Hae Yong Kim

Construção Automática de Operadores Morfológicos por Aprendizagem Computacional

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Engenharia.

São Paulo
1997

Hae Yong Kim

Construção Automática de Operadores Morfológicos por Aprendizagem Computacional

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Engenharia.

Área de Concentração:
Sistemas Eletrônicos

Orientador:
Prof. Dr. Flávio A. M. Cipparrone

São Paulo
1997

Kim, Hae Yong
Construção automática de operadores morfológicos pela aprendizagem computacional. São Paulo, 1997.
204 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Eletrônica.

1. Processamento de imagem 2. Morfologia matemática - Reticulado completo 3. Aprendizagem computacional - PAC I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Eletrônica II. t.

Agradecimentos

Gostaria de manifestar os meus agradecimentos a todos os amigos e amigas que tornaram possível a realização desta tese. Em especial, agradeço:

- Ao prof. Flávio A. M. Cipparrone, pela sua amizade e dedicada orientação.
- Ao prof. José Jaime da Cruz, pela sua orientação inicial e confiança.
- Ao falecido prof. Normonds Alens, que transmitiu-me um pouco da sua longa experiência acadêmica.
- Ao prof. Max Gerken, que como coordenador de pós-graduação, se empenhou para que este trabalho fosse adiante.
- Ao prof. Paulo Antônio Mariotto, que como chefe do Departamento de Engenharia Eletrônica, ajudou-me em diversos momentos.
- Aos amigos e colegas do laboratório prof. Ivandro Sanches, prof. Mário Minami, prof. Miguel Arjona Ramirez, e ao prof. Ronaldo Fumio Hashimoto do Instituto de Matemática e Estatística, pela amizade e compreensão.
- Ao prof. Luiz Jean Lauand, da Faculdade de Educação da USP e ao prof. Elói Medina Galego, do Instituto de Matemática e Estatística da USP, por todo apoio.

- A todos amigos do Centro Cultural Pinheiros que incentivaram esta pesquisa acadêmica.

- Aos meus pais e à minha irmã.

Também gostaria de agradecer àqueles(as) que cuidam dedicadamente das bibliotecas da Engenharia Elétrica da Escola Politécnica e do Instituto de Matemática da USP pois possibilitaram encontrar todos os artigos e livros citados nas referências bibliográficas deste trabalho (na verdade, menos dois artigos em alemão, cujo interesse é meramente histórico).

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

τ-operador	Operador invariante por translação.
Kd-árvore	Árvore *k* dimensional. Tradução do inglês “kd-tree”.
MAE	Erro absoluto médio, do inglês “mean absolute error”.
MEE	Erro euclidiano médio, do inglês “mean euclidean error”.
MSE	Erro quadrático médio, do inglês “mean square error”.
NN	Vizinho mais próximo, do inglês “nearest neighbor”.
PAC	Provavelmente aproximadamente correto, do inglês “probably approximately correct”.
W-operador	Operador restrito à janela. Tradução do inglês “windowed operator”.

Resumo

A Morfologia Matemática é uma teoria que lida com o processamento e análise de imagens, utilizando operadores baseados em conceitos topológicos e geométricos. Tradicionalmente, o usuário constrói um operador morfológico manualmente, compondo operadores elementares, baseado em suas experiências prévias. Porém, muitas vezes, esta tarefa não é trivial, dificultando o uso prático da morfologia. Esta tese introduz novas técnicas para construir operadores em geral (isto é, não necessariamente binários) automaticamente utilizando pares de imagens de entrada e de saída como exemplos de treinamento. Estas técnicas foram implementadas para imagens binárias, em níveis de cinza e coloridas e alguns exemplos de aplicação juntamente com informações de desempenho são expostos. Freqüentemente, um operador gerado automaticamente é composto por um número muito grande de operadores elementares. Técnicas convencionais necessitam de muito tempo para aplicar tal operador numa imagem digital. Esta tese também introduz novos esquemas de representação para operadores, que podem ser armazenados em árvores binárias. O uso de árvores binárias acelera tanto a construção como a aplicação do operador. Finalmente, a construção automática do operador é formalizada utilizando os modelos de aprendizagem PAC (provavelmente aproximadamente correta) generalizada e vizinho mais próximo.

Abstract

Mathematical Morphology is a theory concerned with image processing and analysis, using operators based on topological and geometrical concepts. Traditionally, the user builds morphological operators manually, composing elementary operators, based on his previous experiences. But, in many cases, this is not a trivial task, difficulting the practical use of the morphology. This thesis introduces new techniques to construct generic (not necessarily binary) operators automatically using pairs of input and output images as training samples. These techniques were implemented for binary, gray-scale and color images and some application examples with performance informations are depicted. Frequently, an automatically generated operator is composed of a great number of elementary operators. Conventional techniques take too long to apply such an operator to digital images. This thesis also introduces new representation schemes for operators, which can be stored in binary trees. The use of binary tree quickens both construction and application of the operator. Finally, the automatic construction of operators is formalized using generalized PAC (Probably Approximately Correct) and nearest neighbor learning models.

1 Introdução

1.1 Situando o Problema e os Objetivos

A computação visual, isto é, a manipulação de imagens por computador assume formas diferentes numa grande variedade de aplicações. Ela trabalha basicamente com dois tipos de estruturas de dados que podem ser chamados de modelo e imagem. Um modelo é uma representação abstrata de um objeto, por exemplo, as equações das faces de um objeto. Uma imagem é uma matriz onde cada elemento representa a intensidade luminosa de um ponto. A computação visual pode ser subdividida em quatro grandes áreas de acordo com a estrutura de dados com a qual trabalha: computação gráfica, modelagem geométrica, visão computacional e processamento de imagem.

A computação gráfica trata do problema de converter o modelo em imagem, isto é, a geração da imagem a partir de informação não pictórica. A modelagem geométrica lida com construção de modelos, sem a preocupação de gerar a imagem correspondente. O objetivo da visão computacional ou análise de imagem, é converter uma imagem em modelo correspondente.

Por fim, o processamento de imagem lida com problemas onde tanto a entrada como a saída são imagens. Exemplos clássicos de processamento de imagem são a remoção de ruído, compactação de imagem, ajuste de brilho e contraste e enfatização de borda. Também são conhecidos o processo denominado meio-tom, que permite imprimir fotos em níveis de cinza numa impressora que só possui saída binária (branco ou preto), e o

algoritmo de escolha de mapa de cores, que permite mostrar uma imagem com 16 milhões de cores num monitor com, por exemplo, 256 cores.

As áreas de que trataremos nesta tese são o processamento e a análise de imagem ([Pratt, 1978], [Pavlidis, 1982], [Gonzalez and Woods, 1992] e [Dougherty and Astola, 1994]). Estas áreas vêm sendo abordadas por um conjunto de técnicas de naturezas diversas, que podemos classificar em processamento linear, não-linear e "técnicas diversas" (aquelas não se encaixam nas duas categorias anteriores).

Historicamente, muitos métodos bidimensionais de processamento de imagem evoluíram a partir dos métodos unidimensionais de processamento de sinal, onde a técnica utilizada é principalmente linear. Muitos fenômenos físicos, como som, corrente elétrica alternada, onda eletromagnética, têm uma natureza que torna natural a sua análise pela transformada de Fourier e o conseqüente uso de filtros lineares. Assim, em muitos casos não há necessidade de considerar métodos não-lineares e o moderno processamento digital de sinais desenvolveu-se baseado nessa suposição. Com imagens, a questão é completamente diferente. Uma imagem não pode ser modelada naturalmente como uma somatória de senóides bidimensionais. Mesmo assim, a filtragem linear pode ser utilizada em algumas aplicações de imagem, permitindo por exemplo tornar uma imagem mais nítida ou, no processo inverso, suavizar a imagem. Utilizando métodos lineares também pode-se ampliar ou reduzir imagens para qualquer escala ou mesmo detectar seus gradientes.

Apesar de que, do ponto de vista semântico, qualquer processamento de imagem possa ser chamado de não-linear, historicamente, a terminologia é associada a algumas técnicas específicas. Segundo [Dougherty and Astola, 1994], a filtragem não-linear desenvolveu-se seguindo três linhas independentes. Estes filtros têm sido baseados em estruturas geométricas (operadores morfológicos), ordenação numérica (filtros do tipo mediana) e lógica (filtros de "pilha"). Além dos processamentos linear e não-linear, há uma série de técnicas já consagradas que não se enquadram em nenhuma das duas categorias anteriores. O livro [Pavlidis, 1982] apresenta uma série delas: equalização de

histograma, segmentação pelo crescimento de semente, projeção, reconstrução a partir das projeções, preenchimento de contorno, algoritmo de afinamento, etc.

A morfologia matemática é uma das sub-áreas do processamento não-linear de imagem. Por volta do ano de 1964, na École National Supérieure des Mines de Paris, Matheron e Serra decidiram experimentar uma abordagem original para análise e processamento de imagem: extrair informação de uma imagem a partir de transformações de formas, realizadas através de dois operadores ou filtros elementares, que eles denominaram dilatação e erosão. As bases teóricas foram apresentadas nos livros [Matheron, 1975] e [Serra, 1982].

A morfologia para imagens binárias foi formalizada pelos próprios Matheron e Serra nos primeiros anos de pesquisa. Posteriormente, as idéias estabelecidas para operadores sobre subconjuntos (imagens binárias) foram estendidas para operadores sobre funções (imagens em níveis de cinza) nos trabalhos [Serra, 1982] e [Sternberg, 1986]. A partir da década de oitenta, Matheron e Serra perceberam que os resultados obtidos para conjuntos e funções tinham essencialmente um fator comum: dependiam de uma relação de ordem, a inclusão, no caso de subconjuntos, ou a relação herdada da ordem dos seus contradomínios, no caso das funções. Este fator motivou a generalização da teoria para o domínio dos reticulados completos: conjuntos equipados com uma relação de ordem parcial e tais que qualquer subconjunto possui um supremo e um ínfimo ([Serra, 1988], [Heijmans and Ronse, 1990] e [Ronse and Heijmans, 1990]).

Um problema prático da morfologia matemática é a dificuldade para projetar um operador. Para superar esta dificuldade, algumas ferramentas foram propostas. Estas ferramentas traduzem o conhecimento do usuário, expresso num alto nível de abstração, numa seqüência de operações morfológicas que executa a tarefa desejada. Neste sentido, os trabalhos [Schmitt, 1989a] e [Schmitt, 1989b] propõem utilizar as técnicas de inteligência artificial e o trabalho [Kim et al., 1997] propõe o uso de sistema especialista nebulosa.

Outros trabalhos ([Dougherty, 1992a], [Dougherty, 1992b], [Loce, 1993], [Barrera et al., 1995], [Tomita, 1996] e [Harvey and Marshall, 1996]) expressam o conhecimento do usuário através de exemplos da tarefa a ser executada. Nesses trabalhos, o usuário deve obter algumas imagens de entrada típicas da aplicação juntamente com as imagens de saída correspondentes. Estas imagens alimentam um sistema que constrói automaticamente o operador desejado. Por exemplo, suponha que queremos obter um operador morfológico que elimina ruído com uma determinada distribuição estatística desconhecida. Então primeiro devemos obter uma série de imagens com ruído, juntamente com as imagens limpas correspondentes. A partir destas imagens, um algoritmo de aprendizagem gera automaticamente o operador que elimina o ruído. Os trabalhos [Dougherty, 1992a], [Dougherty, 1992b] e [Loce, 1993] descrevem técnicas para projetar um operador que minimiza o erro médio (quadrático ou absoluto) das imagens exemplos. Os trabalhos [Barrera et al., 1995] e [Tomita, 1996] descrevem um algoritmo para projetar automaticamente operadores binários. O artigo [Harvey and Marshall, 1996] utiliza um algoritmo genético no projeto de operador a partir das imagens-exemplos. Nesta tese, propomos novas técnicas de projeto automático de operadores. Uma parte do trabalho presente está publicada em [Kim, 1997].

Este projeto automático de operador está formalizado nesta tese como uma aprendizagem computacional. De acordo com [Anthony and Biggs, 1992], a teoria de aprendizagem computacional é uma das primeiras tentativas de se construir um modelo matemático do processo cognitivo. Esta pode ser descrita como uma teoria macro, pois fornece o embasamento teórico para estudar uma grande variedade de processos algorítmicos tais como treinamento de redes neurais ou algoritmos genéticos. Esta teoria é relativamente recente e é útil na formalização dos processos de aprendizagem porque captura a essência desses processos, que antes eram formulados em termos somente vagos, e permite demonstrar algumas proposições matemáticas não-triviais.

As bases da teoria de aprendizagem computacional PAC (provavelmente aproximadamente correta) deve ser creditada ao trabalho [Valiant, 1984]. Valiant fornece também uma definição informal para aprendizagem: “Diz-se que um programa de computador

para efetuar uma tarefa foi adquirido pela aprendizagem se foi adquirido por qualquer meio exceto pela programação explícita." Os trabalhos [Barrera et al., 1995] e [Tomita, 1996] utilizam este modelo para formalizar a aprendizagem de operadores morfológicos binários. O modelo de aprendizagem PAC foi generalizado de forma abrangente no trabalho [Haussler, 1992] e é o modelo que será utilizado neste trabalho para descrever a aprendizagem de operadores morfológicos não necessariamente binários. O modelo de aprendizagem vizinho mais próximo ([Cover and Hart, 1967]) é anterior aos modelos de Valiant e Haussler e este modelo pode ser encarado como um caso particular do modelo de Haussler. Porém, a idéia central do modelo vizinho mais próximo, qual seja, a de aproximar uma instância desconhecida por um vizinho mais próximo, inexiste no modelo de Haussler. Utilizaremos esta idéia no projeto dos operadores.

Os algoritmos de busca do vizinho mais próximo têm aplicação em diversas áreas, desde banco de dados até reconhecimento de voz. A área em que este problema é estudado teoricamente é a geometria computacional. Existem vários algoritmos conhecidos para buscar rapidamente o vizinho mais próximo num espaço de dimensão alta, entre eles [Bentley, 1975], [Friedman et al., 1977], [Sproull, 1991] e [Djouadi and Bouktache, 1997]. Também é possível utilizar o diagrama de Voronoi ([Preparata and Shamos, 1985]), apesar de uma implementação prática desta técnica para uma dimensão arbitrária ser extremamente complicada. Nesta tese, utilizaremos o algoritmo conhecido como kd-árvore, descrito em [Friedman et al., 1977]. Descrevemos também uma alteração no algoritmo de Friedman que o torna mais rápido, em detrimento da qualidade, obtendo um operador um pouco "pior" do que o algoritmo original, mas consideravelmente mais veloz.

Existem muitos estudos para acelerar os algoritmos de morfologia matemática. A tese [Vincent, 1990] e os artigos [Bleau et al., 1992a], [Bleau et al., 1992b] e [Jones and Svalbe, 1994a] apresentam estas técnicas. Em particular, o último artigo utiliza uma "look-up table" para acelerar a aplicação de operador morfológico binário. A busca do vizinho mais próximo e o uso da kd-árvore pode ser considerada uma alternativa para acelerar o processamento de imagens não necessariamente binárias.

Resumindo, esta tese tem como tripé de apoio três áreas distintas: morfologia matemática, aprendizagem computacional e os algoritmos de busca do vizinho mais próximo.

1.2 Contribuições desta Tese

Este trabalho pode ser considerado como "multidisciplinar". Podemos citar como a principal contribuição desta tese a integração da morfologia matemática, aprendizagem computacional e os algoritmos de busca do vizinho mais próximo visando resolver um problema concreto de processamento de imagem: projetar um filtro digital automaticamente, utilizando imagens amostras.

Com isso chegamos a novas técnicas, computacionalmente mais eficientes que os algoritmos anteriormente conhecidos, para solucionar este problema. Esta melhora de desempenho pode ser observada (na análise de complexidade e na prática) tanto no projeto do operador como na aplicação do mesmo. Para isso, utilizamos duas estruturas de dados baseadas em árvore binária: kd-árvore e árvore de cortes.

