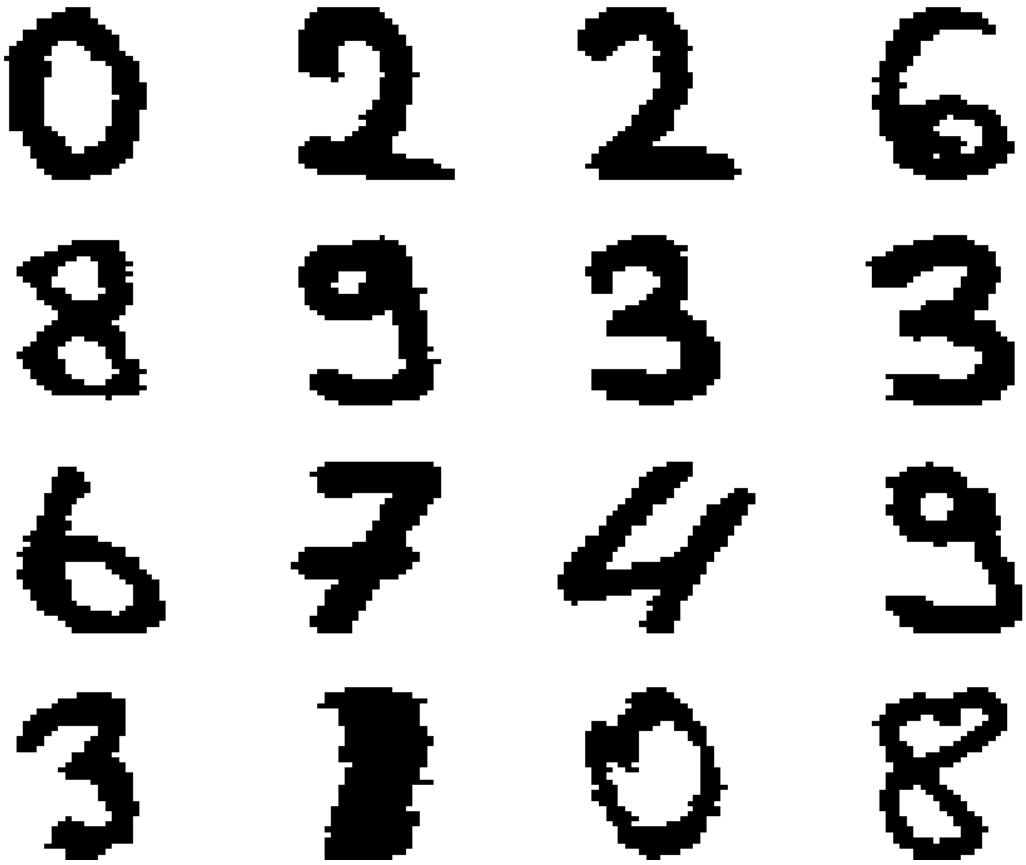


Figura 4.6: Exemplos de amostras de caracteres manuscritos digitalizados.

Neste trabalho foi utilizado apenas o problema da classificação do dígito “0” para efeito de simplificação. Assim o resultado possui um valor binário: classe “1” (dígito “0”) ou classe “0” (qualquer outro dígito). Na Figura 4.7 é mostrado o histograma da freqüência de ativação de cada bit (nível lógico “1” de cada bit do conjunto de dados). Do conjunto original de dados, 386 amostras (20%) foram reservadas para teste. O restante dos dados (1548 amostras - 80%) foram utilizadas para treinamento do classificador kNN.

O gráfico 4.8 apresenta os resultados obtidos pelo classificador kNN em cada métrica, em função do parâmetro k. Conforme mostrado no gráfico, o desempenho do classificador kNN foi melhor utilizando a métrica de distância *overlap*, empatando com a nova métrica proposta. As outras métricas

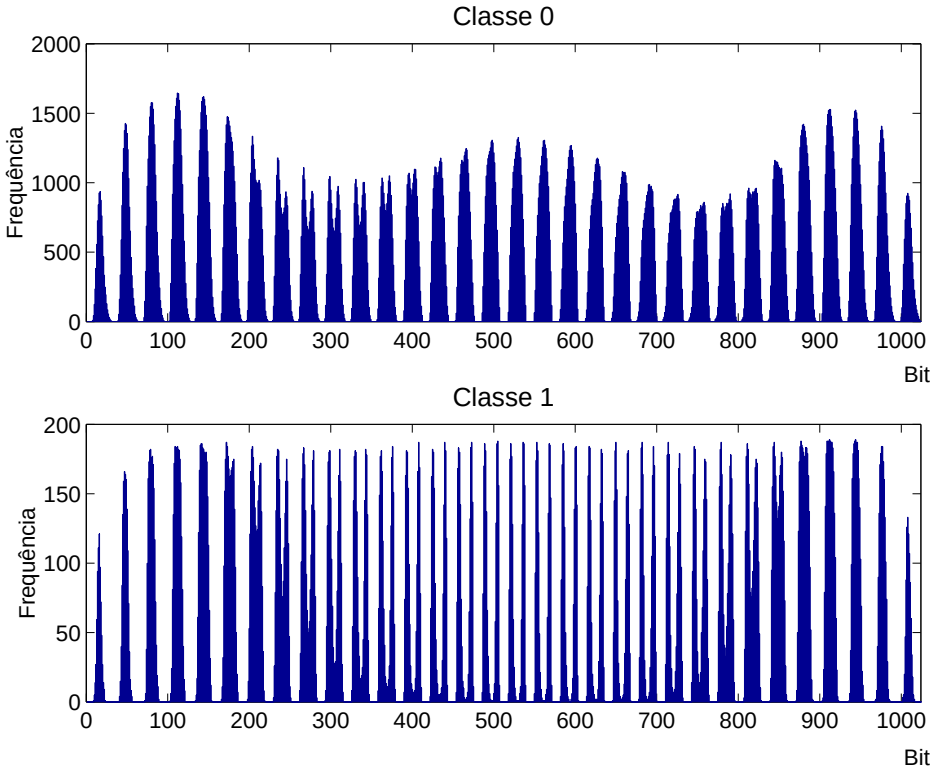


Figura 4.7: Histograma dos dados dos caracteres manuscritos.

obtiveram um resultado bem próximo, excetuando a métrica de Bayes que obteve um resultado pior. A métrica VDM modificada obteve um resultado ligeiramente melhor que a VDM original para baixos valores de k ($k < 11$), entretanto o inverso é notado para valores superiores de k ($k > 13$).

Dados do problema Gene

Este problema se refere a detecção de limites *intron*/*exon* (junções *splice*) em seqüências de nucleotide (Prechelt 1994). De uma janela de 60 elementos da seqüência do DNA, deve-se decidir quando o meio é um limite *intron*/*exon* (*donor*), ou um limite *exon*/*intron* (*acceptor*), ou nenhum destes. Cada nucleotide, o qual é um atributo nominal de 4 valores, é identificado por duas entradas binárias. Há 25% de *donors* e 25% de *acceptors* no conjunto de dados.

São características do conjunto de dados do problema Gene:

- número de entradas booleanas: 120
- número de saídas booleanas: 3
- número de exemplos: 3175

Para efeito de simplificação foi aplicado apenas o problema de detecção da existência de um limite, independente que seja um *donor* ou um *acceptor*.

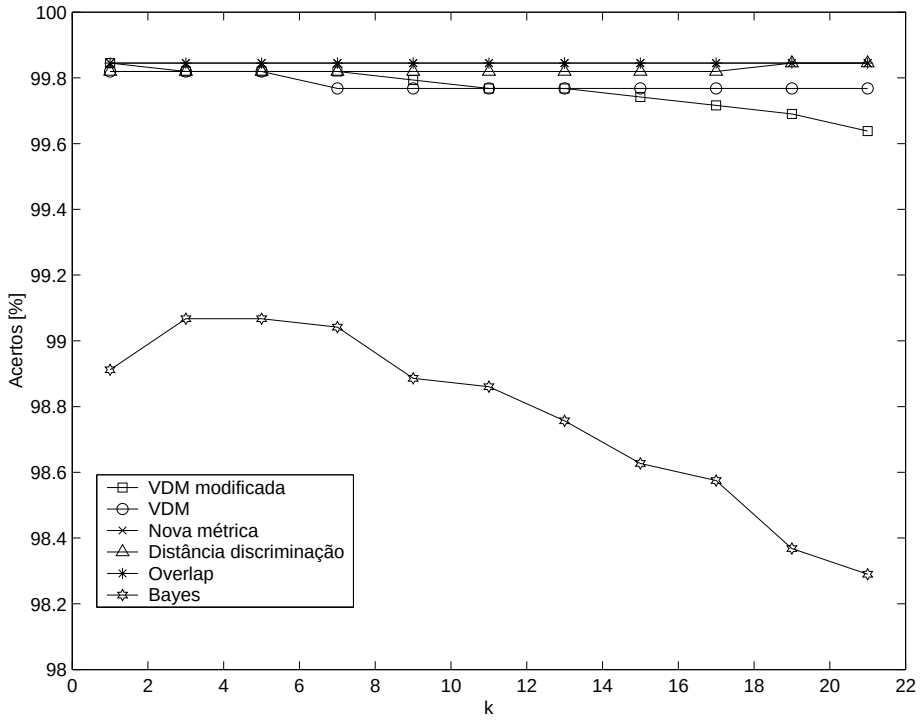


Figura 4.8: Número médio de acertos do classificador kNN em função de k, utilizando diferentes métricas na solução do problema dos caracteres manuscritos.

Assim, foi nomeado classe 0 para as amostras que contenham um limite, e classe 1 para amostras que não possuem um limite. A Figura 4.9 apresenta o histograma de freqüência de ativação (nível lógico "1") de cada bit do conjunto de dados separados para cada classe. Devido a existência de amostras ambíguas, então o conjunto total foi reduzido para 2989 amostras. Destas, 2391 (80%) foram separadas para treinamento do kNN, e o restante (598 amostras, 20%) para teste.

O gráfico 4.10 apresenta os resultados obtidos pelo classificador kNN em cada métrica, em função do parâmetro k para o problema de identificação de limites em genes. Conforme mostrado no gráfico, o desempenho do classificador kNN foi melhor utilizando a nova métrica proposta. A métrica VDM possui um desempenho semelhante para baixos valores de k ($k < 7$), mas para valores superiores de k ($k > 7$) esta métrica tem seu desempenho ligeiramente inferior à melhor métrica. A VDM modificada teve seu desempenho piorado em relação a VDM original. Por sua vez, a métrica *overlap* obteve o pior desempenho para este problema.

Dados do coração

Neste problema foi utilizado um conjunto de amostras de uma base de dados de domínio público (UCI *Repository of Machine Learning Databases and Domain Theories*) da Universidade da Califórnia (Blake & Merz 1998). O con-

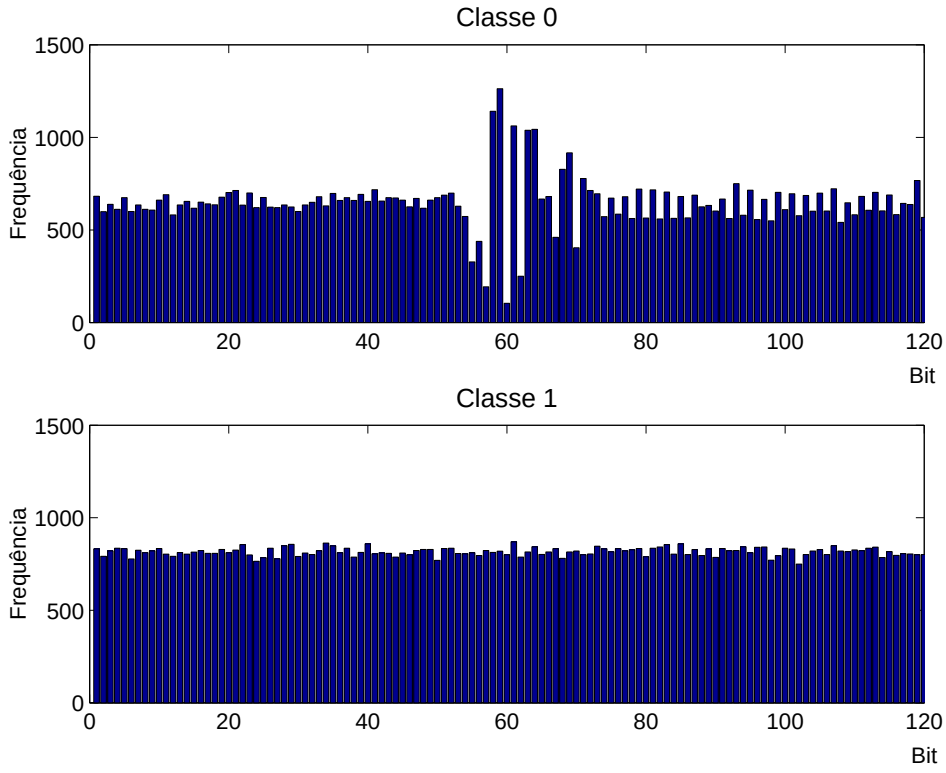


Figura 4.9: Histograma dos dados do problema Gene.

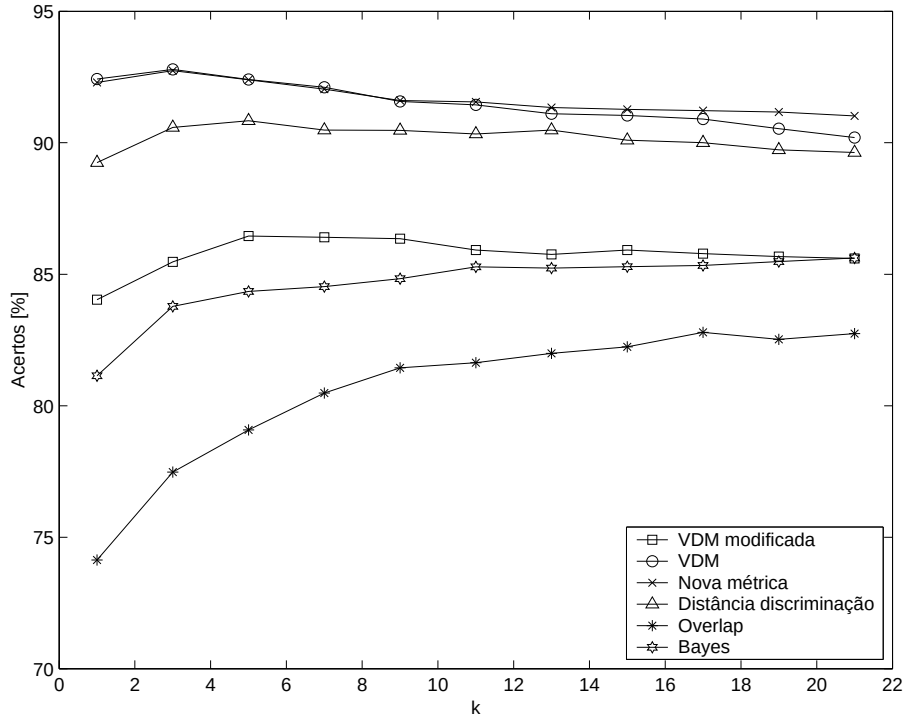


Figura 4.10: Número médio de acertos do classificador kNN em função de k, utilizando diferentes métricas na solução do problema de dados de genes.

junto de dados descreve o diagnóstico cardíaco de imagens de tomografia computadorizada por emissão de próton simples (SPECT - *Single Proton Emission Computed Tomography*). Cada um dos pacientes é classificado dentre duas categorias: normal (classe 1) e anormal (classe 0). A base de dados do conjunto de 267 imagens SPECT foi processada para extrair as características que sumarizam as imagens SPECT originais. Como resultado, 44 características foram criadas para cada paciente. O padrão foi processado posteriormente para obter 22 padrões de características binárias.

São características do conjunto de dados do problema do coração:

- número de entradas booleanas: 22
- número de saídas booleanas: 1
- número de exemplos: 267
- número de exemplos da classe 0: 55
- número de exemplos da classe 1: 212

A Figura 4.11 apresenta o histograma de freqüência de ativação (nível lógico "1") de cada bit do conjunto de dados separados para cada classe. Das 267 amostras, apenas 210 foram utilizadas pois 57 amostras possuíam resultados ambíguos. Para o classificador kNN, 168 (80%) amostras foram utilizadas como referência. Para realização dos testes, 42 (20%) amostras foram reservadas.

O gráfico 4.12 apresenta os resultados obtidos pelo classificador kNN em cada métrica, em função do parâmetro k para o problema do coração. Conforme mostrado no gráfico, o desempenho do classificador kNN variou bastante, não só em função do valor de k, como também em função da métrica escolhida. Para valores mais altos de k ($k > 7$), a métrica de Bayes obteve o melhor desempenho. Para valores específicos de k ($k = 5$ e $k = 7$) a nova métrica proposta obteve o melhor resultado. A métrica de pior desempenho para este problema foi a VDM. A VDM modificada apresentou uma melhora em relação a VDM original.

Dados do cogumelo

Foi utilizado um conjunto de amostras de uma base de dados de domínio público (Proben1) próprio para ser empregado com Redes Neurais na solução de problemas de classificação (Prechelt 1994). O problema escolhido é referente à classificação de cogumelos como venenosos ou não venenosos, levando em conta uma série de atributos. Esta base de dados possui as seguintes características:

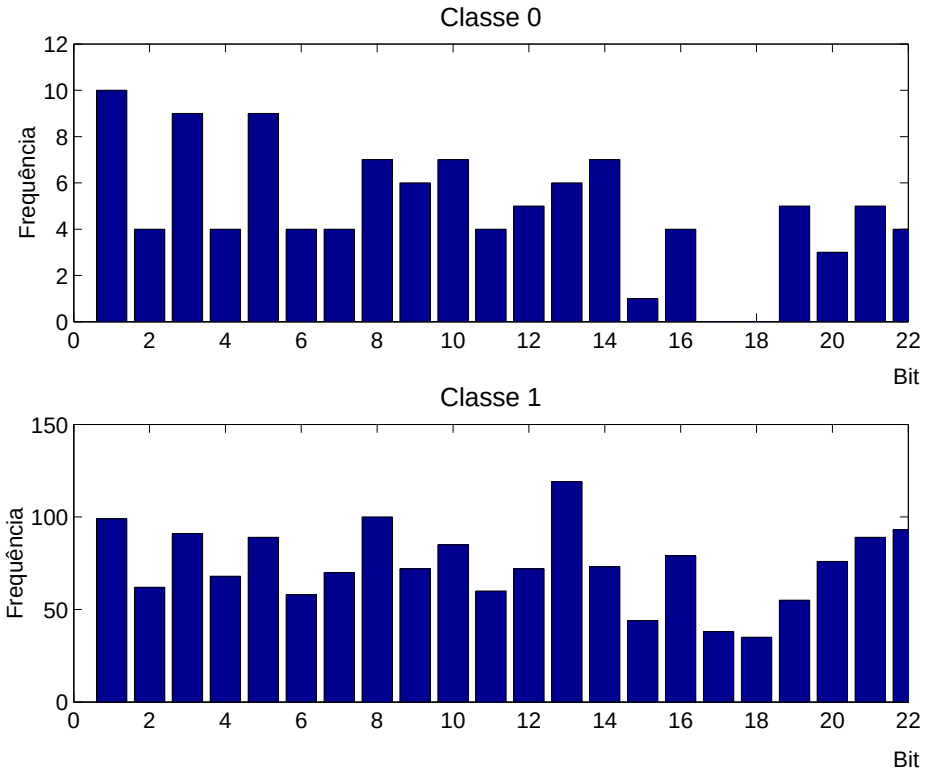


Figura 4.11: Histograma dos dados do problema do coração.

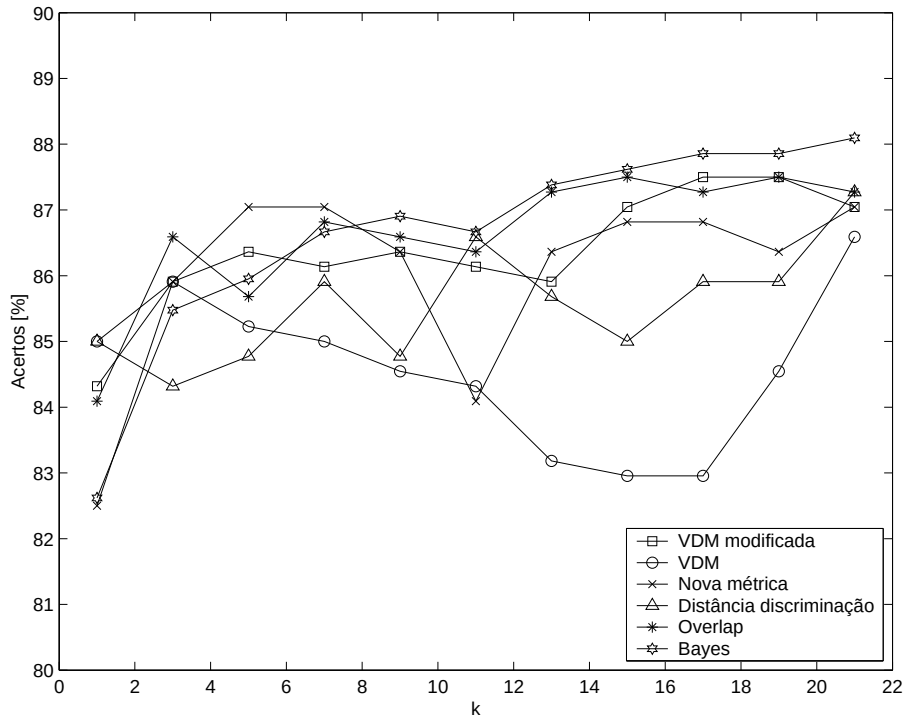


Figura 4.12: Número médio de acertos do classificador kNN em função de k, utilizando diferentes métricas na solução do problema de dados do coração.

- Cada amostra compreende 125 bits em forma de vetor. Cada grupo de bits representa uma característica do cogumelo como: forma, cor, odor, *habitat*, etc.

- Cada amostra é classificada como sendo 0 (não venenoso) ou 1 (venenoso).

- O conjunto total é de 8124 amostras, sendo que 4208 são de cogumelos não venenosos, e 3916 são de cogumelos venenosos. A ordem das amostras é aleatória.

A Figura 4.13 apresenta o histograma de freqüência de ativação (nível lógico "1") de cada bit do conjunto de dados separados para cada classe. Das 8124 amostras originais, 6500 (80%) foram separadas para treinamento do kNN, e o restante (1624 amostras, 20%) para teste.

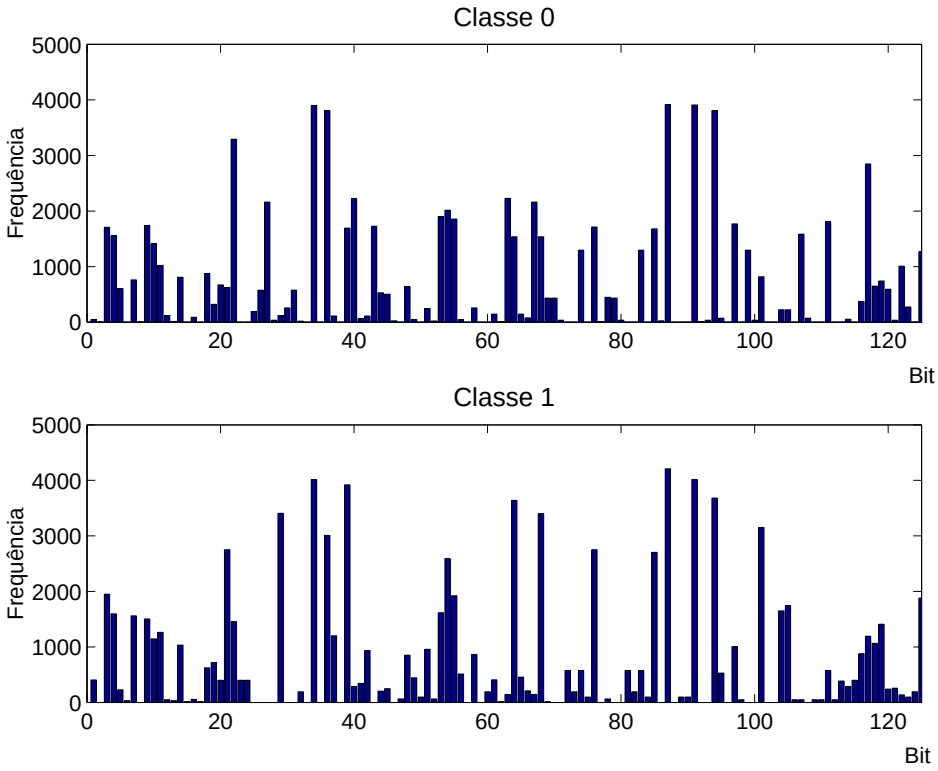


Figura 4.13: Histograma dos dados do cogumelo.

O gráfico 4.14 apresenta os resultados obtidos pelo classificador kNN em cada métrica, em função do parâmetro k para o problema do cogumelo. Conforme mostrado no gráfico, o desempenho do classificador kNN foi melhor utilizando a métrica *overlap*, seguido de perto pela nova métrica e a VDM modificada que empatam com a melhor métrica para $k < 10$. A métrica de Bayes resultou em um desempenho inferior em relação às outras métricas.

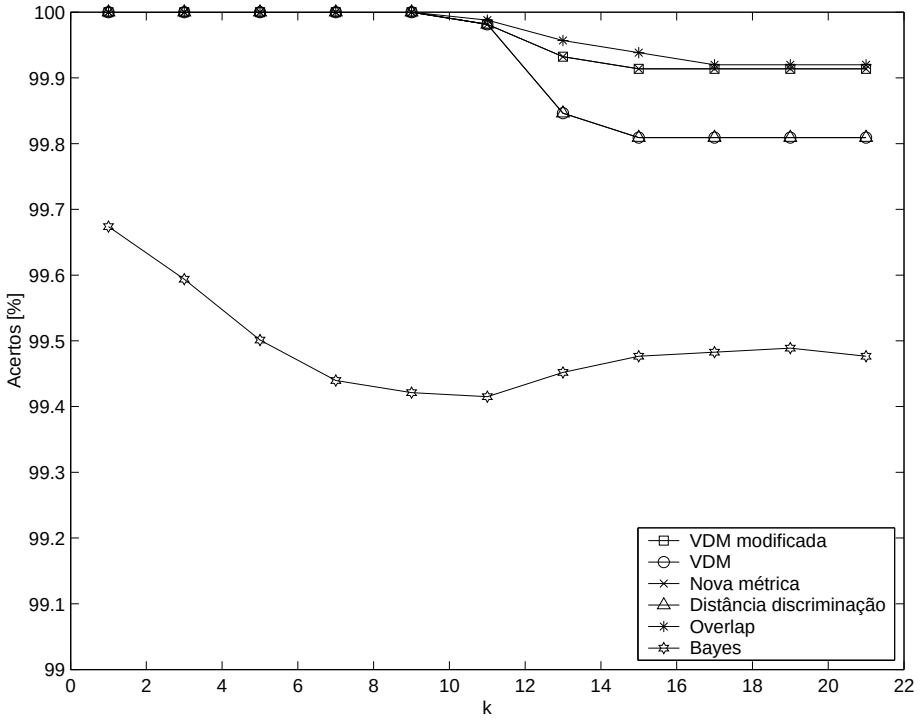


Figura 4.14: Número médio de acertos do classificador kNN em função de k, utilizando diferentes métricas na solução do problema de dados do cogumelo.

Tabela 4.6: Síntese dos resultados.

	kNN			**MLP**	**SVM**
Problema	**Métrica**	**k**	**Acertos (%)**	**Acertos (%)**	**Acertos (%)**
Artificial	Bayes	19	92,9±1,3	92,8±1,0	92,8±0,8
Caracter	overlap,Nova	1	99,9±0,1	99,7±0,2	99,8±0,1
Gene	VDM,Nova	3	92,8±1,2	86,3±1,1	87,6±1,1
Coração	Bayes	21	88,1±1,9	86,2±4,3	88,1±1,9
Cogumelo	overlap	1	100±0	100±0	100±0

4.3.4 Síntese dos Resultados

A Tabela 4.6 apresenta um resumo dos resultados obtidos pelo método kNN na solução dos problemas anteriores com dados binários de entrada. São apresentados a melhor métrica, o melhor valor do parâmetro k e o maior índice de acertos obtidos para os dados de teste em cada problema. Além disto, são mostrados os resultados obtidos para os mesmos problemas utilizando dois outros métodos: Rede Neural (Multilayer Perceptron - MLP) (Haykin 2001) e Máquina de Vetores de Suporte (Support Vector Machine - SVM)(Vapnik 1998). Cabe uma observação de que para validação no treinamento da Rede Neural, são reservados 20% dos dados de treinamento para este fim. Observa-se que o método kNN possui um desempenho comparável aos métodos MLP e SVM, sendo que em alguns problemas foram obtidos resultados superiores.

4.4 Conclusão

Reconhecimento de padrões é uma das principais tarefas de uma máquina de aprendizado. Se é conhecida a distribuição dos dados de treinamento (amostras conhecidas), um método paramétrico de reconhecimento (como a Regra de Bayes) torna-se a melhor solução para o problema. Entretanto, na maior parte das vezes não se conhece a distribuição das amostras de treinamento e métodos não paramétricos podem ser empregados (como a Regra do Vizinho-mais-próximo) na tentativa de se obter uma solução que se aproxima da ótima.

Foi apresentada a formulação da Regra de Bayes, juntamente com uma possível simplificação (*Regra Naive de Bayes*) baseada na independência das características das amostras de treinamento. Geralmente esta simplificação não decrementa significativamente o desempenho do classificador, como é demonstrado a partir de um exemplo.

Métodos não paramétricos dependem de uma medida de distância (ou similaridade) entre as amostras de treinamento. Assim, foram apresentadas algumas métricas de distâncias mais comuns, que tratam de características numéricas e nominais. A Distância de Discriminação (Aleksander, Clarke, & Braga 1994) foi detalhadamente apresentada e desenvolvida uma formulação estatística original para ela. Foi proposta uma modificação na métrica VDM com o objetivo de melhorar o desempenho do classificador kNN para determinados tipos de dados. Foi também proposta uma nova métrica baseada nos coeficientes de cálculo da verossimilhança da equação de Bayes, que se mostrou eficaz quando utilizada pela regra do vizinho-mais-próximo, conforme observado nos resultados dos experimentos. Algumas métricas apresentaram os melhores resultados para alguns problemas, mas para outros problemas as mesmas métricas apresentaram os piores resultados. A nova métrica proposta não apresentou o pior desempenho para nenhum dos problemas utilizados nos experimentos, sendo assim mais adequada para problemas em geral.

A Regra do Vizinho-mais-próximo foi descrita, juntamente com suas vantagens e desvantagens. Um histórico do desenvolvimento desta técnica nos últimos 30 anos também foi apresentado. Alguns experimentos com o classificador kNN utilizando dados binários foram realizados para comparar as métricas apresentadas. Observou-se que, dependendo da métrica escolhida e do valor do parâmetro k adotado, o desempenho do classificador kNN pode ser significativamente melhorado. Entretanto, a melhor escolha da métrica não é bem definida para cada tipo de problema. Também observou-se que o melhor desempenho obtido no kNN se compara aos métodos de classificação baseados em Rede Neural Artificial e Máquina de Vetores de Suporte.

CAPÍTULO 5

Seleção das Amostras

A regra do vizinho-mais-próximo e sua extensão a k vizinhos combinam sua simplicidade conceitual com o fato de que seu erro assintótico (tamanho de amostras infinito) é menor do que o dobro do erro de classificação de Bayes. Entretanto, para um grande conjunto de amostras de alta dimensão, o uso desta metodologia em aplicações reais torna-se computacionalmente pesado, devido ao grande número de distâncias a serem computadas para cada amostra de teste. Além disto, o conjunto de treinamento pode conter ruído ou amostras rotuladas erradas o que usualmente leva a diminuir o seu desempenho (Dasarathy, Sánchez, & Townsend 2000; Sánchez, Barandela, Marqués, Alejo, & Badenas 2003).

Neste capítulo, são descritas as principais técnicas utilizadas para selecionar as amostras de treinamento que melhor identificam o problema, de forma a poupar tanto o custo de armazenamento quanto o custo de processamento dos dados. Em seguida, é apresentada uma nova proposta de metodologia de seleção de amostras baseada na regra do vizinho-mais-próximo. Finalmente são apresentados alguns experimentos realizados e os resultados obtidos com a técnica proposta.

5.1 Métodos de Seleção de Amostras

Para combater o problema de armazenamento e custo computacional, são propostos na literatura alguns esquemas para condensar os dados originais (também referenciados na literatura como redução, refinamento, edição, pré-processamento e seleção de protótipos), de tal forma que poucos vetores de dados necessitam ser armazenados (Wilson & Martinez 2000).

Duas metodologias distintas têm sido propostas para minimizar estes problemas (Lam, Keung, & Liu 2002):

- Geração de protótipos ou **abstração**: cria um novo conjunto de amostras, ou pela geração de protótipos artificiais que resumem características representativas de instâncias similares (Lam, Keung, & Liu 2002), ou usando médias de peso ajustáveis do conjunto de treinamento original.

- Seleção de protótipos ou **filtragem**: seleciona um subconjunto particular dos protótipos e aplica a regra do vizinho-mais-próximo usando apenas as amostras selecionadas. Técnicas de filtragem reduzem o conjunto de dados retendo instâncias representativas do conjunto original dos dados. Dois métodos existem nesta linha:

  - Algoritmos de **redução** ou **condensação** para seleção do subconjunto mínimo de protótipos que levam aproximadamente ao mesmo desempenho que a regra do vizinho-mais-próximo usando todo o conjunto de treinamento.
  - Algoritmos de **edição** que eliminam protótipos rotulados errados do conjunto original e limpam a sobreposição entre regiões de classes diferentes.

Estas técnicas tendem a oferecer melhoramento em desempenho. Entretanto, a natureza heurística de muitos algoritmos de redução contrasta com o forte fundamento estatístico da regra do vizinho-mais-próximo (Dasarathy, Sánchez, & Townsend 2000).

Desde que edição é usada para limpar amostras rotuladas erradas e eliminar os protótipos sobrepostos (Toussaint 1994) do conjunto de treinamento, o principal objetivo é aumentar a precisão de reconhecimento produzindo um conjunto esterelizado. O fato que uma vantagem computacional pode ser ganha é um benefício secundário. Como resultado, a redução do conjunto de dados devido a edição é muitas vezes pequena quando comparado aos métodos de condensação, mas a precisão de reconhecimento para conjunto de treinamento editado é melhor.

Condensação, entretanto, define o método que recupera a fronteira de decisão do vizinho-mais-próximo com um subconjunto idealmente mínimo de protótipos. É utilizado primariamente com o propósito de reduzir o número de amostras para ganhar uma vantagem computacional. Ainda que isto seja feito de modo a minimizar a mudança na precisão do reconhecimento, uma característica infeliz de muitos procedimentos de condensação é que eles podem geralmente resultar em pobre desempenho de reconhecimento marginal.

Enquanto edição é muitas vezes preferível ao analista, condensação é de mais prática importância ao engenheiro desenvolvendo um sistema de reconhecimento de padrões para emprego em aplicações em tempo real. Usando edição e condensação em conjunto, a melhora de reconhecimento induzido pela edição pode ser combinada com a maior redução promovida por ferramentas de condensação para produzir um conjunto que é significantemente menor do que o original com uma capacidade de reconhecimento similar ou melhor. Normalmente, edição é aplicada seguida por condensação, a qual é a ordem natural baseado nos objetivos de limpar e reduzir (Dasarathy, Sánchez, & Townsend 2000).

Apesar de ser observado que as regras de edição do vizinho-mais-próximo são assintoticamente ótimas, elas podem levar a resultados ruins de classificação se o número de protótipos não for grande o bastante comparado à dimensionalidade intrínseca do espaço de característica. Isto torna edição um problema mais crítico do que condensação.

A justificativa para o método do conjunto condensado é que se um ponto é classificado errado pela regra do vizinho-mais-próximo, ele provavelmente está perto da fronteira de decisão, assim ele deve fazer parte do conjunto condensado (Toussaint 1994). Entretanto, a regra do vizinho mais-próximo condensado guarda muitos pontos, que não estão perto da fronteira de decisão por causa de seu passo de inicialização arbitrário.

As técnicas de condensação (Toussaint 1994) também apresentam algumas desvantagens as quais são:

- Em geral, estas técnicas são seqüenciais em natureza e o subconjunto condensado resultante é uma função da ordem na qual o conjunto de treinamento é processado;
- O conjunto condensado determina apenas aproximadamente a fronteira de decisão original determinada pelo conjunto de treinamento;
- Geralmente são utilizadas heurísticas que complicam os algoritmos e dificultam a análise;
- Os métodos geralmente resultam em um conjunto de treinamento consistente (um conjunto condensado que classifica corretamente todos objetos do conjunto de treinamento), em vez de produzir um conjunto condensado consistente em fronteira de decisão (um conjunto condensado que define precisamente a mesma fronteira de decisão do conjunto original de treinamento).

Nas seções que se seguem, são apresentados os métodos de abstração e filtragem de amostras encontrados na literatura.

5.1.1 Abstração de Instâncias

Uma abordagem para encontrar instâncias representativas é o método de abstração das instâncias o qual gera protótipos por abstração das instâncias originais. Protótipos artificiais gerados por técnicas de abstração de instâncias podem ser mais representativos do que protótipos obtidos por técnicas de filtragem. Técnicas de abstração de instâncias usualmente removem pontos da borda de tal forma que fronteiras de decisão mais suaves podem ser obtidas. Técnicas de abstração tipicamente possuem uma maior capacidade de generalização e maior taxa de redução de dados do que técnicas de filtragem (Lam, Keung, & Liu 2002).

Através de abstração de instâncias, as características mais comuns de um conceito podem ser aprendidas e o conceito aprendido tem menos risco para *overfit* do que utilizando os dados originais. Métodos de abstração podem generalizar instâncias com classificação perdida em regiões compactas combinando-as com outras instâncias, obtendo melhor tolerância a ruídos do que apenas selecionando-as como protótipos.

Entretanto, instâncias artificiais geradas por abstração podem não ser possíveis de serem amostradas realmente. Assim, protótipos gerados são menos compreensíveis do que aqueles selecionados por métodos de filtragem. Por exemplo, não é fácil encontrar uma interpretação para valores médios de características discretas. Além disto, abstração simples também falha para descrever conceitos formados por fronteiras de decisão côncavas. Por estes motivos, esta técnica não foi a escolhida para utilização neste trabalho de tese e, por isto, não será detalhada.

5.1.2 Filtragem de Instâncias

Métodos de filtragem são usualmente simples e rápidos. Apesar de sua simplicidade, alguns métodos de filtragem podem obter precisão comparável ou superior ao classificador vizinho-mais-próximo puro. Como as técnicas de filtragem selecionam instâncias representativas do conjunto original, o conceito aprendido é representado de maneira simples comparado às instâncias originais. Então o conceito (ou idéia) aprendido é facilmente compreendido. Desde que regras de filtragem podem ser projetadas para filtrar instâncias diferentes tais como borda (ou margem), pontos centrais e ruído, então podem ser facilmente aplicadas diferentes regras simultâneas ou separadas para filtrar instâncias (Lam, Keung, & Liu 2002).

Entretanto, métodos de filtragem de instâncias têm algumas desvantagens. Eles assumem que exemplos ideais podem ser encontrados no conjunto original dos dados, o que limita a potência dos métodos de filtragem. A potência

de generalização é limitada pela seleção apenas dos dados originais, e alguns métodos também são sensíveis à ordem de apresentação de instâncias.

O método de filtragem pela retenção de instâncias fora da margem descarta instâncias classificadas erradas pela regra do k vizinho-mais-próximo. Como ruído é raramente classificado corretamente pelos seus vizinhos, também ele é usualmente removido. Este método também remove instâncias da margem pois elas usualmente possuem vizinhos de diferentes classes, resultando em limites de classes mais suaves. Ele mantém principalmente instâncias intermediárias e centrais. Geralmente, a taxa de redução de dados é baixa comparada com abstração de instâncias.

O método de filtragem pela seleção do subconjunto consistente possui uma taxa de redução de amostras mais elevado, mas a generalização resultante é menor quando comparada ao método anterior. Subconjunto consistente de um conjunto de amostras é um subconjunto o qual, quando usado como um conjunto de referência armazenado para a regra do vizinho-mais-próximo, classifica corretamente todos os pontos restantes do conjunto de amostras (Hart 1968). Um subconjunto consistente mínimo é um subconjunto consistente com o número mínimo de elementos.

A seguir são apresentados alguns métodos de filtragem encontrados na literatura.

Subconjunto Condensado

O método CNN (*Condensed Nearest Neighbor*) é um método de pré-processamento dos dados para o reconhecimento de padrões proposto por Hart (Hart 1968), baseado na regra do vizinho-mais-próximo. Seu objetivo é reduzir o tamanho do conjunto de dados original D (conjunto das amostras de classificação conhecida) pela eliminação de certas amostras sem afetar significativamente o desempenho da classificação pelo vizinho-mais-próximo. A regra do vizinho-mais-próximo utilizada com os dados condensados E (sendo $E \subset D$) pode resultar em quase o mesmo resultado que a regra do vizinho-mais-próximo com D.