Outra contribuição é que a construção de operador pela aprendizagem computacional foi estendida, com base nas técnicas conhecidas que processam exclusivamente imagens binárias e em níveis de cinza, para reticulado completo. Esta tese utiliza como base teórica a morfologia matemática para reticulados completos, o que torna os algoritmos descritos nesta tese aplicáveis para praticamente todos os tipos de imagens: binárias, em níveis de cinza, coloridas, multi-espectrais, animações, tridimensionais, etc.

Propomos uma nova forma de representar operador morfológico que denominamos representação por intervalos. Demonstramos que o esquema proposto é um caso especial da representação pelo supremo de sup-geradores, descrita em [Banon and Barrera, 1993]. Também mostramos que, no caso binário, os operadores elementares utilizados no novo esquema são exatamente equivalentes à erosão e dilatação tradicionais. Mos-

tramos que nos casos de utilidade prática uma representação por intervalos pode ser armazenada numa árvore binária para acelerar o processamento computacional.

Os trabalhos [Barrera et al., 1995] e [Tomita, 1996] utilizam o modelo PAC clássico (modelo de Valiant) para formalizar a aprendizagem computacional. Isso restringe o campo de trabalho a operadores binários apenas e não permite lidar com amostras que possuam conflitos (veja capítulo 3). Nesta tese, utilizamos o modelo PAC generalizado de Haussler. O modelo de Haussler dá uma sustentação teórica sólida que permite formalizar a aprendizagem de operadores não-binários assim como lidar com conflitos. Introduzimos a função minimizadora de penalidade o que permitiu estender o conceito de consistência do modelo de aprendizagem de Valiant para o de Haussler, utilizado para mostrar a convergência do processo de aprendizagem.

1.3 Organização do Trabalho

Esta tese está dividida em sete capítulos:

1. **Introdução:** Apresenta o problema a ser estudado, ressalta as contribuições e descreve o esqueleto do trabalho.
2. **Morfologia Matemática:** Este capítulo expõe a teoria da morfologia matemática para reticulado. Apresenta brevemente a teoria dos reticulados e define as classes de operadores e as relações entre elas. Expõe os esquemas conhecidos para representar operadores e propõe a representação por intervalos. Define a imagem digital, o operador restrito à janela e o operador invariante por translação. Demonstra que, no caso binário, os operadores elementares utilizados na representação por intervalos são exatamente a erosão e a dilatação tradicionais.
3. **Aprendizagem Computacional:** Após a introdução, apresenta o modelo de aprendizagem de Haussler e analisa as relações entre o problema de aprendizagem e de otimização. Define a função minimizadora de penalidade, o conceito de consistência e o algoritmo de aprendizagem consistente.

4. **Aprendizagem de Operador:** O capítulo começa com um exemplo de aplicação do projeto automático do operador pela aprendizagem. Depois, define formalmente o problema de aprendizagem de operador. Descreve os algoritmos de aprendizagem que podem ser utilizados para resolver o problema. Por fim, discute a escolha conveniente da janela.
5. **Vizinho Mais Próximo:** Apresenta o modelo de aprendizagem vizinho mais próximo como um caso especial do modelo de Haussler. Distingue algoritmo de vizinho mais próximo consistente do não-consistente e demonstra que somente o primeiro é PAC. Define a distribuição suave e argumenta por que o vizinho mais próximo é interessante no projeto automático do operador. Utiliza a kd-árvore como algoritmo de busca do vizinho mais próximo.
6. **Aplicações:** Mostra algumas aplicações das técnicas estudadas e descreve certos detalhes de implementação dos algoritmos.
7. **Conclusão:** Resume os resultados obtidos.

Após os capítulos regulares do texto, apresentamos cinco anexos:

A. **Morfologia Matemática Binária:** Descreve brevemente a teoria original da morfologia matemática binária e mostra um exemplo. Este anexo pode ser de interesse para leitores não familiarizados com a morfologia matemática.
B. **Aprendizagem PAC Clássica:** Apresenta a aprendizagem PAC original. Este anexo destina-se a facilitar a compreensão dos leitores pouco familiarizados com a aprendizagem computacional.
C. **Aprendizagem Computacional e Humana:** Compara o processo de aprendizagem humana com os modelos de aprendizagem computacional. Para descrever o processo de aprendizagem humana, utilizamos a filosofia clássica.
D. **Teoria da Estimação e Aprendizagem de Operador:** O problema de aprendizagem de operador é colocado no contexto da teoria da estimação.

E. KD-Árvore: Apresenta a busca do vizinho mais próximo utilizando a estrutura kd-árvore numa linguagem algorítmica. Colocamos esta descrição num anexo para que os leitores tenham a liberdade de lê-la ou não, pois este algoritmo é bastante intrincado e a sua compreensão não é essencial para a compreensão do resto do trabalho.

2 Morfologia Matemática

2.1 Introdução

A morfologia matemática é uma das sub-áreas do processamento não-linear de imagens digitais. Assim como o projeto de um filtro linear está baseado na análise dos componentes espectrais do sinal, o projeto de um operador morfológico (filtro digital utilizado em morfologia matemática) está baseado no estudo das estruturas geométricas da imagem. Um operador morfológico está relacionado com a teoria da geometria integral [Matheron, 1975] e as estruturas geométricas de uma imagem podem ser analisadas selecionando um operador que é capaz de interagir de modo apropriado com elas.

A teoria original da morfologia matemática foi desenvolvida para imagens binárias e apresentada nos livros [Matheron, 1975] e [Serra, 1982]. Um breve resumo da morfologia binária é apresentado no anexo A. Nesta teoria, uma imagem binária é formalizada como um subconjunto de um grupo abeliano e os operadores elementares utilizados para processar imagens são a erosão e a dilatação. Os filtros digitais mais complexos são descritos utilizando estes dois operadores, juntamente com os operadores complemento, união e intersecção de conjuntos. Fazendo a composição desses operadores elementares, obtemos operadores mais elaborados, como a abertura, o fechamento, "hit-or-miss", etc. Fazendo composições ainda maiores, conseguimos obter operadores para calcular esqueleto, separar componentes por tamanho ou pela forma, eliminar ruídos, detectar linhas de fratura, fazer segmentação, etc.

A morfologia binária foi estendida para níveis de cinza no livro [Serra, 1982] e no artigo [Sternberg, 1986]. Os artigos [Haralick et al., 1987], [Heijmans, 1991], [Dougherty and Sinha, 1995a], [Dougherty and Sinha, 1995b] assim como o livro [Dougherty and Astola, 1994] também trazem a teoria de morfologia em níveis de cinza. Algumas aplicações de morfologia no processamento de imagens em níveis de cinza encontram-se em [Song and Delp, 1990], [Li and Chen, 1991], [Boomgaard and Smeulders, 1994], [Breen and Jones, 1996]. Alguns algoritmos computacionais para acelerar os algoritmos de morfologia em níveis de cinza encontram-se em [Bleau et al., 1992a] e [Bleau et al., 1992b].

Depois, para formalizar a morfologia para imagens digitais não necessariamente binárias, foi utilizada a teoria do reticulado completo. A teoria dos reticulados encontra-se em muitos livros de álgebra, como [Birkhoff and Bartee, 1970] e [MacLane and Birkhoff, 1967] ou nos livros recentes de morfologia matemática como [Heijmans, 1994]. O desenvolvimento detalhado da morfologia para reticulados encontra-se em [Serra, 1988], [Heijmans and Ronse, 1990], [Ronse and Heijmans, 1990], [Serra and Vincent, 1992] e [Heijmans, 1994]. Algumas aplicações de morfologia em reticulado encontram-se em [Overturf et al., 1995] e [Ronse, 1996]. Neste capítulo, apresentamos a morfologia para reticulados.

2.2 Reticulado Completo

Somente os reticulados finitos serão objeto do nosso estudo, pois numa aplicação real do processamento de imagem, todos os conjuntos envolvidos são finitos. Esta suposição simplifica muitos resultados.

Definição (poset): Dado um conjunto não vazio $\mathcal{L}$, uma relação binária $\leq$ em $\mathcal{L}$ é chamada uma ordem parcial se as seguintes propriedades são satisfeitas para todo $x, y, z \in \mathcal{L}$.

1) $x \leq x$; (reflexividade)

2) $x \leq y$ e $y \leq x$ implica $x = y$; (anti-simetria)

3) $x \leq y$ e $y \leq z$ implica $x \leq z$; (transitividade)

Um conjunto $\mathcal{L}$ com a ordem parcial $\leq$ é chamado de *conjunto parcialmente ordenado*, ou *poset* (do inglês "partially ordered set"), e é denotado por $(\mathcal{L}, \leq)$. Quando não há perigo de confusão, escrevemos simplesmente $\mathcal{L}$.

Notação: Dados $x, y \in \mathcal{L}$, adotaremos a seguinte notação:

- Denotamos $x < y$ quando $x \leq y$ e $x \neq y$.
- Denotamos $x \geq y$ quando $y \leq x$.
- Denotamos $x > y$ quando $y \leq x$ e $x \neq y$.

Diagrama de Hasse: Um poset $\mathcal{L}$ pode ser representado graficamente como segue: Se $A < B$, e não existe nenhum elemento $X \in \mathcal{L}$ com $A < X < B$, então colocamos B acima de A, e traçamos uma linha conectando os dois elementos. O diagrama resultante é chamado *diagrama de Hasse*. Alguns exemplos de posets representados como diagrama de Hasse estão na Figura 2.1.

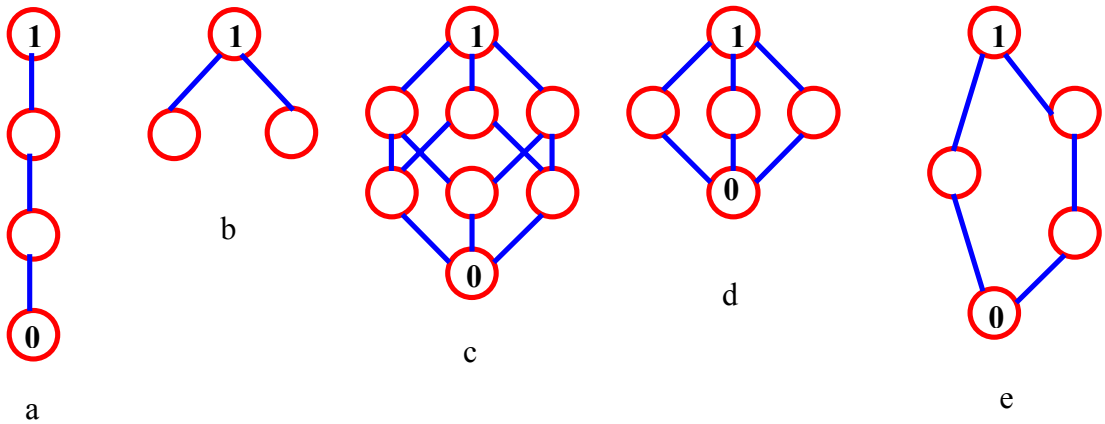


Figura 2.1: Diagrama de Hasse, utilizado para representar um poset.

Definição (cadeia): Dizemos que um poset $(\mathcal{L}, \leq)$ é totalmente ordenado se $x \leq y$ ou $y \leq x$, para todo $x, y \in \mathcal{L}$. Um poset totalmente ordenado é também chamado de *cadeia*.

Uma cadeia $(\mathcal{L}, \leq)$ é limitada se o conjunto $\mathcal{L}$ é finito. Habitualmente, utilizaremos a letra $\mathcal{K}$ para denotar uma cadeia limitada.

O diagrama da ג.a representa uma cadeia.

Princípio da dualidade: Se ≤ é uma ordem parcial em $\mathcal{L}$, então a relação binária ≥ também define uma ordem parcial, chamada *ordem parcial dual*. Se $(\mathcal{L}, \leq)$ é um poset, então $(\mathcal{L}, \geq)$ é também um poset, chamado *poset dual*. Para toda definição, propriedade, proposição em $(\mathcal{L}, \leq)$, existe uma correspondente dual em $(\mathcal{L}, \geq)$.

Definição (mínimo e máximo): Dado um poset $\mathcal{L}$ e um subconjunto $\mathcal{M} \subset \mathcal{L}$, um elemento $a \in \mathcal{M}$ é chamado de *mínimo* de $\mathcal{M}$ se $a \leq x$, para todo $x \in \mathcal{M}$. Da mesma forma, $b \in \mathcal{M}$ é *máximo* de $\mathcal{M}$ se $b \geq x$, para todo $x \in \mathcal{M}$.

Definição (limitante): Dado um poset $\mathcal{L}$ e um subconjunto $\mathcal{M} \subset \mathcal{L}$, um elemento $a \in \mathcal{L}$ é chamado de *limitante inferior* de $\mathcal{M}$ se $a \leq x$, para todo $x \in \mathcal{M}$. Da mesma forma $b \in \mathcal{L}$ *é limitante superior* de $\mathcal{M}$ se $b \geq x$, para todo $x \in \mathcal{M}$.

Definição (ínfimo e supremo): Dado um poset $\mathcal{L}$ e um subconjunto $\mathcal{M} \subset \mathcal{L}$, se o conjunto de limitantes inferiores de $\mathcal{M}$ possui um máximo, este é chamado de *ínfimo* e denotado como $\wedge\mathcal{M}$. Da mesma forma, se o conjunto de limitantes superiores de $\mathcal{M}$ possui um mínimo, este é chamado de *supremo* e denotado como $\vee\mathcal{M}$. O ínfimo de um conjunto de dois elementos $\{a, b\}$ é denotado como $a \wedge b$ e o supremo como $a \vee b$.

Definição (reticulado): Um poset $\mathcal{L}$ é chamado de *reticulado* se todo subconjunto finito de $\mathcal{L}$ possui um ínfimo e um supremo. Um poset $\mathcal{L}$ é chamado de *reticulado completo* se todo subconjunto de $\mathcal{L}$ possui um ínfimo e um supremo.

Por exemplo, a ג.b mostra um poset que não é reticulado. Como neste trabalho, trataremos somente dos conjuntos finitos, todo reticulado será completo.

Notação (0 e 1): Pelas definições acima pode-se ver que todo reticulado completo $\mathcal{L}$ deve possuir um elemento mínimo e um elemento máximo. Denotaremos estes elementos respectivamente por **0** e **1**.

Definição (intervalo): Seja $\mathcal{L}$ um poset. Dados $a, b \in \mathcal{L}$, $a \leq b$, denotaremos o intervalo fechado de $\mathcal{L}$ com extremidades a e b por

$$[a, b] = \{ x \in \mathcal{L} : a \leq x \leq b \}$$

Como o conjunto de números reais $\mathbb{R}$ e o conjunto de números inteiros $\mathbb{Z}$ também são posets podemos utilizar esta notação de intervalo nesses conjuntos. Porém, para distinguir um intervalo de reais do intervalo de inteiros, denotaremos o primeiro como $[a, b]$ e o segundo como $[a \ldots b]$.

Definição (isomorfismo de reticulados): Sejam $\mathcal{L}$ e $\mathcal{M}$ dois reticulados. O mapeamento $\psi : \mathcal{L} \rightarrow \mathcal{M}$ é chamado de isomorfismo de reticulados sse ψ é uma bijeção e se ψ e ψ^{-1} preservam a ordem, isto é, para todo $x, y \in \mathcal{L}$,

$$x \leq y \text{ sse } \psi(x) \leq \psi(y)$$

Proposição 2.1 (cadeias limitadas): Toda cadeia limitada $(\mathcal{K}, \leq)$ é isomorfa ao intervalo $[0 \ldots k\text{-}1] \subset \mathbb{Z}$, onde $k = |\mathcal{K}|$.

Prova: Basta associar o menor elemento de $\mathcal{K}$ ao número 0, o segundo menor elemento ao número 1, assim por diante, e associar o maior elemento de $\mathcal{K}$ ao número inteiro k-1. Esta associação é evidentemente bijetora e preserva a ordem. ■

A definição do que seja exatamente um operador morfológico costuma variar de um autor para outro. A mais aceita parece ser aquela do trabalho [Heijmans e Ronse, 1990]

que apresentamos abaixo. Esta definição é também a mais ampla e abrange as outras definições:

Definição (operador): Um *operador morfológico* ψ (ou abreviadamente *operador*) é uma função entre dois reticulados. Denotamos $\psi : \mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$ ou $\psi \in (\mathcal{L}^{\mathbf{y}})^{\mathcal{L}^{\mathbf{x}}}$.

Exemplo (reticulado das partes): Dado um conjunto finito E, o conjunto das suas partes, denotado por $\mathbb{P}(E)$, é um reticulado completo com a ordem parcial dada pela operação de inclusão. O ínfimo é dado pela interseção de conjuntos, e o supremo pela união. O elemento mínimo é o conjunto vazio e o elemento máximo é E.

Exemplo (reticulado das funções): Dado um reticulado completo $\mathcal{M}$ e um conjunto arbitrário não vazio E, definimos o *reticulado das funções* $\mathcal{L} = \mathcal{M}^E$ (também denotado por $\mathcal{L} = E \to \mathcal{M}$) como o espaço de todas funções de E a $\mathcal{M}$. A relação $\leq$ dada por "$F \leq G$ sse $F(x) \leq G(x)$ para todo $x \in E$" define uma ordem parcial em $\mathcal{L}$.

Exemplo (produto de reticulados): Dados d reticulados completos:

$$(\mathcal{L}_1, \leq_1), (\mathcal{L}_2, \leq_2), ..., (\mathcal{L}_d, \leq_d),$$

definimos

$$\mathcal{L} = \mathcal{L}_1 \times \mathcal{L}_2 \times ... \times \mathcal{L}_d,$$

isto é, $\mathcal{L}$ contém todas as d-uplas $(x_1, x_2, ..., x_d)$ onde $x_k \in \mathcal{L}_k$, $k = 1, 2, ..., d$. Também definimos a relação $\leq$ em $\mathcal{L}$ através de:

$$(x_1, x_2, ..., x_d) \leq (y_1, y_2, ..., y_d), \text{ sse } x_k \leq_k y_k \text{ para todo } k = 1, 2, ..., d.$$

$(\mathcal{L}, \leq)$ assim definido é um reticulado completo e o chamamos de *produto dos reticulados*.

Exemplo (produto de cadeias): Um reticulado muito utilizado no processamento de imagem é o produto de cadeias finitas, pois, praticamente todos os tipos de imagens utilizados na prática podem ser formalizados como um produto de cadeias. Um produto

de cadeias $\mathcal{F}$ é, portanto: $\mathcal{F} = \mathcal{K}_1 \times \mathcal{K}_2 \times ... \times \mathcal{K}_d$, onde cada $\mathcal{K}_i$ é uma cadeia finita e $\mathcal{F}$ herda a relação de ordem parcial das suas cadeias componentes, conforme definido no exemplo produto de reticulados acima.

As relações de ordem parcial definidas nos exemplos acima serão utilizadas daqui em diante neste trabalho, salvo menção em contrário.

2.3 Classes de Operadores

Resumimos nesta seção as diversas classes de operadores morfológicos. Transcrevemos abaixo as definições das classes de operadores dos trabalhos [Heijmans and Ronse, 1990] e [Banon and Barrera, 1993].

2.3.1 Definições de Classes de Operadores

Nas definições abaixo, considere $\mathcal{L}^{\mathbf{x}}$ e $\mathcal{L}^{\mathbf{y}}$ dois reticulados arbitrários.

Definição (crescente): Um operador ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$ é *crescente* (ou *isotônico*) sse para todo x e $x' \in \mathcal{L}^{\mathbf{x}}$, $x \leq x' \Rightarrow \psi(x) \leq \psi(x')$.

Definição (decrescente): Um operador ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$ é *decrescente* (ou *antitônico*) sse para todo x e $x' \in \mathcal{L}^{\mathbf{x}}$, $x' \leq x \Rightarrow \psi(x) \leq \psi(x')$.

Para as definições restantes, utilizaremos a seguinte notação:

Notação: Seja ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$ e $X \subset \mathcal{L}^{\mathbf{x}}$. Denotaremos $\psi(X) = \{ \psi(x) : x \in X \}$.

Definição: Um operador ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$ é *erosão* sse $\forall X \subset \mathcal{L}^{\mathbf{x}}$, $\psi(\wedge X) = \wedge\psi(X)$.

Definição: Um operador ψ: $\mathcal{L}^{\mathbf{x}} \rightarrow \mathcal{L}^{\mathbf{y}}$ é *dilatação* sse $\forall X \subset \mathcal{L}^{\mathbf{x}}$, $\psi(\vee X) = \vee\psi(X)$.

Definição: Um operador ψ: $\mathcal{L}^{\mathbf{x}} \rightarrow \mathcal{L}^{\mathbf{y}}$ é *anti-erosão* sse $\forall X \subset \mathcal{L}^{\mathbf{x}}$, $\psi(\wedge X) = \vee\psi(X)$.

Definição: Um operador ψ: $\mathcal{L}^{\mathbf{x}} \rightarrow \mathcal{L}^{\mathbf{y}}$ é *anti-dilatação* sse $\forall X \subset \mathcal{L}^{\mathbf{x}}$, $\psi(\vee X) = \wedge\psi(X)$.

Definição: Um operador ψ: $\mathcal{L}^{\mathbf{x}} \rightarrow \mathcal{L}^{\mathbf{y}}$ é *sup-gerador* sse $\forall X \subset \mathcal{L}^{\mathbf{x}}$, $X \neq \varnothing$, temos

$$\psi(\wedge X) \wedge \psi(\vee X) = \wedge\psi(X).$$

O conjunto de todos os sup-geradores costuma ser denotado pela letra grega Λ.

Definição: Um operador ψ: $\mathcal{L}^{\mathbf{x}} \rightarrow \mathcal{L}^{\mathbf{y}}$ é *inf-gerador* sse $\forall X \subset \mathcal{L}^{\mathbf{x}}$, $X \neq \varnothing$, temos

$$\psi(\vee X) \wedge \psi(\wedge X) = \vee\psi(X).$$

2.3.2 Relações entre as Classes de Operadores

Proposição 2.2: Erosões e dilatações são crescentes.

Prova: Veja lema 2.1 de [Heijmans and Ronse, 1990].

Utilizando a dualidade, podemos concluir que anti-erosão e anti-dilatação são decrescentes.

Proposição 2.3: Erosões e anti-dilatações são sup-geradores.

Prova: Veja pg. 307 do [Banon and Barrera, 1993].

Utilizando o princípio da dualidade, concluímos que dilatações e anti-erosões são inf-geradores.

Proposição 2.4: Seja ψ: $\mathcal{L}^x \rightarrow \mathcal{L}^y$ um operador sup-gerador, crescente e $\psi(\mathbf{1}) = \mathbf{1}$. Então ψ é uma erosão.

Prova: Veja pg. 307 do [Banon and Barrera, 1993].

Da mesma forma, se ψ: $\mathcal{L}^x \rightarrow \mathcal{L}^y$ é um operador:

- inf-gerador, crescente e $\psi(\mathbf{0}) = \mathbf{0}$ então ψ é uma dilatação;
- inf-gerador, decrescente e $\psi(\mathbf{1}) = \mathbf{0}$ então ψ é uma anti-erosão;
- sup-gerador, decrescente e $\psi(\mathbf{0}) = \mathbf{1}$ então ψ é uma anti-dilatação.

Colocando num diagrama as proposições discutidas acima, obtemos a ד.

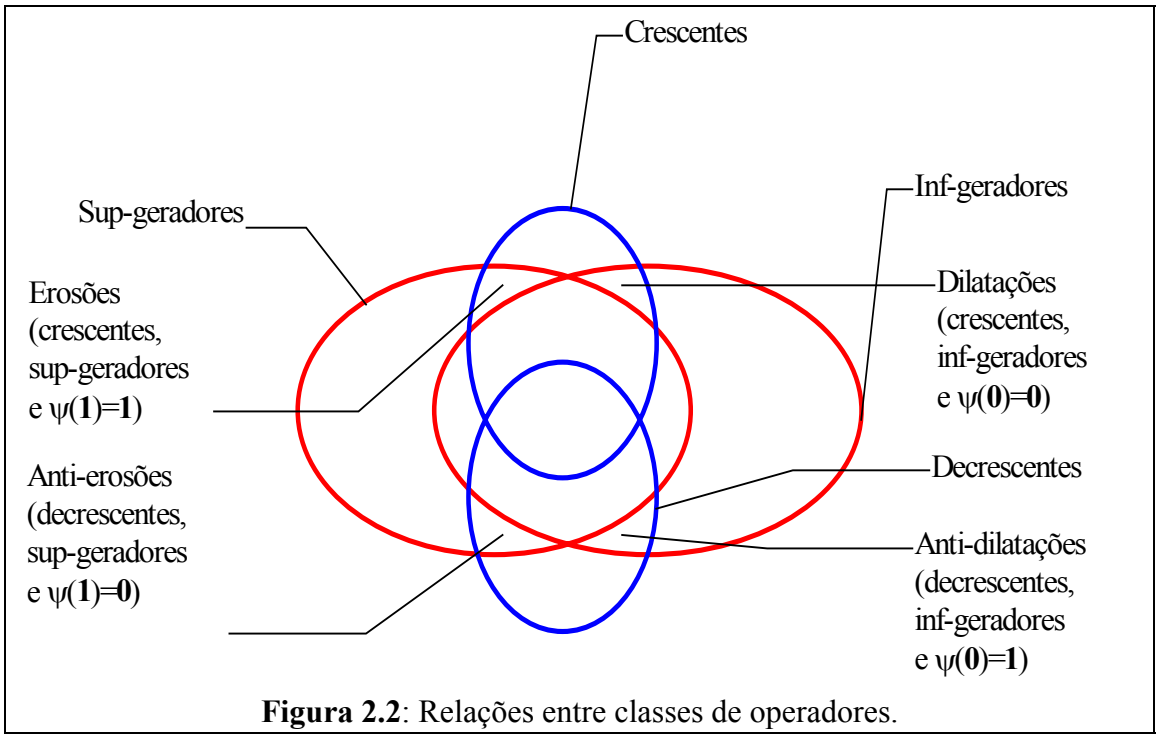


Figura 2.2: Relações entre classes de operadores.

Proposição 2.5: Todo sup-gerador pode ser escrito como o ínfimo de uma erosão e uma anti-dilatação.

Prova: Proposição 5.5 de [Banon and Barrera, 1993].

Pelo princípio da dualidade, concluímos que todo inf-gerador pode ser escrito como o supremo de uma dilatação e uma anti-erosão.

2.4 Imagem Digital e W-Operador

2.4.1 Imagem Digital

O objeto do nosso estudo são as imagens digitais e as transformações definidas sobre elas. A imagem binária está definida no anexo A. Um espaço das imagens digitais (não necessariamente binária) pode ser encarado como um reticulado particular:

Definição (imagem digital): Considere um grupo abeliano E (veja a definição no anexo A) possivelmente infinito, e um reticulado $\mathcal{L}$. Uma *imagem digital* (ou abreviadamente imagem) é uma função $E \to \mathcal{L}$ e o espaço das imagens digitais é o conjunto de todas as funções $E \to \mathcal{L}$.

Observe que, apesar de E poder ser infinito, na prática uma imagem digital é sempre finita. Pois dada uma imagem digital A, na prática sempre existe um subconjunto finito de E denominado suporte de A e denotado $S(A)$, onde a imagem realmente está definida. Fora de $S(A)$, a imagem A possui uma cor de fundo $\gamma(A)$, isto é, $A(x) = \gamma(A)$ se $x \notin S(A)$.

Naturalmente, um operador Ψ entre dois espaços de imagens digitais $E \to \mathcal{L}^{\mathbf{x}}$ e $E \to \mathcal{L}^{\mathbf{y}}$ deve ser definido como uma função entre os dois espaços, isto é, Ψ: $(\mathcal{L}^{\mathbf{x}})^E \to (\mathcal{L}^{\mathbf{y}})^E$. Utilizaremos letras gregas maiúsculas para denotar um operador para diferenciá-lo da função característica de um W-operador que definiremos na próxima subseção.

Cada elemento do conjunto $S(A)$ representa um pixel da imagem. O conjunto $S(A)$ normalmente é um intervalo fechado de $\mathbb{Z}^d$, e a imagem é conhecida como sinal

unidimensional se $d = 1$, imagem bidimensional se $d = 2$, imagem tridimensional se $d = 3$, etc. Cada elemento de $\mathcal{L}$ representa uma cor. Na maioria das imagens digitais, $\mathcal{L}$ é um produto de cadeias limitadas e chamamos cada cadeia de banda. Portanto, a quantidade de bandas de uma imagem é a quantidade de cadeias do reticulado e será denotada por b. A quantidade de elementos de uma banda é o número de níveis de intensidade daquela banda. A quantidade de elementos de $S(A)$ é conhecida como tamanho da imagem.

Observe que um espaço de imagens digitais $E \to \mathcal{L}$ é um reticulado das funções e herda a ordem parcial do reticulado $\mathcal{L}$ (veja a seção 2.2). A seguir, descrevemos alguns tipos de imagens digitais:

Exemplo (sinal digital): Um *sinal digital A* é uma função A: $\mathbb{Z} \to [0\ldots k\text{-}1]$. Observe que, na prática, um sinal digital só está definido num intervalo finito $S(A) \subset \mathbb{Z}$, pois é computacionalmente impossível armazenar um sinal infinito.

Exemplo (imagem binária): Como foi definido no anexo A, uma *imagem binária A* é uma função A: $\mathbb{Z}^2 \to \{0,1\}$. Observe que o espaço $\mathbb{Z}^2 \to \{0,1\}$ é isomorfo ao espaço das partes $\mathbb{P}(\mathbb{Z}^2)$.

Exemplo (imagem em níveis de cinza): Uma *imagem em níveis de cinza A* é uma função A: $\mathbb{Z}^2 \to \mathcal{K}$, onde $\mathcal{K}$ é uma cadeia limitada. Como toda cadeia limitada com k elementos é isomorfa ao intervalo $[0\ldots k\text{-}1] \subset \mathbb{Z}$, podemos imaginar uma imagem em níveis de cinza como uma função A: $\mathbb{Z}^2 \to [0\ldots k\text{-}1]$. k é a quantidade de níveis de cinza.

Exemplo (imagem colorida): Uma *imagem colorida A* é uma função A: $\mathbb{Z}^2 \to \mathcal{K}_1 \times \mathcal{K}_2 \times \mathcal{K}_3$, onde $\mathcal{K}_1$, $\mathcal{K}_2$ e $\mathcal{K}_3$ são três cadeias limitadas que representam as bandas

vermelha, verde e azul. Normalmente, $|\mathcal{K}_1| = |\mathcal{K}_2| = |\mathcal{K}_3| = k$ e dizemos que a imagem colorida tem k níveis de intensidade em cada uma das três bandas.

Exemplo (imagem colorida tridimensional): Uma *imagem colorida tridimensional A* é uma função A: $\mathbb{Z}^3 \rightarrow \mathcal{K}_1 \times \mathcal{K}_2 \times \mathcal{K}_3$.

Exemplo (animação colorida): Uma *animação colorida bidimensional A* é uma função A: $\mathbb{Z}^3 \rightarrow \mathcal{K}_1 \times \mathcal{K}_2 \times \mathcal{K}_3$. As três dimensões do domínio $\mathbb{Z}^3$ são linha, coluna e tempo.

Exemplo (animação multi-espectral): Uma *imagem multi-espectral* ou *multibanda A* é uma função A: $\mathbb{Z}^2 \rightarrow \mathcal{K}_1 \times \mathcal{K}_2 \times ... \times \mathcal{K}_b$, onde b é a quantidade de bandas da imagem.