A regra CNN possui as seguintes propriedades:

- Ela obtém um conjunto E (subconjunto consistente) o qual é um subconjunto do conjunto original, porém bem menor e que requer menor custo computacional de armazenamento e processamento;
- O conjunto E classifica (pela regra do vizinho-mais-próximo) todas as amostras em D corretamente.

O algoritmo a seguir é o proposto por Hart para obtenção do subconjunto consistente pelo método do vizinho mais-próximo-condensado (CNN):

1. A primeira amostra é armazenada em um "buffer" **B1**.

2. A segunda amostra é classificada pela regra NN, usando como conjunto de referência o conjunto de **B1**. Se a segunda amostra é classificada corretamente ela é colocada em um "buffer" **B2**, caso contrário ela é colocada em **B1**.

3. Procedendo indutivamente, a i-ésima amostra é classificada pelo corrente conteúdo de **B1**. Se classificada corretamente é colocada em **B2**, caso contrário é colocada em **B1**.

4. Depois de uma passagem através do conjunto original de amostras, o procedimento repete o laço até terminar, o que pode acontecer de dois modos:

   (a) O **B2** é esgotado, com todos seus membros agora transferidos para **B1** (neste caso, o subconjunto consistente encontrado é o conjunto original de entrada); ou

   (b) Um passo completo é feito por **B2** com nenhuma transferência para **B1**. (Se isto acontece, todos subseqüentes passos através de **B2** resultarão em nenhuma transferência.)

5. O conteúdo final de **B1** é usado como ponto de referência para a regra NN; o conteúdo de **B2** é descartado.

O mesmo algoritmo pode ser declarado de outra forma mais simplificada:

1. A primeira amostra do padrão é copiada do conjunto de treinamento original para o subconjunto reduzido (inicialmente vazio).

2. O subconjunto reduzido é usado para classificar cada padrão de conjunto de treinamento, iniciando com o primeiro. Isto é feito até acontecer um dos seguintes casos:

   (a) Todo padrão do conjunto de treinamento é classificado corretamente, neste caso o processo termina.

   (b) Um dos padrões do conjunto de treinamento é classificado incorretamente, neste caso vá para 3.

3. Some o padrão do conjunto de treinamento que foi incorretamente classificado para o subconjunto reduzido. Vá para 2.

Se as classes possuem pequena sobreposição, então o algoritmo tenderá a separar os pontos perto da margem entre as classes. Pontos profundamente embutidos dentro da classe serão descartados. Se as classes possuem alta sobreposição, então o algoritmo manterá todos os pontos do conjunto original de amostras, e não será realizada redução importante no tamanho das amostras. O subconjunto consistente obtido pode ser utilizado para classificação de padrões desconhecidos pela regra do vizinho-mais-próximo.

Uma desvantagem do CNN é que ele processa amostras de D aleatoriamente (Tomek 1976c), ou seja, move as amostras de D para E aleatoriamente no início e menos mais tarde (quando ele tende a tomar amostras perto da fronteira de classificação). Isto significa que E contém:

- amostras interiores as quais poderiam ser eliminadas, o que implica que E é maior que o necessário;

- amostras que definem uma fronteira em E mas não em D (isto é, amostras não essenciais em D tornam-se pontos da fronteira em E). Isto causa um indesejável deslocamento entre fronteiras.

O método ideal de redução de D usaria pontos perto da fronteira de decisão para gerar E, mas a fronteira de decisão verdadeira é desconhecida. Tomek (Tomek 1976c) propõe duas modificações para melhorar o CNN, porém uma de suas propostas é invalidada por Toussaint (Toussaint 1994) além de corrigir um erro na descrição do algoritmo CNN.

Redução do Subconjunto Consistente

O vizinho-mais-próximo condensado (CNN) não é uma regra de decisão nova uma vez que ainda escolhe a classe pelo vizinho-mais-próximo. A palavra condensado refere-se ao procedimento para escolha de um subconjunto do conjunto de treinamento, o qual pode classificar padrões desconhecidos tão bem quanto o conjunto de treinamento original. Assim, uma menor quantidade de memória é necessária para armazenar o conjunto de treinamento e menor tempo de processamento requerido para chegar a uma decisão.

Um subconjunto consistente de um conjunto de treinamento é um subconjunto que classifica todos os padrões de treinamento corretamente usando a Regra do Vizinho-mais-próximo. O subconjunto consistente mínimo é o menor e mais eficiente subconjunto de treinamento que apropriadamente classifica todos padrões de treinamento. O CNN é consistente, porém não é garantido que seja mínimo. É provado por simulação que CNN não é mínimo, e o subconjunto consistente mínimo é difícil de se obter.

O método do vizinho-mais-próximo reduzido (RNN) é uma extensão da regra CNN (Gates 1972) na tentativa de se obter o subconjunto consistente mínimo. O algoritmo RNN é mostrado a seguir:

1. Copie todo o subconjunto consistente para o subconjunto reduzido.
2. Remova o primeiro padrão do subconjunto reduzido.
3. Use o subconjunto reduzido para classificar os padrões de treinamento:
   (a) Se todos os padrões são classificados corretamente, vá para 4.
   (b) Se um padrão é classificado incorretamente, retorne o padrão que foi removido do subconjunto reduzido em 2 e vá para 4.
4. Se todos padrões do subconjunto reduzido foram removidos uma vez (e possivelmente recolocado) então pare. Caso contrário, remova o próximo padrão e vá para 3.

Gowda (Gowda & Krishna 1979) propôs um método de dois estágios para determinar um subconjunto consistente baseado no CNN modificado. O primeiro estágio é semelhante ao método de obtenção do subconjunto condensado. O segundo estágio é semelhante ao método de redução do subconjunto consistente. O algoritmo é descrito a seguir:

Estágio 1:

1. Para cada amostra x do conjunto de treinamento, encontre o vizinho-mais-próximo y de classe oposta utilizando uma determinada métrica.
2. Ordene as amostras de 1 em ordem crescente da distância e armazene em um conjunto ORDEM.
3. Coloque a primeira amostra de ORDEM em DEPÓSITO.
4. A próxima amostra de ORDEM é classificada pela regra do vizinho-mais-próximo usando as amostras que estão presentes em DEPÓSITO. Se a classificação está errada, some a amostra a DEPÓSITO.
5. Repita o passo 4 até que todas as amostras em ORDEM sejam testadas.
6. Após um passo através de ORDEM, aplique passos 4 e 5 para as amostras retidas em ORDEM. Repita este procedimento até que não haja transferências de amostras de ORDEM para DEPÓSITO em um passo. O conteúdo de DEPÓSITO constitui o conjunto de treinamento condensado modificado.

Estágio 2:

1. Uma amostra de DEPÓSITO é transferida para um conjunto EXAME.

2. Todas as amostras em ORDEM são classificadas pela regra do vizinho-mais-próximo usando as amostras que estão em DEPÓSITO. Se há qualquer erro de classificação, transfira a amostra do passo 1 de volta para DEPÓSITO, caso contrário retenha-a em EXAME.

3. Repita passos 1 e 2 para todas amostras em DEPÓSITO.

Subconjunto Editado

Wilson (Wilson 1972) propôs um método de redução do conjunto de amostras de treinamento para classificação usando a regra do vizinho-mais-próximo. O método basicamente faz a edição das amostras pré-classificadas usando a regra do k vizinho-mais-próximo com $k = 3$, seguido pela classificação usando a regra-do-vizinho mais próximo simples ($k = 1$) com as amostras restantes pré-classificadas. Isto produz um procedimento de decisão o qual o risco se assemelha ao risco de Bayes em muitos problemas com apenas umas poucas amostras pré-classificadas.

O algoritmo proposto por Wilson para a regra do k vizinho-mais-próximo editado é mostrado a seguir:

1. Para toda amostra i pertencente ao conjunto de treinamento

$$\{(x_1, w_1), \ldots, (x_N, w_N)\}$$

   (a) Encontre os k ($k = 3$) vizinhos mais próximos a x_i entre:

$$\{x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_N\}$$

   (b) Encontre a classe associada com o maior número de pontos dentre os k vizinhos mais próximos.

2. Edite o subconjunto (x, w) deletando (x_i, w_i) quando w_i não concorda com o maior número dos k vizinhos mais próximos como determinado anteriormente.

3. Tome uma decisão de classificação sobre uma amostra desconhecida usando o subconjunto editado (conjunto reduzido de amostras pré-classificadas) como referência para a regra do vizinho-mais-próximo simples $(k = 1)$.

O algoritmo resulta em um conjunto de amostras selecionadas do conjunto original, as quais estão fora da margem de separação das classes. Assim, são eliminadas tanto as amostras ruidosas quanto as amostras de classes diferentes sobrepostas na margem o que torna o treinamento do correspondente classificador mais fácil na prática (Ferri, Albert, & Vidal 1999). Entretanto, é impossível remover protótipos ruins sem remover também alguns protótipos bons. Algoritmos de edição constituem diferentes tipos de compromisso entre remover tantos protótipos ruins e ainda manter alguma pequena sobreposição entre as classes.

É possível desenvolver o desempenho assintótico da regra do vizinho-mais-próximo usando edição. O desempenho assintótico é o desempenho quando o número de amostras pré-classificadas é muito grande. A regra k vizinho-mais-próximo editado possui um desempenho assintótico (Wagner 1973) melhor do que a regra do vizinho-mais-próximo pura, e ainda é difícil diferenciar do desempenho da Regra de Bayes em muitas situações. O risco de se usar a regra do vizinho-mais-próximo decrementa na medida em que o número de vizinhos aumenta. O risco de se usar a regra do k vizinho-mais-próximo editado está geralmente entre o risco de usar a regra do vizinho-mais-próximo com o mesmo número k de vizinhos e o risco de Bayes.

Apesar do bom desempenho em classificação, este método resulta em uma taxa baixa de redução de amostras. Assim, métodos derivados da técnica de Wilson tem aparecido na literatura com o objetivo de melhorar seu desempenho (Ferri, Albert, & Vidal 1999; Kim & Oommen 2004). O método de edição pode também ser utilizado com iterações sucessivas para melhorar o desempenho.

Subconjunto seletivo

Ritter (Ritter, Woodruff, Lowry, & Isenhour 1975) propôs um método de pré-processamento dos dados de treinamento (amostras conhecidas) para se encontrar um subconjunto de amostras de forma que o risco de classificação permanece o mesmo. Três critérios servem de base para o método:

- O subconjunto deve ser consistente;
- Todas amostras devem estar mais perto a um vizinho seletivo de mesma classe do que qualquer amostra de outra classe inclusive do subconjunto seletivo (conjunto original);
- Não deve haver nenhum subconjunto que satisfaça os 2 critérios anteriores e que contenha menos membros do que o subconjunto seletivo.

A taxa de erro do subconjunto seletivo está entre a taxa de erro do vizinho-mais-próximo e do vizinho-mais-próximo condensado. Assim, a fronteira de decisão do vizinho-mais-próximo está mais precisamente reproduzida pelo subconjunto seletivo. Quando as amostras estão mais juntas, o critério do vizinho-mais-próximo seletivo parece favorecer as amostras perto da superfície de decisão.

5.1.3 Tipicidade de instâncias

Tipicidade é utilizada para indicar como uma instância é tipificada com respeito à fronteira de decisão (Lam, Keung, & Liu 2002). A tipicidade de uma instância pode ser definida como:

$$T(i) = \frac{\text{média de similaridade de } i \text{ a instâncias de mesma classe}}{\text{média de similaridade de } i \text{ a instâncias de outras classes}}$$

A similaridade de duas instâncias x e y, $Sim(x, y)$, é definida como:

$$Sim(x, y) = 1 - Dist(x, y)$$

onde $Dist(x, y)$ é a medida de distância, onde todas as características contínuas são normalizadas por suas faixas características.

A tipicidade de uma instância possui as seguintes características:

- Instâncias típicas a uma classe específica usualmente possuem tipicidade muito maior do que 1;
- Instâncias fronteiras usualmente possuem tipicidade perto de 1;
- Instâncias ruído possuem tipicidade menor que 1.

Tipicidade é um bom indicador para localizar instâncias na margem e fora da margem. Para instâncias ordinais com tipicidade maior do que 1, há uma tendência de que a tipicidade seja linearmente proporcional à distância da fronteira de decisão. Instâncias ruídos possuem tipicidade menor do que 1, e são inversamente proporcionais à distância da fronteira. Uma instância é dita estar na margem se sua tipicidade é menor do que a tipicidade média de sua classe subtraído do desvio padrão.

5.2 Método Proposto de Seleção de Amostras

O método de projeto do classificador de dados binários proposto neste trabalho de tese é baseado na filtragem dos dados binários de entrada do minimizador de função Booleana. Este pré-processamento envolve a escolha das

amostras de dados binários que melhor representam cada classe, espacialmente localizadas fora da margem de separação das classes, porém perto da margem (Lacerda & Braga 2004).

Partindo da suposição de que, dentro da margem de separação das classes, há uma mistura espacial dos dados de classes diferentes, estes dados são descartados juntamente com os dados que estão bem no interior das classes. Os dados selecionados (restantes) são suficientes para definir a fronteira de separação entre as classes, com uma superfície de decisão suave o bastante para garantir a generalização adequada. Além disto, a quantidade menor de dados resulta em uma diminuição no custo de armazenamento e de processamento.

A seguir, é explicado o procedimento de seleção de amostras proposto neste trabalho de tese. Serão utilizados exemplos com dados reais de duas dimensões para facilitar a compreensão do algoritmo. A métrica utilizada nestes exemplos é a Euclidiana. Na seção seguinte (5.3), são apresentados os resultados obtidos com o método proposto na solução de alguns problemas de classificação com dados binários. Diferentes métricas são testadas com o algoritmo.

5.2.1 Etapas do Método Proposto

Suponha um conjunto X de n amostras reais x_i ($x_i \in \Re^2$, $X = \{x_0, x_1, \cdots, x_{n-1}\}$, $i = \{0, 1, \cdots, n-1\}$) de dimensão $d = 2$ ($x_i \in \Re^2, x_i = \{x_{i0}, x_{i1}\}$) onde cada amostra x_i é associada a uma das possíveis classes. Para efeito de simplificação serão consideradas apenas duas classes (0 ou 1). A Figura 5.1 mostra um exemplo deste conjunto de dados classificados em "círculo" ou "cruz". A função de um classificador é associar uma amostra y desconhecida (sem classe) à sua classe mais provável baseando-se no conjunto X de amostras conhecidas.

A filtragem tem por objetivo não só a redução da quantidade de amostras de treinamento, mas também à melhoria na capacidade de generalização do classificador com projeto baseado nas amostras selecionadas. Para o processo de seleção de amostras a serem utilizadas no projeto do classificador, é proposto um algoritmo de três etapas:

1. Seleção das amostras que estão mais próximas da fronteira de separação das classes (amostras da margem) utilizando um método semelhante ao proposto por Gowda (Gowda & Krishna 1979). Este subconjunto é **consistente**, ou seja, quando utilizado como referência à regra do vizinho-mais-próximo, classifica corretamente todas as amostras de treinamento.

2. Redução do subconjunto consistente, utilizando a técnica proposta por Gates (Gates 1972) onde para cada amostra do subconjunto consistente

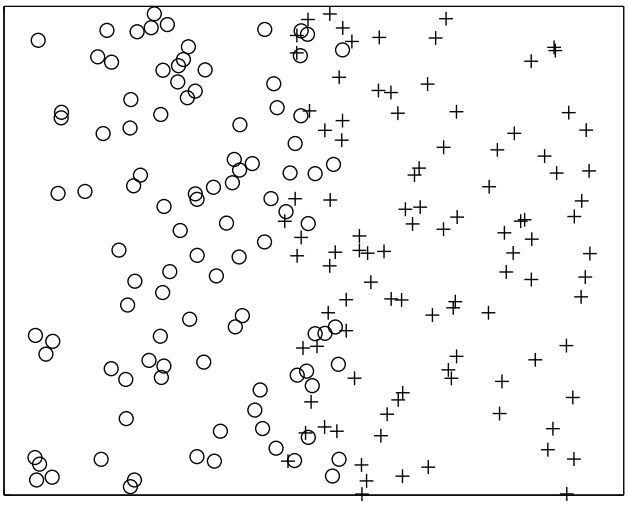


Figura 5.1: Exemplo de conjunto de dados em $\Re^2$ de duas classes: "círculo" ou "cruz".

é testada sua necessidade para manter a consistência do subconjunto. O subconjunto **restante** é automaticamente determinado, pois compreende as amostras de treinamento originais sem as amostras do subconjunto consistente reduzido.

3. Redução das amostras do subconjunto restante pela seleção das amostras que estão mais próximas do subconjunto consistente reduzido. A quantidade de amostras selecionadas é determinada pela quantidade (k) de vizinhos mais próximos escolhidos para cada amostra do subconjunto consistente reduzido (Lacerda & Braga 2004).

A seguir é detalhada cada etapa do algoritmo proposto, denominado RSR (**Redução do Subconjunto Restante**).

5.2.2 Subconjunto Consistente

A obtenção do subconjunto consistente parte da suposição de que as amostras de interesse estão perto da fronteira de decisão da classe. Assim, na região do espaço das amostras em que há uma mistura de classes, ou que pelo menos amostras de classes distintas estão mais próximas segundo uma métrica de distância, são selecionadas gradativamente pares de amostras de classes distintas para fazerem parte do subconjunto consistente, em ordem de distância entre as amostras de classes distintas que fazem parte do par. Quando as amostras selecionadas são capazes de classificar corretamente as demais amostras pela regra do vizinho mais próximo, o processo de seleção termina. Isto é descrito formalmente pelo algoritmo a seguir:

1. Para cada amostra x_i do conjunto de treinamento X, encontre o vizinho-mais-próximo x_j de classe oposta utilizando uma determinada métrica

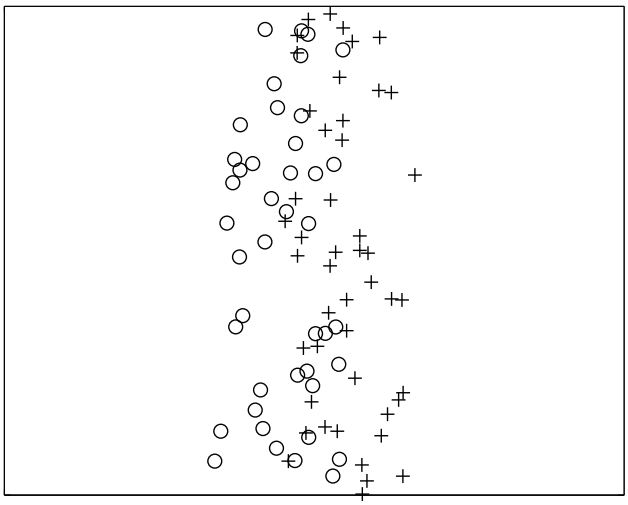


Figura 5.2: Subconjunto consistente.

(calculando a distância entre duas amostras), formando um par (x_i, x_j). Poderão haver pares diferentes com amostras repetidas.

2. Ordene os pares de amostras do passo anterior em ordem crescente da distância entre si e armazene-os em um vetor ORDEM.

3. Retire do conjunto de treinamento as amostras constantes no primeiro par de amostras (de distância menor entre si) de ORDEM e armazene no subconjunto consistente (inicialmente vazio). Pode ser que alguma amostra já tenha sido transferida.

4. Classifique cada amostra do conjunto de treinamento pela regra do vizinho-mais-próximo simples utilizando como referência o subconjunto consistente. Se houver algum erro, retorne a 3 considerando o próximo par de amostras de menor distância entre si. Caso contrário, finalize.

O subconjunto consistente final obtido pelo procedimento acima classifica corretamente todos os dados de treinamento pela regra do vizinho-mais-próximo. A Figura 5.2 mostra o resultado do algoritmo acima utilizando os dados da Figura 5.1 como entrada.

5.2.3 Redução do Subconjunto Consistente

Não é garantido que o subconjunto consistente obtido pelo procedimento anterior seja mínimo, portanto pode ser reduzido utilizando alguma técnica, como o procedimento descrito a seguir:

1. Copie o subconjunto consistente para o subconjunto consistente reduzido (inicialmente vazio).

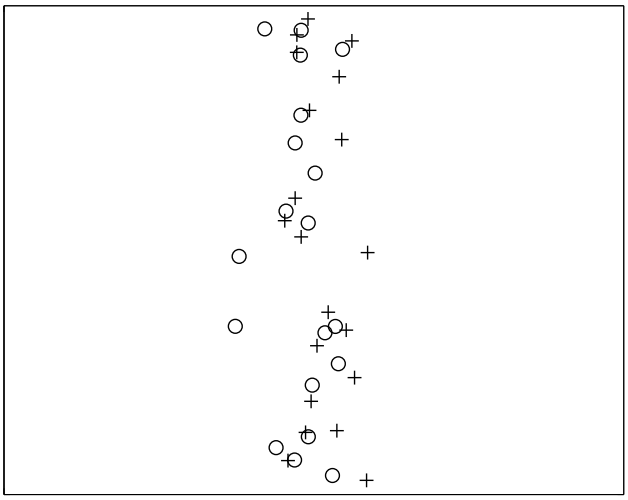


Figura 5.3: Subconjunto consistente reduzido.

2. Remova a última amostra do subconjunto consistente reduzido ordenado em ordem crescente das distâncias entre os pares mais próximos de amostras de classes diferentes.

3. Utilize o subconjunto consistente reduzido como referência para classificar as amostras do conjunto de treinamento pela regra do vizinho-mais-próximo simples:

   (a) Se todas as amostras são classificadas corretamente, vá para 4.

   (b) Se uma amostra é classificada incorretamente, retorne a amostra que foi removida do subconjunto consistente reduzido em 2 e vá para 4.

4. Se todas as amostras do subconjunto consistente reduzido foram removidas uma vez (e possivelmente recolocada) então pare. Caso contrário, remova a próxima amostra do subconjunto consistente reduzido (como em 2) e vá para 3.

Não é garantido também que o resultado deste procedimento seja um subconjunto consistente mínimo, mas será próximo do mínimo. A Figura 5.3 apresenta o resultado do algoritmo acima quando utiliza os dados das Figuras 5.1 e 5.2 como entrada. O conjunto de treinamento original menos o subconjunto consistente reduzido forma o subconjunto restante ou subconjunto editado (Wilson 1972), que é utilizado na próxima etapa. A Figura 5.4 mostra o subconjunto restante obtido com o conjunto de dados anterior.

5.2.4 Redução do Subconjunto Restante

As amostras obtidas pelo procedimento anterior já garantem uma boa generalização para o classificador, por causa da filtragem das amostras que se

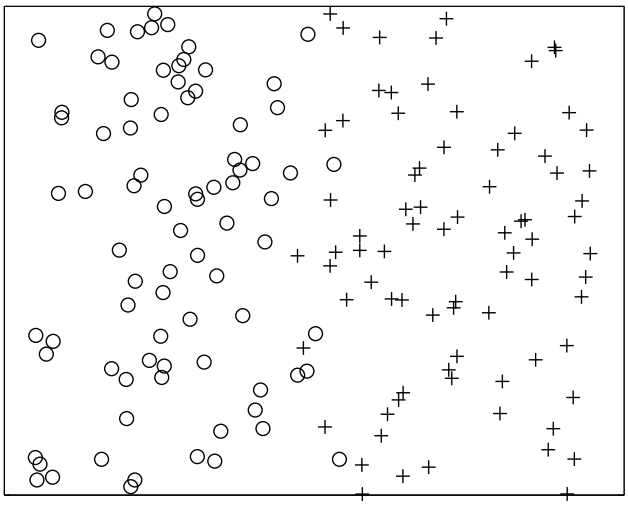


Figura 5.4: Subconjunto restante.

encontram perto da fronteira de classes, tornando a superfície de decisão mais suave. Entretanto, nem todas as amostras presentes no subconjunto restante obtido na etapa anterior são necessárias para manter o mesmo desempenho de classificação quando utilizando o subconjunto restante total. Assim, é desenvolvido um procedimento para redução do subconjunto restante, utilizando a regra do vizinho-mais-próximo. O procedimento é descrito como:

- Obtenha, do subconjunto restante, os k vizinhos mais próximos de cada classe para cada amostra do subconjunto consistente reduzido, e armazene em um subconjunto restante reduzido (inicialmente vazio). Pode ser que alguma amostra do subconjunto restante já esteja no subconjunto restante reduzido, assim ignore-a.

O valor de k pode ser escolhido arbitrariamente ou pode ser ajustado iterativamente pelo algoritmo baseado no desempenho do classificador obtido. A dificuldade está justamente em escolher o valor de k mais apropriado de forma a garantir o melhor desempenho para o classificador. A Figura 5.5 apresenta o resultado final do algoritmo para o subconjunto restante reduzido sendo 3 o valor escolhido para k.

Ao final do procedimento acima (com um valor apropriado para k), o subconjunto restante reduzido será capaz de garantir ao classificador o mesmo desempenho obtido utilizando o subconjunto restante total.

5.3 Resultados do Método Proposto de Seleção das Amostras

Com o objetivo de validar o algoritmo de seleção de amostras proposto (RSR) neste trabalho de tese, foram realizados alguns experimentos utilizando da-

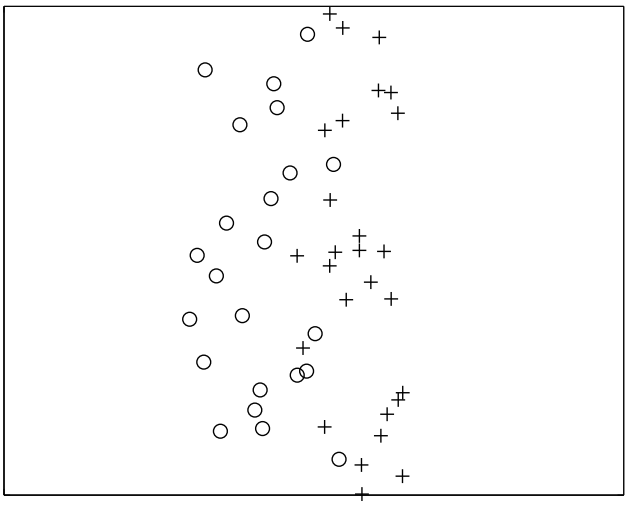


Figura 5.5: Subconjunto restante reduzido ($k = 3$).

dos numéricos e dados binários. Primeiramente, utilizou-se um conjunto de dados de duas dimensões de números reais de duas classes e característica multimodal em forma de tabuleiro de xadrez. Em seguida, foram utilizados os mesmos conjuntos de dados binários apresentados em capítulo anterior.

5.3.1 Tabuleiro de Xadrez

A seguir são apresentados os resultados do algoritmo de seleção de amostras proposto (RSR) utilizando um conjunto de dados de duas dimensões com distribuição multimodal (tabuleiro de xadrez) e duas classes ("círculo" e "cruz") conforme mostrado na Figura 5.6. A métrica utilizada foi a Euclidiana por se tratar de dados numéricos.

Na Figura 5.7 é mostrado o subconjunto consistente obtido pela primeira etapa do algoritmo proposto. Observa-se que as amostras selecionadas nesta etapa estão sobrepostas e perto da fronteira de decisão das classes.

Na Figura 5.8 é mostrado o subconjunto consistente reduzido obtido pela segunda etapa do algoritmo proposto. Percebe-se que a quantidade de amostras reduziu drasticamente em relação ao gráfico anterior.

Na Figura 5.9 é mostrado o subconjunto restante, ou seja, são os dados da Figura 5.6 menos os dados da Figura 5.8. Neste gráfico, os dados de classes diferentes já não estão sobrepostos, entretanto a quantidade de amostras é grande.

Por fim, o algoritmo RSR resulta nos dados mostrados na Figura 5.10, onde a quantidade de amostras é baixa enquanto não há sobreposição de amostras de classes diferentes.

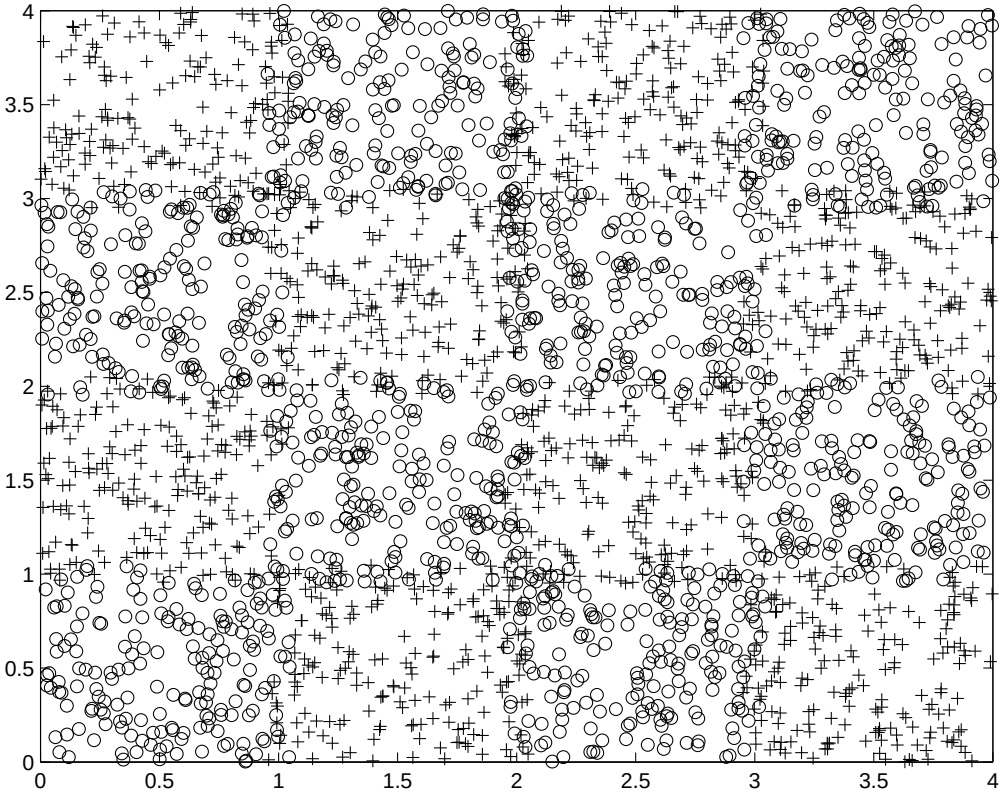


Figura 5.6: Exemplo de conjunto de dados em $\Re^2$ de duas classes ("círculo" e "cruz") e formato multimodal.

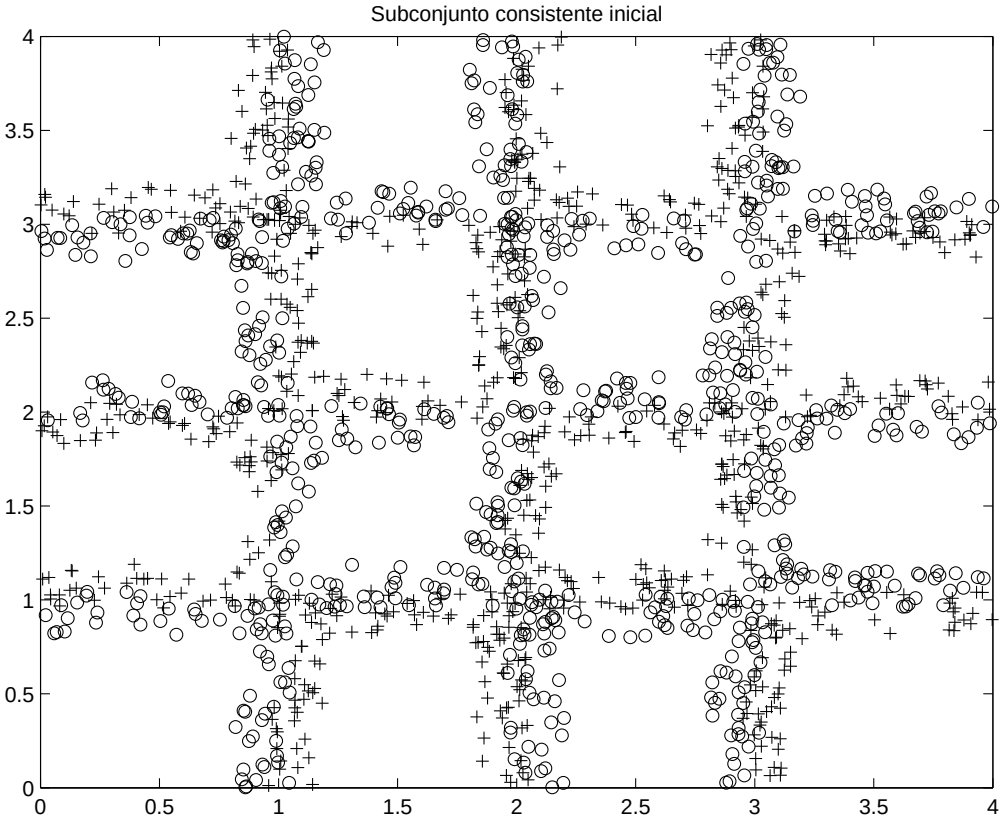


Figura 5.7: Subconjunto consistente obtido pelo algoritmo RSR na solução do problema do tabuleiro de xadrez.

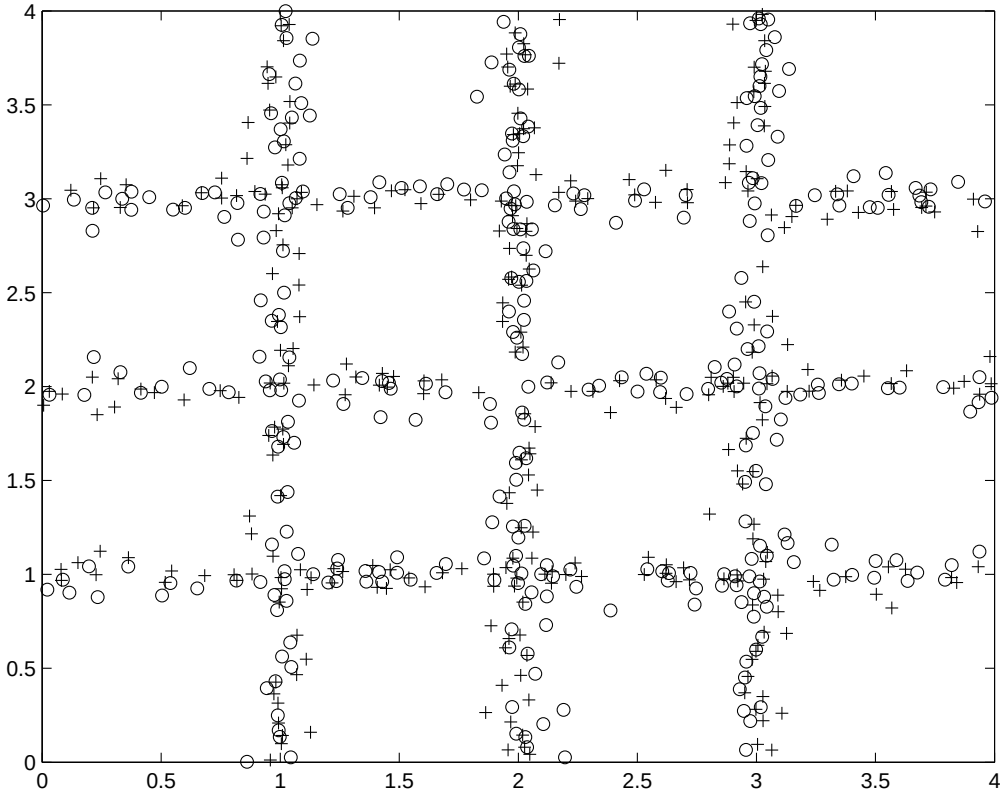


Figura 5.8: Subconjunto consistente reduzido obtido pelo algoritmo RSR na solução do problema do tabuleiro de xadrez.

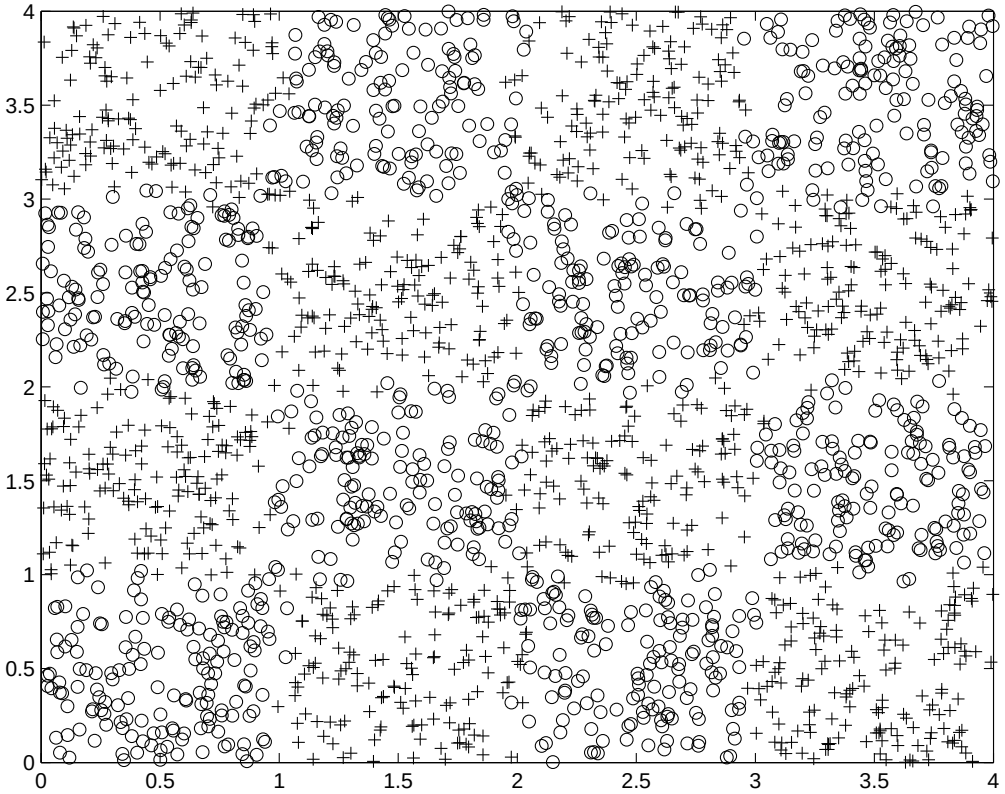


Figura 5.9: Subconjunto restante obtido pelo algoritmo RSR na solução do problema do tabuleiro de xadrez.

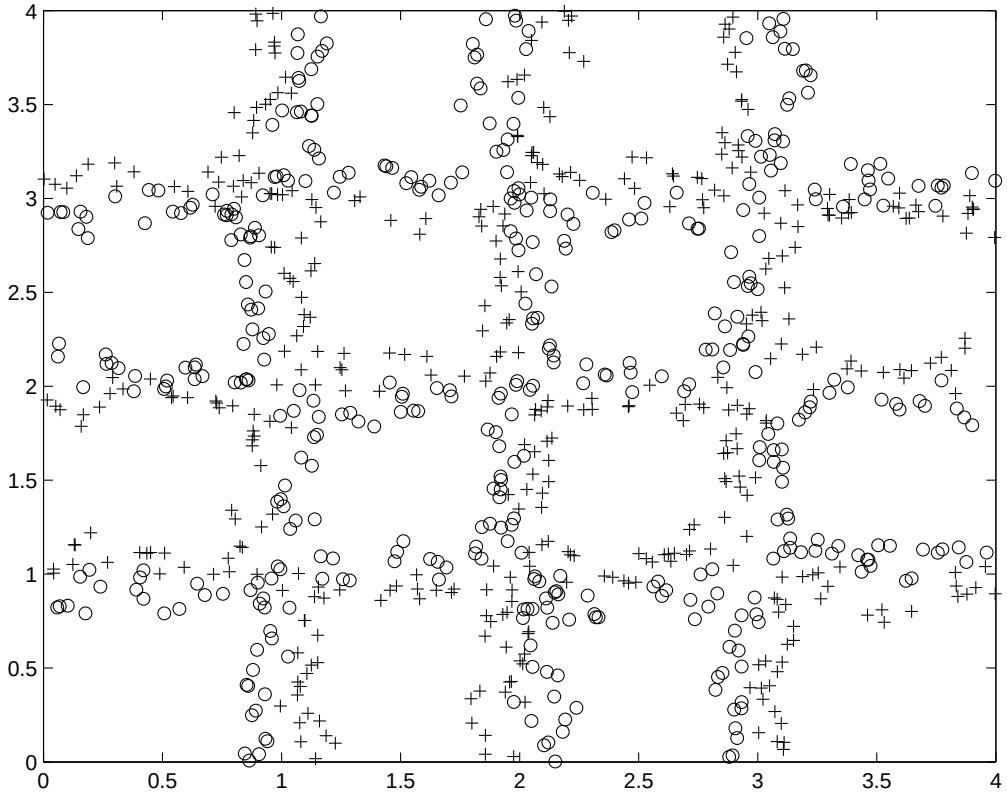


Figura 5.10: Subconjunto restante reduzido obtido pelo algoritmo RSR na solução do problema do tabuleiro de xadrez.

5.3.2 Dados binários

Nesta seção são apresentados os resultados obtidos utilizando os mesmos conjuntos de dados binários do capítulo anterior. Em cada problema foram obtidos 10 conjuntos de dados diferentes de forma a poder obter uma média dos resultados. Os conjuntos de dados também foram particionados da mesma maneira como anteriormente: 80% para treinamento e 20% para teste. Os resultados obtidos utilizando as amostras selecionadas pelo RSR como referência para o classificador 1-NN ($k = 1$) são apresentados em seqüência. Para o cálculo das distâncias entre amostras, utilizou-se algumas métricas já descritas em capítulo anterior para fins de comparação. Antes da execução do algoritmo RSR, é montada a matriz de distâncias entre todas as amostras de treinamento e teste.

Dados sintéticos

Na Figura 5.11 é apresentado o resultado obtido do classificador 1-NN quando utilizando o algoritmo de seleção de amostras RSR em função de k, para os dados binários sintéticos gerados conforme Figura 4.4. O parâmetro k define a quantidade de amostras selecionadas pelo algoritmo RSR. Observa-se que a métrica de melhor desempenho foi a de Bayes. A pior métrica foi a *over-*

lap. A métrica VDM modificada obteve claramente um desempenho ligeiramente superior a métrica VDM original.

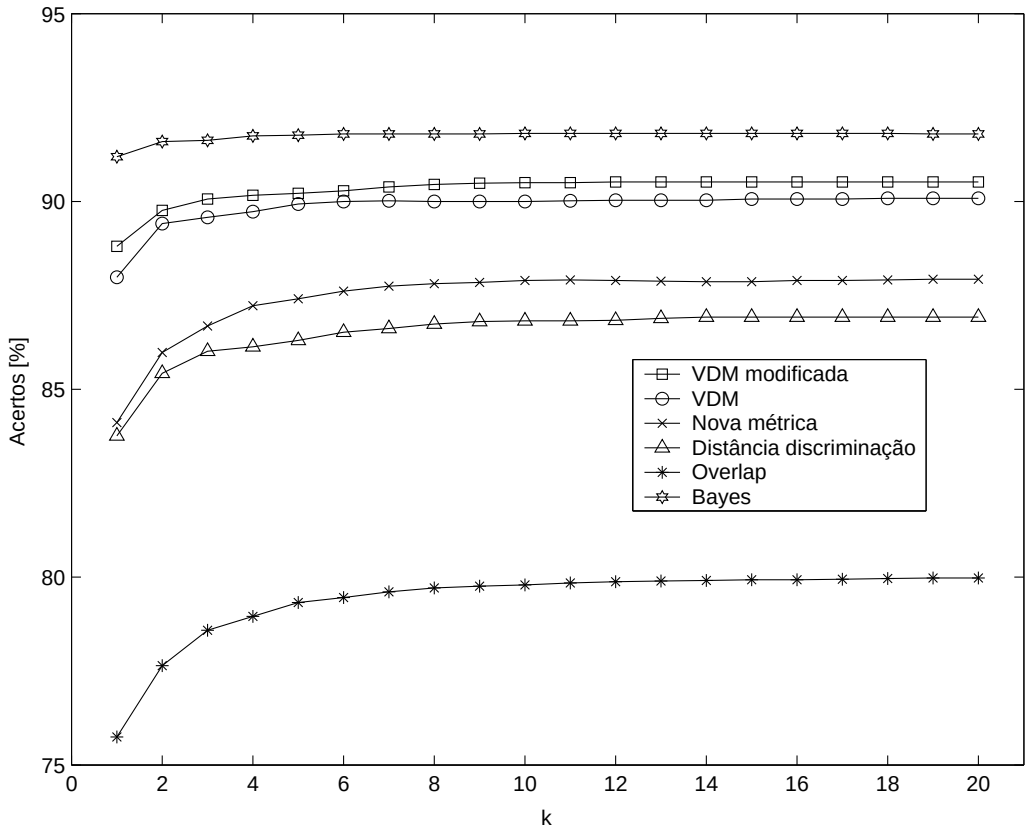


Figura 5.11: Número médio de acertos do classificador 1-NN em função de k, utilizando diferentes métricas na solução do problema de dados sintéticos.

Comparando este gráfico com o resultado obtido do classificador kNN no capítulo anterior para os mesmos dados (Figura 4.5), observa-se que houve uma pequena suavização das curvas de resultados, mas o desempenho geral piorou. Entretanto, utilizando-se a métrica de Bayes, o desempenho melhorou ligeiramente para $k = 1$.

Dados caracteres manuscritos

Na Figura 5.12 é apresentado o resultado obtido do classificador 1-NN quando utilizando o algoritmo de seleção de amostras RSR em função de k, para os dados binários de caracteres manuscritos conforme Figura 4.6. O parâmetro k define a quantidade de amostras selecionadas pelo algoritmo RSR. Observa-se que todas as métricas utilizadas obtiveram desempenho semelhante, excetuando a métrica de Bayes que obteve claramente um resultado inferior às demais.

Comparando este gráfico com o resultado obtido do classificador kNN no capítulo anterior para os mesmos dados (Figura 4.8), observa-se que houve uma suavização das curvas de resultados, mas o desempenho geral piorou.

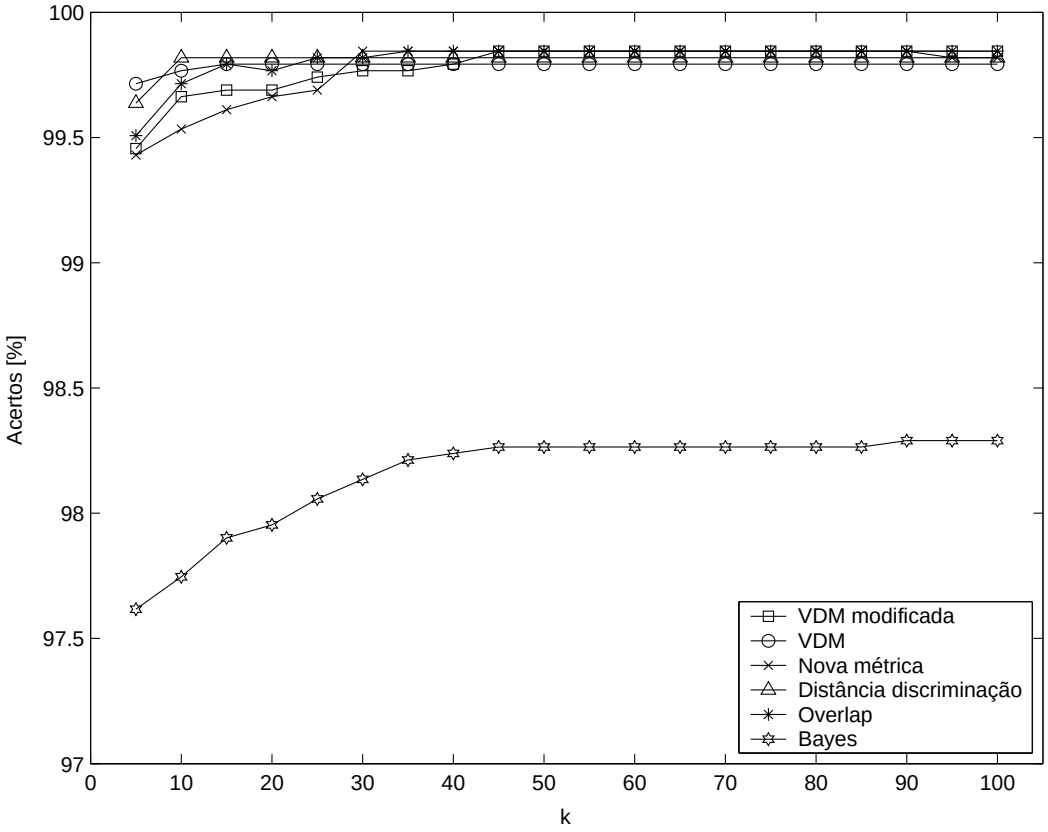


Figura 5.12: Número médio de acertos do classificador 1-NN em função de k, utilizando diferentes métricas na solução do problema dos caracteres manuscritos.

Dados do problema gene

Na Figura 5.13 é apresentado o resultado obtido do classificador 1-NN quando utilizando o algoritmo de seleção de amostras RSR em função de k, para os dados binários do problema gene conforme Figura 4.9. O parâmetro k define a quantidade de amostras selecionadas pelo algoritmo RSR. Observa-se que a métrica VDM obteve o melhor resultado. A métrica VDM modificada obteve um resultado inferior, sendo que a pior métrica foi a *overlap* (abaixo de 75%). A nova métrica proposta obteve um resultado ligeiramente inferior a melhor.

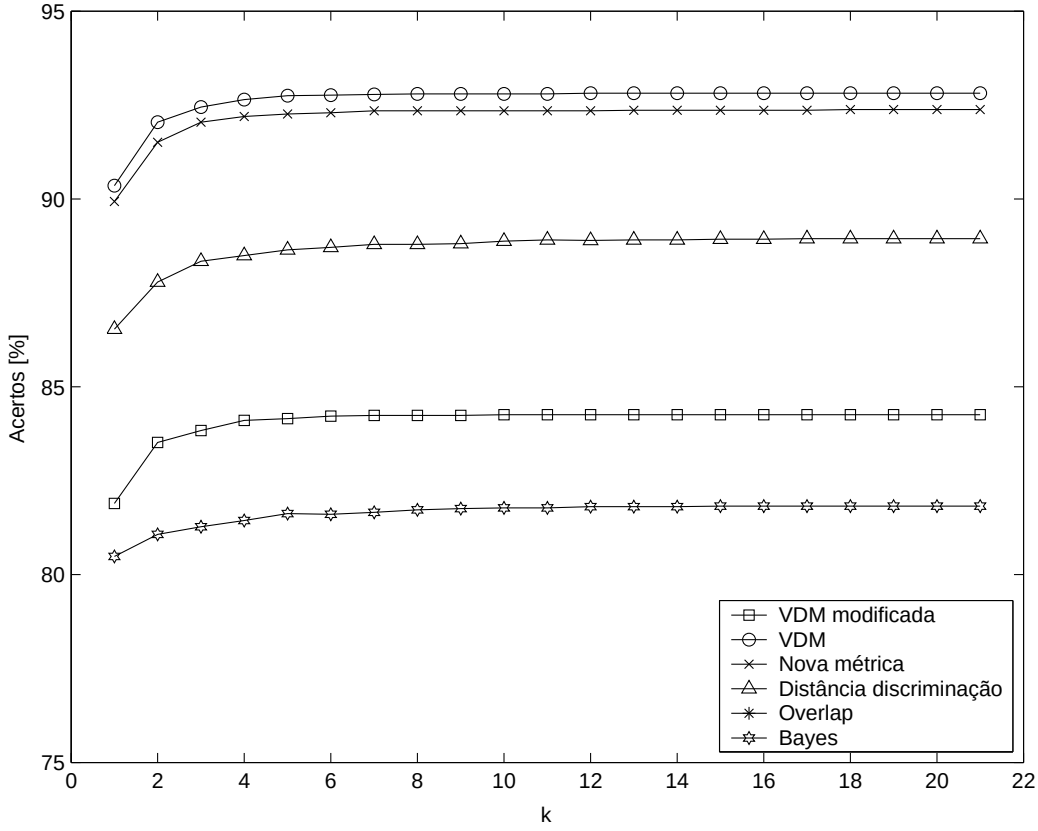


Figura 5.13: Número médio de acertos do classificador 1-NN em função de k, utilizando diferentes métricas na solução do problema de dados de genes.

Comparando este gráfico com o resultado obtido do classificador kNN no capítulo anterior para os mesmos dados (Figura 4.10), observa-se que houve uma suavização das curvas de resultados. Além disto, houve uma inversão de desempenho entre as métricas VDM e a nova métrica proposta. O desempenho geral do classificador melhorou com o algoritmo RSR.

Dados do coração

Na Figura 5.14 é apresentado o resultado obtido do classificador 1-NN quando utilizando o algoritmo de seleção de amostras RSR em função de k,

para os dados binários do problema do coração conforme Figura 4.11. O parâmetro k define a quantidade de amostras selecionadas pelo algoritmo RSR. Observa-se que a métrica de Bayes obteve claramente o melhor resultado. A métrica VDM modificada obteve um resultado claramente melhor que a VDM original. A métrica com o pior desempenho foi a distância de discriminação.

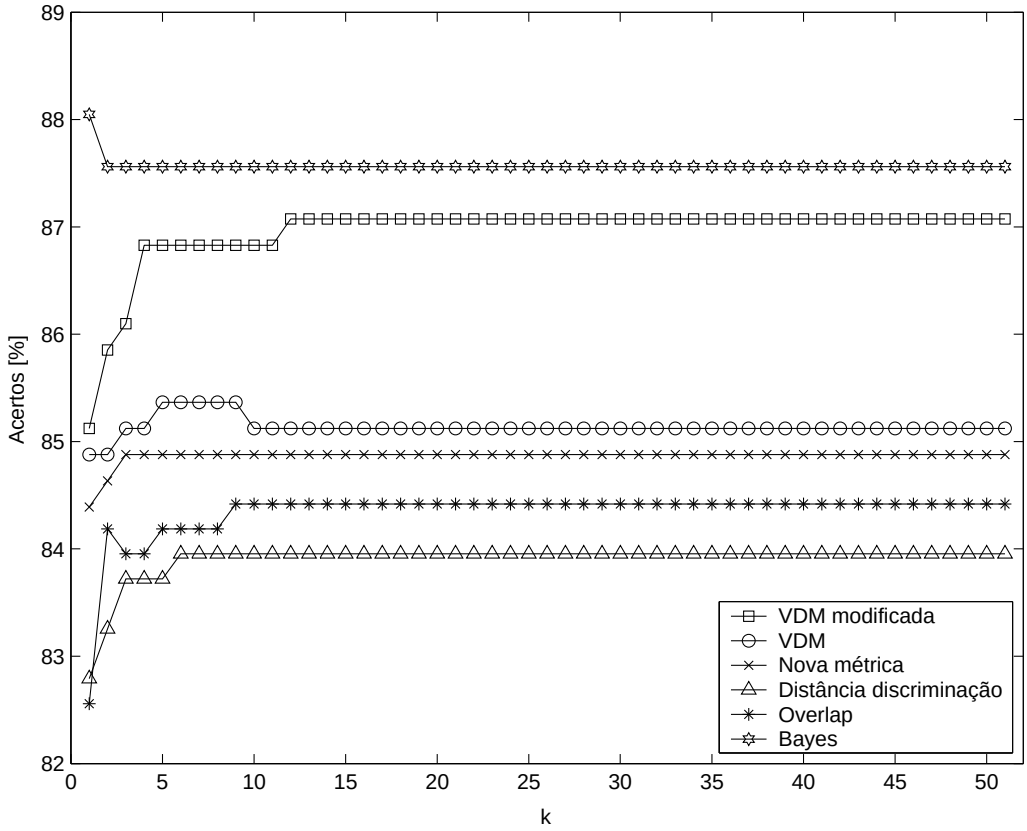


Figura 5.14: Número médio de acertos do classificador 1-NN em função de k, utilizando diferentes métricas na solução do problema de dados do coração.

Comparando este gráfico com o resultado obtido do classificador kNN no capítulo anterior para os mesmos dados (Figura 4.12), observa-se que houve uma clara suavização das curvas de resultados. Além disto, o desempenho geral do classificador piorou ligeiramente com o algoritmo RSR.

Dados do cogumelo

Na Figura 5.15 é apresentado o resultado obtido do classificador 1-NN quando utilizando o algoritmo de seleção de amostras RSR em função de k, para os dados binários do problema do cogumelo conforme Figura 4.13. O parâmetro k define a quantidade de amostras selecionadas pelo algoritmo RSR. Observa-se que as métricas utilizadas obtiveram resultados semelhantes, excetuando a métrica de Bayes que obteve os resultados inferiores.

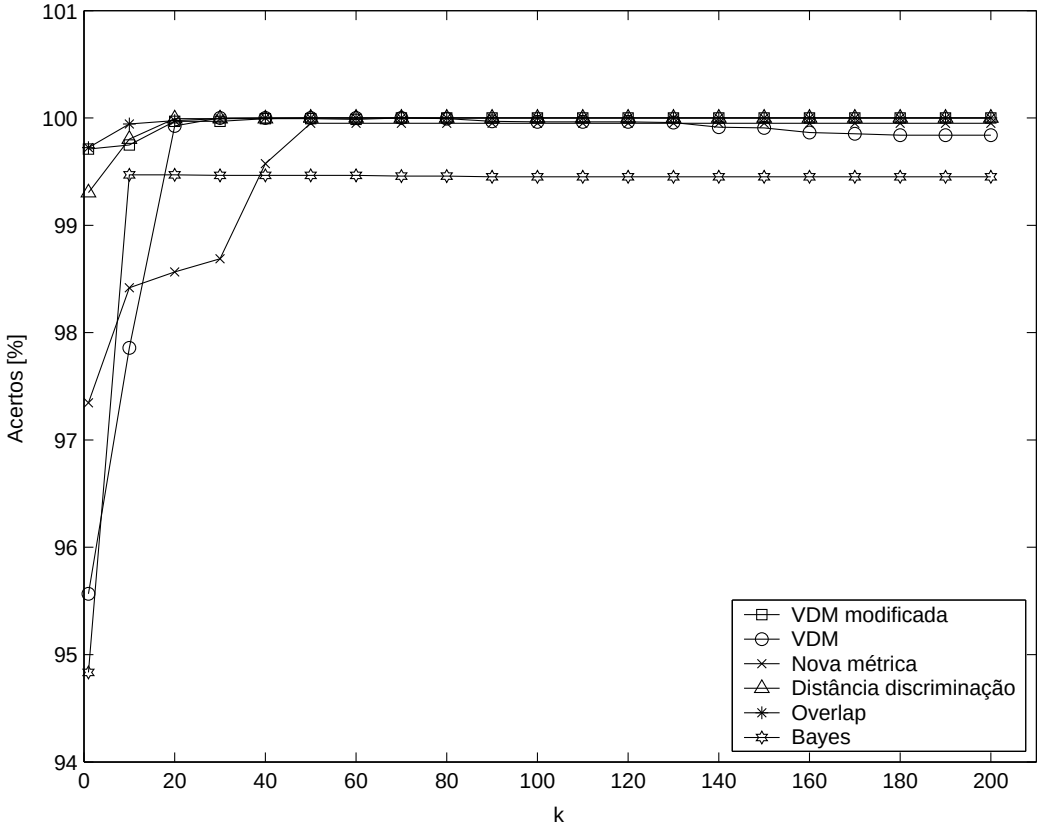


Figura 5.15: Número médio de acertos do classificador 1-NN em função de k, utilizando diferentes métricas na solução do problema de dados do cogumelo.

Comparando este gráfico com o resultado obtido do classificador kNN no capítulo anterior para os mesmos dados (Figura 4.14), observa-se que houve uma suavização das curvas de resultados. Observa-se também que houve uma piora do desempenho geral do classificador com o algoritmo RSR para baixos valores de k ($k < 50$), quando comparado com o kNN para $k = 1$.

5.4 Conclusão

Alguns métodos de seleção das amostras de treinamento que aparecem na literatura foram detalhados neste capítulo. Estas técnicas garantem um custo menor de armazenamento e processamento na aplicação do método de classificação pelo vizinho-mais-próximo. Duas metodologias de seleção de amostras foram apresentadas: abstração e filtragem. Dentre as técnicas de filtragem, o método de edição de amostras proposto por (Wilson 1972) garante uma melhor generalização, entretanto a taxa de redução de amostras é pequena.

As amostras selecionadas pelo método de Wilson (subconjunto editado) são aquelas localizadas fora da margem de separação das classes junto com aquelas bem no interior das classes. Para caracterização das classes, bastaria o subconjunto editado sem as amostras que estão bem no interior das classes, pois estas amostras não contribuem na formação da fronteira de decisão.

Foi apresentado uma proposta de metodologia de seleção de amostras perto da margem de separação das classes, derivado do método proposto por (Wilson 1972), que resulta em uma quantidade menor de amostras. Quanto maior o número (k) de vizinhos mais próximos das amostras do subconjunto consistente reduzido, menor o erro de classificação dos dados de teste, porém sem necessariamente selecionar todas amostras do subconjunto restante para se obter o mesmo desempenho do método de Wilson, conforme comprovado por observações experimentais.

Portanto, é proposto neste trabalho de tese um método de seleção de amostras que melhora a taxa de redução do subconjunto editado sem comprometer significativamente o desempenho do classificador baseado nas amostras selecionadas. Estas amostras selecionadas são utilizadas para o projeto do circuito classificador digital o que resulta em melhor generalização e menor custo computacional para o minimizador de função Booleana. Isto será tratado no capítulo seguinte.

O método de seleção de amostras proposto também pode ser utilizado em conjunto com outras técnicas de classificação com o objetivo de melhorar o desempenho e diminuir o custo computacional (Carvalho, Lacerda, & Braga 2005).

CAPÍTULO 6

Projeto do Circuito Classificador Digital

O método de projeto proposto para geração do circuito classificador digital com capacidade de generalização é baseado na seleção apropriada das amostras binárias do conjunto de dados de treinamento, antes da utilização do minimizador Booleano para obtenção da função Booleana do circuito (Lacerda & Braga 2005). Na Figura 6.1 é apresentado um diagrama descrevendo as etapas do projeto do classificador.

Inicialmente, os dados binários de treinamento são utilizados como entrada para o algoritmo de seleção de amostras apresentado no capítulo anterior (RSR). O algoritmo é claramente dividido em três etapas, conforme mostrado na Figura 6.1. Ao final da terceira etapa, tem-se as melhores amostras que descrevem cada classe dos dados de entrada.

Em seguida, as amostras selecionadas são utilizadas como entrada para o minimizador Booleano (foi utilizado o Espresso). As amostras descartadas pelo algoritmo RSR são consideradas pelo algoritmo de minimização como amostras *don't care*, o que facilita o processo de expansão das coberturas criadas pelo minimizador, além de embutir a característica de generalização na função Booleana resultante.

O resultado do minimizador Booleano, dado em termos de soma de produtos de literais, descreve o circuito digital o qual pode ser implementado em um circuito digital programável tipo PAL (*Programmable Array Logic*), composto por matriz de entrada de portas *AND* e matriz de saída de porta *OR*. A saída do circuito digital, tendo como entrada os dados binários desejados para classificar, compreende o resultado da classificação (neste projeto foi aplicado

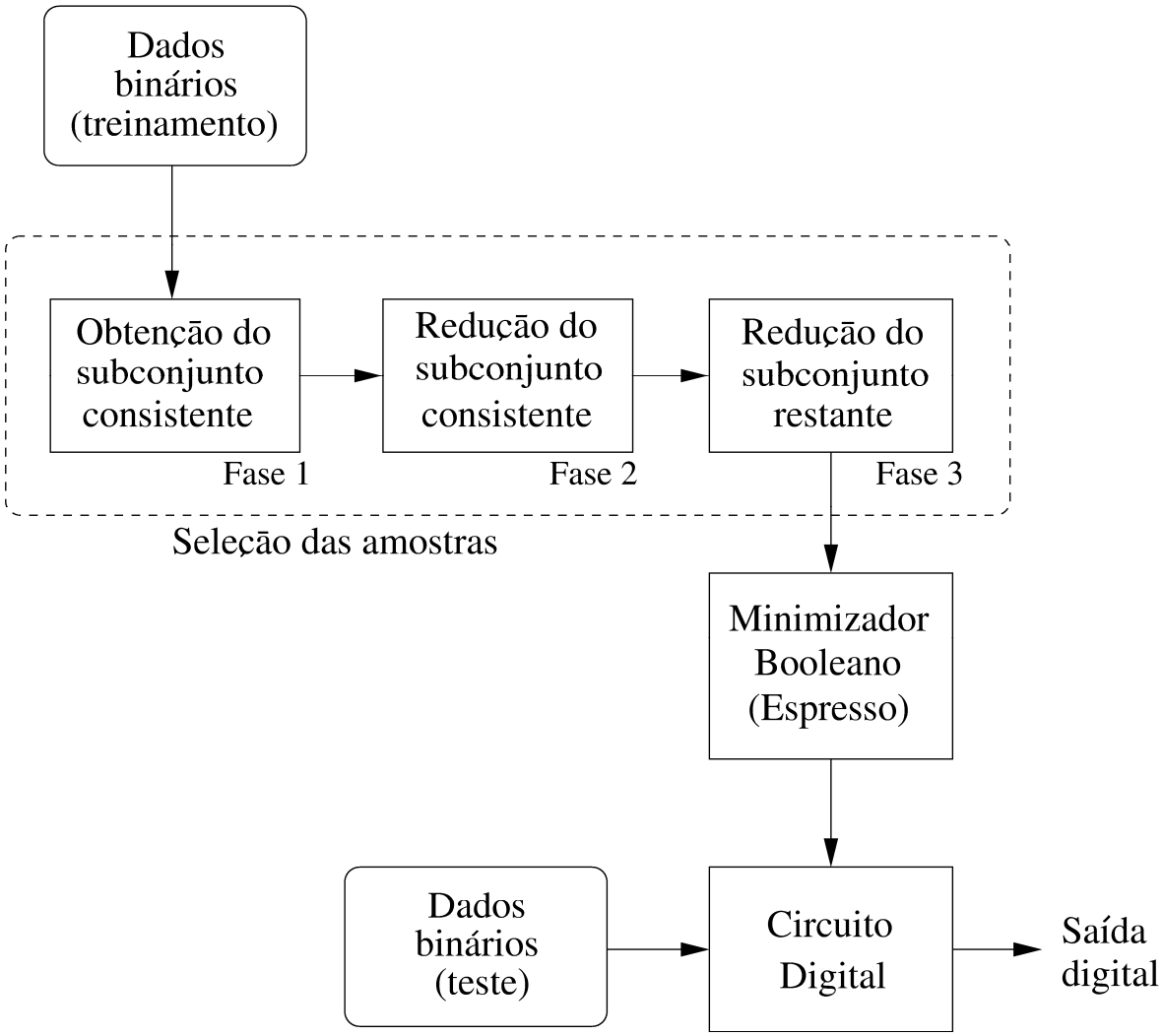


Figura 6.1: Diagrama do método de projeto do circuito digital classificador.

apenas para duas classes).

De forma a obter um método automático para gerar o circuito classificador digital com o melhor desempenho, pode ser aplicado o algoritmo abaixo, de forma a ajustar automaticamente o parâmetro k exigido no algoritmo RSR. Uma vez determinado o melhor parâmetro k, não é mais necessário ajustá-lo para o mesmo problema. O mesmo pode ser feito para definir a melhor métrica de distância utilizada no algoritmo RSR.

1. Faça $k = 0$.
2. Utilizando o minimizador de função Booleana (Espresso), e tendo como entrada todas as amostras do subconjunto restante, gere o circuito classificador digital.
3. Contabilize o número de acertos de classificação do circuito gerado pelo passo anterior utilizando o conjunto de amostras de teste. Armazene o resultado em MAX.
4. Incremente o valor de k de um.
5. Obtenha, do subconjunto restante, os k vizinhos mais próximos de cada classe para cada amostra do subconjunto consistente reduzido, e armazene em um subconjunto restante reduzido (inicialmente vazio). Pode ser que alguma amostra do subconjunto restante já esteja no subconjunto restante reduzido, neste caso ignore-a.
6. Utilizando o minimizador de função Booleana (Espresso), e tendo como entrada o subconjunto restante reduzido obtido no passo anterior, gere o novo circuito classificador digital.
7. Contabilize o número de acertos de classificação do circuito gerado no passo anterior utilizando o conjunto de amostras de teste.
8. Repita os passos 4 a 7 enquanto o número de acertos de classificação dos dados de teste, utilizando o circuito gerado pelo subconjunto restante reduzido, for menor que MAX.

A seguir são apresentados os resultados dos experimentos realizados utilizando esta metodologia de projeto proposta.

6.1 Experimentos

Para a realização dos testes de desempenho e obtenção dos resultados experimentais, foi utilizado um computador PC com processador Intel Pentium IV - 1800MHz com 1 Giga bytes de memória RAM, e sistema operacional Linux.

O método proposto (RSR) no capítulo anterior e outros métodos (para efeito de comparação) foram utilizados em conjunto com o minimizador Booleano para obter a função Booleana de circuitos classificadores digitais na solução de alguns problemas, tanto com dados gerados artificialmente quanto com dados reais. Pelo fato de o classificador projetado ser um circuito digital, é necessário que os dados do problema sejam binários. Foram escolhidos problemas de apenas duas classes (0 ou 1) para efeito de simplificação.

O algoritmo RSR proposto foi implementado em linguagem C++ (Eckel 2000) e utilizou-se dados binários de entrada armazenados em arquivo (formato Espresso). A versão pronta do programa Espresso (University of Texas at Austin – College of Engineering 1990) utilizada foi implementada em linguagem C. O processamento da função Booleana do classificador obtido também foi implementado em C++ para cálculo dos resultados de classificação dos dados de teste (simulação do circuito digital).

Os outros métodos utilizados para efeito de comparação com o método proposto (RSR) foram:

- Sem pré-processamento (filtragem) dos dados de treinamento, ou seja, os dados originais são empregados diretamente como entrada para o Espresso.

- Rede Neural (Haykin 2001) de duas camadas (*Multilayer Perceptron* - MLP) com treinamento multiobjetivo (MOBJ) (Teixeira 2001).

- Máquina de Vetor de Suporte (*Support Vector Machine* - SVM) (Vapnik 1998; Schwaighofer 2002)

Para utilizar a generalização aprendida tanto pela Rede Neural quanto pela SVM, estes foram treinados com os dados de treinamento. Então, os próprios dados de treinamento são reclassificados por estas máquinas já treinadas de modo a passar aos dados a generalização aprendida por estas máquinas. De posse dos dados reclassificados, estes foram utilizados para obter a equação Booleana do circuito digital classificador utilizando o minimizador de função Booleana (Espresso). A implementação utilizada nestes métodos foi para Matlab versão 6.5 (The Mathworks 2001).

Os resultados da utilização dos diferentes métodos (Sem filtragem, RSR, MLP e SVM) para obtenção da função lógica do circuito digital classificador foram contabilizados e comparados em cada problema de classificação. Isto é mostrado a seguir.

6.1.1 Dados Sintéticos

Aplicando-se o método proposto (RSR) nos dados sintéticos descritos em capítulo anterior (Figura 4.4), obteve-se os resultados indicados nas Figuras 6.2 a 6.6. O procedimento foi repetido para cada conjunto de amostras geradas artificialmente (10 conjuntos) e obtido a média para apresentação nos gráficos. O procedimento foi executado para diferentes tipos de métricas de distância indicadas nas correspondentes legendas dos gráficos. A quantidade de vizinhos mais próximos de cada classe para cada amostra do subconjunto consistente reduzido é representada por k nas figuras.

O gráfico da Figura 6.2 indica o número médio de amostras selecionadas para compor o subconjunto restante reduzido para cada valor de k, em cada métrica utilizada. Observa-se que à medida em que o parâmetro k cresce, a quantidade de amostras selecionadas também aumenta, obviamente. A métrica *overlap* apresentou uma taxa maior de crescimento de amostras em função de k, enquanto a métrica de Bayes apresentou a menor taxa. Este subconjunto de dados selecionados são utilizados pelo Espresso para obtenção da função lógica do circuito digital classificador.

O gráfico da Figura 6.3 apresenta o número médio de acertos de classificação dos dados de teste obtidos pela simulação do circuito digital obtido tomando como base de projeto cada subconjunto restante reduzido obtido com cada valor de k, para cada um dos 10 experimentos e para cada uma das 6 métricas testadas. Observa-se pelo gráfico que o número médio de acertos de classificação aumenta inicialmente a partir de $k = 1$ independente da métrica utilizada, e após um certo valor para k o gráfico tende a estabilizar. O melhor resultado é obtido para $k = 7$ e utilizando-se a métrica VDM modificada. O pior desempenho do circuito foi obtido utilizando-se a métrica de Bayes (para baixos valores de k, $k \leq 5$) e *overlap* (para valores mais altos de k, $k \geq 5$). Observa-se que, para o classificador 1-NN com RSR descrito em capítulo anterior (Figura 5.11) e também empregado neste problema, a melhor métrica obtida foi a de Bayes, enquanto que com o método RSR para geração do circuito digital classificador, a métrica VDM modificada apresenta um resultado melhor em relação às outras métricas.

Os gráficos das Figuras 6.4 e 6.5 apresentam os índices de acertos dos circuitos classificadores obtidos para os dados de teste em função da quantidade de amostras utilizadas e número de termos de produto, respectivamente. Observa-se novamente a superioridade da métrica VDM modificada no índice de acertos do circuito obtidos nos dois gráficos.

A Figura 6.6 apresenta o número médio de termos de produtos obtidos em função do parâmetro k. A métrica *overlap* apresentou o maior índice de termos de produto em função de k, o que explica o baixo desempenho desta métrica no

índice de acertos do circuito indicado nos gráficos anteriores. A métrica VDM modificada, por sua vez, apresentou o menor índice de termos de produto em função de k, o que explica a sua capacidade maior de generalização, ou seja, um índice maior de acertos para os dados de teste.

Na Tabela 6.1 são mostrados os resultados com os circuitos digitais simulados e obtidos a partir dos diferentes métodos apresentados, para a solução do problema de classificação dos 10 conjuntos de dados artificiais. São mostrados a quantidade de amostras utilizadas para obter a função Booleana do circuito digital pelo Espresso (minimizador Booleano), o tempo de processamento do programa Espresso, o número de produtos obtidos na função Booleana de cada circuito, e a porcentagem de acertos dos dados de teste pelo circuito obtido em cada método. Os valores apresentados estão no formato de média e desvio padrão.

Tabela 6.1: Resultados obtidos com cada método no problema com dados binários sintéticos.

Método	**Nº amostras**	**T. Espresso [s]**	**Nº prod.**	**Acertos (%)**
Sem filtragem	2380 (100%)	2±0,5	65±3	84,0±1,6
RSR (k=7,VDMm)	1129±44 (47,4±1,8%)	< 1	19±1	89,1±1,2
MLP (32 nodos)	2380 (100%)	6±1	30±2	89,1±1,1
SVM (rbf,w=10,c=1)	2380 (100%)	5±1	31±1	88,6±0,8

Observa-se que o método proposto (RSR) utilizando $k = 7$ e métrica de distância VDM modificada (VDMm) apresenta um resultado de acertos de classificação dos dados de teste com o circuito simulado semelhante aos outros métodos, mas utilizando uma quantidade de amostras menor. Pelo fato do método RSR utilizar uma quantidade menor de amostras como entrada para o programa Espresso, este necessitou significativamente de menos tempo de processamento quando comparado com outros métodos. O método proposto também obtém a função lógica do circuito digital com menos termos de produto, o que economiza portas lógicas na implementação do circuito digital, diminuindo a área ocupada e consumo do circuito gerado.

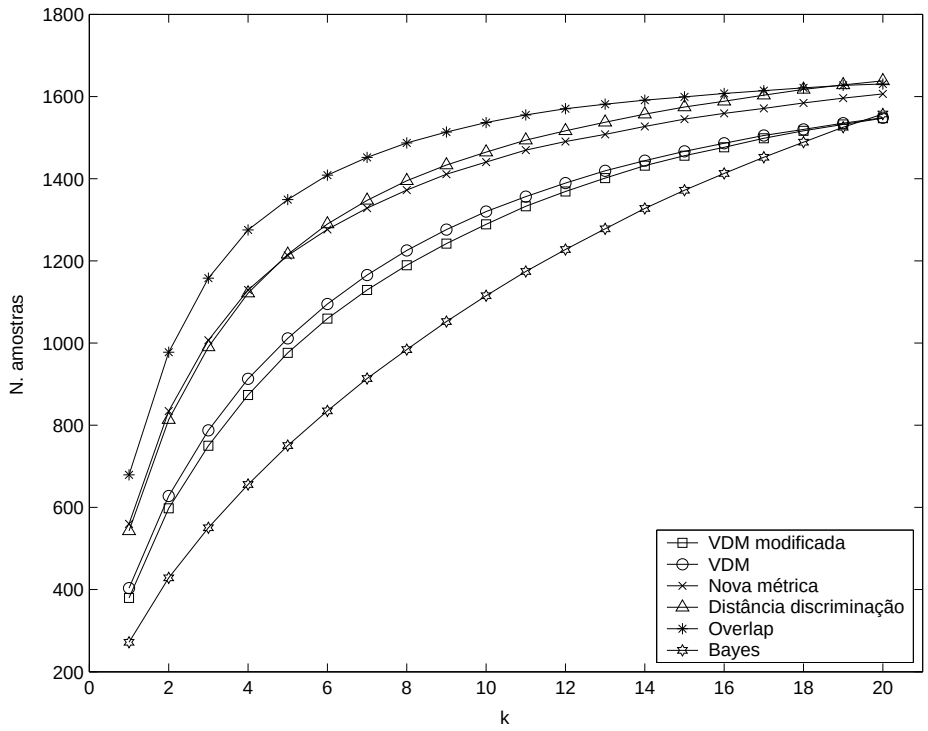


Figura 6.2: Quantidade média de amostras selecionadas para obtenção da função Booleana do circuito digital classificador, para cada valor de k, na solução do problema com dados artificiais utilizando o método proposto (RSR).

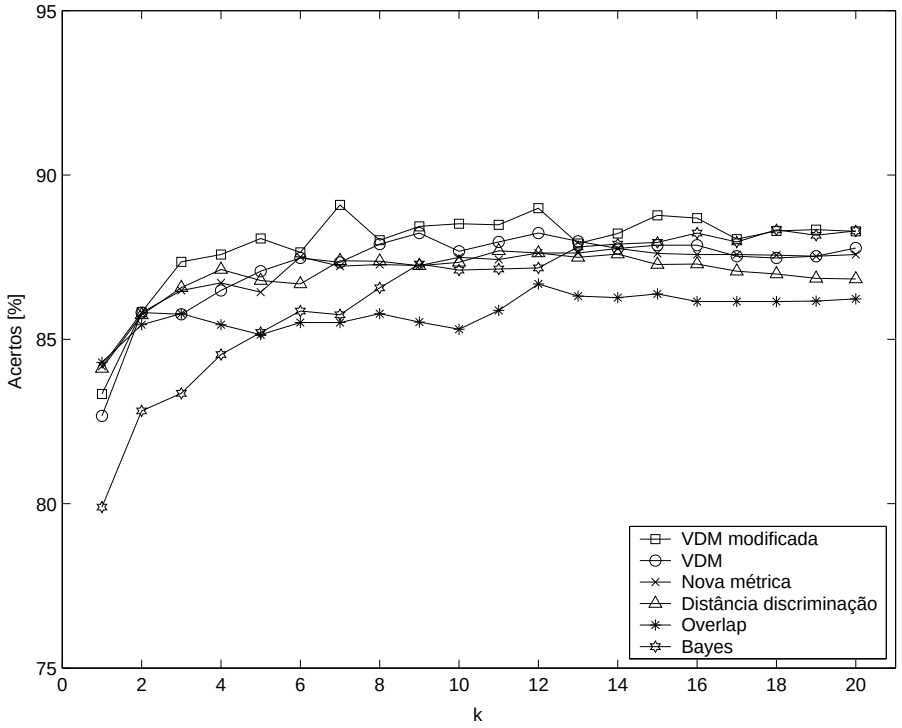


Figura 6.3: Acertos de classificação para os dados de teste em cada circuito digital simulado, para cada valor de k, na solução do problema com dados artificiais utilizando o método proposto (RSR).

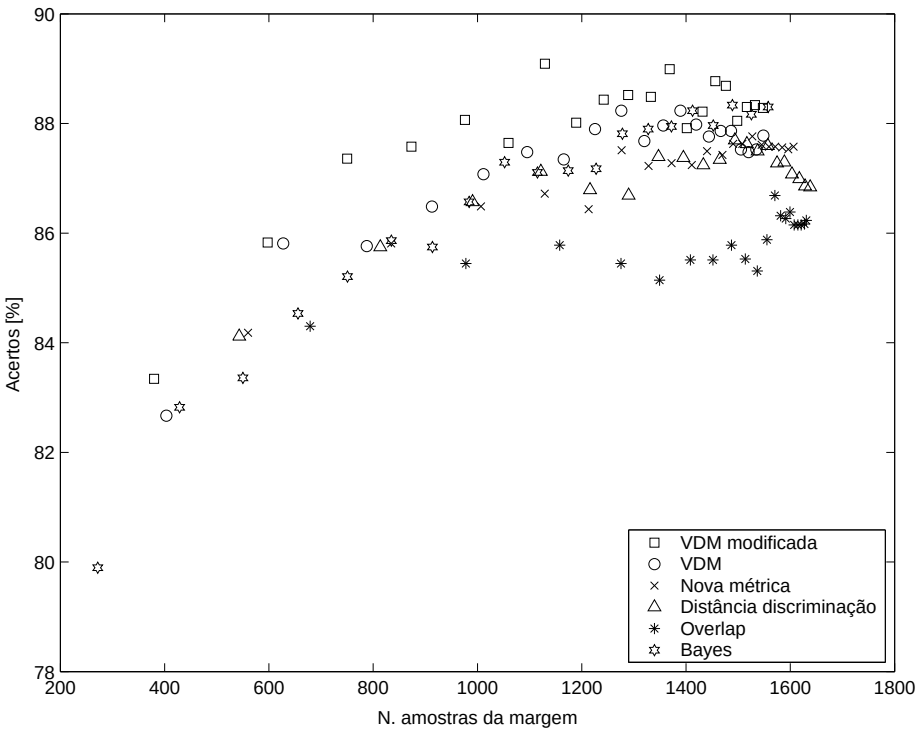


Figura 6.4: Acertos de classificação para os dados de teste em cada circuito digital simulado, para cada quantidade de amostras selecionadas, na solução do problema com dados artificiais utilizando o método proposto (RSR).