2.4.2 τ-Operador e W-Operador

Definição (translação): Dada uma imagem digital $A \in (\mathcal{L})^E$ e um ponto $p \in E$, denota-se por A_p a *translação* de A por p, onde $A_p(x) = A(x\text{-}p)$.

Definição (τ-operador): Um operador Ψ:$(\mathcal{L}^{\mathbf{x}})^E \rightarrow (\mathcal{L}^{\mathbf{y}})^E$ é *invariante por translação* ou *τ-operador* sse, para todo $A \in (\mathcal{L}^{\mathbf{x}})^E$ e $p \in E$, $\Psi(A_p) = (\Psi(A))_p$.

Por exemplo, um operador que detecta as letras ‘a’, pintando-as de vermelho é um τ-operador. Por outro lado, o operador que rotaciona uma imagem não é τ-operador.

Um operador morfológico utilizado num problema de processamento de imagem é formado normalmente por muitos operadores restritos à janela (abreviado neste trabalho como W-operadores, do inglês windowed operators). Por sua vez, cada W-operador é normalmente composto por muitos operadores elementares como erosão, sup-gerador ou int-primitivo. O objetivo deste trabalho é o projeto automático de W-operadores apenas. Um W-operador é caracterizado por uma janela e por uma função característica:

Definição (W-operador): Sejam $\mathcal{L}^{\mathbf{x}}$ e $\mathcal{L}^{\mathbf{y}}$ dois reticulados, um grupo abeliano E e uma seqüência $\vec{W} = (W_1, W_2, ..., W_w)$ chamada janela, onde $W_i \in E$. Um operador $\Psi:(\mathcal{L}^{\mathbf{x}})^E \to (\mathcal{L}^{\mathbf{y}})^E$ é *restrito à janela* $\vec{W}$ (ou *W-operador*) se existe $\psi:(\mathcal{L}^{\mathbf{x}})^w \to \mathcal{L}^{\mathbf{y}}$ denominada função característica que satisfaz:

$$\Psi(A)(p) = \psi(A(W_1 + p), A(W_2 + p), \ldots, A(W_w + p)), \quad \forall A \in (\mathcal{L}^{\mathbf{x}})^E, \quad \forall p \in E$$

Por exemplo, os operadores que podem ser computados utilizando uma vizinhança do pixel são W-operadores. Como dissemos acima, o objetivo deste trabalho é o projeto de W-operadores apenas. Mas isso não restringe a generalidade da abordagem, pois todo operador composto por um número finito de W-operadores pode ser considerado um W-operador numa janela maior. Abaixo, mostramos a relação entre W-operador e τ-operador.

Proposição 2.6: Todo W-operador é τ-operador.

Prova: Seja $\Psi:(\mathcal{L}^{\mathbf{x}})^E \to (\mathcal{L}^{\mathbf{y}})^E$ um operador restrito a uma janela $\vec{W} = (W_1, W_2, ..., W_w)$. Então, existe uma função característica $\psi:(\mathcal{L}^{\mathbf{x}})^w \to \mathcal{L}^{\mathbf{y}}$. Considere dada uma imagem $A \in (\mathcal{L}^{\mathbf{x}})^E$ e um ponto $p \in E$. Então, para todo $x \in E$,

$$\begin{aligned}\Psi(A_p)(x) &= \psi(A_p(W_1 + x), \ldots, A_p(W_w + x)) = \\ &= \psi(A(W_1 + x - p), \ldots, A(W_w + x - p)) = \\ &= \Psi(A)(x - p) = \\ &= (\Psi(A))_p(x)\end{aligned}$$

Logo, Ψ é τ-operador. ■

2.5 Representação de Operador

Nesta seção apresentamos alguns esquemas descritos na literatura para representar operadores. Alguns autores chamam este mesmo assunto de decomposição do operador.

Matheron afirma em [Matheron, 1975] que qualquer operador binário crescente e invariante por translação (anexo A) pode ser representado como uma união de erosões. Depois, os trabalhos [Serra, 1988] e [Heijmans and Ronse, 1990] estenderam este resultado para reticulados completos. O trabalho [Banon and Barrera, 1991] generaliza o trabalho original de Matheron, descrevendo uma representação que permite descrever qualquer operador binário, mesmo aqueles não crescentes. O trabalho [Banon e Barrera, 1993] apresenta uma generalização do resultado anterior para reticulados completos.

Além dos trabalhos já citados, o artigo [Jones and Svalbe, 1994b] descreve alguns algoritmos para decompor operadores em níveis de cinza. Uma abordagem diferente para decompor um operador é a chamada decomposição piramidal. O trabalho [Overturf et al., 1995] utiliza a decomposição piramidal para compactar imagens coloridas.

2.5.1 Representação por Erosões

O teorema de Matheron (veja anexo A) foi estendido para reticulado completo nos trabalhos [Serra, 1988] e [Heijmans and Ronse, 1990]. Este último trabalho demonstra o seguinte teorema:

Teorema 2.1 (supremo de erosões): Seja ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$ um operador entre dois reticulados completos. Então as duas afirmações seguintes são equivalentes:

- ψ é crescente e $\psi(\mathbf{1}) = \mathbf{1}$.
- ψ é o supremo de um conjunto não-vazio de erosões.

Prova: Veja teorema 2.4 de [Heijmans and Ronse, 1990].

Na demonstração do teorema acima no artigo [Heijmans and Ronse, 1990] aparece explicitamente o conjunto de erosões cujo supremo resulta em ψ:

Corolário: Seja ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$ um operador crescente e $\psi(\mathbf{1}) = \mathbf{1}$. Então

$$\psi(x) = \bigvee_{b \neq \mathbf{1}} \varepsilon_b(x), \quad x \in \mathcal{L}^{\mathbf{x}}, \quad b \in \mathcal{L}^{\mathbf{x}},$$

onde,

$$\varepsilon_b(x) = \begin{cases} \mathbf{1}, \text{ se } x = \mathbf{1} \\ \psi(x), \text{ se } b \le x < \mathbf{1} \\ \mathbf{0}, \text{ cc} \end{cases}.$$

(Nota: abreviamos "caso contrário" como cc.)

Prova: É conseqüência direta da demonstração do teorema 2.4 de [Heijmans and Ronse, 1990].

2.5.2 Representação por Sup-Geradores

A representação por erosões descrita na subseção anterior consegue expressar somente operadores crescentes. [Banon e Barrera, 1993] apresenta um esquema para representar operadores não necessariamente crescentes:

Teorema 2.2 (sup-gerador): Seja Λ o conjunto de todos os operadores sup-geradores de $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$. Então, para todo ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$, vale $\psi = \vee \{ \lambda \in \Lambda : \lambda \le \psi \}$.

Prova: Lema 6.1 do artigo [Banon e Barrera, 1993]. Denotaremos $\Lambda_\psi = \vee\{\lambda \in \Lambda : \lambda \le \psi \}$ ■

2.6 Representação por Intervalos

Após apresentar o resultado citado no ד, o artigo [Banon e Barrera, 1993] mostra dois subconjuntos de Λ_ψ cujo supremo continua resultando ψ, que denominaram de decomposição construtiva e base. Porém, não apresentaremos aqui esses resultados pois escolhemos subconjuntos de Λ_ψ, diferentes daqueles citados em [Banon e Barrera, 1993], para representar um operador. A representação proposta por nós tem por finalidade permitir o seu armazenamento numa árvore binária, o que acelera a construção e a aplicação de operadores (capítulo 4).

2.6.1 Operadores de Intervalo

Utilizamos operadores elementares muito simples, baseados na noção de intervalo e por nós denominados de int-primitivos, como blocos construtores dos operadores morfológicos. Claramente, a noção do intervalo é inerente aos conjuntos que possuem uma ordem parcial.

Definição (int-primitivo): Um operador $\lambda : \mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$ é um *primitivo intervalar* (que abreviaremos por *int-primitivo*) se existem $a, b \in \mathcal{L}^{\mathbf{x}}$, $a \leq b$ e $v \in \mathcal{L}^{\mathbf{y}}$ tais que para todo $x \in \mathcal{L}^{\mathbf{x}}$:

$$\lambda(x) = \begin{cases} v \text{ se } a \leq x \leq b \\ \mathbf{0} \text{ cc} \end{cases}$$

Este operador é denotado como $\lambda_{[a,b],v}$ e o conjunto de todos os int-primitivos de $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$ como $\mathcal{I}$.

Abaixo, mostramos a relação do int-primitivo com as classes de operadores conhecidos.

Proposição 2.7 ($\mathcal{I} \subset \Lambda$): Todo int-primitivo é sup-gerador.

Prova: Mostraremos que $\lambda \in \mathcal{I}$ implica $\lambda \in \Lambda$. Seja $\lambda \in \mathcal{I}$. Então, pela definição, $\exists\, a,b \in \mathcal{L}^{\mathbf{x}}$, $a \leq b$ e $\exists v \in \mathcal{L}^{\mathbf{y}}$ tais que

$$\lambda_{[a,b],v}(x) = \begin{cases} v \text{ se } a \leq x \leq b \\ \mathbf{0} \text{ cc} \end{cases} \quad (x \in \mathcal{L}^{\mathbf{x}})$$

Seja $X \subset \mathcal{L}^{\mathbf{x}}$, $X \neq \varnothing$. Então, temos:

$$\bigwedge \lambda\,(X) = \begin{cases} v \text{ se para } \forall x \in X, \text{ temos } a \leq x \leq b \\ \mathbf{0} \text{ cc} \end{cases}$$

Equivalentemente,

$$\bigwedge \lambda\,(X) = \begin{cases} v \text{ se } a \leq \wedge X \leq b \quad \text{e} \quad a \leq \vee X \leq b \\ \mathbf{0} \text{ cc} \end{cases}$$

Por outro lado,

$$\lambda\,(\wedge X) = \begin{cases} v \text{ se } a \le \wedge X \le b \\ \mathbf{0} \text{ cc} \end{cases}$$

$$\lambda\,(\vee X) = \begin{cases} v \text{ se } a \le \vee X \le b \\ \mathbf{0} \text{ cc} \end{cases}$$

Logo, para $\forall X \subset \mathcal{L}^{\mathbf{x}}$, $X \neq \varnothing$, temos $\lambda(\wedge X) \wedge \lambda(\vee X) = \wedge\lambda(X)$ e, portanto, λ é sup-gerador.■

Como pela ט todo int-primitivo $\lambda_{[a,b],v}$ é sup-gerador, utilizando a ר concluímos que $\lambda_{[a,b],v}$ pode ser expresso como o ínfimo de uma erosão e uma anti-dilatação. De fato, $\lambda_{[a,b],v} = \lambda_{[a,\mathbf{1}],v} \wedge \lambda_{[\mathbf{0},b],v}$. Note que $\lambda_{[a,\mathbf{1}],v}$ é uma erosão e $\lambda_{[\mathbf{0},b],v}$ é uma anti-dilatação.

Proposição 2.8 ($\lambda_{[a,\mathbf{1}],v}$ é erosão): Para todo $a \in \mathcal{L}^{\mathbf{x}}$ e $v \in \mathcal{L}^{\mathbf{y}}$, $\lambda_{[a,\mathbf{1}],v}$ é uma erosão.

Prova: Devemos mostrar que $\forall X \subset \mathcal{L}^{\mathbf{x}}$, temos $\lambda_{[a,\mathbf{1}],v}(\wedge X) = \wedge\lambda_{[a,\mathbf{1}],v}(X)$. Seja $X \subset \mathcal{L}^{\mathbf{x}}$. Então:

$$\wedge\lambda_{[a,\mathbf{1}],v}\,(X) = \begin{cases} v \text{ se para } \forall x \in X, \text{ temos } a \le x \\ \mathbf{0} \text{ cc} \end{cases} \tag{1}$$

Por outro lado,

$$\lambda_{[a,\mathbf{1}],v}\,(\wedge X) = \begin{cases} v \text{ se } a \le \wedge X \\ \mathbf{0} \text{ cc} \end{cases} \tag{2}$$

Ora, $a \le \wedge X$ sse para todo $x \in X$, $a \le x$. Portanto, as expressões (1) e (2) são iguais. ■

De forma análoga, demonstra-se que $\lambda_{[\mathbf{0},b],v}$ é anti-dilatação. Denominamos $\lambda_{[a,\mathbf{1}],v}$ e $\lambda_{[\mathbf{0},b],v}$. de erosão intervalar e anti-dilatação intervalar, respectivamente.

É possível definir o operador dual do int-primitivo como segue:

Definição (int-dual): Um operador $\mu : \mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$ é um *primitivo intervalar dual* (que abreviaremos por *int-dual*) se existem $a, b \in \mathcal{L}^{\mathbf{x}}$, $a \le b$ e $v \in \mathcal{L}^{\mathbf{y}}$ tais que para todo $x \in \mathcal{L}^{\mathbf{x}}$:

$$\mu(x) = \begin{cases} v \text{ se } a \le x \le b \\ \mathbf{1} \text{ cc} \end{cases}$$

Este operador é denotado como $\mu_{[a,b],v}$.

É possível demonstrar, seguindo um raciocínio semelhante ao utilizado na demonstração da ʊ, que todo int-dual $\mu_{[a,b],v}$ é inf-gerador. Utilizando a dual da ɾ, concluímos que $\mu_{[a,b],v}$ pode ser expresso como o supremo de uma dilatação e uma anti-erosão. De fato, $\mu_{[a,b],v} = \mu_{[\mathbf{0},b],v} \vee \mu_{[a,\mathbf{1}],v}$. Note que $\mu_{[\mathbf{0},b],v}$ é uma dilatação e $\mu_{[a,\mathbf{1}],v}$ é uma anti-erosão e os denominamos de dilatação intervalar e anti-erosão intervalar.

Na seção 2.6.2 mostraremos que existe um isomorfismo entre erosões binárias tradicionais e erosões intervalares binárias. Da mesma forma, no caso binário, existe um isomorfismo entre dilatações tradicionais e dilatações intervalares. Isto mostra que os operadores intervalares possuem uma forte relação com a morfologia tradicional.

2.6.2 Operadores de Intervalo Binários

Nesta seção, relacionaremos a erosão intervalar e a dilatação intervalar definidos na subseção 2.6.1 com os operadores binários tradicionais: erosão e dilatação. A morfologia binária está apresentada com mais detalhes no anexo A.

Uma imagem binária é tradicionalmente definida em morfologia como um subconjunto de um grupo abeliano *E*, isto é, um elemento de $\mathbb{P}(E)$. Conforme definimos no anexo A, a erosão binária da imagem *A* pela imagem *B* (chamada elemento estruturante) é:

$$\varepsilon_B(A) = A \ominus B = \{x \in E \mid B_x \subset A\}.$$

Em outras palavras,

$$x \in \varepsilon_B(A) \text{ sse } B_x \subset A.$$

Considere que elemento estruturante B=$\{W_1, W_2, ..., W_w\}$. Então:

$$x \in \varepsilon_B(A) \text{ sse } W_1 + x \in A \text{ e } W_2 + x \in A \text{ e } \cdots \text{ e } W_w + x \in A.$$

Podemos considerar uma imagem binária como função $E \to \{0, 1\}$, ao invés de um subconjunto de E. Com essa notação, podemos escrever:

$$\varepsilon_B(A)(x) = A(W_1 + x) \wedge A(W_2 + x) \wedge \cdots \wedge A(W_w + x).$$

Isto é, toda erosão com elemento estruturante B pode ser vista como um operador restrito a uma janela $\vec{W}$ com $\text{conj}(\vec{W}) = B$, onde a $\text{conj}(\vec{W})$ indica o conjunto correspondente à seqüência $\vec{W}$. A sua função característica, que pertence ao espaço $\{0,1\}^w \to \{0,1\}$, é a erosão intervalar $\lambda_{[\mathbf{1},\mathbf{1}],1}$.