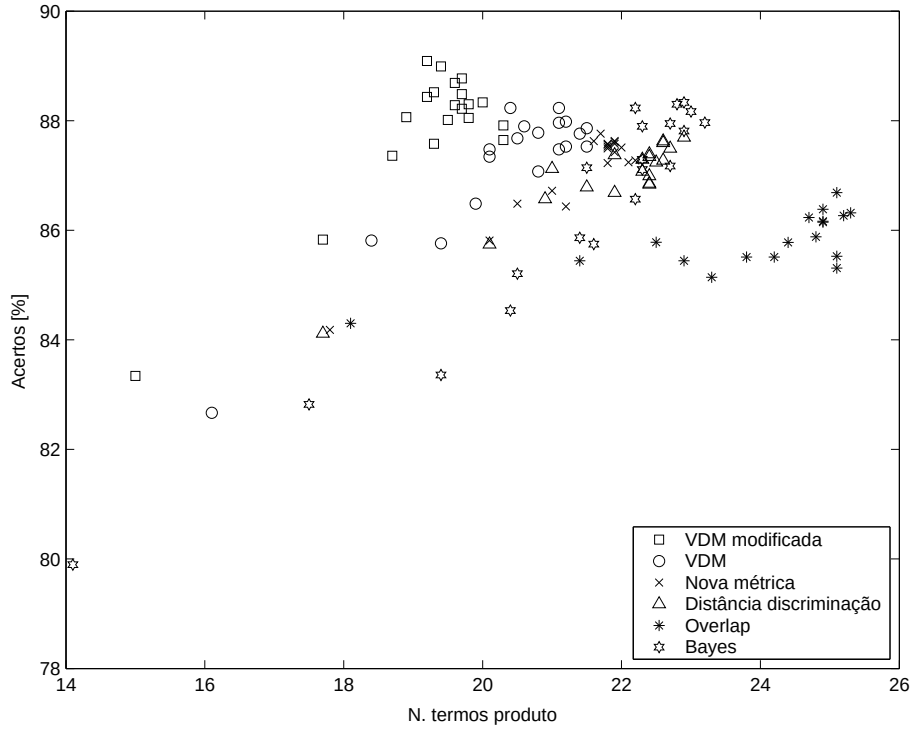


Figura 6.5: Acertos de classificação para os dados de teste em cada circuito digital simulado, para cada valor de termos de produto, na solução do problema com dados artificiais utilizando o método proposto (RSR).

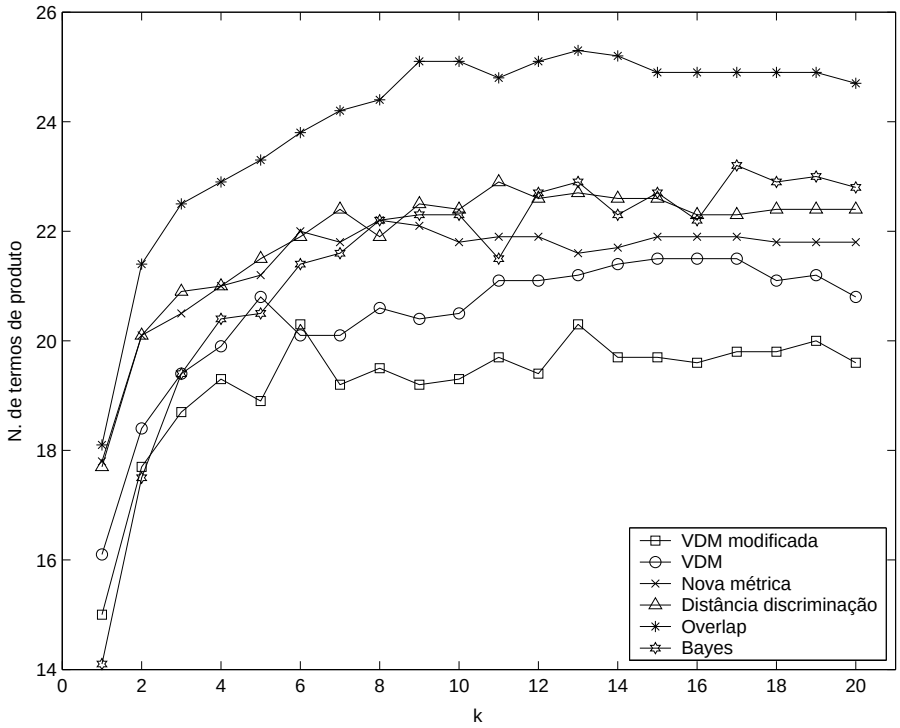


Figura 6.6: Número médio de termos de produto obtidos, para cada valor de k, na solução do problema com dados artificiais utilizando o método proposto (RSR).

6.1.2 Reconhecimento de Caracter Manuscrito

Foram obtidas as funções lógicas dos circuitos digitais classificadores utilizando o método proposto de projeto com os dados de caracteres manuscritos descrito no Capítulo 4 (Figura 4.6). Novamente o problema foi simplificado para a identificação apenas do caracter "0" dentre os caracteres de "0" a "9". O processo foi repetido 10 vezes, sendo que a cada processo os dados eram permutados antes de serem divididos em conjunto de treinamento, teste e validação. Para o método proposto (RSR), são apresentados os resultados nas Figuras 6.7 a 6.11, onde k representa a quantidade de vizinhos mais próximos de cada classe para cada amostra do subconjunto consistente reduzido. Para este problema, o valores escolhidos para k variaram entre 5 e 100, tomados de 5 em 5.

O gráfico da Figura 6.7 indica o número médio de amostras selecionadas para compor o subconjunto restante reduzido para cada valor de k, em cada métrica utilizada. Observa-se novamente que, a medida em que o parâmetro k cresce, a quantidade de amostras selecionadas também aumenta, como já esperado. A métrica VDM modificada apresentou uma taxa maior de crescimento de amostras em função de k, enquanto a métrica de Bayes apresentou a menor taxa. Este subconjunto de dados selecionados é utilizado pelo Espresso para obtenção da função Booleana do circuito digital classificador.

O gráfico da Figura 6.8 apresenta o número médio de acertos de classificação dos dados de teste pelo circuito digital simulado tomando como base de projeto cada subconjunto restante reduzido obtido com cada valor de k, para cada um dos 10 experimentos e para cada uma das 6 métricas testadas. Observa-se pelo gráfico que o número médio de acertos de classificação apresenta uma variação inicialmente a partir de $k = 5$ dependendo da métrica utilizada, e após um certo valor para k o gráfico tende a estabilizar. O melhor resultado é obtido para $k = 50$ e utilizando-se a métrica de Bayes. O pior desempenho geral do circuito foi obtido utilizando-se a métrica *overlap*. Interessante comparar com o resultado obtido em capítulo anterior utilizando o classificador 1-NN com RSR (Figura 5.12). Neste classificador, o pior desempenho foi obtido com a métrica de Bayes enquanto aqui foi o melhor!

Os gráficos das Figuras 6.9 e 6.10 apresentam os índices de acertos dos circuitos classificadores simulados para os dados de teste em função do número de termos de produtos e quantidade de amostras utilizadas, respectivamente. Observa-se novamente a superioridade da métrica de Bayes no índice de acertos do circuito nos dois gráficos, utilizando uma quantidade de termos de produto e amostras menores que o valores máximos utilizados.

A Figura 6.11 apresenta o número médio de termos de produtos obtidos em função do parâmetro k. A métrica de Bayes, apresentou o menor índice de termos de produto em função de k, o que explica a sua capacidade maior de generalização, ou seja, um índice maior de acertos para os dados de teste.

Na Tabela 6.2 são mostrados os resultados obtidos com os circuitos digitais simulados e projetados a partir dos diferentes métodos apresentados, para a solução do problema de classificação dos 10 conjuntos de dados do caracter manuscrito. São mostrados a quantidade de amostras utilizadas para obter pelo Espresso a função Booleana do circuito digital, o tempo de processamento do Espresso para obter a função Booleana do circuito em cada método, o número de produtos da função Booleana de cada circuito, e a porcentagem de acertos dos dados de teste pelo circuito simulado. Os valores apresentados estão no formato de média e desvio padrão.

Tabela 6.2: Resultados obtidos no problema de reconhecimento do caracter "0" para cada método.

Método	**Nº amostras**	**T. Espresso [s]**	**Nº prod.**	**Acertos (%)**
Sem filtragem	1548 (100%)	58715±16972	2,0±0,0	98,6±0,7
RSR (k=50,Bayes)	472±49(30,5±3,2%)	13800±2375	1,3±0,5	98,9±0,5
MLP (4 nodos)	1548 (100%)	43200±16752	1,9±0,3	98,6±0,3
SVM (rbf,w=1,c=1)	1548 (100%)	128924±27514	2,0±0,0	98,9±0,6

Novamente observa-se que o método proposto (RSR) possui resultados de acertos de classificação próximos aos outros métodos utilizando menos amos-

tras e com menor tempo de processamento do Espresso.

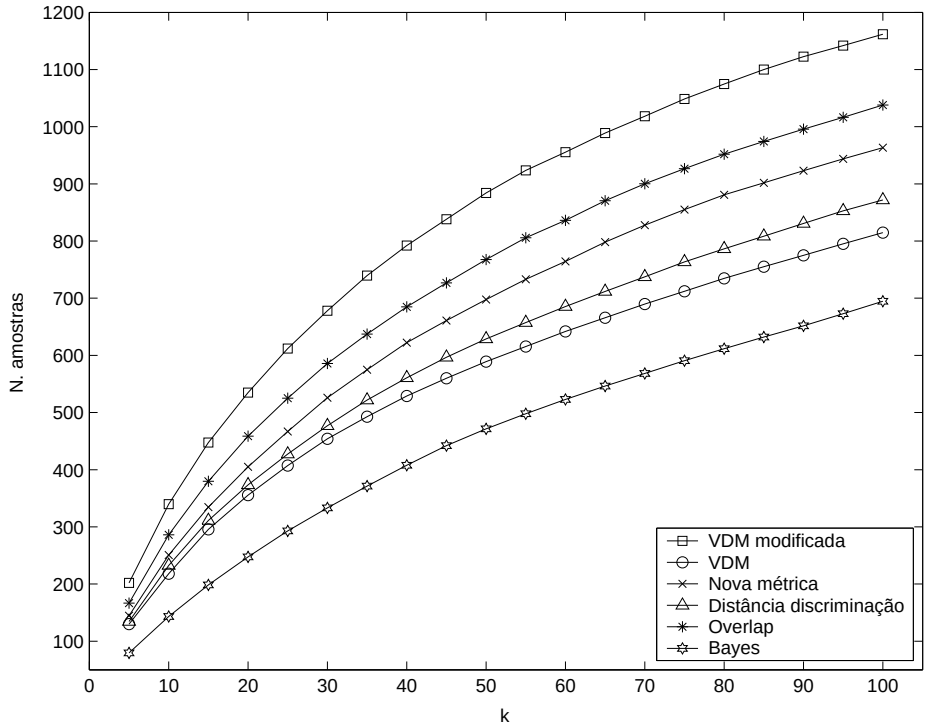


Figura 6.7: Quantidade média de amostras selecionadas para obtenção da função lógica do circuito digital classificador, para cada valor de k, na solução do problema de reconhecimento de caracteres manuscritos utilizando o método proposto (RSR).

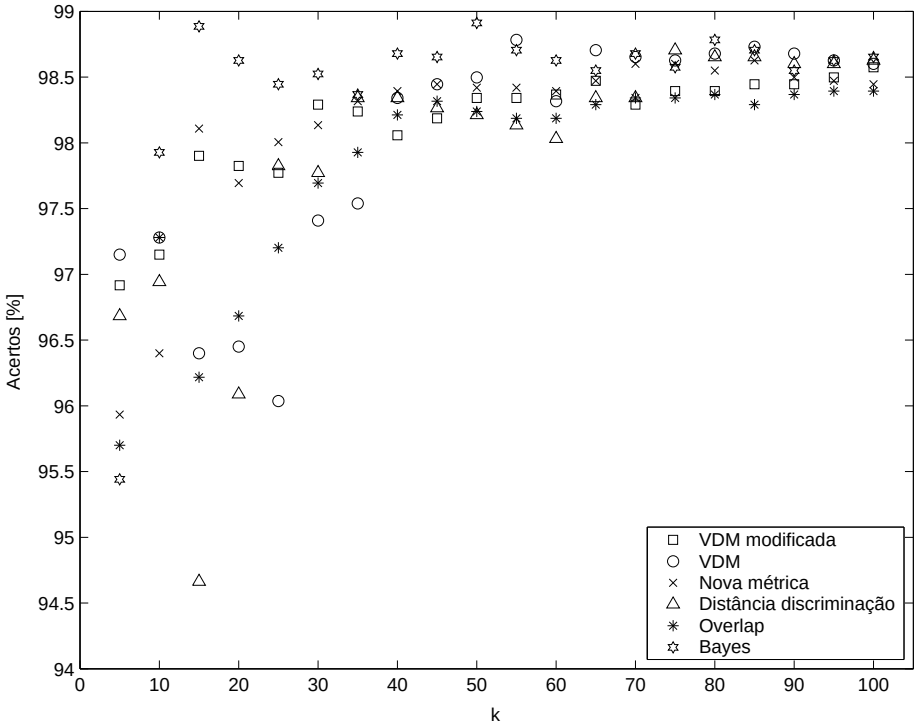


Figura 6.8: Acertos de classificação para os dados de teste em cada circuito digital simulado, para cada valor de k, na solução do problema de reconhecimento de caracteres manuscritos utilizando o método proposto (RSR).

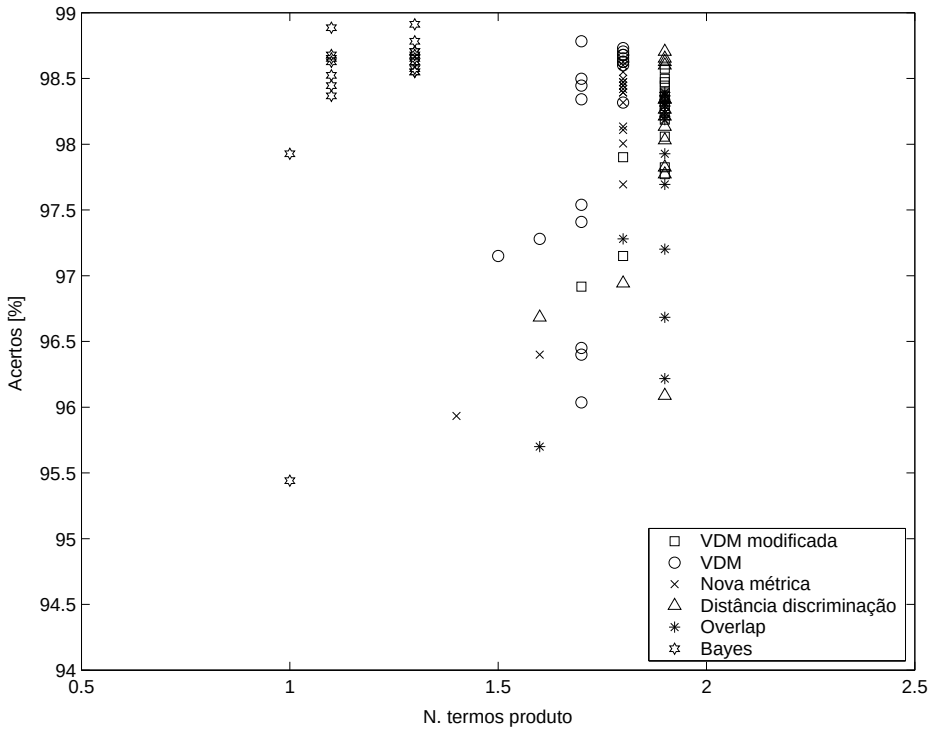


Figura 6.9: Acertos de classificação para os dados de teste em cada circuito digital simulado, para cada valor de termos de produto, na solução do problema de reconhecimento de caracteres manuscritos utilizando o método proposto (RSR).

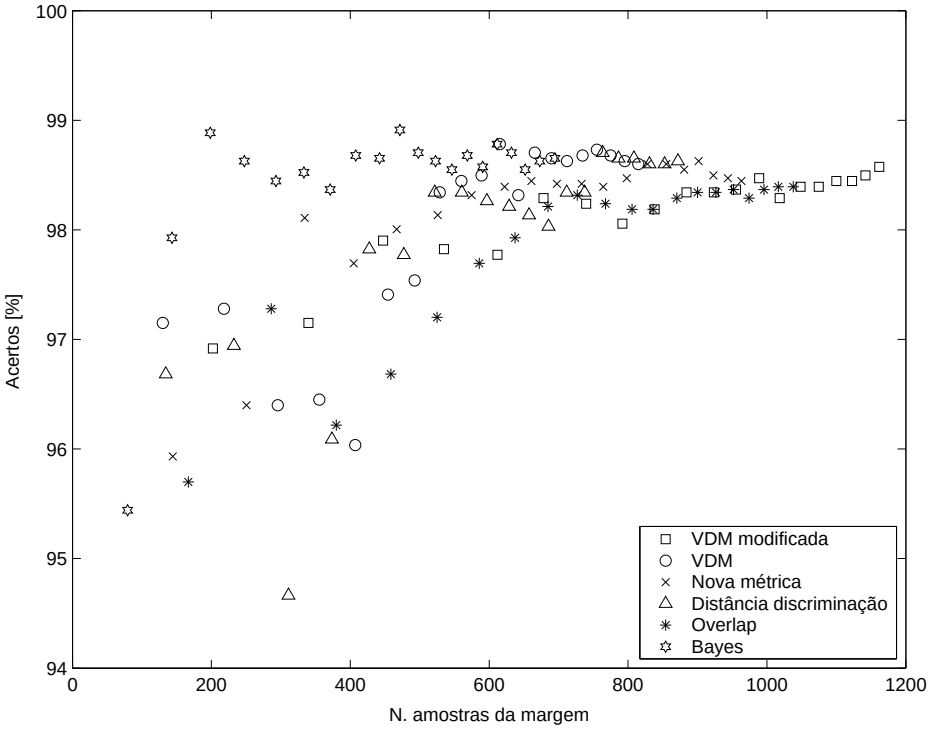


Figura 6.10: Acertos de classificação para os dados de teste em cada circuito digital simulado, para cada quantidade de amostras selecionadas, na solução do problema de reconhecimento de caracteres manuscritos utilizando o método proposto (RSR).

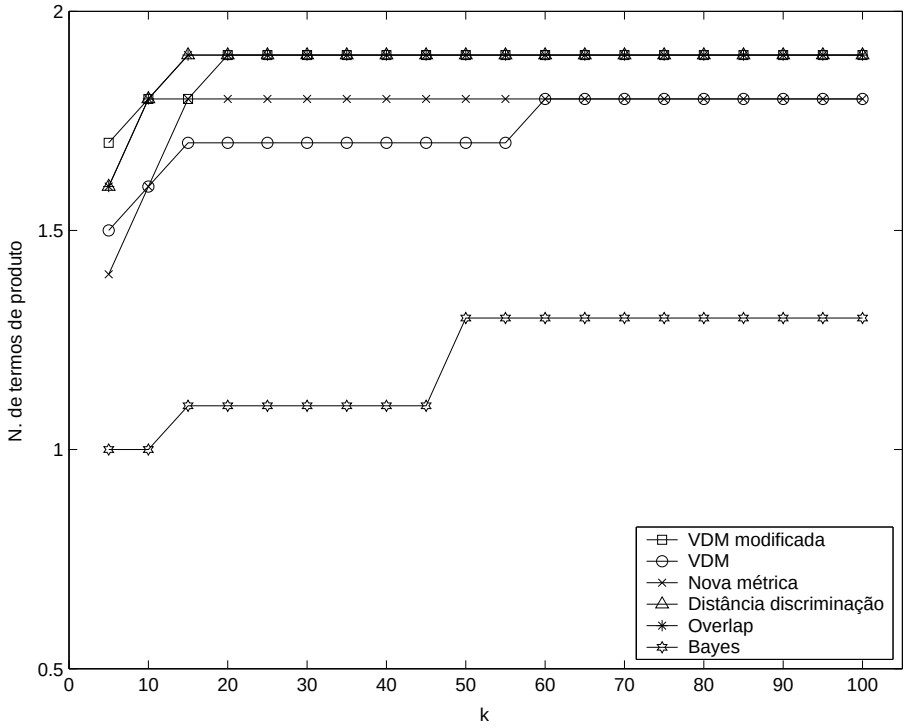


Figura 6.11: Número médio de termos de produto obtidos, para cada valor de k, na solução do problema de reconhecimento de caracteres manuscritos utilizando o método proposto (RSR).

6.1.3 Problema do Gene

Foram obtidas as funções lógicas dos circuitos digitais classificadores utilizando o método proposto com os dados do problema Gene descrito em capítulo anterior (Figura 4.9). Novamente o problema foi simplificado para a identificação de duas classes. O processo foi repetido 10 vezes para cada método, sendo que a cada processo os dados eram permutados antes de serem divididos em conjunto de treinamento, teste e validação. Para o método proposto (RSR), são apresentados os resultados nas Figuras 6.12 a 6.16, onde k representa a quantidade de vizinhos mais próximos de cada classe para cada amostra do subconjunto consistente reduzido.

O gráfico da Figura 6.12 indica o número médio de amostras selecionadas para compor o subconjunto restante reduzido para cada valor de k, em cada métrica utilizada. Observa-se que a medida em que o parâmetro k cresce, a quantidade de amostras selecionadas também aumenta, como já esperado. A métrica Distância de Discriminação apresentou uma taxa maior de crescimento de amostras em função de k. A métrica de Bayes apresentou um crescimento assintótico menor do número de amostras em função de k, sendo que para valores de $k > 5$ a métrica *overlap* apresentou valor quase constante de amostras. Este subconjunto de dados selecionados são utilizados pelo Espresso para obtenção da função lógica do circuito digital classificador.

O gráfico da Figura 6.13 apresenta o número médio de acertos de classificação dos dados de teste pelo circuito digital simulado tomando como base de projeto cada subconjunto restante reduzido obtido com cada valor de k, para cada um dos 10 experimentos e para cada umas das 6 métricas testadas. O melhor resultado é obtido para $k = 21$ e utilizando-se a métrica Distância de Discriminação (DD). O pior desempenho do circuito foi obtido utilizando-se a métrica de *overlap* para grandes valores de k. Interessante comparar com o resultado obtido em capítulo anterior utilizando o classificador 1-NN com RSR (Figura 5.13), onde a métrica VDM apresentou o melhor resultado para o classificador.

Os gráficos das Figuras 6.14 e 6.15 apresentam os índices de acertos dos circuitos classificadores simulados para os dados de teste em função do número de termos de produtos e quantidade de amostras utilizadas, respectivamente. Observa-se novamente a superioridade da métrica Distância de Discriminação no índice de acertos do circuito mostrado nos dois gráficos.

A Figura 6.16 apresenta o número médio de termos de produto obtidos em função do parâmetro k. A métrica *overlap* apresentou o menor índice assintótico médio de termos de produto em função de k, entretanto, isto não foi capaz de garantir o melhor desempenho para o circuito classificador obtido com esta métrica.

Na Tabela 6.3 são mostrados os resultados obtidos com os circuitos digitais simulados e projetados a partir dos diferentes métodos apresentados, para a solução do problema de classificação dos 10 conjuntos de dados do problema dos genes. São mostrados a quantidade de amostras utilizadas pelo Espresso para obter a função Booleana do circuito digital, o tempo de processamento do Espresso em cada método, o número de produtos da função Booleana de cada circuito obtido, e a porcentagem de acertos dos dados de teste pelo circuito simulado. Os valores apresentados estão no formato de média e desvio padrão.

Tabela 6.3: Resultados obtidos no problema Gene.

Método	**Nº amostras**	**T. Espresso [s]**	**Nº produtos**	**Acertos (%)**
Sem filtragem	2391 (100%)	1526±117	80±3	69,5±2,5
RSR (k=21, DD)	1838±13(72,4±0,5%)	995±86	47±5	78,7±2,8
MLP (32 nodos)	2540 (100%)	4486±503	80±6	69,4±2,8
SVM (rbf,w=10,c=1)	2540 (100%)	4493±906	73±4	72,0±2,4

Observa-se que o método proposto utilizando a métrica Distância de Discriminação (DD) com $k = 21$ possui resultados de acertos de classificação melhores que outros métodos utilizando menos amostras e menos tempo de processamento para o Espresso.

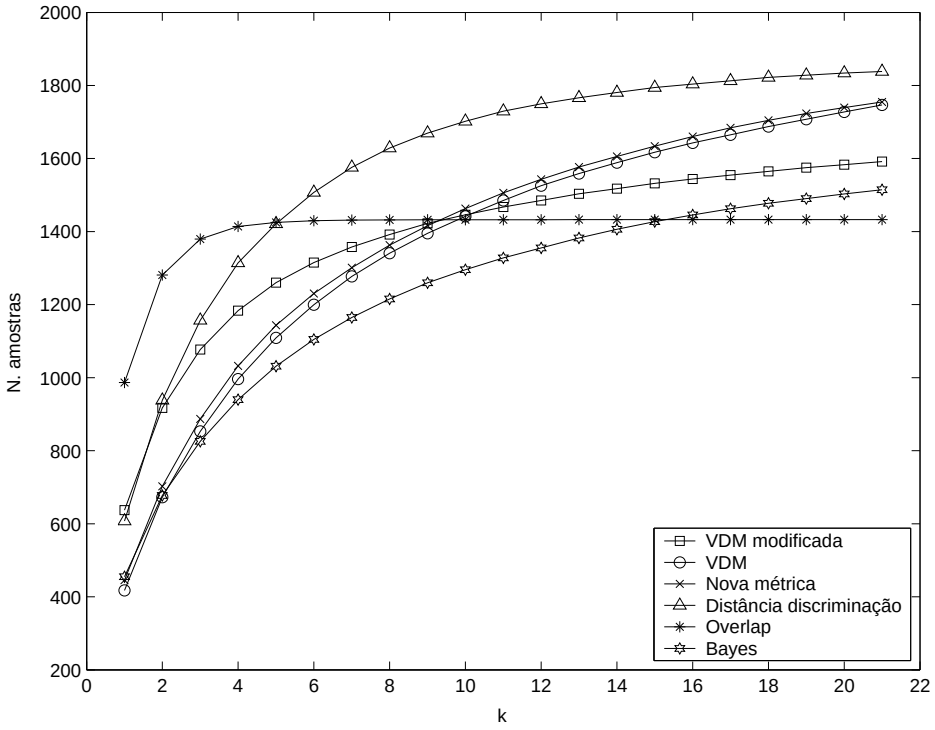


Figura 6.12: Quantidade média de amostras selecionadas para obtenção da função Booleana do circuito digital classificador, para cada valor de k, na solução do problema do gene utilizando o método proposto (RSR).

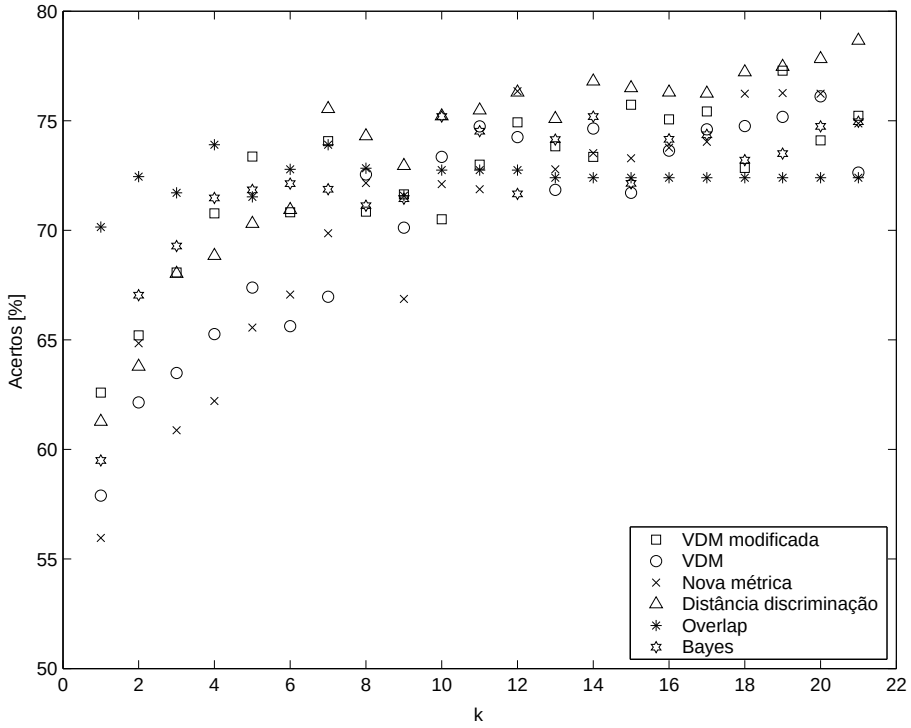


Figura 6.13: Acertos de classificação para os dados de teste em cada circuito digital simulado, para cada valor de k, na solução do problema do gene utilizando o método proposto (RSR).

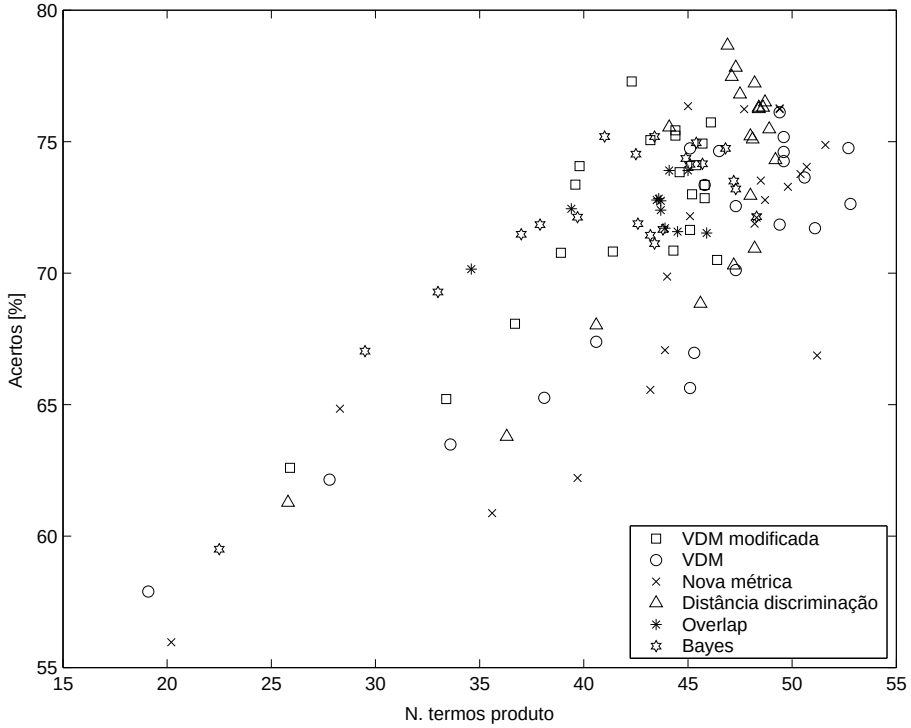


Figura 6.14: Acertos de classificação para os dados de teste em cada circuito digital simulado, para cada valor de termos de produto, na solução do problema do gene utilizando o método proposto (RSR).

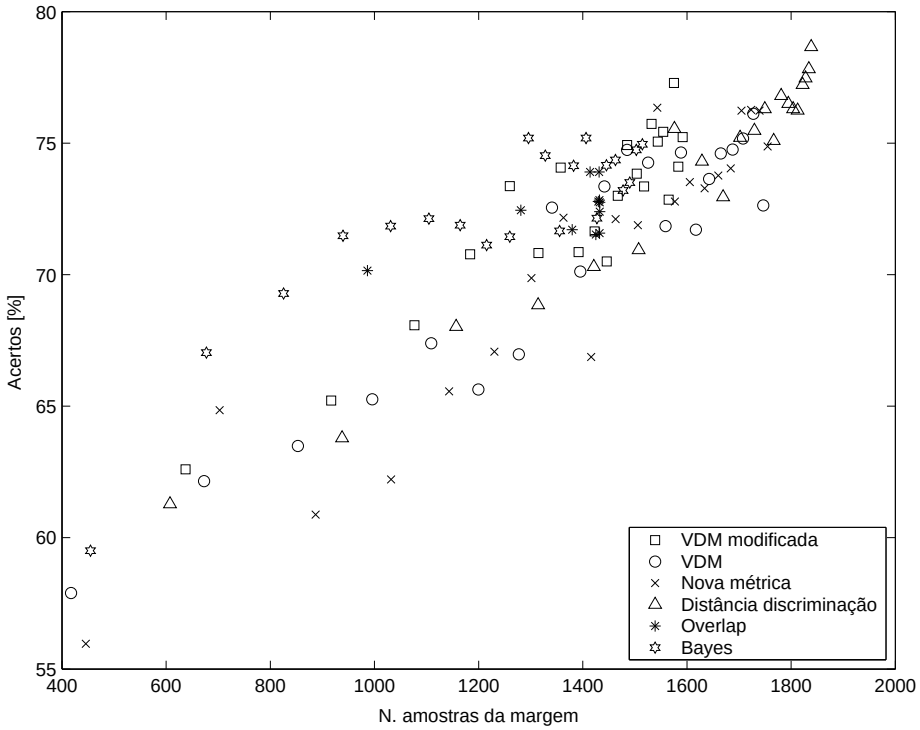


Figura 6.15: Acertos de classificação para os dados de teste em cada circuito digital simulado, para cada quantidade de amostras selecionadas, na solução do problema do gene utilizando o método proposto (RSR).

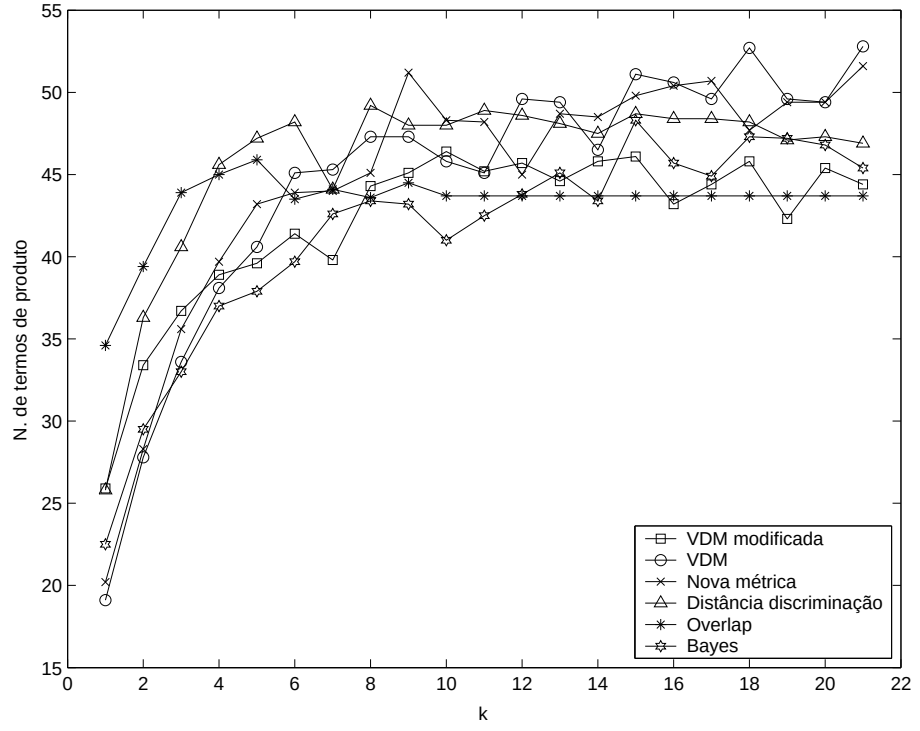


Figura 6.16: Número médio de termos de produto obtidos, para cada valor de k, na solução do problema do gene utilizando o método proposto (RSR).

6.1.4 Problema do Coração

Foram obtidas as funções lógicas dos circuitos digitais classificadores utilizando o método proposto com os dados do problema de diagnóstico do coração descrito em capítulo anterior (Figura 4.11). O processo foi repetido 10 vezes para cada método, sendo que a cada processo os dados eram permutados antes de serem divididos em conjunto de treinamento, teste e validação. Para o método proposto (RSR), são apresentados os resultados nas Figuras 6.17 a 6.21, onde k representa a quantidade de vizinhos mais próximos de cada classe para cada amostra do subconjunto consistente reduzido.

O gráfico da Figura 6.17 indica o número médio de amostras selecionadas para compor o subconjunto restante reduzido para cada valor de k, em cada métrica utilizada. Observa-se que a medida em que o parâmetro k cresce, a quantidade de amostras selecionadas também aumenta, como já esperado. A métrica Distância de Discriminação apresentou uma taxa menor de crescimento de amostras em função de k. A métrica de *overlap* apresentou uma taxa de crescimento maior para o número de amostras selecionadas em baixos valores de k ($k < 37$). Para altos valores de k ($k > 37$), a métrica de Bayes apresentou valores mais altos de amostras selecionadas. Este subconjunto de dados selecionados são utilizados pelo Espresso para obtenção da função Booleana do circuito digital classificador.

O gráfico da Figura 6.18 apresenta o número médio de acertos de classificação dos dados de teste pelo circuito digital simulado tomando como base de projeto cada subconjunto restante reduzido obtido com cada valor de k, para cada um dos 10 experimentos e para cada umas das 6 métricas testadas. O melhor desempenho do circuito projetado é obtido com a métrica de Bayes para $k = 39$, enquanto o pior resultado é obtido com a métrica nova, para $k = 1$. Observa-se que há uma grande variabilidade do resultado em função da métrica escolhida e valor de k. Interessante comparar com o resultado obtido em capítulo anterior utilizando o classificador 1-NN com RSR (Figura 5.14), onde o melhor resultado é obtido para $k = 1$ e utilizando-se também a métrica de Bayes. O pior desempenho do 1-NN com RSR foi obtido utilizando-se a métrica Distância de Discriminação para valores de $k > 1$. Para $k = 1$, a pior métrica foi a *overlap*.

Os gráficos das Figuras 6.19 e 6.20 apresentam os índices de acertos dos circuitos classificadores simulados para os dados de teste em função do número de termos de produtos e quantidade de amostras utilizadas, respectivamente. Observa-se novamente a superioridade da métrica de Bayes no desempenho do circuito mostrado nos dois gráficos.

A Figura 6.21 apresenta o número médio de termos de produtos gerados em função do parâmetro k. A métrica Distância de Discriminação apresentou

o menor índice assintótico médio de termos de produto em função de k, entretanto, isto não foi capaz de garantir um bom desempenho para o circuito classificador obtido com esta métrica.

Na Tabela 6.4 são mostrados os resultados obtidos com os circuitos digitais simulados e projetados a partir dos diferentes métodos apresentados, para a solução do problema de classificação dos 10 conjuntos de dados do problema do coração. São mostrados a quantidade de amostras utilizadas pelo Espresso para obtenção da função Booleana do circuito digital classificador, o tempo de processamento do Espresso em cada método, o número de termos de produto obtidos na função Booleana para cada circuito, e a porcentagem de acertos dos dados de teste obtido pelo circuito simulado. Os valores apresentados estão no formato de média e desvio padrão.

Tabela 6.4: Resultados obtidos no problema do coração.

Método	**Nº amostras**	**T. Espresso [s]**	**Nº produtos**	**Acertos (%)**
Sem filtragem	168 (100%)	<1	$14{,}2\pm1{,}4$	$84{,}4\pm4{,}3$
RSR (k=39, Bayes)	$126\pm4(75\pm2\%)$	<1	$6{,}5\pm1.6$	$86{,}8\pm5{,}2$
MLP (64 nodos)	168 (100%)	<1	$4{,}7\pm4{,}8$	$84{,}5\pm7{,}8$
SVM (rbf;w=0,1;c=0,1)	168 (100%)	<1	1 ± 0	$88{,}1\pm1{,}9$

Observa-se que utilizando SVM, obteve-se a função lógica do circuito classificador com apenas um termo de produto e com o maior índice de acertos dos dados de teste. O método proposto (RSR), mesmo utilizando menos amostras para obtenção da função Booleana, obteve um índice de acertos menor com mais termos de produto. Provavelmente o baixo desempenho do método RSR é devido a quantidade baixa de amostras de treinamento (apenas 168) em relação a alta dimensão dos dados (22 atributos). Devido também a pouca quantidade de amostras, o Espresso consumiu pouco tempo de processamento (<1s) em todos os métodos.