A erosão ε_B pode ser representada como um W-operador, onde a função característica é um int-primitivo, mesmo quando B estiver contido em $\text{conj}(\vec{W})$. Seja $B \subset \text{conj}(\vec{W})$. Vamos definir:

$$w_i = \begin{cases} 1, \text{ se } W_i \in B \\ 0, \text{ cc} \end{cases}$$

Então, a erosão pode ser escrita como:

$$\varepsilon_B(A)(x) = \bigwedge_{i=1}^{w} \left(\overline{w_i} \vee A(W_i + x)\right),$$

onde $\overline{w_i}$ indica o complemento binário de w_i. Isto é, a erosão ε_B é um W-operador onde a função característica é a erosão intervalar:

$$\lambda_{[(w_1, w_2, \ldots, w_w),(1,1,\ldots,1)],1}$$

Por exemplo, considere $E=\mathbb{Z}^2$, $B=\{(0,0), (0,1)\}$ e $\vec{W}=[(0,0), (1,0), (0,1)]$. Então, $w_1=1$, $w_2=0$ e $w_3=1$. Logo,

$$\begin{aligned} \varepsilon_B(A)(x) &= \left(0 \vee A(W_1 + x)\right) \wedge \left(1 \vee A(W_3 + x)\right) \wedge \left(0 \vee A(W_3 + x)\right) = \\ &= A(W_1 + x) \wedge A(W_3 + x) \end{aligned}$$

Assim, este ε_B é um W-operador onde a função característica é a erosão intervalar $\lambda_{[(1,0,1),(1,1,1)],1}$.

Fazendo um raciocínio inverso, chegamos à conclusão de que, qualquer W-operador binário cuja função característica é uma erosão intervalar, é uma erosão binária. Isto

permite-nos afirmar que existe um isomorfismo entre os W-operadores binários restritos a uma janela $\vec{W} \in E^w$, cuja função característica é uma erosão intervalar, e as erosões binárias cujo elemento estruturante está contido em $\text{conj}(\vec{W})$.

De forma análoga, pode-se mostrar que toda dilatação binária δ_B com elemento estruturante B é um W-operador com $\text{conj}(\vec{W}) = -B$ e a sua função característica é a dilatação intervalar $\mu_{[\mathbf{0},\mathbf{0}],0}$. Escolhendo uma janela $\vec{W}$ que contém $-B$, Vamos definir:

$$w_i = \begin{cases} 1, \text{ se } W_i \in -B \\ 0, \text{ cc} \end{cases}$$

Então, a dilatação δ_B pode ser escrita como:

$$\delta_B(A)(x) = \bigvee_{i=1}^{w} \left(w_i \wedge A(W_i + x) \right)$$

Isto é, δ_B é um W-operador com a seguinte função característica:

$$\mu_{[(0,0,\ldots,0),(\overline{w}_1,\overline{w}_2,\ldots,\overline{w}_w)],0}$$

Por exemplo, considere $E=\mathbb{Z}^2$, $B=\{(0,0), (0,1)\}$ e $\vec{W}=[(0,0), (0,1), (0,-1)]$. Então, $w_1=1$, $w_2=0$ e $w_3=1$. Logo,

$$\begin{aligned} \delta_B(A)(x) &= \left(1 \wedge A(W_1 + x)\right) \vee \left(0 \wedge A(W_3 + x)\right) \vee \left(1 \wedge A(W_3 + x)\right) = \\ &= A(W_1 + x) \vee A(W_3 + x) \end{aligned}$$

que é um W-operador onde a função característica é a dilatação intervalar $\mu_{[(0,0,0),(0,1,0)],0}$.

As discussões da presente subseção mostram que os operadores de intervalo definidos na subseção 2.6.1 generalizam a morfologia binária. Pois, quando aplicados a imagens binárias, estes operadores são equivalentes a dilatações e erosões utilizadas tradicionalmente como operadores elementares na morfologia binária.

2.6.3 Representação por Intervalos

A *representação por intervalos*, um novo esquema de representação de operador, é apresentada nesta subseção. A representação por intervalos é capaz de expressar qualquer mapeamento entre dois reticulados completos arbitrários.

Definição (representação por intervalos): Seja $\psi \in \mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$ e $\mathcal{B} \subset \mathcal{I}$. Diremos que a expressão $\vee\mathcal{B}$ é uma *representação por intervalos* de ψ se $\vee\mathcal{B} = \psi$.

A seguir, descrevemos como construir algumas representações por intervalos.

Definição (int-núcleo): Dado um operador ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$, chamaremos de *int-núcleo* de ψ, denotado por $\mathcal{I}_\psi$, o seguinte conjunto de int-primitivos.

$$\mathcal{I}_\psi = \{ \lambda_{[a,b],v} \in \mathcal{I} : a \le x \le b \Rightarrow \psi(x) = v \}$$

Proposição 2.9 (int-núcleo): Seja $\mathcal{I}_\psi$ o int-núcleo de um operador ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$. Então $\vee\mathcal{I}_\psi = \psi$, isto é, $\vee\mathcal{I}_\psi$ é uma representação por intervalos de ψ.

Prova: ($\le$) Qualquer que seja $\lambda_{[a,b],v} \in \mathcal{I}_\psi$,

$$\lambda_{[a,b],v}(x) = \begin{cases} \psi(x) \text{ se } a \le x \le b \\ \mathbf{0} \text{ cc} \end{cases}$$

Logo, $\forall x \in \mathcal{L}^{\mathbf{x}}$, $\lambda_{[a,b],v}(x) \le \psi(x)$ para todo $\lambda_{[a,b],v} \in \mathcal{I}_\psi$ e conseqüentemente $(\vee\mathcal{I}_\psi)(x) \le \psi(x)$.

($\ge$) Por outro lado, $\forall x \in \mathcal{L}^{\mathbf{x}}$, $\lambda_{[x,x],\psi(x)} \in \mathcal{I}_\psi$ e $\lambda_{[x,x],\psi(x)}(x) = \psi(x)$. Logo, $(\vee\mathcal{I}_\psi)(x) \ge \psi(x)$. ■

[Heijmans, 1994] diz que um subconjunto $\mathcal{H}$ de $\mathcal{L}$ é chamado de família sup-geradora se todo elemento de $\mathcal{L}$ é um supremo de elementos de $\mathcal{H}$. Neste sentido, a א mostra que o conjunto $\mathcal{I}$ de todos os int-primitivos é uma família sup-geradora do espaço $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$ e

está contida em Λ. De forma dual, o conjunto de todos os int-duais forma uma família inf-geradora.

O seguinte Corolário mostra a relação entre o int-núcleo e a representação Λ_ψ.

Corolário (int-núcleo é representação por sup-geradores): Para todo operador ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$ tem-se $\mathcal{I}_\psi \subset \Lambda_\psi$.

Prova: Seja $\lambda_{[a,b],v} \in \mathcal{I}_\psi$. Já sabemos (ט) que $\lambda_{[a,b],v} \in \Lambda$. Por outro lado, na primeira parte da demonstração da א demonstramos que $\forall x \in \mathcal{L}^{\mathbf{x}}$, $\lambda_{[a,b],v}(x) \le \psi(x)$. Portanto, $\lambda_{[a,b],v} \in \{ \lambda \in \Lambda : \lambda \le \psi \}$. ■

Dado um operador ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$, provamos que o seu int-núcleo $\mathcal{I}_\psi$ é uma representação de ψ, pois $\vee\mathcal{I}_\psi = \psi$. Será que não existe uma representação mais econômica? A resposta é positiva, ou seja, existe um subconjunto próprio de $\mathcal{I}_\psi$, digamos B, tal que $\vee B = \psi$. De fato, há mais de uma forma de se construir este subconjunto B. O int-núcleo maximal definido a seguir é uma delas.

Definição (int-núcleo maximal): Dado um operador ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$, chamaremos de *int-núcleo maximal* de ψ, denotado por $\mathcal{I}_\psi^*$, ao conjunto de todos os int-primitivos maximais contidos em $\mathcal{I}_\psi$. Um int-primitivo $\lambda_{[a,b],v} \in \mathcal{I}_\psi$ é maximal (em $\mathcal{I}_\psi$) sse $\nexists \lambda_{[c,d],v} \in \mathcal{I}_\psi$ tal que $[a,b] \subsetneq [c,d]$.

Proposição 2.10 (int-núcleo maximal): Seja $\mathcal{I}_\psi^*$ o int-núcleo maximal de um operador ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$. Então $\vee\mathcal{I}_\psi^* = \psi$.

Prova: Como já sabemos que $\vee\mathcal{I}_\psi = \psi$ (א), basta demonstrar que $\vee\mathcal{I}_\psi = \vee\mathcal{I}_\psi^*$.

(≥) Como $\mathcal{I}_\psi \supset \mathcal{I}_\psi^*$, temos $\vee\mathcal{I}_\psi \ge \vee\mathcal{I}_\psi^*$.

($\leq$) Por outro lado, para todo $\lambda_{[a,b],v} \in \mathcal{I}_\psi$, existe um $\lambda_{[c,d],v} \in \mathcal{I}_\psi^*$ tal que $[a,b] \subset [c,d]$. Logo, $\vee\mathcal{I}_\psi \leq \vee\mathcal{I}_\psi^*$. ■

A seguir, apresentamos uma outra forma de se construir um subconjunto de $\mathcal{I}_\psi$ cujo supremo resulta em ψ. Para isso, particionaremos o espaço $\mathcal{L}^{\mathbf{x}}$ em intervalos disjuntos. O livro [Serra, 1988, pg. 15] define a partição de um conjunto E e observa que a família de todas as partições com a relação de ordem parcial definida abaixo forma um reticulado.

Definição (reticulado das partições): Seja E um conjunto arbitrário. Podemos representar uma partição T como um mapeamento de E em $\mathbb{P}(E)$ tal que:

1. $\forall x \in E$ tem-se $x \in T(x)$,
2. $\forall (x, y) \in E^2$ tem-se $T(x) = T(y)$ ou $T(x) \cap T(y) = \varnothing$,

onde a imagem $T(x)$ é a classe de partição que contém o ponto x. Pode-se associar a seguinte relação de ordem entre duas partições T, T' de E:

$T \leq T'$ sse $T(x) \subset T'(x)$, $\forall x \in E$.

Definição (int-partição): Dado um operador ψ: $\mathcal{L}^{\mathbf{x}} \rightarrow \mathcal{L}^{\mathbf{y}}$, diremos que um conjunto de int-primitivos

$$\breve{\mathcal{I}}_\psi = \left\{\lambda_{[a_1,b_1],v_1}, \lambda_{[a_2,b_2],v_2}, \ldots, \lambda_{[a_n,b_n],v_n}\right\}$$

é uma representação de ψ pela partição por intervalos do domínio $\mathcal{L}^{\mathbf{x}}$, abreviadamente *int-partição* de ψ, se as duas condições abaixo são satisfeitas:

1. O mapeamento definido $T(x) = [a_i, b_i]$, se $x \in [a_i, b_i]$, é uma partição de $\mathcal{L}^{\mathbf{x}}$.
2. $\psi = \vee\breve{\mathcal{I}}_\psi$.

Proposição 2.11 (int-partição): Dado qualquer operador ψ: $\mathcal{L}^{\mathbf{x}} \rightarrow \mathcal{L}^{\mathbf{y}}$, existe um conjunto de int-primitivos que é uma int-partição de ψ.

Prova: Considere o seguinte conjunto:

$$\left\{\lambda_{[a,a],v} \in \mathcal{I} : a \in \mathcal{L}^{\mathbf{x}}, \psi(a) = v\right\}$$

Ora, esse conjunto de int-primitivos particiona $\mathcal{L}^{\mathbf{x}}$ e o seu supremo resulta ψ. ■

Proposição 2.12 (int-partição ⊂ int-núcleo): Dado um operador ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$, toda int-partição $\breve{\mathcal{I}}_{\psi}$ de ψ é um subconjunto do int-núcleo $\mathcal{I}_{\psi}$ de ψ.

Prova: Suponha, por absurdo, que $\exists\lambda_{[a,b],v} \in \breve{\mathcal{I}}_{\psi}$ mas $\lambda_{[a,b],v} \notin \mathcal{I}_{\psi}$. Então, pela definição de int-núcleo, $\exists x \in \mathcal{L}^{\mathbf{x}}$, $a \le x \le b$ mas $\psi(x) \ne v$. Como, pela hipótese, $\breve{\mathcal{I}}_{\psi}$ é uma int-partição de ψ, pela definição de int-partição,

$$\forall x \in \mathcal{L}^{\mathbf{x}},\ a \le x \le b \Rightarrow (\vee\breve{\mathcal{I}}_{\psi})(x) = \lambda_{[a,b],v}(x) = v\ .$$

Ora, pela hipótese, $\psi(x) \ne v$ e portanto $(\vee\breve{\mathcal{I}}_{\psi})(x) \ne \psi(x)$ o que contraria a condição 2 da definição de int-partição. ■

Evidentemente, a int-partição de ψ utilizada na demonstração da גי não é útil na prática, uma vez que a quantidade de int-primitivos é muito grande e o intervalo de cada int-primitivo é contém apenas um elemento. De fato, estamos interessados em que o intervalo $[a,b]$ de cada int-primitivo seja o maior possível, pois isso diminuiria o número de int-primitivos.

Definição (int-partição maximal): Dado um operador ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$, a int-partição maximal de ψ é a maior int-partição de ψ e será denotada por $\breve{\mathcal{I}}^{*}_{\psi}$.

Evidentemente, dado um operador ψ qualquer, como existe uma int-partição de ψ (גי) e como todos os conjuntos envolvidos são finitos, tem que existir uma int-partição maximal. O seguinte resultado mostra como obtê-lo.

Proposição 2.13 (int-partição maximal): Dado qualquer operador ψ: $\mathcal{L}^{\mathbf{x}} \to \mathcal{L}^{\mathbf{y}}$, existe um conjunto de int-primitivos que é uma int-partição maximal de ψ.

Prova: Considere o conjunto:

$$\breve{\mathcal{I}}_{\psi} = \left\{ \lambda_{[a,a],v} \in \mathcal{I} : a \in \mathcal{L}^{\mathbf{x}}, \psi(a) = v \right\}$$

Já vimos (גי) que esse conjunto é uma int-partição de ψ. Se não existir um conjunto $B = \left\{ \lambda_{[a_1,b_1],v}, \ldots, \lambda_{[a_n,b_n],v} \right\} \subset \breve{\mathcal{I}}_{\psi}$, $|B|>1$, tal que $[a_1,b_1] \cup \cdots \cup [a_n,b_n]=[a,b]$, $a,b \in \mathcal{L}^{\mathbf{x}}$ então $\breve{\mathcal{I}}_{\psi}$ é uma int-partição maximal. Caso contrário, considere a nova int-partição $\breve{\mathcal{I}}_{\psi}$ formado com a operação $\breve{\mathcal{I}}_{\psi} \leftarrow \left(\breve{\mathcal{I}}_{\psi} \setminus B \right) \cup \left\{ \lambda_{[a,b],v} \right\}$ e repita os passos. Esta repetição não pode ser executada infinitamente pois o conjunto $\breve{\mathcal{I}}_{\psi}$ inicial é finito e a cada passo o número de elementos de $\breve{\mathcal{I}}_{\psi}$ diminui. Logo, o último conjunto $\breve{\mathcal{I}}_{\psi}$ tem que ser uma int-partição maximal de ψ. ■

Todos os resultados descritos na subseção presente possuem respectivos resultados duais. Utilizando o int-primitivo e a int-partição, mostramos que o problema de decomposição de operador morfológico pode ter uma interpretação bastante natural. Neste trabalho, propomos representar operadores de uma forma muito simples: particionando o domínio em intervalos de forma que qualquer ponto de um determinado intervalo tenha o mesmo valor de saída.

2.6.4 Exemplo de Representação

Para que os conceitos vistos até agora fiquem mais claros, mostraremos como exemplo um operador representado de diversas formas. Considere um W-operador

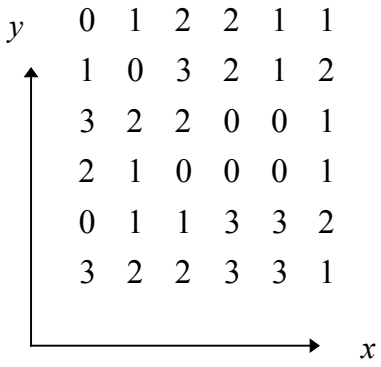
$$\Psi: (\mathbb{Z}^2 \to [0\ldots3]) \to (\mathbb{Z}^2 \to [0\ldots3]).$$

Suponha que a janela desse operador seja $\vec{W} = \left[(x_1,y_1),(x_2,y_2) \right] = \left[(0,0),(1,0) \right]$ e que a sua função característica ψ: $[0\ldots3]^2 \to [0\ldots3]$ seja definida por:

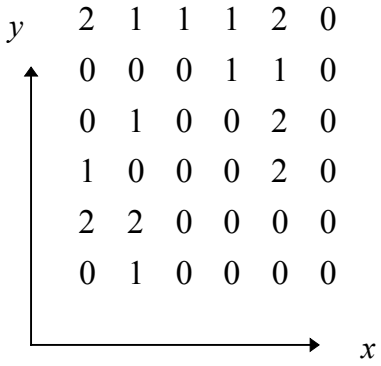
$$\psi = \begin{bmatrix} 0 & 2 & 2 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Isto é, ψ(linha=0, coluna=0) = ψ(0, 0) = 0, ψ(0, 1) = 2, ψ(2, 1) = 1, e assim por diante.

Aplicando esse operador na imagem abaixo (supondo 0 a sua cor de fundo):



Obtemos a seguinte imagem:



Pode-se imaginar os números das duas imagens acima como representando diferentes níveis de cinza. Por exemplo, 0 seria preto, 1 seria cinza escuro, 2 seria cinza claro e 3 seria branco.

O int-núcleo maximal deste operador é:

$$\mathcal{I}_{\psi}^{*} = \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 2 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 2 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right\}.$$

Isto é,

$$\mathcal{I}_{\psi}^{*} = \left\{ \lambda_{[(2,1),(2,2)],1}, \lambda_{[(1,2),(2,2)],1}, \lambda_{[(0,1),(0,2)],2}, \lambda_{[(0,1),(1,1)],2} \right\}.$$

Pela definição, $\lambda_{[(0,0),(3,0)],0}$, $\lambda_{[(3,0),(3,3)],0}$ e $\lambda_{[(0,3),(3,3)],0}$ pertenceriam ao int-núcleo maximal. Mas, como qualquer int-primitivo $\lambda_{[a,b],v}$ com valor v=**0** é uma função nula em todo o domínio, podem ser eliminados do int-núcleo, pois isto não irá alterar o valor do supremo. Assim sendo, não enumeramos tais int-primitivos. Também adotamos a mesma política na representação que segue.

Uma das int-partições maximais desse operador é

$$\psi = \bigvee \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 2 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right\},$$

que também pode ser escrita como

$$\psi = \bigvee \left\{ \lambda_{[(2,1),(2,1)],1}, \lambda_{[(1,2),(2,2)],1}, \lambda_{[(0,1),(0,2)],2}, \lambda_{[(1,1),(1,1)],2} \right\}.$$

2.7 Produto de Cadeias

Na seção 2.2 definimos um operador Ψ entre dois espaços de imagens digitais como uma função entre ambos, isto é, $\Psi: (\mathcal{L}^{\mathbf{x}})^E \rightarrow (\mathcal{L}^{\mathbf{y}})^E$. Para não sobrecarregar desnecessariamente a notação, suporemos $\mathcal{L}^{\mathbf{x}} = \mathcal{L}^{\mathbf{y}} = \mathcal{L}$. Como observamos na subseção 2.4.1, em praticamente todos os tipos de imagens digitais $\mathcal{L}$ é um produto de b cadeias limitadas. Também para não sobrecarregar a notação, suporemos que todas as b cadeias limitadas que compõe $\mathcal{L}$ são iguais e assim podemos escrever $\mathcal{L}=(\mathcal{K})^b$.

Se trabalharmos somente com W-operadores, o espaço $\mathcal{L}^E \rightarrow \mathcal{L}^E$ torna-se mais restrito, facilitando o processamento computacional. Supondo fixa uma janela $\vec{W} = (W_1, W_2, ..., W_w)$, cada operador Ψ é caracterizado inequivocamente por uma função característica $\psi: (\mathcal{K})^{bw} \rightarrow (\mathcal{K})^b$ e, denotando por $d = bw$ a dimensão do domínio das funções características, o espaço das funções características torna-se $\mathcal{K}^d \rightarrow \mathcal{K}^b$.

Ora, o nosso espaço de trabalho com isso torna-se o espaço das funções de produto de cadeias limitadas a um outro produto de cadeias limitadas. Portanto, na prática, não estamos interessados em reticulados arbitrários, mas somente em produtos de cadeias.

2.8 Conclusão

Neste capítulo apresentamos rapidamente a morfologia matemática para reticulado completo. Esta teoria é utilizada para formalizar o processamento de imagens digitais não necessariamente binárias. Depois, definimos a imagem digital e o W-operador. Descrevemos dois esquemas conhecidos de representação de operador: supremo de erosões e supremo de sup-geradores. Propusemos um novo esquema, a representação por intervalos, que utiliza os operadores de intervalo como operadores básicos (entre eles a erosão e a dilatação de intervalo). Demostramos que toda erosão binária tradicional é um W-operador binário, onde a função característica é uma erosão intervalar, e vice-versa. Resultado dual pode ser demonstrado para a dilatação binária. Mostramos que a representação por intervalos é um caso particular da representação por sup-geradores. Obtivemos (capítulo 4) algoritmos eficientes para projetar operadores automaticamente devido ao uso deste esquema. Em seguida, como exemplo, mostramos um W-operador expresso utilizando os esquemas de representação propostos. Em seguida, argumentamos por que o produto de cadeias é uma estrutura algébrica adequada para formalizar o processamento de imagem.

3 Aprendizagem Computacional

3.1 Introdução

O objetivo desta tese é a construção de operadores morfológicos pela aprendizagem computacional. O estudo da teoria de aprendizagem computacional é necessário para formalizar o problema.

Para construir operadores pelo treinamento automático, adotamos como modelo teórico a aprendizagem provavelmente aproximadamente correta generalizada (abreviada como PAC generalizada) descrita em [Haussler, 1992]. Como diz o nome, este modelo é uma generalização do modelo PAC clássico introduzido por [Valiant, 1984].

No anexo B, apresentamos o modelo de Valiant. Neste modelo, entende-se por *conceito* um subconjunto de um domínio X pré-definido. Um *exemplo* de conceito é um objeto do domínio juntamente com um rótulo indicando se este pertence ou não ao conceito. A aprendizagem do conceito é o processo na qual um *aprendiz*, também chamado *algoritmo de aprendizagem*, constrói uma boa aproximação de um conceito desconhecido, a partir de uma seqüência de exemplos de treinamento. Um critério de "boa aprendizagem" é PAC (provavelmente aproximadamente correto). Informalmente, podemos dizer que um processo de aprendizagem é PAC se, à medida em que cresce a quantidade de exemplos de treinamento, o conceito aprendido tende ao conceito alvo. Este modelo permite formalizar apenas a aprendizagem de operadores binários utilizando exemplos sem *conflitos*. Dois exemplos são *conflitantes* se ambos referem-se ao mesmo objeto do domínio mas têm rótulos contraditórios. Os trabalhos [Barrera et al., 1995] e [Tomita,

1996] utilizam o modelo de Valiant para formalizar a aprendizagem de operadores binários.

Para formalizar a aprendizagem de um operador genérico, precisamos de um modelo de aprendizagem mais abrangente. O modelo de Haussler (PAC generalizado) presta-se muito bem a esta tarefa. Nele, um conceito não é mais definido como uma função de X em $\{0,1\}$, mas como uma distribuição de probabilidade conjunta em $X \times Y$, onde X e Y, denominados respectivamente de espaço das instâncias e espaço das saídas ou rótulos, são conjuntos arbitrários. Com esta generalização, pode-se formalizar a aprendizagem de funções multi-valoradas discretas, reais e até vetoriais. O modelo generalizado também permite lidar com conflitos, ou seja, quando existem exemplos que se referem ao mesmo elemento em X mas que têm rótulos em Y diferentes. Os conflitos podem ter diversas origens, por exemplo, o ruído nos exemplos de treinamento e a escolha da janela demasiadamente pequena.

O teorema de consistência do modelo de Valiant (enunciado no anexo B) fornece-nos uma condição suficiente para que um algoritmo de aprendizagem seja uma solução PAC. Ele diz que se um algoritmo é consistente então é PAC, se os conjuntos envolvidos no processo são todos finitos. Dizemos que um algoritmo é consistente quando o conceito h por ele aprendido sempre concorda com os exemplos de treinamento, isto é, se (x, y) é um exemplo de treinamento então $h(x) = y$. A generalização do conceito da consistência para o modelo de Haussler não é imediata pois, como neste modelo podem existir conflitos, não é possível exigir que o conceito aprendido sempre concorde com os exemplos de treinamento. Permitir que o conceito aprendido concorde com um dos exemplos de treinamento também não é suficiente para assegurar a convergência do processo de aprendizagem. Na subseção 3.2.5, expomos como é possível definir a consistência no modelo de Haussler e demonstramos um resultado análogo ao teorema de consistência clássica.

No anexo C, comparamos o mecanismo de aprendizagem computacional dos modelos de Valiant, Haussler e vizinho mais próximo com o mecanismo de aprendizagem huma-

na. No anexo D, colocamos a aprendizagem computacional, em particular a aprendizagem do operador, no contexto da teoria de estimação.

3.2 Modelo de Haussler

3.2.1 Introdução

O modelo de Haussler é considerado como a primeira generalização abrangente do modelo PAC. Apesar deste trabalho ser relativamente recente, já existem dezenas de trabalhos que o citam. A referência para estes trabalhos pode ser encontrada na Web no endereço:

http://theory.lcs.mit.edu/~iandc/References/haussler1992:78.html.

Nesta seção, expomos as idéias de Haussler de uma forma mais detalhada que no artigo original, pois este apresenta alguns conceitos muito sucintamente. Relembramos que somente conjuntos finitos são objetos do nosso estudo, pois não se lidam com conjuntos infinitos no processamento de imagens prático.

3.2.2 Problema de Aprendizagem Generalizado

No modelo de Haussler, um *aprendiz*, também chamado de *algoritmo de aprendizagem* recebe *exemplos de treinamento* escolhidos aleatoriamente. Cada exemplo consiste de uma *instância* $x \in X$ e uma *saída* $y \in Y$, onde X e Y são conjuntos arbitrários chamados espaço das instâncias e espaço das saídas, respectivamente. Estes exemplos são gerados de acordo com uma *distribuição conjunta de probabilidade P* (também chamada de *distribuição alvo*) em $X \times Y$. Após o treinamento, o aprendiz recebe mais exemplos escolhidos aleatoriamente utilizando a mesma distribuição conjunta *P*. Para cada exemplo (x, y), é mostrado ao aprendiz somente a instância x. O aprendiz deve escolher uma *ação a* dentre um conjunto de possíveis ações A. Para cada ação a e saída y, o aprendiz

sofrerá uma *penalidade*, que é medida por uma função penalidade com valores reais l: $Y \times A \rightarrow [0, M]$, onde M é um número real positivo. Assumimos que esta função penalidade é conhecida pelo aprendiz. O aprendiz tenta escolher sua ação de modo a minimizar a penalidade esperada.

Baseado nos exemplos de treinamento, o aprendiz desenvolve uma estratégia determinística h que especifica o que ele acredita ser a ação apropriada a para cada instância x de X. Ele usa esta estratégia em todos os exemplos futuros. A estratégia do aprendiz h, que é uma função de $X \rightarrow A$, é chamada de regra de decisão. Assumimos que a regra de decisão é escolhida de um espaço fixo de regras de decisão $\mathcal{H}$. O objetivo da aprendizagem é achar uma regra de decisão em $\mathcal{H}$ que minimize a penalidade esperada, quando os exemplos são escolhidos aleatória e independentemente utilizando a mesma distribuição conjunta P utilizada para gerar os exemplos de treinamento.

Como um exemplo, podemos imaginar que o aprendiz é um médico, o conjunto X são os sintomas do paciente, o conjunto Y são as doenças e o conjunto A são os remédios. Então, dados os sintomas $x \in X$ de um paciente, provocados por uma doença $y \in Y$ desconhecida, o médico deve escolher um remédio $a \in A$ que minimize a penalidade conhecida $l(y, a)$. O fato do médico conhecer a penalidade indica que, se ele conhecesse a doença y, saberia qual é o remédio mais adequado. O médico dispõe de uma tabela sintomas × doenças (exemplos de treinamento) e, baseado nessa tabela, procura escolher o remédio a que julga ser mais adequado ao paciente. Evidentemente, a finalidade deste exemplo é somente ilustrar o processo de aprendizagem computacional, pois um médico não raciocina da forma descrita.

Formalizando estes conceitos, temos:

Definição (instância, saída e ação): X, Y e A são três conjuntos arbitrários chamados respectivamente de *espaço das instâncias*, *espaço das saídas* e *espaço das ações*. Seus membros são chamados de *instância*, *saída* e *ação*, respectivamente.

Definição (exemplo e distribuição de probabilidade): O *espaço das amostras* é o espaço $X \times Y$ e será denotado por *Z*. Um elemento deste espaço é chamado de *exemplo*. O espaço das *distribuições de probabilidades das amostras* $\mathcal{P}$ é uma família de distribuições de probabilidades em *Z*. Uma seqüência de exemplos gerado de acordo com uma distribuição desconhecida $P \in \mathcal{P}$ será chamada de *amostra de P* e a distribuição *P* será chamada de *distribuição alvo*. Uma amostra de comprimento *m* será denotada como $\vec{z} = ((x_1, y_1), (x_2, y_2), ..., (x_m, y_m))$. Também denotaremos $\vec{z}^{\,x} = (x_1, x_2, ..., x_m)$ e $\vec{z}^{\,y} = (y_1, y_2, ..., y_m)$.

Com as definições acima, podemos concluir que podem haver "conflitos" nos exemplos de uma amostra $\vec{z} = ((x_1, y_1), (x_2, y_2), ..., (x_m, y_m))$. Em outras palavras, podem existir um ou mais pares de exemplos $(x_i, y_i), (x_j, y_j) \in \vec{z}$ tal que $x_i = x_j$ mas $y_i \neq y_j$.

Definição (regra de decisão): O conjunto $\mathcal{H}$, chamado de *espaço das regras de decisão*, é um subconjunto do espaço de funções $X \to A$.

Definição (algoritmo de aprendizagem): Um *algoritmo de aprendizagem* $\mathcal{A}$ é uma função de $\bigcup_{m \geq 1} Z^m \to \mathcal{H}$. Isto é, dada uma amostra $\vec{z} \in \bigcup_{m \geq 1} Z^m$, o algoritmo de aprendizagem cria uma regra de decisão $h \in \mathcal{H}$. Denotamos este processo por $h = \mathcal{A}(\vec{z})$.

Definição (penalidade): Chamamos uma função l: $Y \times A \to [0, M]$, para algum $M > 0$ real, de *penalidade*.

Exemplo (penalidades): Suponha $Y = A = \mathbb{R}$. Se queremos que *y* seja igual a *a*, a penalidade costuma ser uma das funções abaixo:

1. $l(y, a) = (y - a)^2$
2. $l(y, a) = | y - a |$
3. $l(y,a) = \begin{cases} 1 \text{ se } y \neq a \\ 0 \text{ cc} \end{cases}$

Para exemplificar algumas penalidades para espaços $Y = A = \mathbb{R}^n$, vamos recordar primeiro a definição da *p*-norma (veja, por exemplo, [Golub and Van Loan, 1989, pg. 53]).