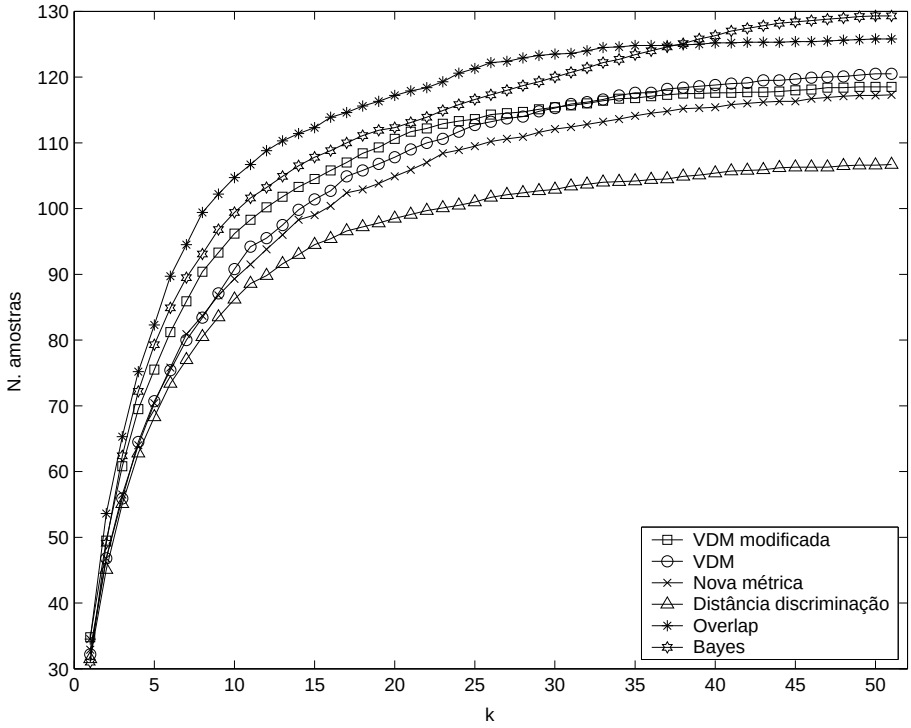


Figura 6.17: Quantidade média de amostras selecionadas para obtenção da função lógica do circuito digital classificador, para cada valor de k, na solução do problema do coração utilizando o método proposto (RSR).

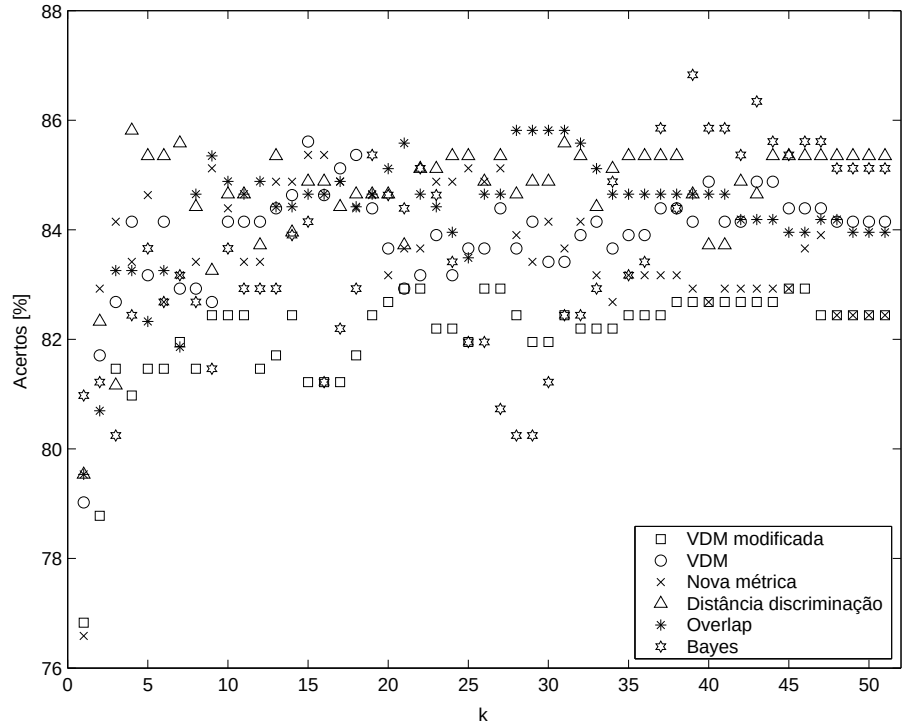


Figura 6.18: Acertos de classificação para os dados de teste em cada circuito digital simulado, para cada valor de k, na solução do problema do coração utilizando o método proposto (RSR).

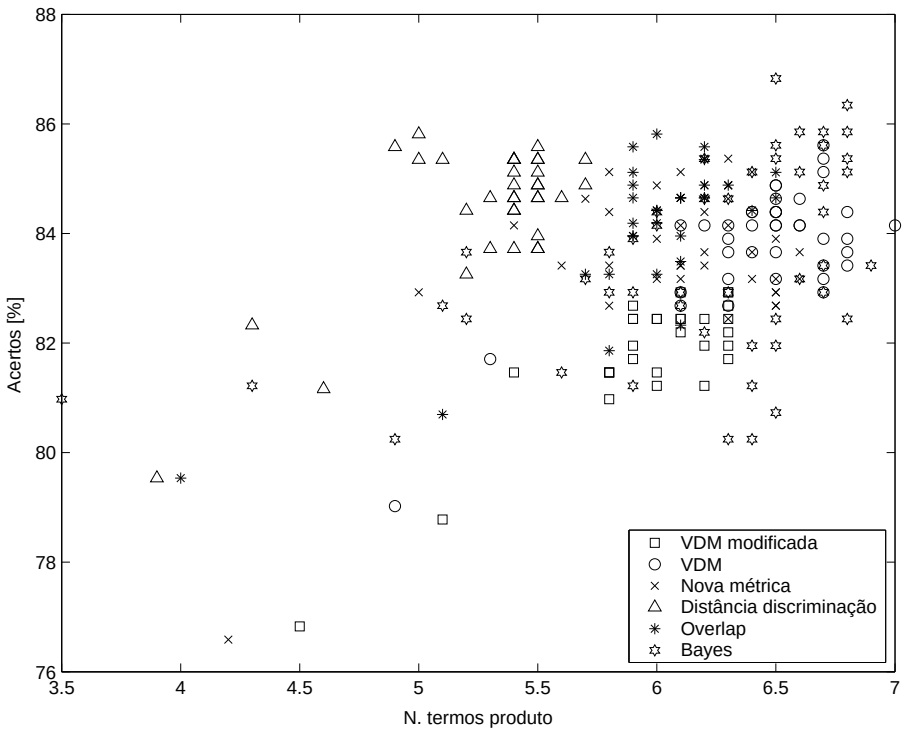


Figura 6.19: Acertos de classificação para os dados de teste em cada circuito digital simulado, para cada valor de termos de produto, na solução do problema do coração utilizando o método proposto (RSR).

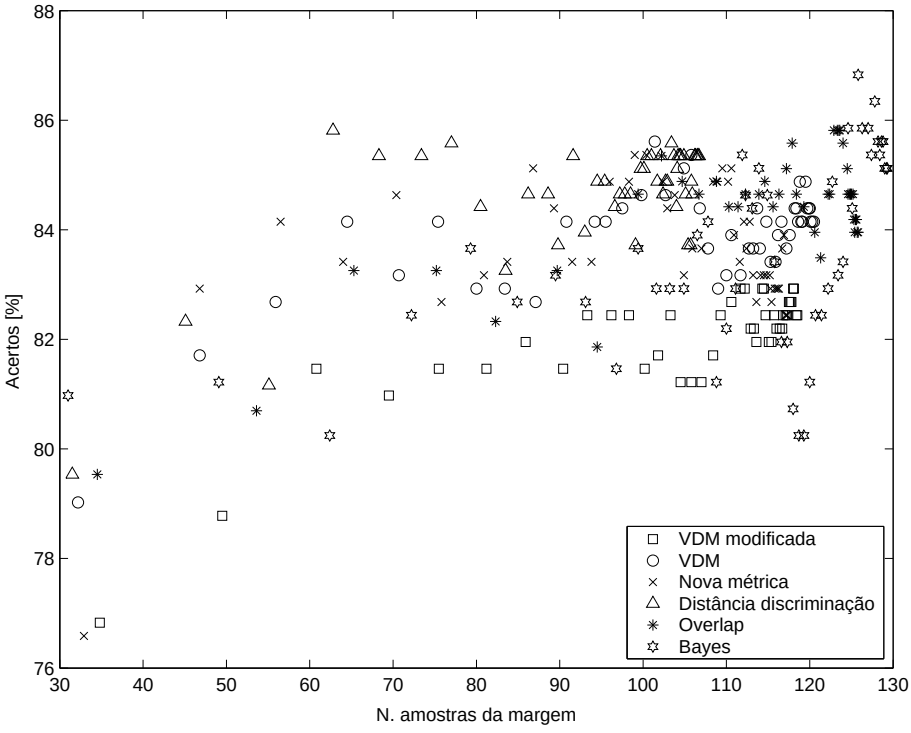


Figura 6.20: Acertos de classificação para os dados de teste em cada circuito digital simulado, para cada quantidade de amostras selecionadas, na solução do problema do coração utilizando o método proposto (RSR).

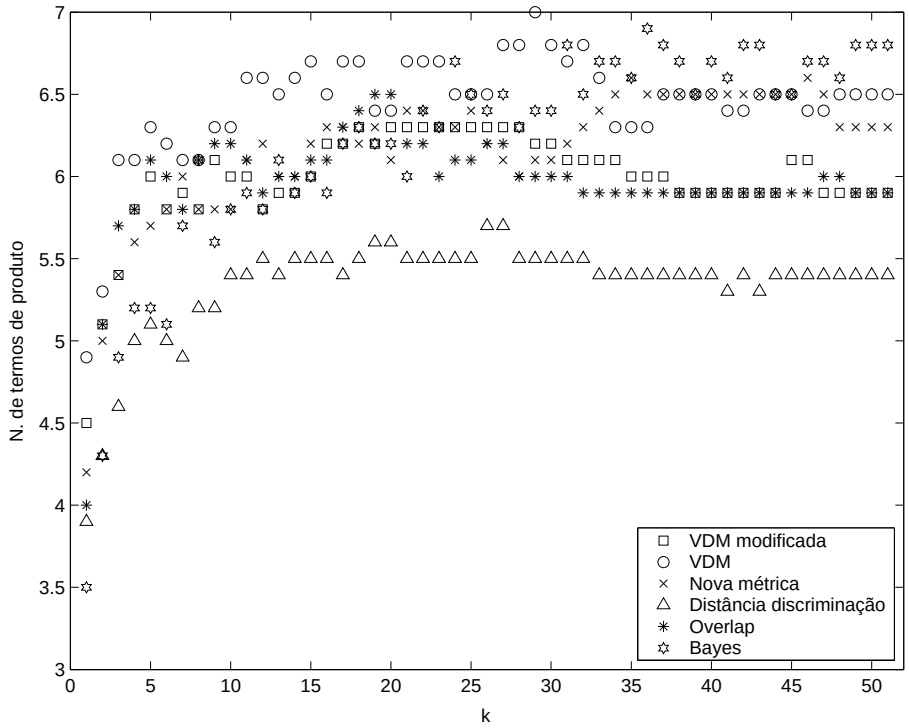


Figura 6.21: Número médio de termos de produto obtidos, para cada valor de k, na solução do problema do coração utilizando o método proposto (RSR).

6.1.5 Classificação de Cogumelos

Foram obtidas as funções lógicas dos circuitos digitais classificadores utilizando o método proposto com os dados do problema de classificação de cogumelos descrito em capítulo anterior (Figura 4.13). O processo foi repetido 10 vezes para cada método, sendo que a cada processo os dados eram permutados antes de serem divididos em conjunto de treinamento, teste e validação. Para o método proposto (RSR), são apresentados os resultados nas Figuras 6.22 a 6.26, onde k representa a quantidade de vizinhos mais próximos de cada classe para cada amostra do subconjunto consistente reduzido.

O gráfico da Figura 6.22 indica o número médio de amostras selecionadas para compor o subconjunto restante reduzido para cada valor de k, em cada métrica utilizada. Observa-se que a medida em que o parâmetro k cresce, a quantidade de amostras selecionadas também aumenta, como já esperado. A métrica *overlap* apresentou uma taxa maior de crescimento de amostras em função de k. A métrica de Bayes apresentou um crescimento assintótico menor do número de amostras em função de k. Este subconjunto de dados selecionados são utilizados pelo Espresso para obtenção da função Booleana do circuito digital classificador.

O gráfico da Figura 6.23 apresenta o número médio de acertos de classificação dos dados de teste pelo circuito digital simulado tomando como base de projeto cada subconjunto restante reduzido obtido com cada valor de k, para

cada um dos 10 experimentos e para cada uma das 6 métricas testadas. O melhor resultado é obtido para $k = 140$ e utilizando-se a métrica *overlap*. O pior desempenho do circuito foi obtido utilizando-se a métrica de Bayes. Interessante comparar com o resultado obtido em capítulo anterior utilizando o classificador 1-NN com RSR (Figura 5.15). A métrica de Bayes também apresentou o pior resultado.

Os gráficos das Figuras 6.24 e 6.25 apresentam os índices de acertos dos circuitos classificadores simulados para os dados de teste em função do número de termos de produtos e quantidade de amostras utilizadas, respectivamente. Observa-se novamente a superioridade da métrica *overlap* no índice de acertos do circuito mostrado nos dois gráficos.

A Figura 6.26 apresenta o número médio de termos de produto obtidos em função do parâmetro k. A métrica de Bayes, apresentou o menor índice assintótico médio de termos de produto em função de k, entretanto, isto não foi capaz de garantir um bom desempenho para o circuito classificador obtido com esta métrica.

Na Tabela 6.5 são mostrados os resultados obtidos com os circuitos digitais simulados e projetados a partir dos diferentes métodos apresentados, para a solução do problema de classificação dos 10 conjuntos de dados do problema do cogumelo. São mostrados a quantidade de amostras utilizadas pelo Espresso para obtenção da função Booleana do circuito digital classificador, o tempo de processamento do Espresso em cada método, o número de termos de produto da função Booleana de cada circuito obtido, e a porcentagem de acertos dos dados de teste pelo circuito simulado. Os valores apresentados estão no formato de média e desvio padrão.

Tabela 6.5: Resultados obtidos no problema do cogumelo para cada método.

Método	**Nº amostras**	**T. Espresso [s]**	**Nº prod.**	**Acertos (%)**
Sem filtragem	6500 (100%)	37±1	2±0	100±0
RSR (k=140,overlap)	3207±112(49,3±1,7%)	14±3	2±0	100±0
MLP (64 nodos)	6500 (100%)	185±6	2±0	100±0
SVM (rbf,w=1,c=1)	6500 (100%)	309±23	2±0	100±0

Novamente observa-se que o método proposto (RSR) possui resultados de acertos de classificação idênticos aos outros métodos utilizando menos amostras e menos tempo de processamento para o Espresso.

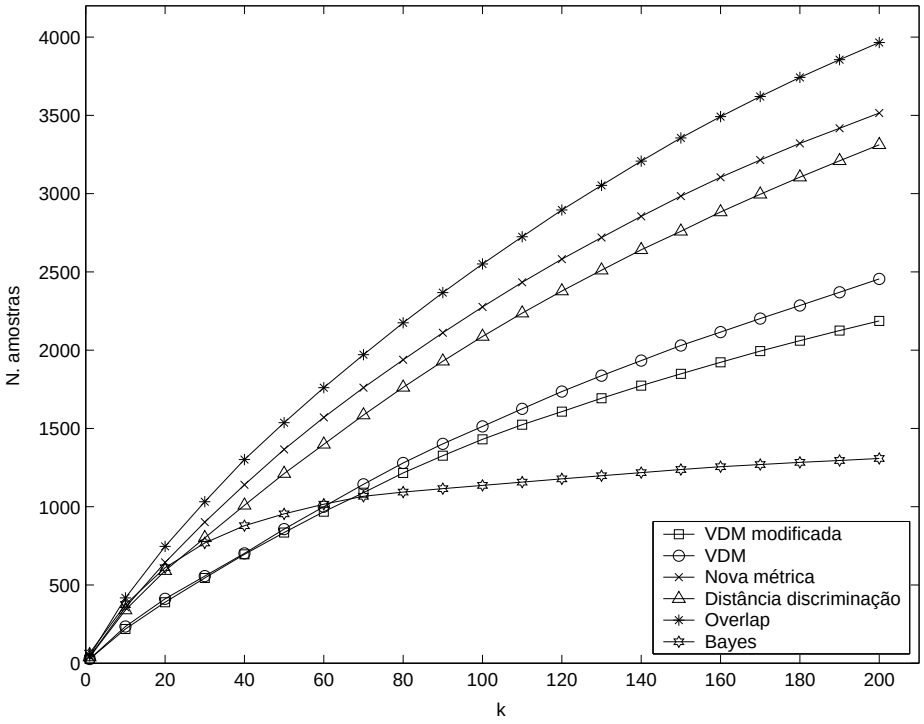


Figura 6.22: Quantidade média de amostras selecionadas para obtenção da função Booleana do circuito digital classificador, para cada valor de k, na solução do problema do cogumelo utilizando o método proposto (RSR).

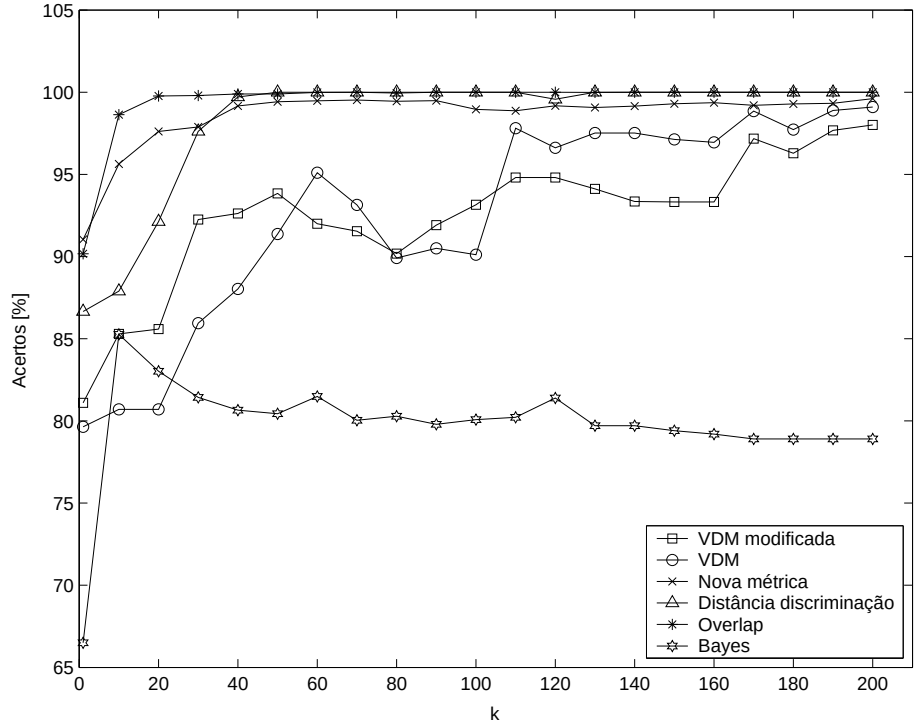


Figura 6.23: Acertos de classificação para os dados de teste em cada circuito digital simulado, para cada valor de k, na solução do problema do cogumelo utilizando o método proposto (RSR).

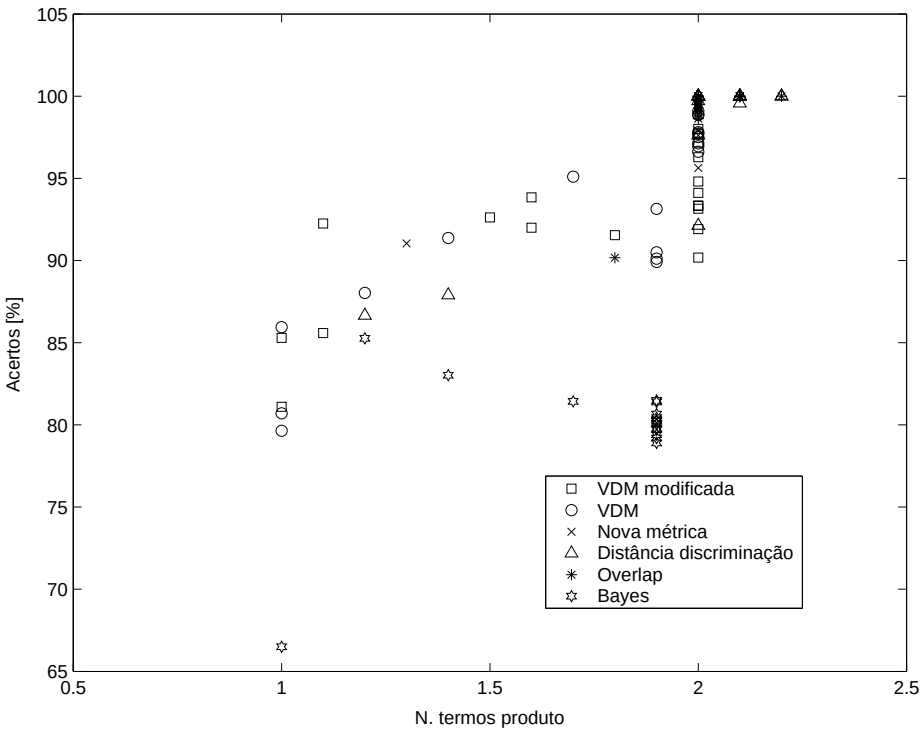


Figura 6.24: Acertos de classificação para os dados de teste em cada circuito digital simulado, para cada valor de termos de produto, na solução do problema do cogumelo utilizando o método proposto (RSR).

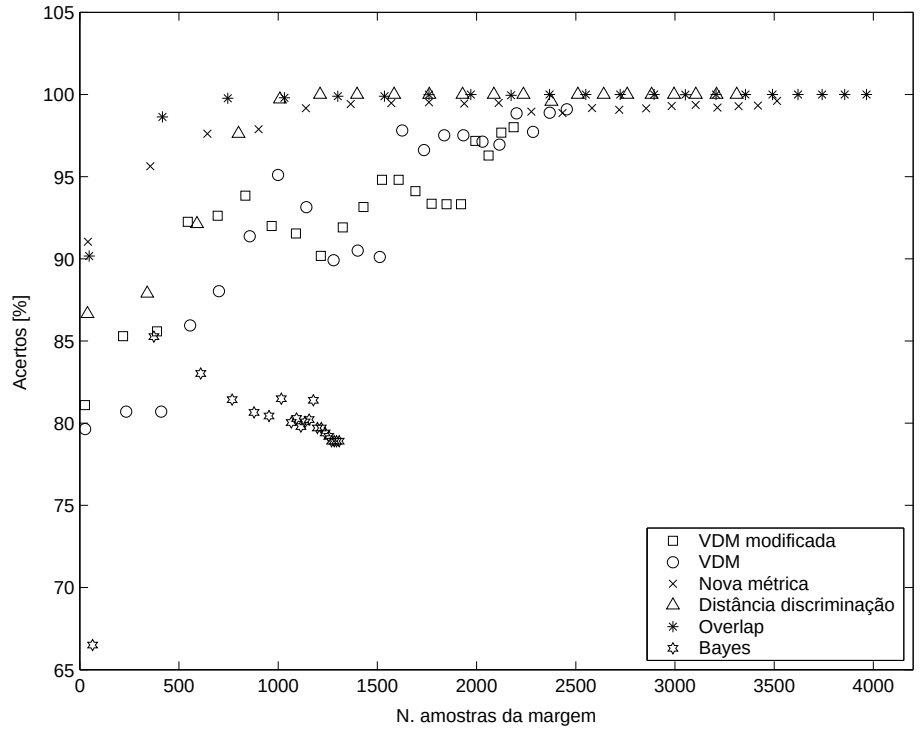


Figura 6.25: Acertos de classificação para os dados de teste em cada circuito digital simulado, para cada quantidade de amostras selecionadas, na solução do problema do cogumelo utilizando o método proposto (RSR).

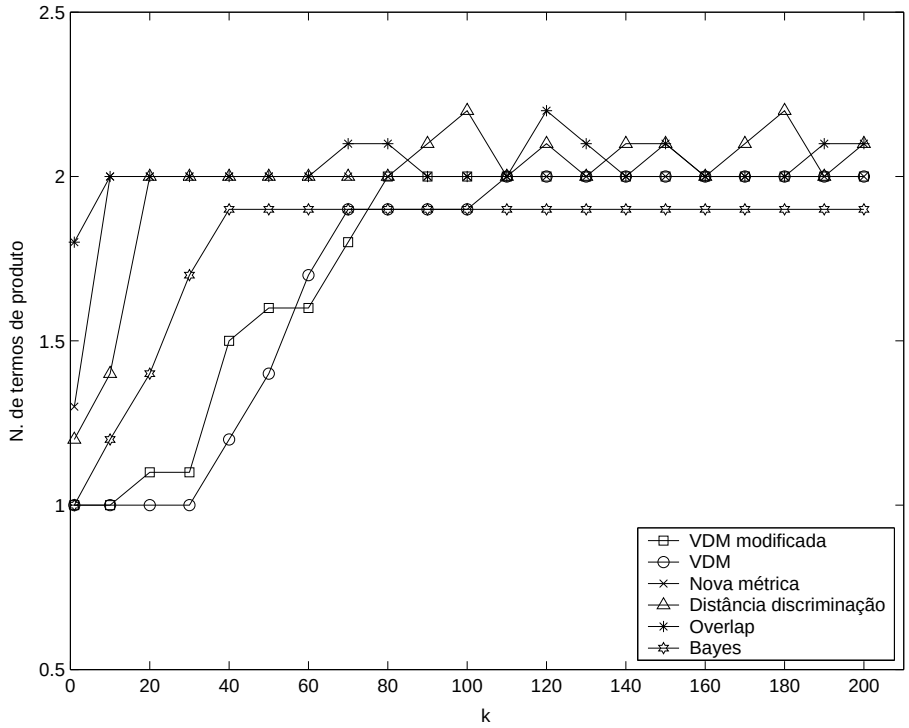


Figura 6.26: Número médio de termos de produto obtidos, para cada valor de k, na solução do problema do cogumelo utilizando o método proposto (RSR).

6.2 Conclusão

O circuito classificador digital projetado por meio dos dados selecionados pelo método RSR proposto apresentou desempenho simulado de classificação semelhante aos melhores métodos de classificação utilizados na atualidade para obter o correspondente circuito, porém com menos dados de treinamento. Alguns testes foram implementados utilizando dados artificiais e dados reais, cujos resultados comprovam as observações. Além disto, a utilização do método proposto exigiu menos esforço computacional para o minimizador de função Booleana (o Espresso) devido ao menor número de amostras de treinamento.

O método proposto possui um custo computacional alto, pois é derivado da Regra do Vizinho-mais-próximo. A complexidade é devida principalmente ao cálculo das distâncias entre todas as amostras de treinamento para escolha dos vizinhos mais próximos. Além disto, o método utiliza um processo de ordenação dos valores das distâncias calculadas (foi utilizado o *Quicksort*) para a tomada de decisão. A complexidade do método também é aumentada por causa das 3 diferentes etapas a serem executadas em seqüência, utilizando iteração para cada etapa. Apesar disto, o gargalo maior de processamento para determinação do circuito digital está justamente no minimizador de função Booleana (Espresso).

Não foi indicado o tempo de processamento dos métodos utilizados nos experimentos pelos seguintes motivos: o algoritmo implementado para o método proposto ainda não está otimizado para efeito de comparação; utilizou-se diferentes plataformas (C++, Matlab) para implementação dos métodos; o tempo maior de processamento, em geral, é devido a execução do Espresso (exceto no método de Rede Neural com algoritmo de treinamento MOBJ). Assim, foi apresentado nos resultados dos experimentos apenas o tempo de processamento do Espresso, por ser o mesmo programa utilizado em conjunto com cada método.

CAPÍTULO 7

Implementação do Circuito Classificador

Para determinadas aplicações, um classificador de padrões binários necessita ser implementado em circuito digital de modo a ter características de portabilidade, autonomia e execução em tempo real. Na literatura especializada encontram-se alguns exemplos de implementação em *hardware* de circuitos classificadores de padrões binários como em (Tzionas, Tsalides, & Thanailakis 1992; Tzionas, Tsalides, & Thanailakis 1994; Lipman & Yang 1997; Ferrari, Borgatti, & Guerrieri 2000; Silva, Lacerda, & Braga 2004).

Após a realização dos testes do método de projeto proposto nesta tese implementados em *software* e executados em um sistema de computador tipo PC (monoprocessado), decidiu-se fazer a implementação do sistema do classificador de padrões binários em *hardware* reconfigurável. O principal objetivo desta etapa da tese foi comprovar a possibilidade de implementar a proposta em um sistema embutido (vide (Wolf 2001)), o que o tornaria viável para as mais diversas aplicações, principalmente em problemas que exigem solução com sistemas portáteis e com processamento em tempo real, conforme (Wilmshurst 2001).

A seguir será dada uma explanação geral sobre sistemas embutidos, evidenciando as suas principais características de projeto. Em seguida, será apresentada a arquitetura escolhida para implementação do sistema proposto baseado no processador NIOS II (Altera Corporation 2003b) em FPGA. Os detalhes de *hardware* e *software* necessários para implementação do sistema proposto usando a arquitetura escolhida são mostrados em conjunto com al-

guns resultados de operação do sistema.

7.1 Projeto de Sistemas embutidos

Sistemas embutidos são sistemas que geralmente fazem interface direta com o ambiente, reagindo com o mesmo, ou aparecem como parte de um sistema computacional implementando aplicações específicas (Etrusco 2003). Sistemas embutidos são geralmente construídos agrupando-se processadores convencionais e circuitos dedicados de aplicação específica, denominados ASIC's (*Application Specific Integrated Circuits*), além de um sistema associado de memória e interfaces com o ambiente.

ASIC's são dispositivos de *hardware* dedicados e possuem alto desempenho, sendo geralmente destinados a partes complexas das aplicações, porém implementam apenas funções fixas determinadas durante a fabricação. Por outro lado, o uso de processadores convencionais em sistemas embutidos fornece uma interface de desenvolvimento flexível através de programação via *software* .

Os sistemas embutidos se caracterizam por restrições de projeto e das próprias funções implementadas pelos mesmos pela necessidade de interação direta com o ambiente (vide (Vahid & Givargis 2002)). As restrições de projeto envolvem processamento em tempo real e capacidade de controle simultâneo de fluxo de dados em taxas de ocorrência diferentes (vide (Wilmshurst 2001)).

O desenvolvimento de sistemas embutidos é baseado principalmente na divisão de seus componentes de *hardware* e componentes de *software*. Estes componentes são implementados utilizando-se diversas tecnologias como:

- ASIC's;
- processadores de uso geral;
- combinações híbridas de processadores de uso geral e ASIC's;
- *System on a chip*.

A seguir são apresentados alguns conceitos sobre projeto e implementação de sistemas embutidos.

7.1.1 Particionamento hardware/software

Uma abordagem tradicional para o projeto de sistemas embutidos é o particionamento entre *hardware* e *software* . O *hardware* é implementado através de ASIC's e o *software* é implementado através de processadores de uso geral,

microcontroladores ou DSP's (*Digital Signal Processor*). As partes que necessitam de desempenho são implementadas em *hardware*, e partes que necessitam de flexibilidade são implantadas em *software*.

A principal métrica que define a necessidade de um componente do sistema ser implementado em *hardware* é o desempenho requerido segundo (vahid2002). As tarefas que lidam diretamente com processamento em tempo real são implementadas em *hardware*. A métrica utilizada para definir a parte a ser implementada em *software* é a flexibilidade. O custo do sistema é utilizado como métrica global.

7.1.2 Sistema reconfigurável

Sistemas reconfiguráveis são construídos através de utilização de dispositivos de *hardware* programáveis tais como as FPGA's (*Field Programmable Gate Arrays*) e configurados com a utilização de ferramentas de projeto. Os sistemas reconfiguráveis podem ser considerados uma alternativa aos sistemas compostos de processadores convencionais e ASIC's pois agregam tanto a flexibilidade dos primeiros quanto o desempenho dos últimos. A partir destes dispositivos é possível desenvolver sistemas com características de desempenho de ASIC's e a flexibilidade de processadores de uso geral. Esta combinação de flexibilidade e desempenho coloca as arquiteturas reconfiguráveis como uma importante área no desenvolvimento de sistemas computacionais.

A computação reconfigurável traz modificações no particionamento *hardware/software* uma vez que a mesma se encontra em um ponto intermediário onde temos o desempenho de *hardware* associado à flexibilidade do *software* . Além disto, toda a implementação ocorre estritamente em *hardware*, ou seja, não há efetivamente um particionamento entre *hardware* e *software*.

Em sistemas implementados espacialmente como os sistemas reconfiguráveis temos um modelo de computação diferente que ao invés de realizar operações em instantes de tempo, seqüencialmente, estabelece a alocação de cada operação a uma área diferente do dispositivo programável e, desta forma, permite a execução de quantas operações possam ser alocadas no dispositivo, em paralelo. A grande vantagem de computação espacial é a possibilidade de se explorar todo o paralelismo intrínseco a um determinado algoritmo desde que se tenha recursos de *hardware* suficientes para tanto.

7.1.3 System on a chip (SoC)

System on a chip, conforme (Berger 2002), é uma solução para integração de sistemas em um único circuito integrado. Estas soluções são divididas por área de aplicação e em blocos de propriedade intelectual implementados em

hardware comumente chamados *cores*.

A FPGA permite a integração de sistemas complexos em um único *chip*, agregando flexibilidade à implementação baseada em SoCs. As soluções de integração são divididas em duas plataformas diferentes:

- um processador convencional integrado à parte lógica programável em dispositivos específicos;
- processadores implementados na forma de *softcores* que podem ser instanciados em diversas famílias de FPGA's.

Os periféricos podem ser conectados aos processadores através de barramentos disponíveis nos dispositivos. Na seção seguinte é apresentado um processador disponibilizado em forma de *softcore* para aplicações de uso geral.

7.2 Processador NIOS II

O processador NIOS II é um *core* de processador RISC de propósito geral (Altera Corporation 2003a), contendo:

- conjunto de instruções, dados e espaço de endereçamento de 32 bits;
- 32 registradores de propósito geral;
- 32 fontes de interrupção externa;
- instruções simples de multiplicação e divisão em 32 bits;
- instruções dedicadas para processamento de produtos de multiplicação de 64 bits e 128 bits;
- instruções simples de deslocamento;
- acesso a uma variedade de periféricos *on-chip*, e interfaces para memória e periféricos *off-chip*;
- módulo de depuração por hardware assistido, habilitando o processador a iniciar, parar, e executar passo a passo sob controle de um ambiente de desenvolvimento;
- ambiente de desenvolvimento de *software* baseado na ferramenta GNU C/C++ e Eclipse IDE;
- desempenho em torno de 150DMIPS (*Dhrystone Millions of Instruction per Second*).

O processador NIOS II é um processador com núcleo configurável por *software* . Configurável significa que características podem ser somadas ou removidas. *Softcore* significa que o núcleo da CPU é disponibilizado em forma de projeto de *software* , e pode ser implementado em alguma família de FPGA. É o usuário quem configura o processador NIOS II e periféricos segundo suas necessidades e então programa o sistema dentro de uma FPGA.

Um sistema de processador NIOS é equivalente a um microcontrolador em "computador *on-chip*" que inclui uma CPU e uma combinação de periféricos e memória em um simples *chip*. A Figura 7.1 apresenta um exemplo de um sistema que faz uso do NIOS II.

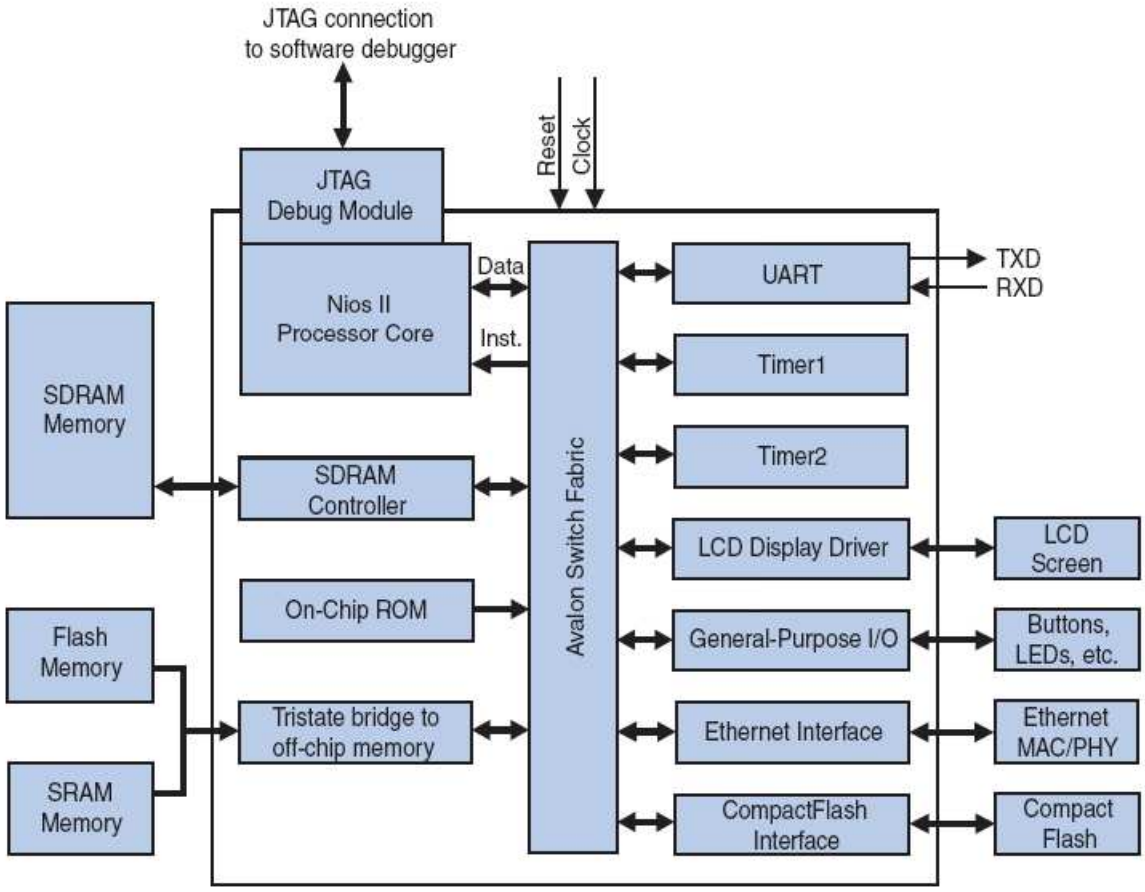


Figura 7.1: Exemplo de sistema com processador NIOS II.
FONTE: Altera Corporation

Projetos com processadores *softcore* permitem configurar o *hardware* do processador baseado em necessidades de projeto do sistema (Altera Corporation 2003c; Altera Corporation 2003b). Algumas das características disponíveis que podem ser usadas para incrementar o desempenho do sistema NIOS são:

- A CPU NIOS de 32 bits oferece duas instruções de multiplicação aceleradas por *hardware* (MSTEP, MUL) o qual aumentam até dez vezes o desempenho da implementação apenas em *software* .

- O processador NIOS II pode ser configurado para usar a memória *on-chip* com cache de instruções e dados, o qual pode aumentar o desempenho do sistema.

- É possível acrescentar funções personalizadas ao NIOS II para incrementar o seu desempenho.

- Um processador NIOS II pode ser criado com qualquer combinação de periféricos, interfaces, memória ou CPU's. Geralmente é mais eficiente duplicar recursos de *hardware* do que fazer *software* compartilhar um simples recurso. O único fator limitante para o número de CPU's, periféricos e interfaces de memória em um sistema é a quantidade de elementos lógicos disponíveis dentro da FPGA.

O projetista pode personalizar o sistema com processador NIOS II até chegar ao custo e desempenho desejados. FPGA's do fabricante Altera[1] providenciam uma flexibilidade para acrescentar características e incrementar desempenho ao sistema do processador. De outro modo, características não necessárias ao processador e periféricos podem ser eliminados para ajustar o projeto em um dispositivo menor e mais barato. Muitas personalizações são possíveis devido aos pinos e recursos lógicos dos dispositivos FPGA serem programáveis. Em geral, é possível utilizar:

- pinos extras e recursos lógicos no *chip* para serem usados em funções não relacionadas ao processador. Um sistema de processador NIOS II consome em torno de 5% de uma FPGA de alta densidade, deixando o resto dos recursos do *chip* disponíveis para implementar outras funções.

- pinos extras e lógica no *chip* para implementar periféricos adicionais ao sistema com processador NIOS II. Altera disponibiliza uma variedade de bibliotecas de periféricos que podem ser facilmente conectados ao sistema de processamento do NIOS II.