Definição (*p*-norma): Uma *p*-norma vetorial no espaço $\mathbb{R}^n$ é definida como segue:

$$\|x\|_p = \left(|x_1|^p + \cdots + |x_n|^p\right)^{1/p}, x \in \mathbb{R}^n.$$

Entre elas, 1, 2, e ∞ normas são as mais importantes. A 1-norma é a distância "soma dos lados", a 2-norma é a distância euclidiana e a ∞-norma é a distância "maior lado".

Exemplo (penalidades): Suponha $Y = A = \mathbb{R}^n$. Então, abaixo temos alguns exemplos de penalidades:

1. $l(y, a) = \left(\|y - a\|_2\right)^2$
2. $l(y, a) = \|y - a\|_1$
3. $l(y, a) = \|y - a\|_\infty$
4. $l(y,a) = \begin{cases} 1 \text{ se } y \neq a \\ 0 \text{ cc} \end{cases}$

Definição (risco): Dada uma regra de decisão $h \in \mathcal{H}$ e uma distribuição *P* no espaço das amostra *Z*, o *risco* de *h* ou a *penalidade esperada* de *h* é o valor esperado de $l(y, h(x))$, quando os exemplos (x, y) são gerados aleatoriamente pela distribuição alvo *P*:

$$r_{h,l}(P) = r_h(P) = E(\, l(y, h(x)) \,) = \sum_{(x,y) \in Z} l(\, y, h(x) \,)\, P(x, y).$$

Denotaremos a regra de decisão que gera o menor risco de h^* e o risco ótimo como $r^*(P) = r_{h^*}(P) = \wedge \{r_h(P) : h \in \mathcal{H}\} = E(\, l(y,\ h^*(x)) \,)$. A regra h^* é conhecida na literatura como o procedimento de decisão de Bayes. Para maiores detalhes veja, por exemplo, [Cover and Hart, 1967] e [Spiegel, 1979, pg. 12].

Podemos dizer, informalmente, que o objetivo de um problema de aprendizagem é, dada uma amostra escolhida de acordo com uma distribuição de probabilidade $P \in \mathcal{P}$ e uma função penalidade *l*, achar uma regra de decisão *h* em $\mathcal{H}$ tal que $r_h(P)$ seja "próximo" ao $r^*(P)$.

Definição (arrependimento): O *arrependimento L* é uma função $\mathcal{P} \times \mathcal{H} \rightarrow \mathbb{R}^+$ e mede o quanto falhamos em produzir uma regra de decisão próxima da ótima.

Normalmente, o arrependimento é uma função derivada do risco, que por sua vez é uma função derivada da penalidade *l*. Apesar de ser possível definir uma variedade muito grande de funções arrependimento, a mais usual, denotada como L_ε, é definida como segue. Seja *d* uma métrica em $\mathbb{R}$ e seja dado um parâmetro de acuidade ε>0. Então:

$$L_\varepsilon(P,h) = \begin{cases} 1 \text{ se } d\left(r_h(P), r^*(P)\right) \geq \varepsilon \\ 0 \text{ cc} \end{cases}$$

Neste trabalho, utilizaremos como arrependimento somente a função L_ε definida acima.

Definição (grande risco): O *grande risco* é definido por:

$$R_{L,\mathcal{A},m}(P) = \sum_{\vec{z} \in Z^m} L(P, \mathcal{A}(\vec{z}))\ P^m(\vec{z}) = P^m\left\{\vec{z} \in Z^m : d(r_{\mathcal{A}(\vec{z})}(P), r^*(P)) \geq \varepsilon\right\}$$

Isto é, o grande risco é o arrependimento esperado quando a regra de decisão é gerada por um algoritmo de aprendizagem $\mathcal{A}$ utilizando para treinamento uma amostra de tamanho *m*. Denotamos por $P^m(\vec{z})$ a probabilidade da seqüência $\vec{z}$ quando cada um dos *m* componentes de $\vec{z}$ é gerado independentemente por *P*.

Exemplo (PAC clássico): O modelo PAC clássico é um caso particular do modelo PAC generalizado. Isto fica claro fazendo a seguinte consideração. Vamos introduzir um parâmetro de acuidade $\varepsilon > 0$ e definir a função de arrependimento como:

$$L_\varepsilon(P,h) = \begin{cases} 1 \text{ se } r_h(P) \geq \varepsilon \\ 0 \text{ cc} \end{cases}$$

Assim, sofreremos um arrependimento somente quando a regra de decisão *h* produzida pelo algoritmo de aprendizagem possuir um risco de pelo menos ε a mais que a ótima, que no caso do modelo de Valiant tem risco zero ($r^*(P)=0$), pois não existem conflitos nos exemplos. Por esta razão, o grande risco $R_{L_\varepsilon,\mathcal{A},m}(P)$ mede a probabilidade de que a regra de decisão produzida pelo $\mathcal{A}$ tenha um risco de pelo menos ε a mais que a ótima, isto é,

$$R_{L_\varepsilon,\mathcal{A},m}(P) = P\{\vec{z} \in Z^m : (r_{\mathcal{A}(\vec{z})}(P)) \geq \varepsilon\}.$$

Ou seja, obtivemos o modelo de Valiant. ■

Com os conceitos vistos até agora, já estamos aptos a definir formalmente um problema de aprendizagem generalizado.

Definição (problema de aprendizagem): Um *problema de aprendizagem generalizado* é definido por uma sêxtupla ordenada (X, Y, A, $\mathcal{H}$, $\mathcal{P}$, L). Os três primeiros componentes são espaços das instâncias, saídas e ações que são conjuntos arbitrários. A quarta, $\mathcal{H}$, é o espaço das regras de decisão, uma família de funções $X \to A$. O quinto componente, $\mathcal{P}$, é uma família de distribuições de probabilidades em $Z = X \times Y$ que governa a geração de exemplos. O último componente, L, é uma família de funções de arrependimento, isto é, um subconjunto de $\mathcal{P} \times \mathcal{H} \to \mathbb{R}^+$.

Definição (aprendizagem PAC): Dizemos que um algoritmo de aprendizagem $\mathcal{A}$: $\bigcup_{m\geq l} Z^m \to \mathcal{H}$ é uma solução *provavelmente aproximadamente correta* (PAC) de um problema de aprendizagem generalizado (X, Y, A, $\mathcal{H}$, $\mathcal{P}$, L) sse para todo $L \in$ L, e para todo $0 < \delta < 1$ existe um tamanho finito de amostra $m_0 = m_0(L, \delta)$ tal que para toda distribuição alvo $P \in \mathcal{P}$ e para todo tamanho finito $m \geq m_0$, $R_{L,\mathcal{A},m}(P) \leq \delta$. A função $m_0(L, \delta)$ é chamada *complexidade de amostra*.

Como já foi discutido, estas definições generalizam um problema de aprendizagem clássico pois este pode ser formalizado como:

$$(X, \{0, 1\}, \{0, 1\}, X \rightarrow \{0, 1\}, \mathcal{P}, \{L_\varepsilon\}),$$

onde a função penalidade l é definido como segue:

$$l(y,a) = \begin{cases} 1 & \text{se } y \neq a \\ 0 & \text{cc} \end{cases}$$

Além disso, as distribuições de probabilidades $P \in \mathcal{P}$ não podem possuir "conflitos", isto é, se $P \in \mathcal{P}$,

$$P\{(x, y)\} > 0 \;\Rightarrow\; P\{(x, \bar{y})\} = 0, \text{ onde } \bar{y} \text{ é a negação de } y.$$

Note também que podemos reescrever a condição $R_{L,\mathcal{A},m}(P) \leq \delta$ como:

$$P^m\{ \vec{z} \in Z^m : r_{\mathcal{A}(\vec{z})}(P) < \varepsilon \} > 1 - \delta$$

3.2.3 Problema de Otimização

Nesta e na próxima subseção respondemos à seguinte pergunta: "Existe um teste para verificar se um dado algoritmo de aprendizagem é uma solução PAC?" Para isso, definimos o problema de otimização associado a um problema de aprendizagem. [Haussler, 1992] mostra que, se um algoritmo de aprendizagem é uma solução PAC de um problema de otimização, ele é também uma solução PAC do problema de aprendizagem associado, sempre que estivermos trabalhando com os conjuntos X, Y e A finitos. Desenvolvemos os conceitos desta subseção mais extensamente que no artigo original, esperando com isso facilitar a compreensão dos mesmos. A ٦ ilustra os relacionamentos entre um problema de aprendizagem e um problema de otimização.

Definição (risco empírico): Fixada uma penalidade l, o risco empírico de uma regra de decisão $h \in \mathcal{H}$ sobre uma amostra $\vec{z} = ((x_1, y_1), (x_2, y_2), ..., (x_m, y_m))$ é definido:

$$\tilde{r}_h(\vec{z}) = \frac{1}{m} \sum_{i=1}^{m} l(y_i, h(x_i)) .$$

O risco é uma avaliação estatística do valor do risco verdadeiro desconhecido utilizando uma amostra de tamanho *m*.

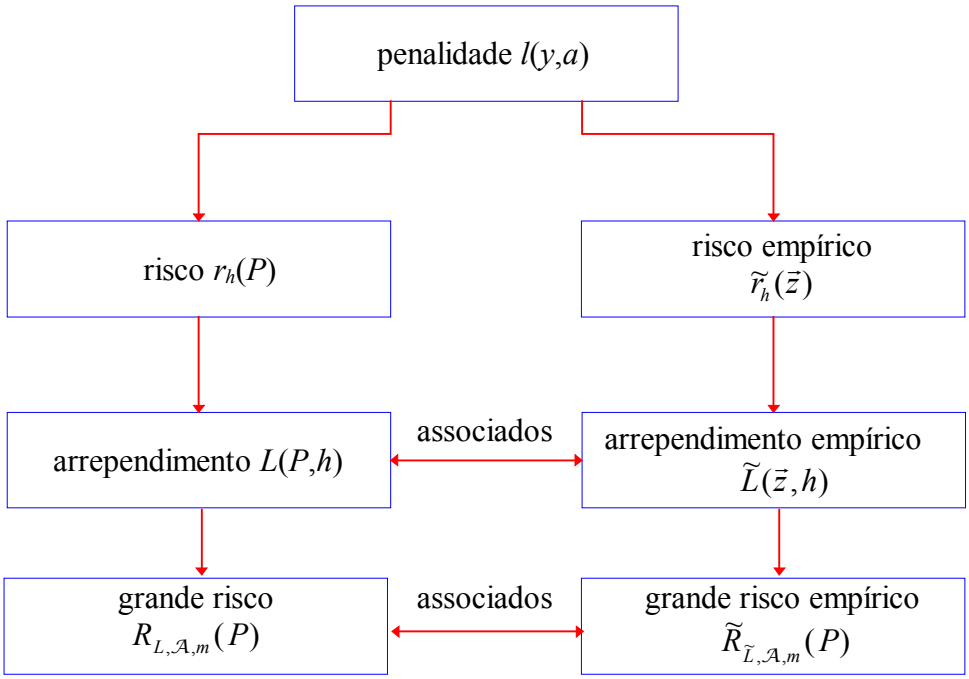


Figura 3.1: Problemas de aprendizagem e de otimização associados.

Definição (regra ótima empírica): Dada uma amostra $\vec{z}$, a regra de decisão ótima empírica é aquela que gera o menor risco empírico e será denotada por $\tilde{h}^* = \tilde{h}^*(\vec{z})$. O seu risco será denotado por:

$$\tilde{r}^*(\vec{z}) = \tilde{r}_{\tilde{h}^*}(\vec{z}) = \wedge\left\{\tilde{r}_h(\vec{z}) : h \in \mathcal{H}\right\}.$$

Definição (arrependimento empírico): O arrependimento empírico $\tilde{L}$ é uma função $Z^m \times \mathcal{H} \rightarrow \mathbb{R}^+$. O arrependimento empírico avalia o valor do arrependimento verdadeiro, normalmente desconhecido, numa amostra de comprimento *m*.

Assim como o arrependimento normalmente é uma função derivada do risco, o arrependimento empírico é, quase sempre, uma função derivada do risco empírico. Neste traba-

lho utilizaremos unicamente a função $\widetilde{L}_\varepsilon$ definida abaixo como o arrependimento empírico. Sejam dadas uma métrica *d* em $\mathbb{R}$, um parâmetro de acuidade $\varepsilon > 0$ e o risco empírico $\widetilde{r}_h(\vec{z})$. Então definimos:

$$\widetilde{L}_\varepsilon(\vec{z},h) = \begin{cases} 1 \text{ se } d\left(\widetilde{r}_h(\vec{z}),\widetilde{r}^*(\vec{z})\right) \geq \varepsilon \\ 0 \text{ cc} \end{cases}$$

Dizemos que o arrependimento verdadeiro L_ε e o arrependimento empírico $\widetilde{L}_\varepsilon$ são associados entre si quando ambos utilizam o risco verdadeiro e o empírico associados entre si, a mesma função distância *d* e o mesmo parâmetro de acuidade ε.

Definição (grande risco empírico): O grande risco empírico pode ser definido por:

$$\widetilde{R}_{\widetilde{L},\mathcal{A},m}(P) = \sum_{\vec{z}\in Z^m} \widetilde{L}(\vec{z},\mathcal{A}(\vec{z}))\ P^m(\vec{z}) = P^m\left\{\vec{z} \in Z^m : d\left(\widetilde{r}_{\mathcal{A}(\vec{z})}(\vec{z}),\widetilde{r}^*(\vec{z})\right) \geq \varepsilon\right\}$$

O grande risco empírico avalia o grande risco verdadeiro desconhecido numa amostra de *m* elementos. Diremos que o grande risco empírico e o grande risco verdadeiro estão associados entre si quando ambos utilizam o arrependimento empírico e o arrependimento verdadeiro associados entre si.

Com estes conceitos, podemos definir um problema de otimização associado a um problema de aprendizagem:

Definição (problema de otimização): Um *problema de otimização* é uma sêxtupla ordenada $(X, Y, A, \mathcal{H}, \mathcal{P}, \widetilde{\mathsf{L}})$ onde os cinco primeiros componentes são definidos da mesma forma que num problema de aprendizagem e o sexto componente, $\widetilde{\mathsf{L}}$, é uma família de arrependimentos empíricos.

Definição (problema de otimização associado): Um problema de aprendizagem (X, Y, A, $\mathcal{H}$, $\mathcal{P}$, L) e um problema de otimização (X, Y, A, $\mathcal{H}$, $\mathcal{P}$, $\mathsf{L}^{\sim}$) são associados sse para todo arrependimento em L, o seu arrependimento associado pertence ao $\mathsf{L}^{\sim}$ e vice-versa.

Um algoritmo de aprendizagem $\mathcal{A} : \bigcup_{m\geq 1} Z^m \rightarrow \mathcal{H}$ será uma solução PAC de um problema de otimização se achar, dada uma amostra de treinamento $\vec{z}$, uma regra de decisão $h \in \mathcal{H}$ tal que $\tilde{r}_h(\vec{z})$ seja "próximo" ao $\tilde{r}^*(\vec{z})$. Este conceito pode ser formalizado como segue:

Definição (otimização PAC): Dizemos que um algoritmo $\mathcal{A} : \bigcup_{m\geq 1} Z^m \rightarrow \mathcal{H}$ é uma solução *provavelmente aproximadamente correta* (PAC) para um problema de otimização sse, para todo $\tilde{L} \in \mathsf{L}^{\sim}$ e para todo $0 < \delta < 1$ existe um tamanho de amostra finito $m_0 = m_0(\tilde{L}, \delta)$ tal que para todo $P \in \mathcal{P}$ e $m \geq m_0$, $\tilde{R}_{\tilde{L},\mathcal{A},m}(P) \leq \delta$. A função $m_0(\tilde{L}, \delta)$ é chamada *complexidade de amostra*.

3.2.4 Relações entre Aprendizagem e Otimização

Nesta subseção estudaremos as relações entre os problemas de aprendizagem e otimização associados. Veremos que sob certa hipótese (convergência uniforme), uma solução PAC de um será também uma solução PAC do outro. Com isso, construímos um teste para verificar se um algoritmo de aprendizagem é uma solução PAC de um problema de aprendizagem. Pois, para isso basta que o algoritmo seja uma solução PAC do problema de otimização associado.