Altera disponibiliza um conjunto de periféricos comumente utilizados em microcontroladores, tais como timers, interfaces de comunicação serial, portas gerais de E/S, controladores de SDRAM e outras interfaces de memória. Projetistas podem também criar seus próprios periféricos e integrá-los ao sistema de processamento NIOS II. Para sistemas de desempenho crítico que gastam muitos ciclos de CPU executando uma seção específica do código, é uma técnica comum criar um periférico personalizado que implementa a mesma

[1]Altera, Quartus, NIOS IDE, SOPC Builder, Cyclone, Stratix são marcas registradas da Altera Corporation

função em hardware. Esta metodologia oferece um duplo benefício em desempenho: a implementação em *hardware* é mais rápida que em *software* , e o processador está livre para executar outras funções em paralelo enquanto o periférico personalizado processa os dados.

Como periféricos personalizados, instruções personalizadas são um método para incrementar o desempenho do sistema pela melhoria do processador com *hardware* personalizado. A natureza de *core* em *software* do processador NIOS II permite aos projetistas integrar lógica personalizada à unidade lógica aritmética (ULA). Similar as instruções nativas do NIOS II, instruções personalizadas podem formar valores de até 2 registradores fonte e opcionalmente salvar o resultado em um registrador destino.

Do ponto de vista de *software*, instruções personalizadas aparecem como macros em assembly ou funções em linguagem C. Assim programadores não necessitam conhecer assembly para usar instruções personalizadas.

Nas próximas subseções, são apresentados alguns detalhes de hardware do processador NIOS II e das ferramentas disponíveis para desenvolvimento de sistemas. Também são descritas as características de sistemas multiprocessados.

7.2.1 Arquitetura do processador NIOS II

O *core* do processador inclui apenas os circuitos necessários para implementar a arquitetura do NIOS II. Na Figura 7.2 são apresentados os principais componentes internos do processador NIOS II. A arquitetura do NIOS II define as seguintes unidades funcionais visíveis ao usuário:

- registradores
- ULA
- interfaces para instruções lógicas personalizadas
- controlador de exceção
- controlador de interrupção
- barramento de instruções
- barramento de dados
- memória cache de instruções e dados
- interfaces de memória acopladas para instruções e dados
- módulo de depuração JTAG

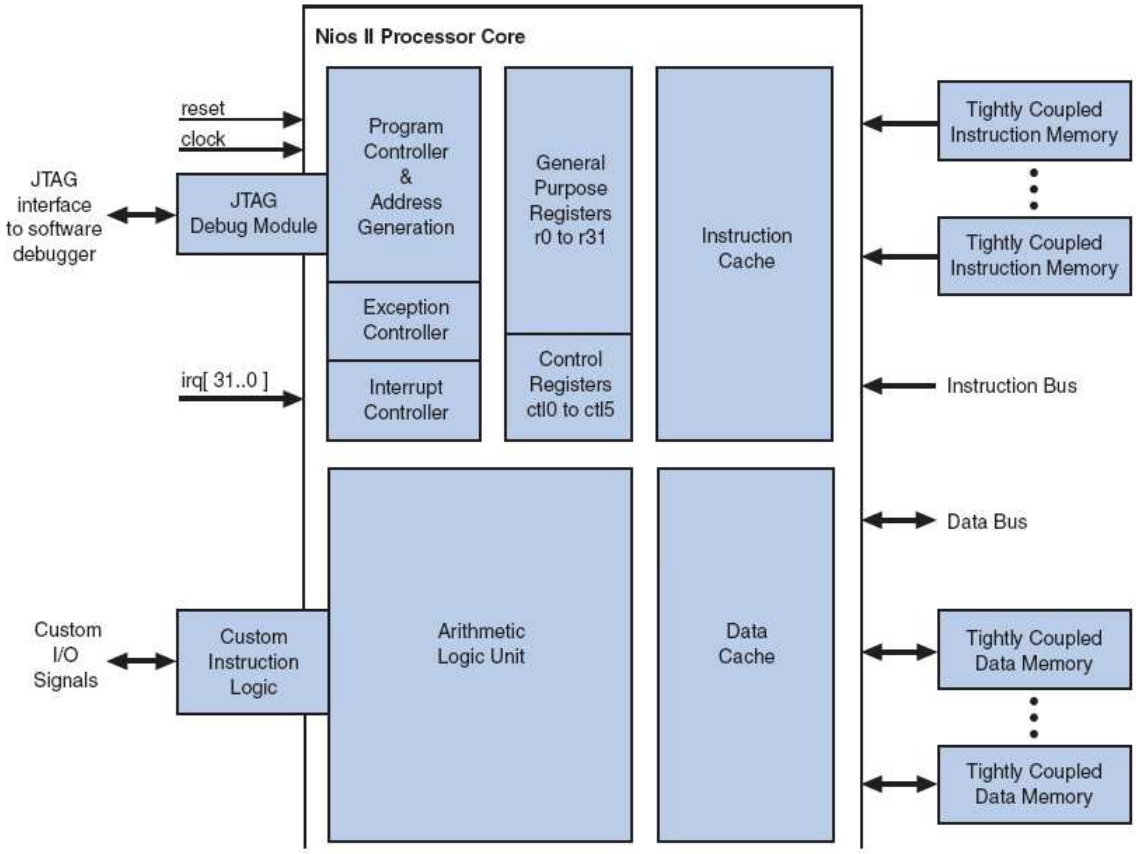


Figura 7.2: Estrutura interna do processador NIOS II.
FONTE: Altera Corporation

A arquitetura NIOS II suporta barramento de instruções e dados separados, classificando-o como uma arquitetura Harvard. A arquitetura do NIOS II disponibiliza acesso E/S mapeado em memória. Dados de memória e periféricos são mapeados em espaço de endereço. A arquitetura do NIOS II é *litlle endian*, isto é, palavras e meias palavras são armazenados em memória com os bytes mais significativos no endereço mais alto.

Tipicamente um sistema de processamento com NIOS II contém uma mistura de memória *on-chip* rápida e memória *off-chip* mais lenta. Periféricos tipicamente residem *on-chip*, embora interfaces para periféricos *off-chip* também existem.

A arquitetura NIOS II suporta memórias cache para instruções e dados. Memória cache reside *on-chip* como uma parte do *core* do processador NIOS II. O *core* do processador NIOS II pode incluir um, ambos, ou nenhuma das memórias cache.

A arquitetura NIOS II suporta um módulo de depuração JTAG que disponibiliza emulação *on-chip* para controlar o processador remotamente de um *host*. Ferramentas de *software* de depuração no *host* comunicam com o módulo JTAG e disponibilizam facilidades tais como:

- *download* de programas para a memória;
- início e término de execução;
- configuração de *breakpoints* e *watchpoints*;
- análise de registradores e memória;
- coleta de dados em tempo de execução.

O processador NIOS II possui ainda 3 modos de operação:

- Modo supervisor: todas funções do processador são disponíveis e irrestritas. O processador está em modo supervisor assim que é "*resetado*".
- Modo usuário: providencia um subconjunto restrito de funções do modo supervisor.
- Modo debug: é usado por ferramentas de depuração para implementar características tais como *breakpoints* e *watchpoints*. Código do sistema e código de aplicação nunca executam em modo debug.

A arquitetura do NIOS suporta os seguintes modos de endereçamento:

- endereçamento de registrador;

- endereçamento indexado;
- endereçamento imediato;
- endereçamento indireto;
- endereçamento absoluto.

O formato das palavras de instruções do NIOS II estão em três categorias:

- tipo I: contém um valor imediato embutido dentro da palavra de instrução;
- tipo R: todos argumentos e resultados são especificados como registradores;
- tipo J: implementa a instrução CALL.

7.2.2 Tipos de cores do NIOS II

Altera disponibiliza 3 tipos de núcleos para o NIOS II:

- NIOS II/f - o *core* "*fast*" é projetado para desempenho rápido;
- NIOS II/s - o *core* "*standard*" é projetado para tamanho pequeno enquanto mantêm o desempenho;
- NIOS II/e - o *core* "*economic*" é projetado para alcançar o menor tamanho possível.

Todos *cores* NIOS II suportam as seguintes famílias de FPGA da Altera:

- Stratix
- Stratix II
- Cyclone
- Cyclone II

A seguir é detalhado 2 tipos de *core*: NIOS II/f e NIOS II/e. O NIOS II/s possui características intermediárias entre os dois *cores* anteriores, sem degradar substancialmente o seu desempenho.

core NIOS II/f

O tipo de *core* NIOS II/f possui as seguintes características:

- Possui caches separadas para instruções e dados;
- Pode acessar até 2GBytes de espaço de endereço externo;
- Opção de suportar memória fortemente acoplada para instruções e dados;
- Emprega 6 estágios de pipeline;
- Multiplicação, divisão e deslocamento em hardware;
- Suporta a adição de instruções personalizadas;
- Suporta módulo de depuração JTAG.

core NIOS II/e

O objetivo do projeto no *core* NIOS II/e é reduzir recursos de utilização o máximo quanto possível, enquanto mantém compatibilidade com a arquitetura do conjunto de instruções do NIOS II. O *core* do NIOS II/e é quase metade do tamanho do *core* NIOS II/s, mas o desempenho de execução é substancialmente menor.

7.2.3 Memória cache

O *core* do processador NIOS II contém cache de dados e instruções (Altera Corporation 2005d). Em todos *cores* atuais do NIOS II não há um mecanismo de coerência de cache em hardware. Desta forma, se há dispositivos acessando memória compartilhada, o *software* deve explicitamente manter a coerência através de todos os dispositivos.

O conteúdo do cache de dados de todos processadores que acessam a memória compartilhada deve ser gerenciado pelo *software* para garantir que todos os processadores lêem os mesmos valores recentes e não sobreescrevam novos dados com dados ultrapassados. Este mecanismo é feito atualizando os dados do cache e utilizando facilidades de “bypassar” a memória cache para mover dados entre a memória compartilhada e os dados de cache necessários.

A biblioteca de funções C do NIOS II disponibiliza algumas funções para acesso de dados da memória principal “bypassando” a cache.

7.2.4 Instruções personalizadas

Com o processador NIOS II embutido da Altera, projetistas de sistemas podem acelerar algoritmos em *software* de tempo crítico adicionando instruções personalizadas ao conjunto de instruções do NIOS II (Altera Corporation 2005b). Com instruções personalizadas, projetistas de sistemas podem reduzir uma máquina complexa de instruções padrões a uma simples instrução implementada em hardware.

Com o processador NIOS II de instruções personalizadas, projetistas de sistemas são capazes de levar vantagem da flexibilidade de FPGA's para encontrar necessidades de desempenho de sistemas. Isto dá ao projetista de sistemas a habilidade para modificar o *core* do processador NIOS II para atender as necessidades de uma aplicação particular. A Figura 7.3 apresenta o modo de inserção de um *hardware* à ULA do NIOS II de forma a criar uma instrução personalizada.

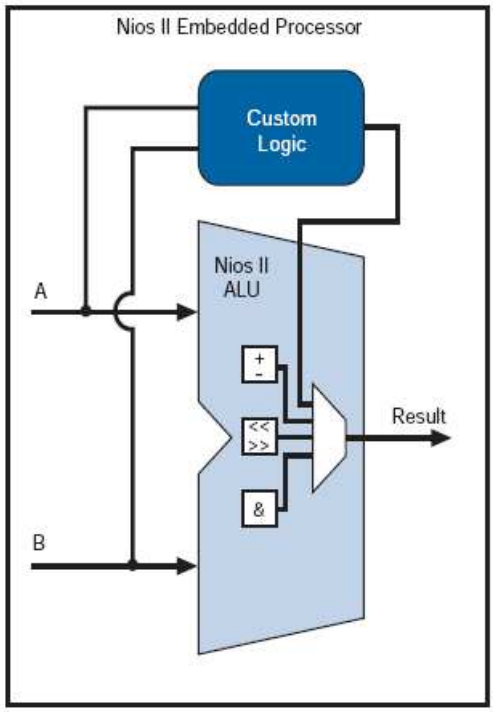


Figura 7.3: Instrução personalizada no processador NIOS II.
FONTE: Altera Corporation

Projetistas de sistemas tem a habilidade para acelerar tempo crítico de *software* pela conversão do algoritmo em blocos lógicos de hardware. Como é fácil alterar o projeto de um processador NIOS II baseado em FPGA, instruções personalizadas providenciam um modo fácil para particionar projetos em *hardware*/*software* durante a implementação de um sistema embutido.

Para cada instrução personalizada, o ambiente de desenvolvimento integrado do NIOS II (IDE) produz uma macro que é definida no cabeçalho do pro-

grama do sistema. Assim é possível chamar uma macro de uma aplicação em código de linguagem C como uma chamada normal à função e não é necessário programar em assembly para acessar as instruções personalizadas.

As macros definidas pelo NIOS II IDE fazem uso de tipos inteiros em C. As instruções personalizadas do processador NIOS II permitem a definição de macros personalizadas que permitem tipos de 32 bits de entrada para interfacear com instruções personalizadas.

O NIOS II usa funções embutidas do compilador gcc para mapear instruções personalizadas. Usando funções embutidas permitem a tipos além de inteiros para serem usados com instruções personalizadas.

7.2.5 Ambiente de desenvolvimento

A pressão crescente para entrega de produtos robustos no mercado em um tempo hábil tem aumentado a importância da verificação de projetos de processadores embutidos (Altera Corporation 2004). Projeto com o processador embutido NIOS II suporta uma ampla faixa de soluções de verificação incluindo:

- NIOS II IDE (*Integrated Development Environment*) pode ser usado para verificar projetos rodando em placas em desenvolvimento ou personalizadas usando seu *debugger* incluso.

- ISS (*Instruction Set Simulator*) é usado para modelar o conjunto de instruções do processador NIOS II em um *software* baseado em modelo de simulação. Isto permite projetistas rodarem a imagem executável de seus projetos de *software* no ISS e depurar o *software* usando o NIOS II IDE Debugger. O ISS é principalmente utilizado quando uma placa de desenvolvimento não está disponível.

- *Register Transfer Level* (RTL) *Simulation*. RTL habilita projetistas para observar qualquer registrador e sinal dentro do projeto. Sistemas baseados em NIOS II podem facilmente serem simulados em ModelSim[2] usando um ambiente automaticamente criado pelo SOPC Builder e NIOS II IDE.

- ModelSim é uma ferramenta de simulação e depuração para VHDL (*Very High Speed Integrated Circuit Hardware Description Language*), VERILOG e projetos de linguagens mixadas.

Simulação e verificação são partes vitais do processo de projeto. O processador NIOS II pode ser verificado compreensivamente usando depuração, emulação por *software* usando NIOS II ISS e simulação RTL usando ModelSim.

[2]ModelSim é marca registrada da Mentor Graphics Corporation

Simulação RTL é uma parte importante do processo de projeto para sistemas configuráveis uma vez que ele permite observar sinais embutidos dentro do processador e seus conjuntos de periféricos.

A Figura 7.4 apresenta as ferramentas utilizadas para o projeto, compilação, simulação e configuração da FPGA da Altera com um sistema de processador NIOS II. A seguir são detalhadas algumas destas ferramentas de *software* .

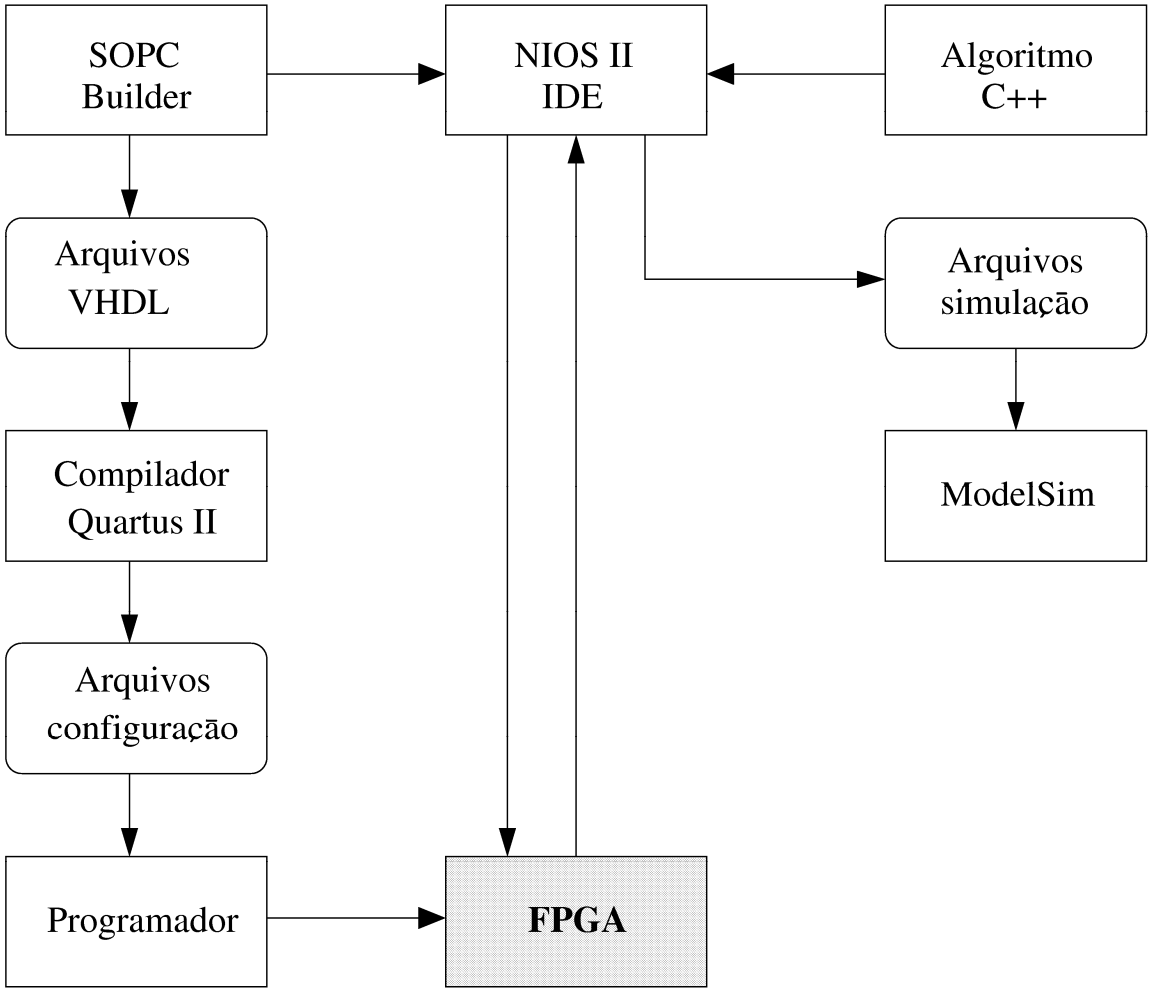


Figura 7.4: Sistema de desenvolvimento para o processador NIOS II.

Quartus II

O *software* de projeto Quartus II da Altera disponibiliza um ambiente de projeto completo e multiplataforma que facilmente adapta as necessidades do projetista (Altera Corporation 2005e). O *software* inclui soluções para todas as fases de projeto de FPGA da Altera (Figura 7.5).

O *software* QuartusII permite o uso de interface gráfica de projeto, interface para ferramenta de análise, e interface em linha de comando para cada fase do projeto.

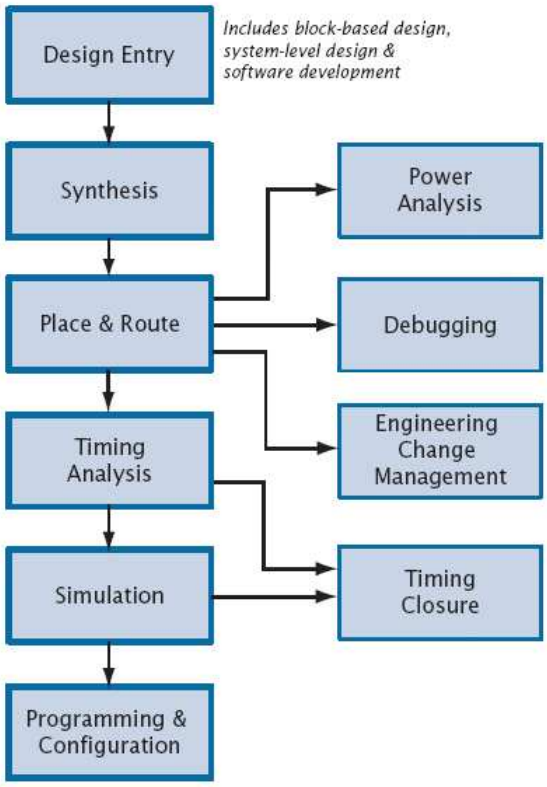


Figura 7.5: Fases de projeto de um sistema em FPGA.
FONTE: Altera Corporation

SOPC Builder

SOPC Builder (*System On a Programmable Chip*) é uma ferramenta de *software* que permite ao usuário criar um microcontrolador embutido totalmente funcional e personalizado chamado módulo do sistema NIOS II. Um sistema completo NIOS II contém um processador embutido NIOS II e seus sistemas de periféricos associados. SOPC Builder permite ao usuário criar facilmente e rapidamente um módulo de sistema *multi-master* (*master*, *slave*, lógica de árbitro de barramento, etc.) que é conectado e pronto uso em FPGA's da Altera.

NIOS II IDE

O NIOS II IDE (*Integrated Development Environment*) é um *software* com interface gráfica para desenvolvimento de programas em C/C++ para o processador NIOS II. Todo desenvolvimento das tarefas de programas podem ser acompanhadas de dentro do NIOS II IDE, incluindo edição, compilação e depuração de programas. O NIOS II IDE é uma janela através da qual todas outras ferramentas podem ser acessadas.

O NIOS II IDE é baseado na popular ferramenta Eclipse IDE e no kit de ferramenta de desenvolvimento Eclipse C. O NIOS II IDE é uma interface do usuário que manipula outras ferramentas livrando o usuário dos detalhes de

ferramentas de linha de comando e apresenta um ambiente de desenvolvimento unificado.

ModelSim

ModelSim é uma ferramenta de simulação e depuração para VHDL, VERILOG e projetos de linguagens mixadas. As seguintes etapas são utilizadas para simular um projeto no ModelSim:

1. Criar uma biblioteca de trabalho;
2. Compilar arquivos de projeto;
3. Executar simulação;
4. Analisar resultados.

Em ModelSim, todos projetos, sejam eles em VHDL, VERILOG ou uma combinação dos dois são compilados em uma biblioteca. Uma nova simulação é sempre iniciada pela criação de uma biblioteca de trabalho, após a qual é realizada a compilação das unidades de projeto. Tendo o projeto compilado, o simulador é invocado em um módulo de alto nível (VERILOG) ou uma configuração em par de entidade/arquitetura (VHDL). Pela análise dos resultados obtidos da simulação, são feitas as correções no projeto até se obter os resultados desejados.

7.2.6 Multiprocessamento

Sistema multiprocessado é qualquer sistema o qual incorpora 2 ou mais microprocessadores trabalhando juntos para executar uma tarefa (Altera Corporation 2005a). O processador NIOS II da Altera e a ferramenta SOPC Builder permitem aos desenvolvedores de sistemas projetar e construir sistemas multiprocessados que compartilham recursos rapidamente. Usando o NIOS II IDE, o projetista pode criar e depurar 3 projetos de *software* , um para cada processador do sistema.

Sistemas multiprocessados possuem o benefício de incrementar o desempenho, quando comparado com sistemas monoprocessados, porém com aumento do custo geral. FPGA's providenciam uma plataforma ideal para desenvolvimento de sistemas multiprocessados assimétricos embutidos desde que o *hardware* possa ser facilmente modificado e ajustado usando a ferramenta SOPC Builder de modo a providenciar ótimo desempenho do sistema. Múltiplos processadores NIOS II são hábeis para compartilhar recursos de sistema graças a capacidade de arbitração.

NIOS II IDE e SOPC Builder providenciam um esquema simples de particionamento de memória que permite múltiplos processadores rodarem seu *software* em diferentes regiões separadas da mesma memória física. O NIOS II IDE providencia a cada processador sua própria seção dentro da memória na qual ele pode executar seu *software* .

NIOS II IDE inclui um número de características que podem ajudar no desenvolvimento de *software* para sistemas multiprocessados, incluindo a depuração simultânea de múltiplos processadores *on chip*. Múltiplas sessões de depuração podem rodar ao mesmo tempo em um sistema multiprocessado e podem parar e continuar cada processador independentemente. *Breakpoints* também podem ser configurados individualmente para cada processador.

Sistemas multiprocessados NIOS II podem ser divididos em 2 principais categorias:

1. aqueles que compartilham recursos;

2. aqueles que cada processador é autônomo e não compartilha recurso com outros. Estes múltiplos processadores não comunicam entre si, sendo impossível de interferirem com a operação de outro.

7.2.7 Compartilhamento de recursos

Recursos são considerados compartilhados quando eles são disponíveis para serem acessados por mais do que um processador. Recursos podem ser compartilhados simplesmente conectando eles no barramento principal dos múltiplos processadores na matriz de conexão do SOPC Builder. Mas só isto não garante que os processadores que compartilham os recursos trabalharão não destrutivamente. O *software* executando em cada processador é responsável por coordenar o acesso aos recursos compartilhados com os outros processadores do sistema.

O tipo mais comum de compartilhamento de recurso em sistemas multiprocessados é a memória. Memória compartilhada pode ser usada desde como simples "*flag*" cujo propósito é comunicar *status* entre processadores, até estruturas de dados complexas que são coletivamente acessadas por muitos processadores simultaneamente.

Se um componente de memória é para ser compartilhado para propósito de dados, sua porta de acesso necessita ser conectada ao barramento mestre dos processadores que estão compartilhando a memória. Se um processador está escrevendo em uma área particular da memória de dados compartilhada no mesmo instante que outro processador está lendo ou escrevendo naquela área, provavelmente ocorrerão corrupção de dados, causando no mínimo aplicação de erros, e possivelmente um "*crash*" no sistema.

Para prevenir que múltiplos processadores interfiram um com o outro, um *core* de *hardware* é incluído no sistema que permite a diferentes processadores obterem a posse de recursos compartilhados por um período de tempo. Isto previne que o recurso compartilhado se torne corrompido pela ação de outro processador.

O processador NIOS II permite a manipulação de proteção de recursos compartilhados com sua característica de *core* de hardware mutex (*mutual exclusion*). Este *core* é um pequeno componente do SOPC Builder. Um mutex permite que um dos processadores possa acessar exclusivamente o recurso de *hardware* do sistema. Isto é melhor detalhado na próxima seção.

7.2.8 Mutex

Ambientes multiprocessados podem usar o *core* mutex para coordenar acessos a dispositivos compartilhados. O *core* mutex providencia um protocolo para garantir propriedade mutuamente exclusiva de recurso compartilhado. O *core* mutex providencia uma operação atômica de teste e ativação baseados em hardware, permitindo ao *software* em um ambiente multiprocessado determinar qual processador apropria o mutex. O *core* mutex pode ser usado em conjunto com memória compartilhada para implementar características de coordenação interprocessadores. O *core* mutex é projetado para uso em sistemas processados baseados em Avalon, tais como um sistema de processador NIOS II. Altera disponibiliza *device drivers* para o processador NIOS II para habilitar o uso do *hardware* mutex.

O *core* mutex disponibiliza acesso a dois registradores de 32 bits mapeados em memória. Supondo que há múltiplos processadores acessando um *core* simples de mutex, e cada processador tem um único identificador (ID):

1. quando o campo valor é 0x0000, o mutex está disponível (isto é, destravado). Caso contrário, o mutex está indisponível (travado).

2. o registrador mutex está sempre legível. Um processador pode ler o registro mutex para determinar seu estado corrente.

3. o registrador mutex é escrito apenas sob condições específicas. Uma operação de escrita troca o registrador mutex apenas se uma ou ambas das seguintes condições são verdadeiras:

   (a) o campo valor do registrador mutex é zero;

   (b) o campo proprietário do registrador mutex coincide com o campo proprietário no dado para ser escrito.

4. um processador tenta adquirir o mutex escrevendo seu ID no campo proprietário, e escrevendo um valor não zero no campo valor. O processador então verifica se a aquisição sucedeu pela verificação do campo proprietário.

5. depois do reset do sistema, o bit de reset no registrador reset é ativo. Escrevendo um "1" neste bit limpa-o.

Há algumas funções da biblioteca C do NIOS II próprias para leitura e escrita do registrado mutex.

7.3 Sistema Classificador Proposto

A solução adotada para implementação em *hardware* do classificador de padrões binários proposto foi um sistema multiprocessado com NIOS II implementado em FPGA Cyclone II da Altera, disponível em forma de kit de desenvolvimento e programação em linguagem C/C++. Nas seções seguintes são apresentados os detalhes de *hardware* e *software* do sistema completo desenvolvido.

7.3.1 O hardware do sistema

Para implementação do sistema proposto, foi utilizado um kit de desenvolvimento da Altera: NIOS II Development Kit Cyclone II Edition (Altera Corporation 2005c). O principal componente do kit é a FPGA da Altera Cyclone II EP2C35 com sinal de *clock* de 50MHz, conforme pode ser visto na Figura 7.6 (*chip* U62).

Na Tabela 7.1 são apresentados os componentes principais do kit mostrados na Figura 7.6. Estes componentes estão disponíveis ao projetista para a implementação de um sistema NIOS II completo em FPGA com várias opções de periféricos e memória tanto *on-chip* quanto *off-chip*.

As características principais da FPGA Cyclone II EP2C35 presente no kit pode ser resumida na Tabela 7.2.

Na Figura 7.7 pode ser visto um diagrama simplificado com os componentes e conexões do kit de desenvolvimento. Na Figura 7.8 é apresentado um diagrama simplificado do *hardware* do sistema classificador de padrões binários proposto implementado no kit. O sistema proposto é composto por:

- 3 CPU's NIOS II implementadas na FPGA do kit junto com barramentos de comunicação e periféricos (contadores de tempo), não mostrados para simplificar a Figura 7.8;

Tabela 7.1: Descrição dos componentes do kit de desenvolvimento.

Sigla	**Descrição**
U62	Cyclone II FPGA EP2C35F672C6
SW0-SW3	Push-button para entrada de sinais pelo usuário
D0-D7	LEDs de saída da FPGA
U8,U9	2 LEDs de sete segmentos de saída da FPGA
U74	Memória SSRAM de 2 MBytes (synchronous SRAM)
U5	Memória Flash de 16 MBytes
LED7	Indicador de memória Flash com configuração
U63	Memória DDR SDRAM de 32 MBytes
U4 RJ1	Interface Ethernet MAC/PHY 10/100
RJ1	Conector RJ-45 para Ethernet
J19	Conector serial RS232 com buffers
PROTO1	Conector de expansão
PROTO2	Conector de expansão
CON3	Conector para expansão de memória CompactFlash
JH1, JH2	Conector de expansão PMC para um PCI mezzanine card
J25	Conector Mictor com acesso a 27 pinos E/S da FPGA
TP1-TP8	Pontos de teste com acesso a 8 pinos E/S da FPGA
J24	Conector JTAG com acesso a FPGA
J5	Conector JTAG com acesso ao controlador MAX
J27	Conector para o dispositivo de configuração serial
U3	Controlador de configuração Altera MAX EPM7256AE
U69	Dispositivo de configuração serial Altera EPCS64 para a FPGA
SW8	Push-button para resetar a CPU Nios II configurada na FPGA
SW9	Push-button para reconfigurar a FPGA com a configuração de fábrica
SW10	Push-button para resetar o kit
LED0-3,LED6	LEDs de status de configuração do kit
Y2	Oscilador de 50 MHz, sinal de clock para a FPGA
J4	Entrada de clock externo para a FPGA
J26	Entrada de energia para alimentação do kit, 16V DC não regulado
D34	Retificador em ponte para a alimentação do kit

Tabela 7.2: Descrição da FPGA EP2C35 do kit de desenvolvimento.

Característica	**Quantidade**
Elementos lógicos (LE)	33.216
Blocos de memória de 4kbits	105
Total de bits da RAM	483.840
Blocos multiplicadores embutidos 18x18	35
PLLs	4
Pinos de E/S do usuário	475

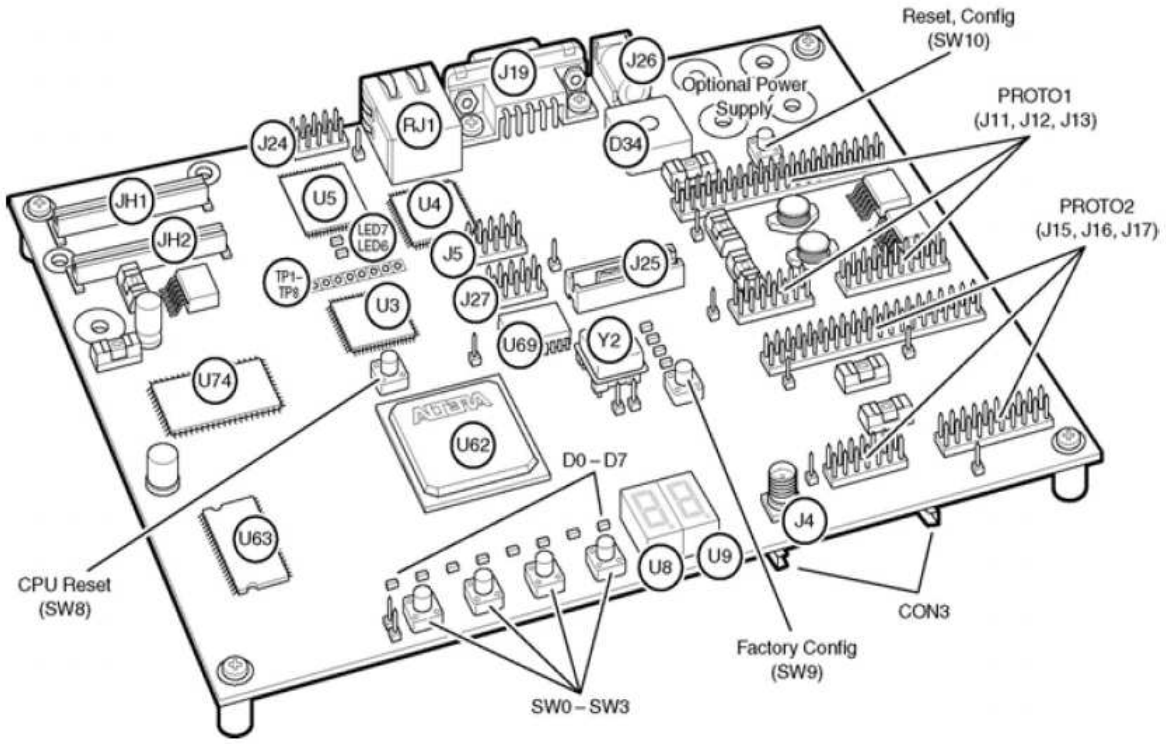


Figura 7.6: Vista superior do kit de desenvolvimento.
FONTE: Altera Corporation

- memória DDR (não indicada na Figura 7.8) e memória SRAM (memória comum) incluídas no kit;
- interface JTAG incluída na própria FPGA;
- display LCD disponível no kit;
- PAL implementada dentro da FPGA;
- mutex (não indicada na Figura 7.8).

No Apêndice A encontra-se o diagrama de projeto do sistema proposto utilizando o *software* Quartus II e SOPC Builder. Cada componente possui as seguintes características e funções dentro do sistema implementado no kit:

- CPU 1: é um NIOS II/e responsável por receber os dados do computador tipo PC via interface JTAG e encaminhar os dados para a memória comum do sistema (SRAM). Também realiza a monitoração das outras CPU's indicando no display LCD o status do sistema. Ao final de cada ciclo de processamento, a CPU 1 realiza a configuração da PAL com a função lógica do circuito classificador obtido.
- CPU 2: é um NIOS II/f responsável pela execução do algoritmo de seleção de amostras (RSR). Os dados são lidos da memória comum, e ao final do processamento os dados selecionados são devolvidos à mesma memória.

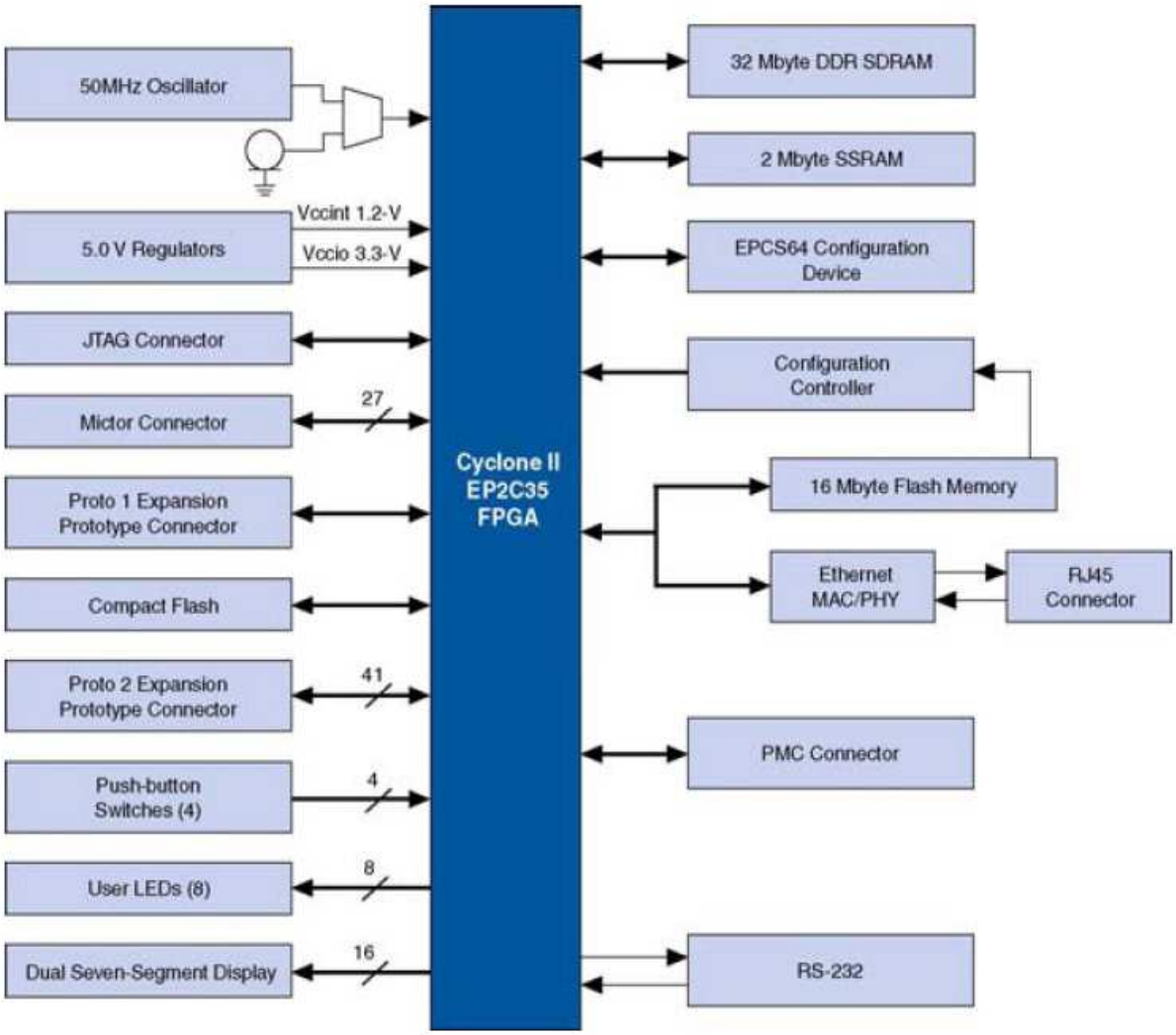


Figura 7.7: Diagrama em blocos do kit de desenvolvimento.
FONTE: Altera Corporation

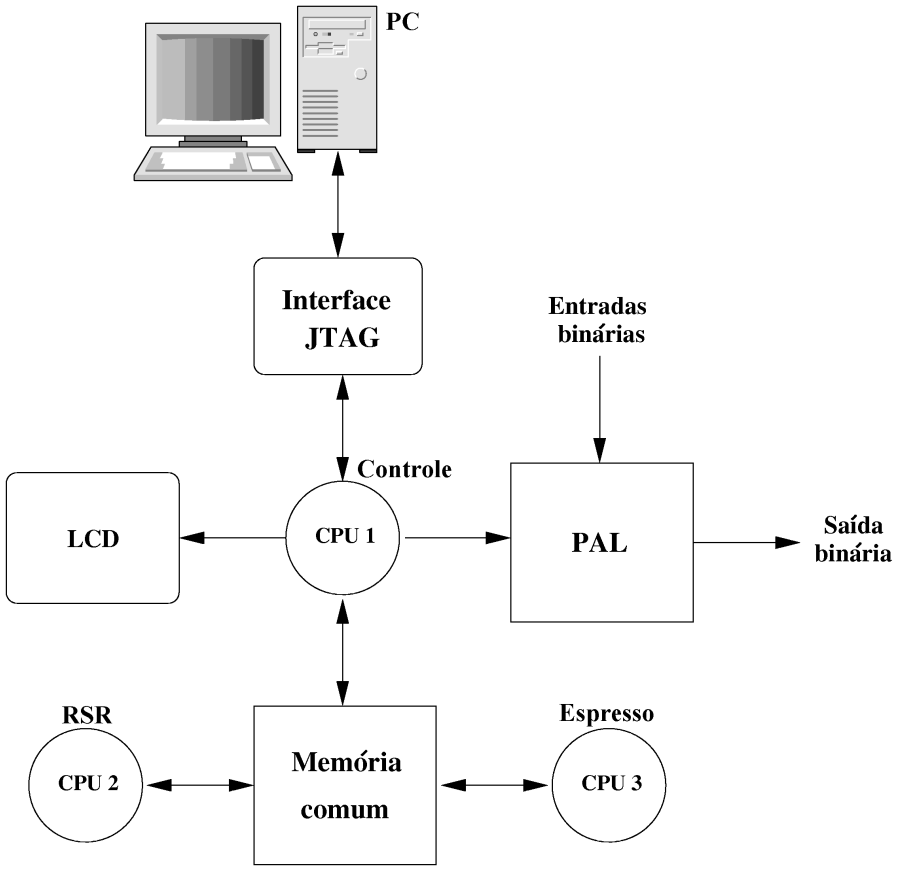


Figura 7.8: Implementação em *hardware* do Sistema Classificador.

- CPU 3: é um NIOS II/f responsável pela execução do minimizador de função Booleana (Espresso). Os dados são lidos da memória comum assim que a CPU 2 os disponibiliza. Ao final do processamento, o resultado é armazenado na mesma memória para que a CPU 1 possa utilizá-los na configuração da PAL.

- Memória DDR: responsável pelo armazenamento do programa de cada CPU e respectivos dados locais. Regiões de memória são reservadas para cada CPU de modo que uma CPU não interfere na região de memória de outra CPU.

- Memória comum: é uma memória SRAM disponível no kit de desenvolvimento e que é compartilhada com as 3 CPU's para troca de dados e informações. Para evitar conflito de dados, o mutex é acessado via *software* por cada CPU.

- Interface JTAG: responsável pela transferência de dados do computador tipo PC para o sistema NIOS, além de permitir o envio e depuração dos programas executados por cada CPU.

- Display LCD: controlado pela CPU 1, é responsável por mostrar o status do sistema.

- PAL: fornece um ambiente de configuração de função Booleana em *hardware* para implementação do resultado do processamento da CPU 3 na forma de soma de produtos.

No Apêndice B, encontra-se a descrição em VHDL da PAL implementada em FPGA, bem como um exemplo de simulação de seu funcionamento. Uma visão simplificada da implementação da PAL é mostrada na Figura 7.9. Ela contém uma matriz de portas lógicas *AND* e *OR*, em conjunto com registradores para armazenar a configuração das entradas binárias na matriz *AND*. A PAL foi projetada para comportar uma capacidade máxima de 85 termos de produto de 24 variáveis binárias de entrada e uma saída. Uma vez configurada, a PAL funciona como um classificador de dados binários provenientes de suas entradas, e fornece em sua saída o resultado da classificação. Foi aproveitado 4 botões existentes no kit para funcionarem como entradas binárias para a PAL, enquanto um LED foi aproveitado como indicador da saída da PAL.

A fim de melhorar o desempenho da CPU 2, responsável pela execução do algoritmo RSR, foi implementada uma instrução personalizada em *hardware* para cálculo da distância entre dois operandos em ponto flutuante (vide Apêndice C). A distância é justamente o cálculo do valor absoluto da diferença entre os dois operandos. Como o processador necessita de realizar este cálculo

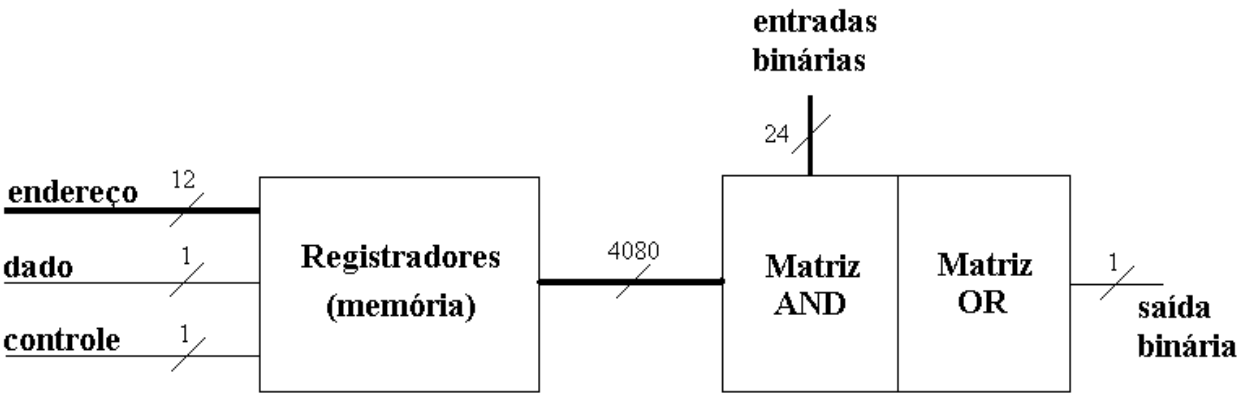


Figura 7.9: Diagrama simplificado da PAL implementada na FPGA.

várias vezes, intenciona-se diminuir o tempo total de execução desta operação. Assim, duas instruções de alto nível (diferença entre dois operandos e cálculo do valor absoluto do resultado) que são executas em 35 ciclos de *clock* foram substituídas por uma única instrução de baixo nível que é executada em apenas 10 ciclos de *clock*.

O sistema completo ocupou 49% dos elementos lógicos da FPGA (total de 33.216) e 45% dos pinos de E/S (total de 475). A implementação da instrução personalizada ocupou apenas 2% da FPGA, enquanto só a implementação da PAL ocupou 25%. O baixo rendimento na implementação da PAL se deve ao desperdício de elementos lógicos para implementar registradores de memória que memorizam o estado da matriz AND na PAL. Além disto, a PAL ocupa vários elementos lógicos para implementar as funções lógicas AND e OR.

A seguir são apresentados os detalhes do *software* que são executados em cada CPU.

7.3.2 O software do sistema

O sistema classificador proposto possui 3 processadores NIOS II que podem trabalhar independentemente um do outro. Cada um possui seus próprios recursos de memória para execução de programas e armazenamento de dados, além de uma memória comum para troca de dados. Cada processador pode acessar 3 tipos de memórias existentes no kit de desenvolvimento:

- Memória Flash
- Memória DDR
- Memória SRAM

A seguir é detalhada cada tipo de memória utilizada pelos processadores.

Memória Flash

A memória Flash possui um tamanho total de 16MBytes e tem por função:

- armazenar o programa de inicialização de cada CPU NIOS II;
- armazenar dados quando necessário (não utilizado no sistema implementado);
- armazenar a configuração da FPGA se desejado pelo usuário (não implementado).

Na Figura 7.10 é apresentado o mapa de memória da Flash. Para cada um dos 3 processadores NIOS II implementados no kit, há uma região de memória de 1MByte na Flash dedicada para armazenar um programa de inicialização da CPU. Esta parte do programa do sistema é definida automaticamente pelo NIOS II IDE.

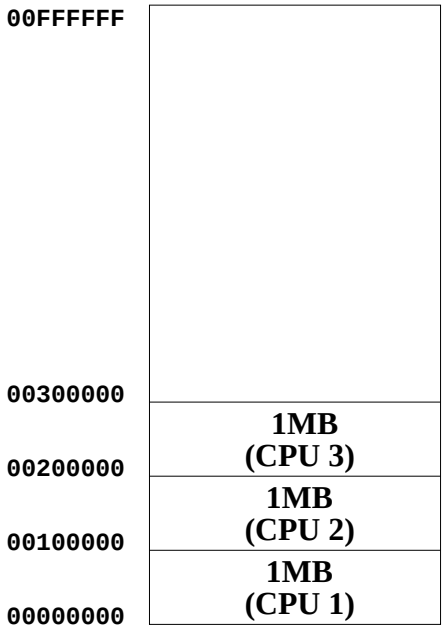


Figura 7.10: Mapa de memória Flash do sistema (inicialização).

O restante da memória Flash não é utilizado pelo sistema implementado.

Memória SDRAM

A memória SDRAM possui um tamanho total de 32MBytes e é responsável por armazenar os aplicativos e dados de cada CPU. Ela foi particionada da maneira indicada na Figura 7.11 para cada CPU.

Cada região da memória SDRAM recebe, via interface JTAG, o programa já compilado correspondente a cada CPU. A ferramenta de *software* NIOS II IDE

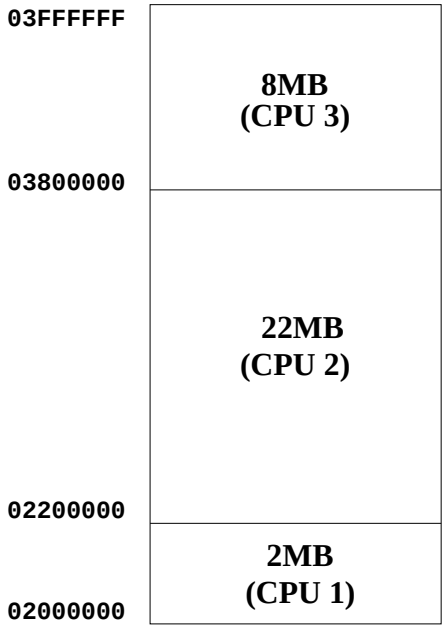


Figura 7.11: Mapa de memória SDRAM do sistema (programa e dados).

executada em um PC se encarrega de compilar, transferir e iniciar a execução de cada programa desenvolvido em linguagem C++ para cada CPU NIOS II.

O programa já compilado em cada CPU em conjunto com os dados inicializados possuem o seguinte tamanho para cada CPU (indicado também o restante de espaço livre de memória):

- CPU 1: 83 KBytes (1964 KBytes livres para *stack* e *heap*);
- CPU 2: 123 KBytes (22404 KBytes livres para *stack* e *heap)*;
- CPU 3: 329 KBytes (7862 KBytes livres para *stack* e *heap*).

Memória SRAM

A memória SRAM foi escolhida para servir de elo de comunicação entre os processadores. Por meio desta memória, os processadores passam os dados uns para os outros sem ocasionar conflito de dados. Possui um tamanho total de 2MBytes e foi particionada para melhor organizar os dados, mas qualquer um dos 3 processadores pode acessar qualquer região desta memória. Na Figura 7.12 é apresentado o seu mapa de memória.

Cada região (buffer) alocada da memória SRAM possui a seguinte característica e função:

- Buffer 1: aproximadamente 1,43MBytes de tamanho, permitindo armazenar 50.000 linhas por 30 colunas de tabela de dados binários no formato do Espresso. É utilizado pela CPU 1 para armazenar os dados binários provenientes do PC e conseqüentemente é lido pela CPU 2 para execução do algoritmo de seleção das amostras (RSR).

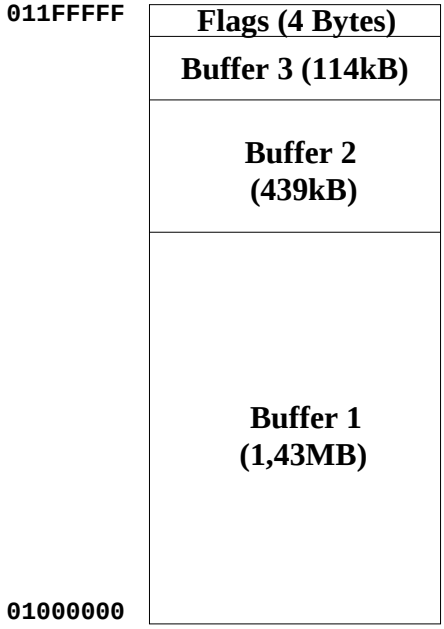


Figura 7.12: Mapa de memória SRAM do sistema (dados comuns).

- Buffer 2: aproximadamente 439kBytes de tamanho, permitindo armazenar 15.000 linhas por 30 colunas de tabela de dados binários no formato Espresso. É utilizado pela CPU 2 para armazenar o resultado das amostras selecionadas pelo algoritmo RSR e conseqüentemente é lido pela CPU 3 para execução do algoritmo Espresso.

- Buffer 3: aproximadamente 114kBytes de tamanho, permitindo armazenar 4.000 linhas por 30 colunas de tabela de dados binários no formato Espresso. É utilizado pela CPU 3 para armazenar o resultado do programa Espresso e conseqüentemente é lido pela CPU 1 para configurar a PAL.

- Flags: são pequenos espaços reservados da memória para que cada processador possa armazenar o seu *status* corrente, de modo que a CPU 1 possa identificar o que cada processador está fazendo num dado momento e apresentar a informação no display LCD.

O tamanho de cada buffer pode ser alterado de acordo com as necessidades do problema. A seguir é apresentado um exemplo de utilização do sistema classificador implementado.

7.4 Resultados Experimentais

O sistema classificador de dados binários baseado nas 3 CPU's NIOS II foi implementado com sucesso no kit de desenvolvimento com FPGA da Altera. As 3 CPU´s funcionam de maneira independente e em paralelo, permitindo que

o sistema funcione de maneira contínua. Assim, enquanto novos dados estão sendo recebidos pela CPU 1, os dados anteriores estão sendo processados pela CPU 2 (algoritmo RSR), dados mais antigos já processados pela CPU 2 estão sendo processados pela CPU 3 (algoritmo Espresso), e a PAL já está configurada implementando o classificador com o resultado anterior dos dados já processados pela CPU 3.

Para testar o sistema, foi utilizado como entrada o mesmo conjunto de dados sintéticos binários descrito no Capítulo 4. Porém, foi utilizado uma quantidade de amostras menor devido a limitação da memória do sistema para armazenamento da matriz de distâncias. Assim foram utilizadas 1385 amostras binárias de 24 bits de atributos e 2 classes. A métrica de distância escolhida para cálculo dos vizinhos-mais-próximos foi a VDM e o parâmetro k com valor 10 para o algoritmo RSR.

Na tabela 7.3 são mostrados algumas medições de tempo de execução do sistema classificador implementado no kit (formato de média e desvio padrão), bem como o mesmo sistema implementado em *software* em um computador tipo PC Pentium IV 1.8GHz com 1GByte de RAM. Foram medidos o tempo de execução de cada etapa do sistema classificador (RSR e Espresso) e também os resultados obtidos da quantidade de amostras selecionadas pelo algoritmo RSR e termos de produto da função Booleana obtida do minimizador Booleano (Espresso). Também foram realizados testes do Espresso sem a execução do algoritmo de seleção de amostras (RSR) tanto no PC quanto no sistema NIOS II implementado em kit. O sistema NIOS II também foi testado sem a instrução personalizada para cálculo das distâncias.

Tabela 7.3: Resultados de tempos de execução.

Plataforma	**RSR [s]**	**N. amostras**	**Espresso [s]**	**N. produtos**
PC P4 1.8GHz (sem RSR)	-	1385	1,8±0,4	37
PC P4 1.8GHz	49±2	831	1,0±0,0	13
NIOS II (sem RSR)	-	1385	486,5±0,5	37
NIOS II	21403±39	831	425,8±43,6	13
NIOS II personalizado	20097±25	831	425,8±43,6	13

A velocidade de processamento do sistema classificador no ambiente do computador PC foi bem maior que no sistema NIOS II proposto conforme tabela anterior. Isto é devido principalmente a velocidade superior do *clock* do PC ($\frac{1800}{50}$MHz $= 36$ vezes superior) e aos recursos de *hardware* adicionais existentes no PC (memória cache rápida de dados e instruções, barramento de dados de 64 bits, pipeline de múltiplos estágios, preditor de *branch*, etc). Por outro lado, o sistema NIOS II foi implementado em uma FPGA de baixo custo e com recursos inferiores de hardware. Contudo, o sistema implementado em FPGA possui a característica de ser completamente embutido, autônomo

e compacto.

Observa-se pela tabela que, com a instrução personalizada no sistema NIOS II, o tempo de execução do algoritmo RSR (21403 s) diminuiu aproximadamente 6% quando comparado com o tempo de execução do mesmo algoritmo sem a instrução personalizada (20097 s). Assim, é possível diminuir mais o tempo de execução do algoritmo RSR e Espresso, implementando mais instruções personalizadas em hardware.

O uso do algoritmo de seleção de amostras (RSR) como pré-processamento dos dados antes de aplicação do Espresso, faz com que a quantidade de amostras caia de 1385 para 831 (40% de diminuição). Isto acarreta uma diminuição no tempo de processamento do Espresso no sistema NIOS II em 12,5%, além de diminuir substancialmente a quantidade de termos de produto da função Booleana do circuito digital classificador (de 37 para 13). Isto evidencia o ganho da capacidade de generalização do circuito gerado quando utilizado em conjunto com o algoritmo RSR. O custo para isto está no tempo de processamento do RSR, que para este exemplo de dados não compensou o ganho de tempo de processamento do Espresso.

O tempo de atraso de resposta da PAL, implementada na FPGA, para ativação da resposta de classificação (saída) referente à amostra de entrada foi de 20ns, conforme medido no simulador do *software* Quartus II (vide Apêndice B). Este tempo de resposta não é possível de ser batido por um sistema implementado em *software* como em um microcomputador tipo PC.

Na Figura 7.13 é mostrado um exemplo do resultado da função Booleana processada pelo sistema proposto (utilizando o algoritmo RSR e Espresso) para implementação na PAL, no formato do programa Espresso.

7.5 Conclusão

A utilização de *hardware* reconfigurável viabiliza a implementação de um sistema multiprocessado de forma rápida, fácil e segura. O conjunto de ferramentas de *software* disponíveis atualmente livra o projetista de um árduo trabalho de se preocupar com vários detalhes ao mesmo tempo. Isto permite que o projetista se preocupe com outras questões mais relevantes do projeto como, por exemplo, o desenvolvimento do algoritmo que está sendo implementado.

O sistema classificador de padrões binários foi implementado em uma FPGA, evidenciando a sua característica de ser possível a sua implementação em um sistema embutido para aplicações específicas. Além do treinamento do classificador ser *on-line*, o tempo de resposta do sistema treinado depende apenas do atraso de portas lógicas da FPGA. Isto é devido ao componente PAL do sistema

```
.i 24
.o 1
.p 13
--0---0-1----1---0-0--00 1
0-0-0--------0--0000000- 1
0--0--000---0-0-100--000 1
00--00----------0-000000 1
-000-0--00------0--00-0- 1
-00000---0---0--000--00- 1
--000-01----0-----0-000- 1
000-0--0-1------0----000 1
-0---00-----1-----000-00 1
00000--------0---0-00000 1
000000---------0---00-00 1
00---------0----00000000 1
0-00----0--1----0-0--00- 1
.e
```

Figura 7.13: Função Booleana processada pelo sistema para implementação na PAL, no formato do programa Espresso.

classificador ser implementado utilizando apenas circuitos combinatoriais.

A utilização de *hardware* reconfigurável permite que o sistema classificador possa ser facilmente adaptado para aplicações diferentes, de acordo com as necessidades do problema. Para aplicações que exigem velocidade maior de processamento do NIOS II, pode ser necessário substituir a FPGA por uma de família mais poderosa, como por exemplo a Stratix II da Altera. Ou então desenvolver mais instruções personalizadas em hardware, justamente para substituir a parte do *software* onde apresenta atrasos maiores.

Sistemas mono ou multiprocessados implementados em hardware reconfigurável permitem que algoritmos desenvolvidos inicialmente em plataformas monoprocessadas e não portáteis (tipo PC) possam ser diretamente migradas para equipamentos embutidos. De acordo com a necessidade, partes do *software* original pode ser convertido gradativamente para *hardware* com a finalidade de melhorar o desempenho do sistema. O custo para isto é o investimento em tempo de projetista e área de FPGA. E graças as ferramentas de projeto atuais, isto tudo é possível!

CAPÍTULO 8

Conclusão Final e Propostas de Continuidade

Sistemas de aprendizado podem ser implementados em *software* ou *hardware*. A escolha de uma metodologia ou outra depende das necessidades e características do problema alvo. Geralmente o método em *hardware* é mais adequado quando é necessário maior rapidez de resposta do sistema e quando a compactação e portabilidade são essenciais. Assim, circuitos digitais são atrativos para implementação de tais sistemas. Além disto, com o advento de dispositivos digitais reconfiguráveis, é possível a implementação remota e em tempo real de sistemas digitais.

Informações em diversos tipos e formatos podem ser codificadas em dados binários para serem utilizadas em sistemas digitais. O código de conversão utilizado influencia diretamente a capacidade do sistema em extrair conhecimento pelos dados. Dados binários podem ser obtidos e armazenados facilmente para serem utilizados diretamente em sistemas de aprendizado.

Circuitos digitais podem trabalhar como classificadores e ter boa generalização desde que, durante a fase de projeto, os dados binários de treinamento sejam pré-processados (ou filtrados), de forma a eliminar as amostras que prejudiquem o desempenho de classificação do circuito para dados desconhecidos. Os melhores dados selecionados para geração do circuito são justamente aqueles que se encontram perto da margem de separação das classes. A maior dificuldade é, justamente, determinar que dados devem ser aproveitados e quais devem ser ignorados para o projeto do circuito digital.

Algoritmos de minimização de função Booleana utilizados em síntese de circuitos digitais não se preocupam em apresentar generalização para dados

desconhecidos, uma vez que a função deles é gerar um circuito digital que seja fiel à tabela de dados que o gerou (dados de treinamento), utilizando o mínimo de portas lógicas. Entretanto, os algoritmos de minimização se aproveitam dos dados desconhecidos (*don't care*) para otimizar o circuito gerado, durante a etapa de expansão dos hipercubos. Esta forma de expansão é semelhante à determinação dos vizinhos-mais-próximos de uma amostra de treinamento.

Caso os dados binários sejam cuidadosamente selecionados para utilização no algoritmo de minimização Booleana, então os dados eliminados são tratados como uma situação de (*don't care*). Assim, durante o processo de expansão dos hipercubos na execução do algoritmo de minimização, este obtém a melhor superfície de separação das classes (saídas do circuito) numa posição intermediária entre as classes. Isto caracteriza a generalização do circuito gerado.

A Regra do Vizinho-mais-próximo possui ótimo desempenho em problemas de classificação e pode ser utilizada como método de pré-processamento dos dados a serem utilizados no projeto do classificador digital. Vários algoritmos baseados na regra do vizinho-mais-próximo se propõem a identificar e selecionar as amostras perto da margem de modo ao classificador apresentar ótima generalização. O método de seleção proposto neste trabalho se baseia na regra desenvolvida por Wilson (Wilson 1972) para selecionar as amostras perto da margem.

A Regra do Vizinho-mais-próximo depende diretamente de uma medida de similaridade dos dados. Neste trabalho foi deduzida a formulação estatística da Distância de Discriminação (Aleksander, Clarke, & Braga 1994), uma métrica de distância própria para ser utilizada com dados binários. Além disto, foi proposto uma modificação na métrica VDM para melhoria do seu desempenho para determinados tipos de problemas, conforme comprovado por experimentos. Também foi desenvolvida uma nova métrica baseada nos coeficientes da verossimilhança da Regra de Bayes. Esta nova métrica se mostrou mais adequada para alguns tipos de problemas pois apresentou um comportamento mais estável para o classificador kNN do que outras métricas apresentadas.

O método de seleção de amostras (denominado RSR) proposto neste trabalho de tese filtra (retira) os dados entre as margens de separação das classes e seleciona (retém) os dados mais próximos às margens que melhor caracterizam cada classe. A quantidade de amostras selecionadas é controlada pelo parâmetro k, o que influencia na generalização da resposta do classificador. A aplicação destes dados selecionados como entrada ao minimizador de função Booleana faz com que os hipercubos sejam naturalmente expandidos de tal forma a se aproximarem da fronteira de separação das classes, além da expansão ser também em direção ao interior do agrupamento de cada classe.

O método de seleção de amostras proposto reduz a quantidade armazenada de amostras necessárias para a obtenção do circuito digital pelo minimizador Booleano, o que implica na redução do custo computacional de processamento pelo minimizador, além de garantir uma melhora na generalização quando comparado ao processo de obtenção do circuito sem a seleção dos dados.

O método de seleção RSR também pode ser utilizado em conjunto com outros métodos de classificação formando um sistema híbrido. O objetivo do pré-processamento das amostras com o algoritmo RSR seria não só a redução da quantidade de amostras e custo computacional do método subseqüente, mas também a melhoria no desempenho da classificação final. O método RSR pode ser utilizado com dados binários, mas também com dados nominais e números reais ou inteiros.

O sistema classificador de dados binários foi implementado em um sistema de *hardware* reconfigurável baseado em FPGA, tornando-o um sistema embarcado que pode ser utilizado em aplicações específicas que exigem portabilidade, autonomia e compactação. O sistema classificador implementado funciona em tempo real, com treinamento *on-line* e resposta de classificação rápida com tempo de atraso devido apenas ao atraso intrínseco das portas lógicas.

A característica de *hardware* reconfigurável em FPGA permite que o sistema classificador implementado possa ser modificado facilmente para se adaptar a diferentes problemas, além de permitir modificações para melhoria do desempenho. Por exemplo, partes do *software* podem ser convertidas para *hardware* com um custo de área de FPGA e tempo de projeto. Ou então, pode-se utilizar dispositivos mais velozes para implementação do sistema como alguma família de FPGA mais rápida (e possivelmente mais cara).

O uso de multiprocessamento para implementação do sistema em *hardware* reconfigurável torna-o flexível para ser utilizado tanto por desenvolvedores de *software* quanto de *hardware*. Assim, espera-se que este trabalho de tese tenha continuidade tanto no desenvolvimento de *software* quanto no de *hardware*.

Propostas de Continuidade

Sugere-se como propostas para continuação deste trabalho de tese, investir nos seguintes problemas relacionados ao tema:

- Modificação do algoritmo de minimização lógica (Quine-McCluskey ou Espresso) para melhorar a generalização do circuito digital gerado, ou seja, minimização lógica e controle da generalização implementados no mesmo algoritmo;
- Testar o método proposto utilizando outros bancos de dados binários, inclusive dados reais com características multimodais;
- Utilizar o método de seleção de amostras proposto para aplicação em SVM, com o objetivo de diminuir o custo computacional do SVM e de outros métodos de classificação de dados;
- Criar um esquema de atualização (ou modificação) *on-line* do circuito digital quando os dados de treinamento são disponibilizados em partes, utilizando o sistema proposto;
- Comparar o método proposto com outros métodos tipo: Regra de Bayes, Árvore de Decisão, Sistemas Fuzzy, etc;
- Personalizar mais instruções do processador NIOS II em *hardware*, tanto para o algoritmo RSR quanto para o Espresso, com o objetivo de diminuir o tempo de processamento;
- Desenvolver outras métricas específicas para padrões binários;
- Implementar recepção de dados do sistema NIOS utilizando a porta serial e/ou *ethernet*;
- Desenvolver um programa monitor para a CPU 1 do sistema implementado, de forma que um computador PC possa monitorar o funcionamento do sistema e inclusive enviar comandos;
- Implementar cache de dados do sistema NIOS II com coerência de cache;
- Determinar as condições de contorno do problema para as quais cada métrica descrita é mais indicada que outra;
- Desenvolver o método de seleção de amostras para mais de duas classes (Myles & Hand 1990) e implementar no sistema NIOS;

- Melhorar o desempenho do classificador pelo desenvolvimento do algoritmo de seleção de amostras, aproveitando as idéias dos mais novos métodos de seleção de amostras (Sánchez, Barandela, Marqués, Alejo, & Badenas 2003);
- Utilizar todas as amostras de treinamento para obter a função Booleana do circuito classificador e então melhorar a generalização pela manipulação da expressão lógica por meio de algoritmos genéticos (*hardware* evolutivo).

Referências

Aha, D. W., D. Kibler, & M. K. Albert (1991). Instance-based learning algorithms. *Machine Learning 6*, 37–66.

Aleksander, I. (1995). *An Introduction to Neural Computing* (2a ed.). London: International Thomson Computer Press. 284 p.

Aleksander, I., T. J. W. Clarke, & A. P. Braga (1994). Binary neural systems: combining weighted and weightless properties. *Intelligent Systems Engineering 3*(4), 211–221.

Altera Corporation (2003a, July). *NIOS Handbook*. San Jose: Altera Corporation. Version 1.3.

Altera Corporation (2003b). *NIOS II Processor Reference Handbook*. San Jose: Altera Corporation. Version 5.1.

Altera Corporation (2003c, July). *NIOS Software Development Tutorial*. San Jose: Altera Corporation. Version 1.3.

Altera Corporation (2004, May). *AN 351: Simulating Nios II Embedded Processor Designs*. San Jose: Altera Corporation. Version 1.0.

Altera Corporation (2005a, May). *Creating Multiprocessor Nios II Systems Tutorial*. San Jose: Altera Corporation. Version 1.0.

Altera Corporation (2005b, January). *NIOS II Custom Instruction User Guide*. San Jose: Altera Corporation. Version 1.2.

Altera Corporation (2005c). *NIOS II Development Kit - Getting Start User Guide*. San Jose: Altera Corporation. Version 2.0.

Altera Corporation (2005d). *NIOS II Software Developer's Handbook*. San Jose: Altera Corporation. Version 5.2.

Altera Corporation (2005e). *Quartus II Development Software Handbook*. San Jose: Altera Corporation. Version 5.1.

Avesani, P., E. Blanzieri, & F. Ricci (1999, September). Advanced metrics for class-driven similarity search. In *Tenth International Workshop on Database and Expert Systems Applications*, pp. 223–227.

Berger, A. (2002). *Embedded Systems Design - An Introduction to Process, Tools and Techniques*. Lawrence, Kansas: CMP Books. 237p.

Blake, C. L. & C. J. Merz (1998). UCI Repository of machine learning databases, University of California, Irvine, Dept. of Information and Computer Sciences. Disponível em http://www.ics.uci.edu/~mlearn/MLRepository.html. <Acesso em janeiro de 2004>.

Blanzieri, E. & F. Ricci (1999a). A minimum risk metric for nearest neighbor classification. In *Proc. 16th International Conf. on Machine Learning*, pp. 22–31. Morgan Kaufmann, San Francisco, CA.

Blanzieri, E. & F. Ricci (1999b). Probability based metrics for nearest neighbor classification and case-based reasoning. *Lecture Notes in Computer Science 1650*, 14–29.

Braga, A. P. (1995, November). *Design models for recursive binary neural networks*. Tese de Doutorado, Imperial College of Science, Technology and Medicine - University of London, London. 163p.

Cachin, C. (1994). Pedagogical pattern selection strategies. *Neural Network 7*(1), 175–181.

Carvalho, B. P. R., W. S. Lacerda, & A. P. Braga (2005, November). A hybrid approach for sparse least squares support vector machines. In *Fifth International Conference on Hybrid Intelligent Systems HIS'2005*, Rio de Janeiro, Brazil, pp. 323–328. IEEE Computer Society.

Chang, C.-L. (1974, November). Finding prototypes for nearest neighbor classifiers. *IEEE Transactions on Computers 23*(11), 1179–1184.

Chaudhuri, D., C. A. Murthy, & B. B. Chaudhuri (1994, September). Finding a subset of representative points in a data set. *IEEE Transactions on Systems, Man and Cybernetics 24*(9), 1416–1424.

Cherkassky, V. S. & F. Mulier (1998). *Learning from Data: Concepts, Theory, and Methods*. New York: John Wiley & Sons. 441p.

Cost, S. & S. Salzberg (1993). A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning 10*, 57–78.

Cover, T. (1968, January). Estimation by the nearest neighbor rule. *IEEE Transactions on Information Theory IT-14*(1), 50–55.

Cover, T. & P. Hart (1967, January). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory IT-13*(1), 21–27. Reprinted in Pattern Recognition, Chinese University Press, Hong Kong, 1980. ed. by K.S. Fu.

Dasarathy, B. V. (1994, March). Minimal consistent set (MCS) identification for optimal nearest neighbor decision systems design. *IEEE Transactions on Systems, Man and Cybernetics 24*(3), 511–517.

Dasarathy, B. V., J. S. Sánchez, & S. Townsend (2000). Nearest neighbour editing and condensing tools - synergy exploitation. *Pattern Analysis Applications 3*, 19–30. Springer-Verlag London Limited.

De Micheli, G. (1994). *Synthesis and Optimization of Digital Circuits*. Cingapura: McGraw-Hill International. 579 p.

Domeniconi, C. & D. Gunopulos (2002). Adaptive nearest neighbor classification using support vector machines. In T. G. Dietterich, S. Becker, & Z. Ghahramani (Eds.), *Advances in Neural Information Processing Systems 14*, Cambridge. MIT Press.

Domeniconi, C., J. Peng, & D. Gunopulo (2000). An adaptive metric machine for pattern classification. In *Advances in Neural Information Processing Systems*, pp. 458–464. Neural Information Processing Systems: MIT Press.

Duda, R. O., P. E. Hart, & D. G. Stork (2000). *Pattern Classification*. New York: John Wiley & Sons. 654 p.

Dudani, S. A. (1976, April). The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics 6*, 325–327.

Eckel, B. (2000). *Thinking in C++, Volume 1: Introduction to Standard C++* (2th ed.). Toronto: Prentice Hall. 814p.

Ercegovac, M., T. Lang, & J. H. Moreno (2000). *Introdução aos Sistemas Digitais*. Porto Alegre: Bookman. 453 p.

Etrusco, L. F. (2003, dezembro). *Arquiteturas Reconfiguráveis Aplicadas ao Desenvolvimento de Sistemas Embutidos*. Tese de doutorado, Escola de Engenharia da Universidade Federal de Minas Gerais, Belo Horizonte.

Ferrari, A., M. Borgatti, & R. Guerrieri (2000, December). A complete system for nn classification based on a vlsi array processor. *Pattern Recognition 33*, 2083–2093.

Ferri, F. J., J. V. Albert, & E. Vidal (1999). Considerations about sample-size sensitivity of a family of edited nearest-neighbor rules. *IEEE Transactions on Systems, Man and Cybernetics, Part B 29*(5), 667–672.

Fiser, P. & J. Hlavicka (2001, June). Boom: a boolean minimizer. Technical report, CTU, Faculty of Electrical Engineering, Department of Computer Science and Engineering, Prague - Czech Republic. 33 p.

Fukunaga, K. & T. E. Flick (1984). An optimal global nearest neighbor metric. *IEEE Trans. Pattern Analysis and Machine Intelligence PAMI-6*(3), 314–318.

Gates, G. W. (1972). The reduced nearest neighbor rule. *IEEE Transactions on Information Theory 18*(3), 431–433.

Geva, S. & J. Sitte (1991, March). Adaptive nearest neighbor pattern classification. *IEEE Transactions on Neural Networks 2*(2), 318–322.

Gowda, K. C. & G. Krishna (1979, July). The condensed nearest neighbor rule using the concept of mutual nearest neighborhood. *IEEE Transactions on Information Theory 25*(4), 488–490.

Hall, M. A. (1999, April). *Correlation-based Feature Selection for Machine Learning*. Tese de Doutorado, Department of Computer Science - University of Waikato, Hamilton - New Zealand. 178p.

Hamza, A. B., H. Krim, & B. Karacali (2003, April). Structural risk minimization using nearest neighbor rule. In *IEEE International Conference on Acoustics, Speech, and Signal Processing ICASSP '03*, Volume 6, pp. 161–164. IEEE.

Hart, P. E. (1968). The condensed nearest neighbor. *IEEE Transaction on Information Theory IT*(14), 515–516.

Hastie, T. & P. Y. Simard (1997). Models and metrics for handwritten character recognition. *Statistical Science 13*(1), 54–65.

Hastie, T. & R. Tibshirani (1996, June). Discriminant adaptive nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*(6), 607–616.

Hattori, K. & M. Takahashi (1999). A new nearest-neighbor rule in the pattern classification problem. *Pattern Recognition 32*(3), 425–432.

Hattori, K. & M. Takahashi (2000). A new edited k-nearest neighbor rule in the pattern classification problem. *Pattern Recognition 33*(3), 521–528.

Haykin, S. (2001). *Redes Neurais: Princípios e Prática* (2ª ed.). Porto Alegre: Bookman. 900 p.

Hlavicka, J. & P. Fiser (2001). Boom: a heuristic boolean minimizer. In *ICCAD '01: Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, Piscataway, NJ, USA, pp. 439–442. IEEE Press.

Jain, A. K., M. N. Murty, & P. J. Flynn (1999, September). Data clustering: A review. *ACM Computing Surveys 31*(3), 264–323.

Jaskolski, J. V. (1992, June). Construction of neural network classification expert systems using switching theory algorithms. In *International Joint Conference on Neural Networks - IJCNN*, Volume 1, pp. 1–6. IEEE.

Kanerva, P. (1988). *Sparse Distributed Memory*. London: A Bradford Book. 155 p.

Kim, S.-W. & B. J. Oommen (2004, June). Enhancing prototype reduction schemes with recursion: A method applicable for large data sets. *IEEE Transactions on Systems, Man, and Cybernetics-Part B 34*(3), 1384–1397.

Kolcz, A. & N. M. Allinson (1994, May). Application of the cmac input encoding scheme in n-tuple approximation network. In *Proceedings on Computers and Digital Technique*, Volume 141:3, pp. 177–183. IEE.

Koplowitz, J. & T. A. Brown (1981). On the relation of performance to editing in nearest neighbor rules. *Pattern Recognition 13*(3), 251–255.

Kovács, Z. M. V. & R. Guerrieri (1991, November). A generalization technique for nearest-neighbor classifiers. In *IEEE International Joint Conference on Neural Networks*, Volume 2, pp. 1782–1788. IEEE.

Kuncheva, L. I. & J. C. Bezdek (1998, Feb.). Nearest prototype classification: clustering, genetic algorithms, or random search? *IEEE Transactions on Systems, Man and Cybernetics, Part C 28*(1), 160–164.

Lacerda, W. S. & A. P. Braga (2004, September). Um novo método para determinação de margens utilizando a regra do vizinho mais próximo modificada. In *VIII Brazilian Symposium on Artificial Neural Networks SBRN'2004*, São Luis/Maranhão. SBC.

Lacerda, W. S. & A. P. Braga (2005, November). Design of digital classifier circuits with nearest neighbour prior sample selection. In *Fifth International Conference on Hybrid Intelligent Systems HIS'2005*, Rio de Janeiro, Brazil, pp. 531–533. IEEE Computer Society.

Lam, W., C.-K. Keung, & D. Liu (2002). Discovering useful concept prototypes for classification based on filtering and abstraction. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*(8), 1075–1090.

Lipman, A. & W. Yang (1997, September). Vlsi hardware for example-based learning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 5*(3), 320–328.

Liu, T., A. W. Moore, & A. Gray (2004). Efficient exact k-nn and nonparametric classification in high dimensions. In S. Thrun, L. Saul, &

B. Schölkopf (Eds.), *Advances in Neural Information Processing Systems 16*. Cambridge, MA: MIT Press.

Liu, W. Z. & A. P. White (1997). Metrics for nearest neighbour discrimination with categorical attributes. In *Research and Development in Expert Systems XIV: Proceedings of the 17th Annual Technicial Conference of the BCES Specialist Group*, pp. 51–59.

Lorena, A. C., G. E. A. P. A. Batista, & A. C. P. L. F. de Carvalho (2002). The influence of noisy patterns in the performance of learning methods in the splice junction recognition problem. In I. C. Society (Ed.), *Proceedings of the VII Brazilian Symposium on Neural Networks - SBRN'02*.

Lorena, A. C. & A. C. de Carvalho (2003, Abril). Introdução às máquinas de vetores suporte (*Support Vector Machines*). Technical Report 192, Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação, São Carlos.

Luk, A. & J. E. S. Macleod (1986). An alternative nearest neighbour classification scheme. *Pattern Recognition Letters 4*, 375–382.

Mahamud, S. & M. Hebert (2003). Minimum risk distance measure for object recognition. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, Washington, DC, USA, pp. 242. IEEE Computer Society.

Munro, P. W. (1992). Repeat until bored; a pattern selection strategy. In J. Moody & R. Lippmann (Eds.), *Advances in Neural Information Processing Systems*, Volume 4, pp. 1001–1008. San Mateo, CA: Morgan Kaufmann.

Myles, J. P. & D. J. Hand (1990). the multiclass metric problem in nearest neighbour discrimination rules. *Pattern Recognition 23*(11), 1291–1297.

Nene, S. A. & S. K. Nayar (1997, september). A simple algorithm for nearest neighbor search in high dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*(9), 989–1003.

Paredes, R. & E. Vidal (1998, July). A nearest neighbor weighted measure in classification problems. In *Proceedings of VIII Simposium Nacional de Reconocimiento de Formas y Análisis de Imágenes*, Bilbao, Spain.

Paredes, R. & E. Vidal (2000). A class-dependent weighted dissimilarity measure for nearest neighbor classification problems. *Pattern Recognition Letters 21*(12), 1027–1036.

Partridge, M.; Jabri, M. (2000, December). Face recognition using a new distance metric. In *Proceedings of the X Neural Networks for Signal Processing Workshop*, Volume 2, pp. 584 – 593. IEEE Signal Processing Society.

Payne, T. R. & P. Edwards (1998). Implicit feature selection with the value difference metric. In H. Prade (Ed.), *Machine Learning and Data Mining*, pp. 450–454. 13th European Conference on Artificial Intelligence (ECAI98): John Wiley & Sons.

Peebles, P. Z. (1993). *Probability, Random Variables, and Random Signal Principles* (3rd ed. ed.). Singapore: McGraw-Hill. 401 p.

Peng, J., D. R. Heisterkamp, & H. K. Dai (2002, August). Adaptive kernel metric nearest neighbor classification. In *16th International Conference on Pattern Recognition*, pp. 33–36. IEEE.

Peng, J., D. R. Heisterkamp, & H. K. Dai (2003, July). LDA/SVM driven nearest neighbor classification. *IEEE Transactions on Neural Networks 14*(4), 940–942.

Prechelt, L. (1994, September). Proben1: A set of neural network benchmark problems and benchmarking rules. Technical Report 21, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany. 38p.

Ricci, F. & P. Avesani (1999, April). Data compression and local metrics for nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence 21*(4), 380–384.

Ritter, G. L., H. B. Woodruff, S. R. Lowry, & T. L. Isenhour (1975, November). An algorithm for a seletive nearest neighbor decision rule. *IEEE Transactions on Information Theory 21*, 665–669.

Rohwer, R. & M. Morciniec (1995, May). The theoretical and experimental status of the n-tuple classifier. NCRG 4347, Aston University - Dept. of Computer Science and Applied Mathematics, UK. 24 p.

Schalkoff, R. J. (1992). *Pattern Recognition: Statistical, Structural and Neural Approaches*. John Wiley & Sons. 364 p.

Schapire, R. E. (2002, March). The boosting approach to machine learning: An overview. In MSRI Workshop on Nonlinear Estimation and Classification. Berkeley, CA.

Schwaighofer, A. (2002, January). SVM toolbox for Matlab - version 2.51. URL http://www.cis.tugraz.at/igi/aschwaig/software.html. Acesso em março de 2004.

Short, R. D. & K. Fukunaga (1981, September). The optimal distance measure for nearest neighbor classification. *IEEE Transactions on Information Theory 27*(5), 622–627.

Silva, M. T. P., W. S. Lacerda, & A. P. Braga (2004). Reconfigurable coprocessor for kanerva's sparse distributed memory. *Microprocessors and Microsystems 28*(3), 127–134.

Snapp, R., D. Psaltis, & S. S. Venkatesh (1991). Asymptotic slowing down of the nearest-neighbor classifier. In R. P. Lippmann, J. E. Moody, & D. S. Touretzky (Eds.), *Advances in Neural Information Processing Systems*, pp. 932–938. San Mateo: Morgan Kaufmann.

Sánchez, J., R. Barandela, R. Alejo, & A. I. Marqués (2001, October). Performance evaluation of prototype selection algorithms for nearest neighbor classification. In *Proceedings of XIV Brazilian Symposium on Computer Graphics and Image Processing*, pp. 44–50. IEEE.

Sánchez, J. S., R. Barandela, A. I. Marqués, R. Alejo, & J. Badenas (2003, April). Analysis of new techniques to obtain quality training sets. *Pattern Recognition Letters 24*(7), 1015–1022.

Stanfill, C. & D. Waltz (1986, December). Toward memory-based reasoning. *Communications of the ACM 29*(12), 1213 – 1228.

Teixeira, R. A. (2001, agosto). *Treinamento de Redes Neurais Artificiais Através de Otimização Multi-objetivo: Uma Nova Abordagem para o Equilíbrio entre a Polarização e a Variância*. Tese de Doutorado, Escola de Engenharia da Universidade Federal de Minas Gerais, Belo Horizonte. 144 p.

Teixeira, R. A., A. P. Braga, R. H. C. Takahashi, & R. R. Saldanha (2000, November). Improving generalization of mlps with multi-objective optimization. *Neurocomputing 35*(1), 189–194.

The Mathworks (2001). *MATLAB User's Guide*. The Mathworks. URL http://www.mathworks.com <Acesso em julho de 2003>.

Tomek, I. (1976a, June). An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man and Sybernetics 6*(6), 448–452.

Tomek, I. (1976b, February). A generalization of the k-NN rule. *IEEE Transactions on Systems, Man and Sybernetics 6*(2), 121–126.

Tomek, I. (1976c, November). Two modifications of CNN. *IEEE Transactions on Systems, Man and Sybernetics 6*(11), 769–772.

Toussaint, G. T. (1994, August). A counter-example to tomek's consistency theorem for a condensed nearest neighbor decision rule. *Pattern Recognition Letters 15*, 797–801.

Tzionas, P., P. Tsalides, & A. Thanailakis (1992, December). Design and vlsi implementation of a pattern classifier using pseudo 2d cellular automata. In *IEE Proceedings-G Circuits, Devices and Systems*, Volume 139, pp. 661–668.

Tzionas, P. G., P. G. Tsalides, & A. Thanailakis (1994, September). A new, cellular automaton-based, nearest neighbor pattern classifier and its vlsi implementation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 2*(3), 343 – 353.

University of Texas at Austin – College of Engineering (1990). *Espresso*. University of Texas at Austin – College of Engineering. Disponível em http://www.ece.utexas.edu/projects/logic_synthesis/sis/espresso <Acesso em agosto de 2003>.

Vahid, F. & T. Givargis (2002). *Embedded System Design*. USA: John Wiley & Sons. 324p.

Vapnik, V. N. (1998). *Statistical Learning Theory*. New York: John Wiley & Sons, Inc. 736p.

Vincent, P. & Y. Bengio (2001, June). K-local hyperplane and convex distance nearest neighbor algorithms. Technical report, Dept. IRO, Université de Montréal, C.P. 6128, Montreal, Canada. Techinical Report 1197.

Wagner, T. J. (1973, September). Convergence of the edited nearest neighbor. *IEEE Transactions on Information Theory 19*(5), 696–697.

Weinshall, D., D. W. Jacobs, & Y. Gdalyahu (1999). Classification in non-metric spaces. In S. A. S. Michael S. Kearn & D. A. Cohn (Eds.), *Advances in Neural Information Processing Systems - Proceedings of the 1998 Conference*, Volume 11, London, England, pp. 838–844. MIT Press.

Wilmshurst, T. (2001). *An Introduction to the Design of Small-scale Embedded Systems*. New York: Palgrave. 411p.

Wilson, D. L. (1972, July). Asymptotic properties of nearest neighbor rules using edited data. *IEEE - Transactions on Systems, Man, and Cybernetcs SMC-2*(3), 408–421.

Wilson, D. R. & T. R. Martinez (1997). Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research 6*, 1–34.

Wilson, D. R. & T. R. Martinez (2000). Reduction techniques for instance-based learning algorithms. *Machine Learning 38*(3), 257–286.

Wolf, W. (2001). *Computers as Components - Principles of Embedded Computing System Design*. USA: Morgan Kaufmann. 662p.

Wu, Y., K. Ianakiev, & V. Govindaraju (2002, October). Improved k-nearest neighbor classification. *Pattern Recognition 35*(10), 2311–2318.

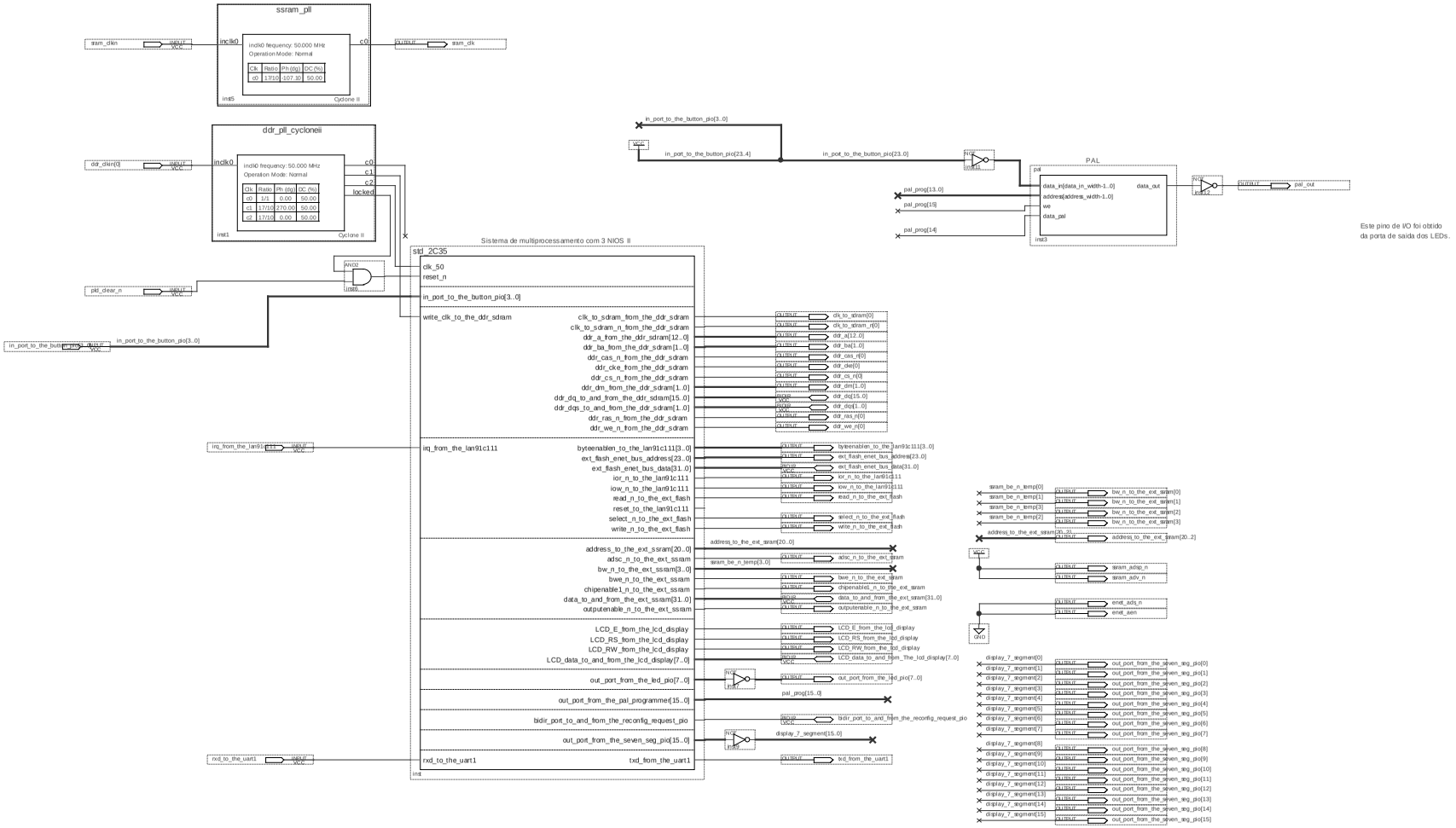
Xie, Q., C. A. Laszlo, & R. K. Ward (1993, December). Vector quantization technique for nonparametric classifier design. *IEEE Transactions on Pattern Analysis and Machine Intelligence 15*(12), 1326–1330.

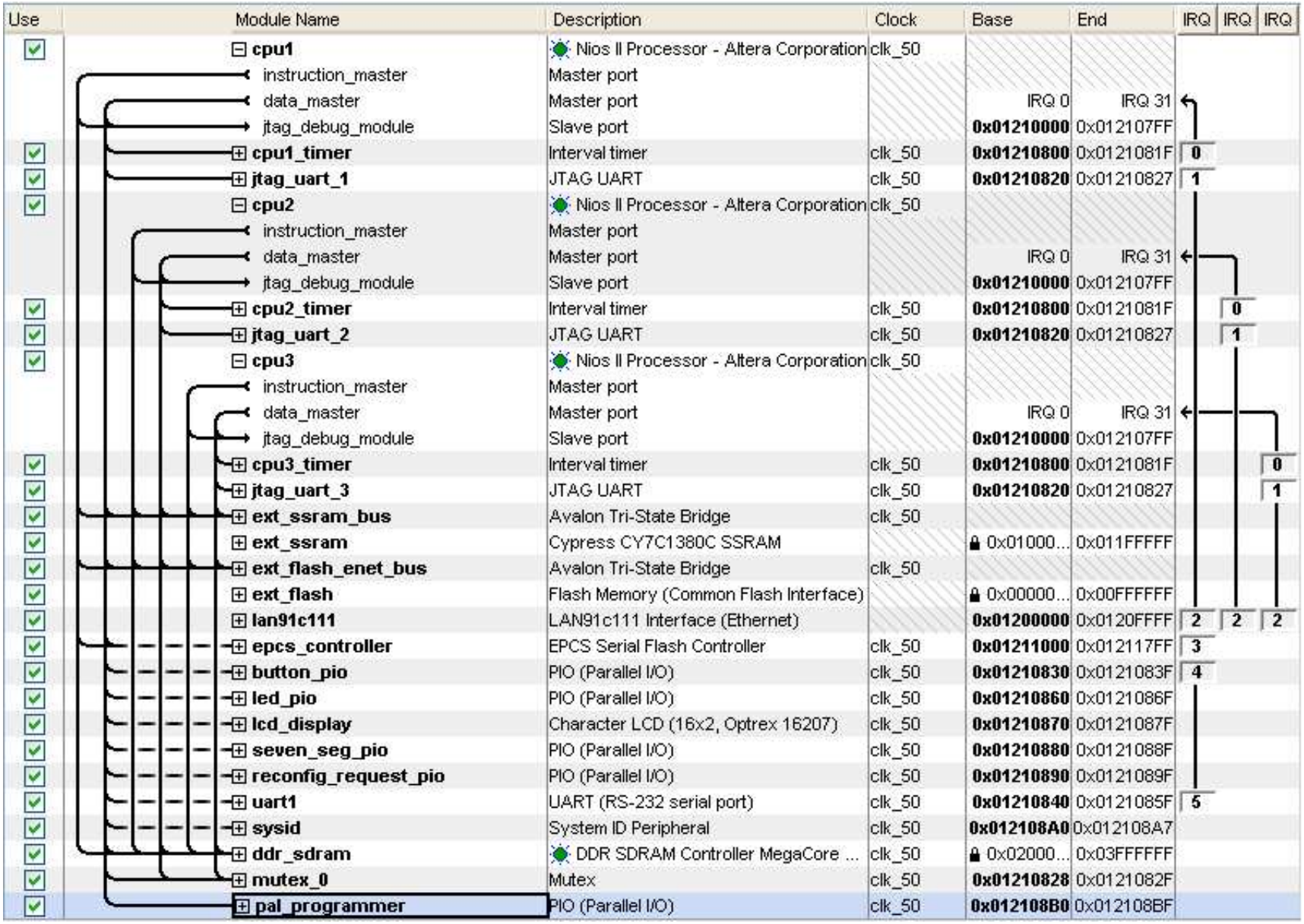
Xing, E. P., A. Y. Ng, M. I. Jordan, & S. Russell (2003). Distance metric learning with application to clustering with side-information. In S. T. S. Becker & K. Obermayer (Eds.), *Advances in Neural Information Processing Systems 15*, pp. 505–512. Cambridge: MIT Press.

Yen, C.-W., C.-N. Young, & M. L. Nagurka (2004). A vector quantization method for nearest neighbor classifier design. *Pattern Recognition Letters 25*(6), 725–731.

Zhang, H. & C. X. Ling (2003, June). A fundamental issue of naive bayes. In Y. Xiang & B. Chaib-draa (Eds.), *Proceedings of Canadian Conference on Artificial Intelligence*, Volume 2671 of *Lecture Notes in Computer Science*, Halifax, Nova Scotia, Canada, pp. 591–595. Springer.

Apêndice A - Projeto do sistema proposto usando o Quartus II e SOPC


ssram_pll
sram_dkin
inclk0
inclk0 frequency: 50.000 MHz
Operation Mode: Normal
Clk | Ratio | Ph (dg) | DC (%)
c0 | 17/10 | 107.30 | 50.00
c0
sram_clk
inst5
Cyclone II
ddr_pll_cycloneii
ddr_clkin[0]
inclk0
inclk0 frequency: 50.000 MHz
Operation Mode: Normal
Clk | Ratio | Ph (dg) | DC (%)
c0 | 1/1 | 0.00 | 50.00
c1 | 17/10 | 270.00 | 50.00
c2 | 17/10 | 0.00 | 50.00
c0
c1
c2
locked
inst1
Cyclone II
pld_clear_n
in_port_to_the_button_pio[3..0]
in_port_to_the_button_pio[23..4]
in_port_to_the_button_pio[23..0]
pal_prog[13..0]
pal_prog[15]
pal_prog[14]
PAL
pal
data_in[data_in_width-1..0]
address[address_width-1..0]
we
data_pal
data_out
inst3
pal_out
Este pino de I/O foi obtido
da porta de saida dos LEDs.
Sistema de multiprocessamento com 3 NIOS II
std_2C35
clk_50
reset_n
in_port_to_the_button_pio[3..0]
write_clk_to_the_ddr_sdram
clk_to_sdram_from_the_ddr_sdram
clk_to_sdram_n_from_the_ddr_sdram
ddr_a_from_the_ddr_sdram[12..0]
ddr_ba_from_the_ddr_sdram[1..0]
ddr_cas_n_from_the_ddr_sdram
ddr_cke_from_the_ddr_sdram
ddr_cs_n_from_the_ddr_sdram
ddr_dm_from_the_ddr_sdram[1..0]
ddr_dq_to_and_from_the_ddr_sdram[15..0]
ddr_dqs_to_and_from_the_ddr_sdram[1..0]
ddr_ras_n_from_the_ddr_sdram
ddr_we_n_from_the_ddr_sdram
irq_from_the_lan91c111
byteenablen_to_the_lan91c111[3..0]
ext_flash_enet_bus_address[23..0]
ext_flash_enet_bus_data[31..0]
ior_n_to_the_lan91c111
iow_n_to_the_lan91c111
read_n_to_the_ext_flash
reset_to_the_lan91c111
select_n_to_the_ext_flash
write_n_to_the_ext_flash
address_to_the_ext_ssram[20..0]
adsc_n_to_the_ext_ssram
bw_n_to_the_ext_ssram[3..0]
bwe_n_to_the_ext_ssram
chipenable1_n_to_the_ext_ssram
data_to_and_from_the_ext_ssram[31..0]
outputenable_n_to_the_ext_ssram
LCD_E_from_the_lcd_display
LCD_RS_from_the_lcd_display
LCD_RW_from_the_lcd_display
LCD_data_to_and_from_the_lcd_display[7..0]
out_port_from_the_led_pio[7..0]
out_port_from_the_pal_programme[15..0]
bidir_port_to_and_from_the_reconfig_request_pio
out_port_from_the_seven_seg_pio[15..0]
rxd_to_the_uart1
txd_from_the_uart1
inst
pal_prog[15..0]
display_7_segment[15..0]
sram_be_n_temp[0]
sram_be_n_temp[1]
sram_be_n_temp[3]
sram_be_n_temp[2]
address_to_the_ext_sram[20..31]
sram_adsp_n
sram_adv_n
enet_adsc_n
enet_aen
display_7_segment[0]
display_7_segment[1]
display_7_segment[2]
display_7_segment[3]
display_7_segment[4]
display_7_segment[5]
display_7_segment[6]
display_7_segment[7]
display_7_segment[8]
display_7_segment[9]
display_7_segment[10]
display_7_segment[11]
display_7_segment[12]
display_7_segment[13]
display_7_segment[14]
display_7_segment[15]


Use
Module Name
Description
Clock
Base
End
IRQ
IRQ
IRQ
cpu1
Nios II Processor - Altera Corporation
clk_50
instruction_master
Master port
data_master
Master port
IRQ 0
IRQ 31
jtag_debug_module
Slave port
0x01210000
0x012107FF
cpu1_timer
Interval timer
clk_50
0x01210800
0x0121081F
0
jtag_uart_1
JTAG UART
clk_50
0x01210820
0x01210827
1
cpu2
Nios II Processor - Altera Corporation
clk_50
instruction_master
Master port
data_master
Master port
IRQ 0
IRQ 31
jtag_debug_module
Slave port
0x01210000
0x012107FF
cpu2_timer
Interval timer
clk_50
0x01210800
0x0121081F
0
jtag_uart_2
JTAG UART
clk_50
0x01210820
0x01210827
1
cpu3
Nios II Processor - Altera Corporation
clk_50
instruction_master
Master port
data_master
Master port
IRQ 0
IRQ 31
jtag_debug_module
Slave port
0x01210000
0x012107FF
cpu3_timer
Interval timer
clk_50
0x01210800
0x0121081F
0
jtag_uart_3
JTAG UART
clk_50
0x01210820
0x01210827
1
ext_ssram_bus
Avalon Tri-State Bridge
clk_50
ext_ssram
Cypress CY7C1380C SSRAM
0x01000...
0x011FFFFF
ext_flash_enet_bus
Avalon Tri-State Bridge
clk_50
ext_flash
Flash Memory (Common Flash Interface)
0x00000...
0x00FFFFFF
lan91c111
LAN91c111 Interface (Ethernet)
0x01200000
0x0120FFFF
2
2
2
epcs_controller
EPCS Serial Flash Controller
clk_50
0x01211000
0x012117FF
3
button_pio
PIO (Parallel I/O)
clk_50
0x01210830
0x0121083F
4
led_pio
PIO (Parallel I/O)
clk_50
0x01210860
0x0121086F
lcd_display
Character LCD (16x2, Optrex 16207)
clk_50
0x01210870
0x0121087F
seven_seg_pio
PIO (Parallel I/O)
clk_50
0x01210880
0x0121088F
reconfig_request_pio
PIO (Parallel I/O)
clk_50
0x01210890
0x0121089F
uart1
UART (RS-232 serial port)
clk_50
0x01210840
0x0121085F
5
sysid
System ID Peripheral
clk_50
0x012108A0
0x012108A7
ddr_sdram
DDR SDRAM Controller MegaCore ...
clk_50
0x02000...
0x03FFFFFF
mutex_0
Mutex
clk_50
0x01210828
0x0121082F
pal_programmer
PIO (Parallel I/O)
clk_50
0x012108B0
0x012108BF

Fitter report for standard
Wed Dec 07 16:42:53 2005
Version 5.0 Build 168 06/22/2005 Service Pack 1.17 SJ Full Version
Additional patches: 1.04

Fitter Summary	
Fitter Status	Successful - Wed Dec 07 16:42:50 2005
Quartus II Version	5.0 Build 168 06/22/2005 SP 1.17 SJ Full Version
Revision Name	standard
Top-level Entity Name	standard
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Preliminary
Total logic elements	16,412 / 33,216 (49 %)
Total pins	218 / 475 (45 %)
Total virtual pins	0
Total memory bits	110,208 / 483,840 (22 %)
Embedded Multiplier 9-bit elements	8 / 70 (11 %)
Total PLLs	2 / 4 (50 %)

Fitter Resource Usage Summary	
Resource	Usage
Total logic elements	16,412 / 33,216 (49 %)
-- Combinational with no register	7823
-- Register only	2918
-- Combinational with a register	5671
Logic element usage by number of LUT inputs	
-- 4 input functions	6255
-- 3 input functions	5586
-- <=2 input functions	1653
-- Register only	2918
-- Combinational cells for routing	1722
Logic elements by mode	
-- normal mode	12312
-- arithmetic mode	1182
Total registers	8,589 / 33,216 (25 %)
Total LABs	2,041 / 2,076 (98 %)
User inserted logic elements	0
Virtual pins	0
I/O pins	218 / 475 (45 %)
-- Clock pins	2 / 8 (25 %)
Global signals	16
M4Ks	40 / 105 (38 %)
Total memory bits	110,208 / 483,840 (22 %)
Total RAM block bits	184,320 / 483,840 (38 %)
Embedded Multiplier 9-bit elements	8 / 70 (11 %)
Global clocks	16 / 16 (100 %)
Maximum fan-out node	ddr_pll_cycloneii:inst1|altpll:altpll_component|_clk2~clkctrl
Maximum fan-out	5737
Total fan-out	80345
Average fan-out	3.33

Apêndice B - Projeto da PAL em VHDL

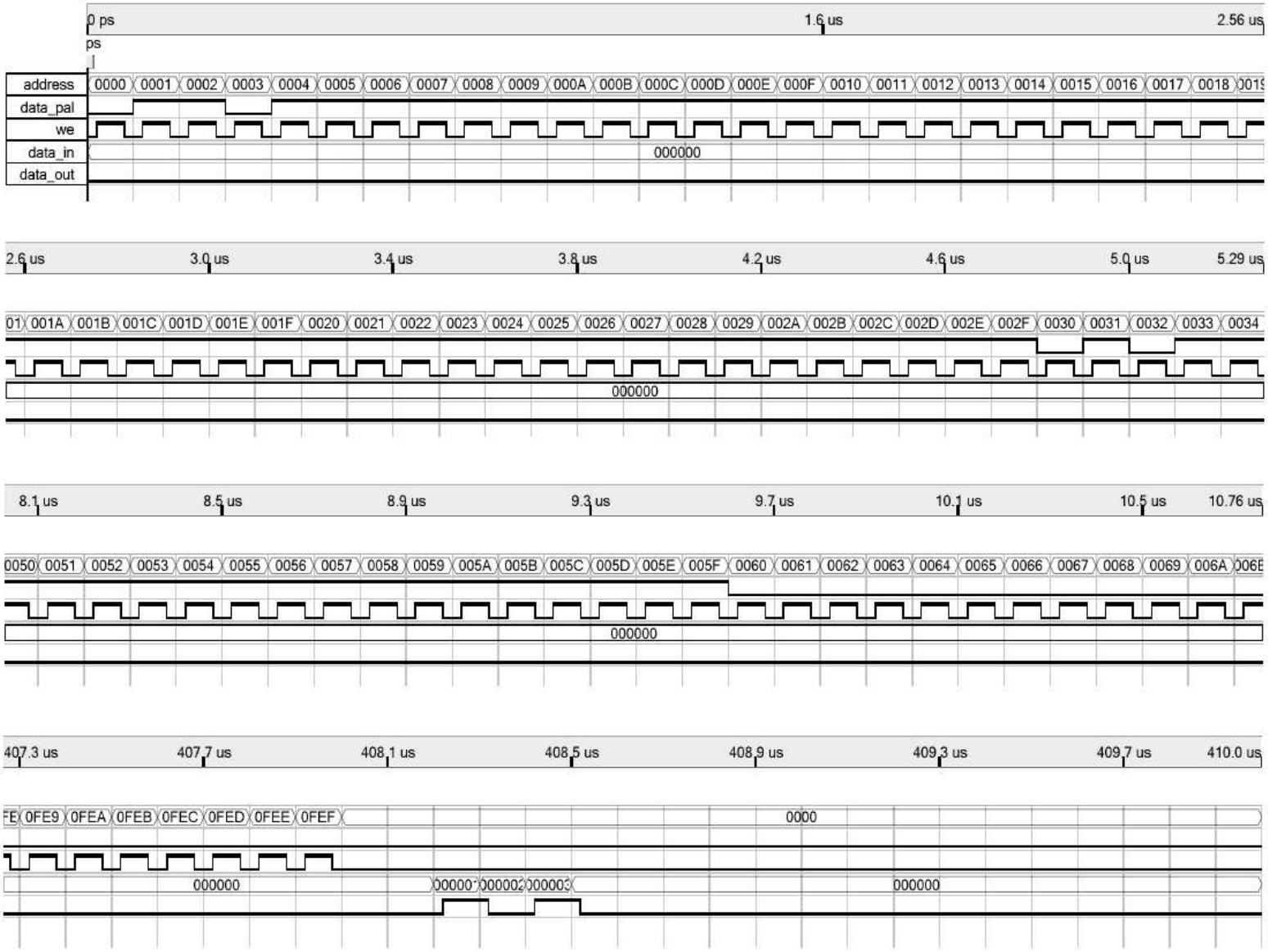
```
----------------------------------------------------------------------
-- Programa de implementacao de uma PAL com uma saida binaria
-- Autor: Wilian Soares Lacerda
-- Data: 24/10/2005
----------------------------------------------------------------------
Library IEEE ;
use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use IEEE.std_logic_unsigned.all ;
----------------------------------------------------------------------
entity pal is
  generic (
    -- Numero de variaveis binarias de entrada em bits
    data_in_width   : natural := 24 ;
    -- Numero de bits de endereco do literal usado para
    -- acessar matriz and
    address_width   : natural := 12 ;
    -- Numero de literais = dobro do numero de variaveis de entrada
    num_literais    : natural := 48;
    -- Numero de bits da matriz and da PAL = num_literais*produtos
    size_matriz_and : natural := 4080;
    -- Numero de termos de produto = numero de portas and
    produtos        : natural := 85 );
  port (
    data_in  : in  UNSIGNED((data_in_width - 1) downto 0) ;
    address  : in  UNSIGNED((address_width - 1)  downto 0) ;
    we       : in  std_logic ;
    data_pal : in  std_logic;
    data_out : out std_logic );
end pal;
----------------------------------------------------------------------

architecture Comportamental of pal is
  type reg_type is array ((size_matriz_and - 1) downto 0) of std_logic;
  signal registrador : reg_type ;
  signal saida_and : std_logic_vector((produtos - 1) downto 0);
```

```
  begin
    ----------------------------------------------------------------
    memoria : process (we)
      begin
        if (we = '1') then
          registrador(conv_integer(address)) <= data_pal ;
        end if ;
      end process ;
    ----------------------------------------------------------------
    matriz_and : process (registrador,data_in)
      variable resposta_and : std_logic;
      variable fusivel : integer;
      begin
        for j in 0 to (produtos - 1) loop
          resposta_and := '1';
          for i in 0 to (data_in_width - 1) loop
            for k in 0 to 1 loop
              fusivel := (j * num_literais) + (i * 2) + k;
              if (registrador(fusivel) = '0') then
                -- fusivel sem queimar
                if (k = 0) then -- entrada nao invertida
                  resposta_and := resposta_and and data_in(i);
                else  -- entrada invertida
                  resposta_and := resposta_and and (not (data_in(i)));
                end if;
              end if;
            end loop;
          end loop;
          saida_and(j) <= resposta_and;
        end loop;
      end process;
    ----------------------------------------------------------------
    matriz_or : process (saida_and)
      variable resposta_or : std_logic;
      begin
        resposta_or := '0';
        for j in 0 to (produtos - 1) loop
          resposta_or := resposta_or or saida_and(j);
        end loop;
        data_out <= resposta_or;
      end process;
    ----------------------------------------------------------------
end Comportamental;
```

Fitter report for pal
Thu Oct 27 19:42:53 2005
Version 5.0 Build 168 06/22/2005 Service Pack 1.17 SJ Full Version
Additional patches: 1.04

Fitter Summary	
Fitter Status	Successful - Thu Oct 27 19:42:51 2005
Quartus II Version	5.0 Build 168 06/22/2005 SP 1.17 SJ Full Version
Revision Name	pal
Top-level Entity Name	pal
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Preliminary
Total logic elements	8,562 / 33,216 (25 %)
Total pins	41 / 475 (8 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Fitter Resource Usage Summary	
Resource	Usage
Total logic elements	8,562 / 33,216 (25 %)
-- Combinational with no register	4482
-- Register only	2040
-- Combinational with a register	2040
Logic element usage by number of LUT inputs	
-- 4 input functions	1379
-- 3 input functions	5143
-- <=2 input functions	0
-- Register only	2040
-- Combinational cells for routing	567
Logic elements by mode	
-- normal mode	6522
-- arithmetic mode	0
Total registers	4,080 / 33,216 (12 %)
Total LABs	2,040 / 2,076 (98 %)
User inserted logic elements	0
Virtual pins	0
I/O pins	41 / 475 (8 %)
-- Clock pins	2 / 8 (25 %)
Global signals	1
M4Ks	0 / 105 (0 %)
Total memory bits	0 / 483,840 (0 %)
Total RAM block bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Global clocks	1 / 16 (6 %)
Maximum fan-out node	data_pal
Maximum fan-out	4080
Total fan-out	33754
Average fan-out	3.01


0 ps
1.6 us
2.56 us
ps
address
data_pal
we
data_in
data_out
0000 0001 0002 0003 0004 0005 0006 0007 0008 0009 000A 000B 000C 000D 000E 000F 0010 0011 0012 0013 0014 0015 0016 0017 0018
000000
2.6 us
3.0 us
3.4 us
3.8 us
4.2 us
4.6 us
5.0 us
5.29 us
001A 001B 001C 001D 001E 001F 0020 0021 0022 0023 0024 0025 0026 0027 0028 0029 002A 002B 002C 002D 002E 002F 0030 0031 0032 0033 0034
000000
8.1 us
8.5 us
8.9 us
9.3 us
9.7 us
10.1 us
10.5 us
10.76 us
0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 005A 005B 005C 005D 005E 005F 0060 0061 0062 0063 0064 0065 0066 0067 0068 0069 006A
000000
407.3 us
407.7 us
408.1 us
408.5 us
408.9 us
409.3 us
409.7 us
410.0 us
0FE9 0FEA 0FEB 0FEC 0FED 0FEE 0FEF
0000
000000
000000
000000
000000

Apêndice C - Projeto em *hardware* da instrução personalizada no NIOS II

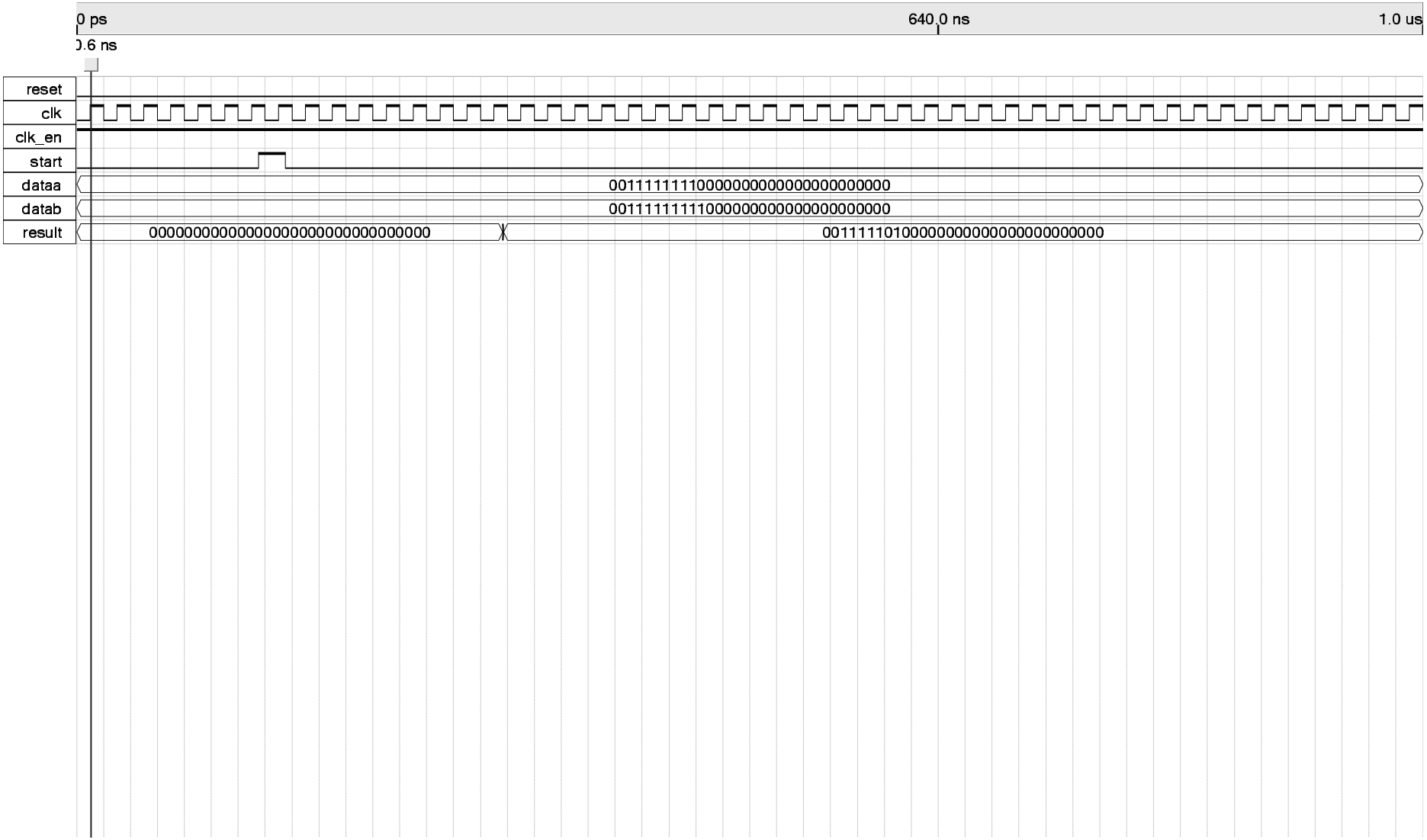
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
--------------------------------------------------------------------------------
ENTITY dif_abs_float IS
   PORT
    (
    dataa       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    datab       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    clk_en      : IN STD_LOGIC ;
    reset           : IN STD_LOGIC ;
    clk     : IN STD_LOGIC ;
    start       : IN STD_LOGIC ;
    result      : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
   );
END dif_abs_float;
--------------------------------------------------------------------------------
ARCHITECTURE RTL OF dif_abs_float IS
  ATTRIBUTE synthesis_clearbox: boolean;
  ATTRIBUTE synthesis_clearbox OF RTL: ARCHITECTURE IS TRUE;
  SIGNAL sub_wire0  : STD_LOGIC ;
  SIGNAL sub_wire1  : STD_LOGIC ;
  SIGNAL sub_wire2  : STD_LOGIC ;
  SIGNAL sub_wire3  : STD_LOGIC_VECTOR (31 DOWNTO 0);
  SIGNAL dataar     : STD_LOGIC_VECTOR (31 DOWNTO 0);  -- registered dataa
  SIGNAL databr     : STD_LOGIC_VECTOR (31 DOWNTO 0);  -- registered datab

  COMPONENT dif_abs_float_altfp_add_sub_rkg
    PORT (
    dataa       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    datab       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    overflow    : OUT STD_LOGIC ;
    underflow   : OUT STD_LOGIC ;
    nan     : OUT STD_LOGIC ;
    clk_en      : IN STD_LOGIC ;
    clock       : IN STD_LOGIC ;
    aclr        : IN STD_LOGIC ;
    result      : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
  END COMPONENT;
--------------------------------------------------------------------------------
```

```
BEGIN

  dif_abs_float_altfp_add_sub_rkg_component : dif_abs_float_altfp_add_sub_rkg
  PORT MAP (
    dataa       => dataar,
    datab       => databr,
    clk_en      => clk_en,
    clock       => clk,
    aclr        => reset,
    overflow    => sub_wire0,
    underflow   => sub_wire1,
    nan         => sub_wire2,
    result      => sub_wire3
  );
--------------------------------------------------------------------------------

  PROCESS (clk)
  BEGIN

    IF (clk'EVENT AND clk = '1') THEN

        IF (start = '1') THEN
            dataar <= dataa;
            databr <= datab;
        ELSE
            dataar <= dataar;
            databr <= databr;
        END IF;

    END IF;

  END PROCESS;

  result(31) <= '0';    -- Transform value to positive
  result(30 DOWNTO 0) <= sub_wire3(30 DOWNTO 0);

END RTL;
--------------------------------------------------------------------------------
```

Fitter report for dif_abs_float
Mon Nov 28 14:59:23 2005
Version 5.0 Build 168 06/22/2005 Service Pack 1.17 SJ Full Version
Additional patches: 1.04

Fitter Summary	
Fitter Status	Successful - Mon Nov 28 14:59:23 2005
Quartus II Version	5.0 Build 168 06/22/2005 SP 1.17 SJ Full Version
Revision Name	dif_abs_float
Top-level Entity Name	dif_abs_float
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Preliminary
Total logic elements	790 / 33,216 (2 %)
Total pins	100 / 475 (21 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Fitter Resource Usage Summary	
Resource	Usage
Total logic elements	790 / 33,216 (2 %)
-- Combinational with no register	366
-- Register only	92
-- Combinational with a register	332
Logic element usage by number of LUT inputs	
-- 4 input functions	254
-- 3 input functions	332
-- <=2 input functions	112
-- Register only	92
-- Combinational cells for routing	73
Logic elements by mode	
-- normal mode	582
-- arithmetic mode	116
Total registers	424 / 33,216 (1 %)
Total LABs	52 / 2,076 (2 %)
User inserted logic elements	0
Virtual pins	0
I/O pins	100 / 475 (21 %)
-- Clock pins	2 / 8 (25 %)
Global signals	2
M4Ks	0 / 105 (0 %)
Total memory bits	0 / 483,840 (0 %)
Total RAM block bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Global clocks	2 / 16 (12 %)
Maximum fan-out node	clk~clkctrl
Maximum fan-out	424
Total fan-out	3979
Average fan-out	3.06


0 ps
640.0 ns
1.0 us
0.6 ns
reset
clk
clk_en
start
dataa
datab
result
00111111110000000000000000000000
00111111111000000000000000000000
00000000000000000000000000000000
00111111010000000000000000000000