Definição (convergência uniforme): Dizemos que o risco empírico converge uniformemente ao risco verdadeiro sse para todo $0 < \varepsilon, \delta < 1$, existe um tamanho de amostra m_0 tal que para todo $m \geq m_0$, quando cada componente z_i de $\vec{z}$ é escolhido independente e aleatoriamente de Z de acordo com uma distribuição alvo P, temos, com probabilidade de pelo menos $1 - \delta$:

$$|\tilde{r}_h(\vec{z}) - r_h(P)| < \varepsilon ,$$

para todo $h \in \mathcal{H}$.

O teorema a seguir é uma conseqüência direta do Lema 1 de [Haussler, 1992].

Teorema 3.1 (convergência uniforme + otimização PAC ⇒ aprendizagem PAC): Se um algoritmo $\mathcal{A}$ é uma solução PAC de um problema de otimização (X, Y, A, $\mathcal{H}$, $\mathcal{P}$, $\{\tilde{L}_\varepsilon\}$) e se a estimativa do risco empírico converge uniformemente ao risco verdadeiro então $\mathcal{A}$ é uma solução PAC do problema de aprendizagem (X, Y, A, $\mathcal{H}$, $\mathcal{P}$, $\{L_\varepsilon\}$) associado.

Prova: Suponha dados $0 < \varepsilon, \delta < 1$ e que existe $m_o = m_o(\varepsilon, \delta)$ tal que para todo $P \in \mathcal{P}$ e todo $m > m_o$:

$$\Pr\left(\exists h \in \mathcal{H} : d(\tilde{r}_h(\vec{z}), r_h(P)) \geq \varepsilon / 3 \right) \leq \delta / 2 . \quad (1)$$

Suponha também que o algoritmo $\mathcal{A}$ é tal que para todo $P \in \mathcal{P}$ e todo $m > m_o$:

$$\Pr\left(d(\tilde{r}_{\mathcal{A}(\vec{z})}(\vec{z}), \tilde{r}^*(\vec{z})) \geq \varepsilon / 3 \right) \leq \delta / 2 . \quad (2)$$

Então devemos demonstrar que para todo $P \in \mathcal{P}$ e todo $m > m_o$:

$$\Pr\left(d(r_{\mathcal{A}(\vec{z})}(P), r^*(P)) \geq \varepsilon \right) \leq \delta . \quad (3)$$

Demonstrando que (1) e (2) implicam (3) teremos demonstrado o teorema. Pois a convergência uniforme do risco empírico ao risco verdadeiro implica (1); se o algoritmo $\mathcal{A}$ é uma solução PAC do problema de otimização então (2) torna-se verdadeiro; e sendo (3) verdadeiro, o algoritmo $\mathcal{A}$ será uma solução PAC do problema de aprendizagem. Considere as seguintes afirmações:

$$d(r_{\mathcal{A}(\vec{z})}(P), \tilde{r}_{\mathcal{A}(\vec{z})}(\vec{z})) < \varepsilon / 3 \quad (4)$$

$$d(\tilde{r}_{\mathcal{A}(\vec{z})}(\vec{z}), \tilde{r}^*(\vec{z})) < \varepsilon / 3 \quad (5)$$

$$d(\tilde{r}^*(\vec{z}), r^*(P)) < \varepsilon / 3 \quad (6)$$

A hipótese (2) implica que (5) é verdadeiro com probabilidade maior que $1 - \delta/2$ e a hipótese (1) implica que (4) é verdadeiro com probabilidade maior que $1 - \delta/2$. A hipó-

tese (1) também implica que (6) é verdadeiro com probabilidade maior que 1 - δ/2, pois se (6) é falso, podemos achar um $h \in \mathcal{H}$ tal que $d(\tilde{r}_h(\vec{z}), r_h(P)) \geq \varepsilon / 3$. Para isso basta escolher:

$$h = \begin{cases} \tilde{h}^*, \text{ se } \tilde{r}^*(\vec{z}) < r^*(P) \\ h^*, \text{ cc} \end{cases} \tag{7}$$

As hipóteses (1) e (2) implicam que (4), (5) e (6) são verdadeiros com probabilidade maior que 1 - δ. Logo, concluímos que a afirmação abaixo é verdadeira com probabilidade maior que 1 - δ.

$$d(r_{\mathcal{A}(\vec{z})}(P), r^*(P)) < \varepsilon . \tag{8}$$

Daí, concluímos que (3) é verdadeiro e que $\mathcal{A}$ é uma solução PAC do problema de aprendizagem. ■

Para prosseguir, necessitamos da desigualdade de Hoeffding que enunciamos abaixo:

Teorema 3.2 (desigualdade de Hoeffding): Sejam Y_1, Y_2, ..., Y_n variáveis aleatórias independentes com média zero e delimitadas nos intervalos $a_i \leq Y_i \leq b_i$. Então para todo η>0,

$$\Pr\left\{\left|Y_1 + \cdots + Y_n\right| \geq \eta\right\} \leq 2\exp\left[-2\eta^2 \Big/ \sum_{i=1}^{n} (b_i - a_i)^2\right]$$

Prova: Veja Corolário 3 do [Pollard, 1984, pg. 192]. ■

O teorema a seguir é uma conseqüência direta do Teorema 1 do [Haussler, 1992].

Teorema 3.3 ($\mathcal{H}$ finito ⇒ convergência uniforme): O risco empírico converge uniformemente ao risco verdadeiro sempre que o conjunto $\mathcal{H}$ for finito.

Prova: Devemos mostrar que se $\mathcal{H}$ é finito então dados quaisquer 0 < ε, δ < 1, existe um tamanho de amostra m_0 tal que para todo $m \geq m_0$, quando cada componente z_i de $\vec{z}$ é escolhido independente e aleatoriamente de Z de acordo com uma distribuição alvo P, temos para todo $h \in \mathcal{H}$:

$$\Pr\left\{\left|\widetilde{r}_h(\vec{z}) - r_h(P)\right| < \varepsilon\right\} > 1-\delta\,.$$

Ou seja,

$$\Pr\left\{\left|\left(\frac{1}{m}\sum_{z\in\vec{z}} l(h(x),y)\right) - r_h(P)\right| < \varepsilon\right\} > 1-\delta\,.$$

Supondo fixa uma regra de decisão $h \in \mathcal{H}$, $l(h(x),y)$ é uma variável aleatória no intervalo real [0,M] e $r_h(P)$ é a sua esperança. Vamos criar m outras variáveis aleatórias fazendo

$$Y_i = l(h(x_i),y_i) - r_h(P), \quad i = 1,\ldots,m$$

Ora, estas variáveis aleatórias têm média 0 e estão limitadas no intervalo real

$$[-r_h(P), M - r_h(P)]\,.$$

Assim, podemos aplicar a desigualdade de Hoeffding enunciado no teorema acima. Dado qualquer número real $\eta>0$,

$$\Pr\left\{\left|Y_1+\cdots+Y_m\right| \geq \eta\right\} \leq 2\exp\left[-2\eta^2/mM^2\right].$$

Chamando $\varepsilon m = \eta$, obtemos:

$$\Pr\left\{\left|Y_1+\cdots+Y_m\right| \geq \varepsilon m\right\} \leq 2\exp\left[-2(\varepsilon m)^2/mM^2\right].$$

Ou seja,

$$\Pr\left\{\left|\left(\frac{1}{m}\sum_{z\in\vec{z}} l(h(x),y)\right) - r_h(P)\right| \geq \varepsilon\right\} \leq 2\exp\left[-2m\varepsilon^2/M^2\right].$$

Esta é a probabilidade de uma regra de decisão fixa. Segue que a probabilidade de que exista alguma regra de decisão $h \in \mathcal{H}$ tal que

$$\left|\left(\frac{1}{m}\sum_{z\in\vec{z}} l(h(x),y)\right) - r_h(P)\right| \geq \varepsilon$$

é no máximo

$$2\left|\mathcal{H}\right|\exp\left[-2m\varepsilon^2/M^2\right].$$

Isto demonstra a convergência uniforme do risco empírico. ■

Corolário (otimização PAC ⇒ aprendizagem PAC): Se $\mathcal{H}$ é finito, toda solução PAC de um problema de otimização também é uma solução PAC do problema de aprendizagem associado.

Prova: Conseqüência direta do ד e do ו. ■

3.2.5 Consistência Generalizada

No anexo B, tomando por base o livro [Anthony and Biggs, 1992], apresentamos o teorema de consistência do modelo PAC clássico. Este teorema diz que todo algoritmo consistente é uma solução PAC do problema de aprendizagem clássico, caso *X* seja finito. Será que existe um resultado análogo no modelo de Haussler? Exporemos nesta subseção como a consistência pode ser generalizada para o modelo de Haussler.

Intuitivamente, um algoritmo PAC deve "convergir" para a regra de decisão ótima (regra de Bayes) à medida em que o número de exemplos de treinamento cresce. O procedimento de decisão de Bayes de um problema de aprendizagem pode ser criado explicitamente quando a probabilidade de distribuição *P* é totalmente conhecida a priori. Claramente, a regra ótima h^* deve escolher, para cada instância x_o, a ação a^* com a menor esperança de erro. Vamos denotar a esperança de penalidade ao tomar uma ação *a*, dada uma instância x_o, por $\boldsymbol{r}_{x_o}(a)$, isto é:

$$\boldsymbol{r}_{x_o}(a) = \sum_{(x_o,y)\in Z} l(y,a)\,P(x_o,y)$$

Então, a regra de Bayes é definida, para cada $x_o \in X$, como $h^*(x_o) = a^*$, onde a^* é o argumento que minimiza $\boldsymbol{r}_{x_o}(a)$, isto é, $\boldsymbol{r}_{x_o}(a^*) = \min_{a\in A}\left\{\boldsymbol{r}_{x_o}(a)\right\}$.

De forma semelhante, a regra ótima empírica $\tilde{h}^*$ deve escolher, dada uma instância $x_o \in \vec{z}^{\,\mathbf{x}}$, a ação a^* com o menor risco empírico. Para explicitar esta regra, adotaremos a seguinte notação:

Notação (subseqüência coincidente): Dados $x \in X$ e uma seqüência $\vec{z} = ((x_1, y_1), (x_2, y_2), ..., (x_m, y_m)) \in Z^m$, chamaremos de *subseqüência de* $\vec{z}$ *coincidente com x* e denotaremos por $C_{\vec{z}}(x)$ a subseqüência $((x'_1, y'_1), (x'_2, y'_2), ..., (x'_n, y'_n))$ de $\vec{z}$ composta por todos os elementos $(x_i, y_i) \in \vec{z}$ onde $x_i = x$. Além disso, denotaremos a seqüência $(x'_1, x'_2, ..., x'_n)$ por $C^x_{\vec{z}}(x)$ e a seqüência $(y'_1, y'_2, ..., y'_n)$ por $C^y_{\vec{z}}(x)$.

Vamos denotar a esperança empírica da penalidade, ao tomar uma ação *a*, dada uma instância $x_o \in \vec{z}^{\mathbf{x}}$, como $\tilde{\boldsymbol{r}}_{x_o}(a)$, isto é:

$$\tilde{\boldsymbol{r}}_{x_o}(a) = \frac{1}{\left|C_{\vec{z}}(x_o)\right|} \sum_{y \in C^y_{\vec{z}}(x_o)} l(y,a)$$

Com essa notação, a regra empírica ótima deve ser definida, para cada $x_o \in X$, como $\tilde{h}^*(x_o) = a^*$, onde a^* é o argumento que minimiza $\tilde{\boldsymbol{r}}_{x_o}(a)$, isto é, $\tilde{\boldsymbol{r}}_{x_o}(a^*) = \min_{a \in A} \left\{ \tilde{\boldsymbol{r}}_{x_o}(a) \right\}$.

Como se pode escolher este a^* na prática? Utilizando a função minimizadora de penalidade definida abaixo.

Definição (minimizadora de penalidade): Diremos que uma função

$$\mathcal{M} : \bigcup_{m \geq 1} Y^m \rightarrow A$$

é minimizadora da penalidade *l* sse, para cada $\vec{y} = (y_1 , y_2 , ..., y_m) \subset Y^m$, o valor $a^* = \mathcal{M}(\vec{y})$ tiver a seguinte característica:

$$\sum_{i=1}^{m} l(y_i, a^*) = \min_{a \in A} \left(\sum_{i=1}^{m} l(y_i, a) \right).$$

Apresentamos a seguir três exemplos de funções minimizadoras de penalidade.

Exemplo (média aritmética): Sejam $Y = A = \mathbb{R}$ e seja a penalidade $l(y, a) = (y - a)^2$. Então, a função $\mathcal{M}(\vec{y})$ é a média aritmética de $\vec{y}$.

Prova: Seja $\vec{y} = (y_1 , y_2 , ..., y_n)$. Queremos achar $a \in A$ que minimiza a seguinte função:

$$f(a) = (y_1 - a)^2 + (y_2 - a)^2 + ... + (y_n - a)^2$$

Ora, esta é uma função de segundo grau. Calculando a sua derivada e igualando a zero, acharemos um ponto de mínimo (já que o coeficiente de grau dois é positivo).

$$f'(a) = -2(y_1 - a) - 2(y_2 - a) - ... - 2(y_n - a) = 0$$

A igualdade acima é verdadeira quando $a = (y_1 + y_2 + ... + y_n) / n$, ou seja, a média aritmética de $\vec{y}$. ■

Exemplo (mediana): Sejam $Y = A = \mathbb{R}$ e seja a penalidade $l(y, a) = |y - a|$. Então, a função $\mathcal{M}(\vec{y})$ é a mediana de $\vec{y}$. A mediana de uma seqüência de n números ordenados é o valor central da seqüência, se n é ímpar e é a média aritmética dos dois valores centrais, se n é par. Neste último caso, qualquer número que esteja situado entre os dois valores centrais, inclusive, minimiza a penalidade.

Prova: Vamos demonstrar para o caso n par, pois quando n é ímpar, a demonstração é inteiramente similar. Seja, pois, $\vec{y} = (y_1, y_2, ..., y_{2m})$ uma seqüência ordenada em ordem crescente. Queremos minimizar a função $f: [y_1, y_{2m}] \rightarrow \mathbb{R}$ definida por:

$$f(a) = |y_1 - a| + |y_2 - a| + ... + |y_{2m} - a|$$

Esta função pode ser reescrita para $a \in [y_j, y_{j+1}]$, $1 \leq j \leq 2m\text{-}1$ como:

$$f(a) = -(y_1 - a) - (y_2 - a) - ... - (y_j - a) + (y_{j+1} - a) + ... + (y_{2m} - a)$$

$$f(a) = -(y_1 + y_2 + ... + y_j) + (y_{j+1} + ... + y_{2m}) + (2j - 2m)\, a$$

Calculando a derivada, temos:

$$f'(a) = (2j - 2m)$$

Ora, a derivada acima é negativa quando $j < m$, é nula quando $j = m$ e é positiva quando $j > m$. Logo, a função f é decrescente no intervalo $[y_1, y_m]$, é constante no intervalo $[y_m, y_{m+1}]$ e é crescente no intervalo $[y_{m+1}, y_{2m}]$. Portanto, podemos concluir que o mínimo da função é atingido no intervalo $[y_m, y_{m+1}]$. ■

Exemplo (moda): Sejam $Y = A = \mathbb{R}$ e seja a penalidade tipo "acerto ou erro"

$$l(y,a) = \begin{cases} 0 & \text{se } y = a \\ 1 & \text{cc} \end{cases}.$$

Então, a função $\mathcal{M}(\vec{y})$ é a moda dos $y \in \vec{y}$. A moda de uma seqüência de números é o valor que ocorre com maior freqüência. Observe que a moda pode não ser única. A demonstração deste resultado é imediata.

Exemplo (média aritmética vetorial): Sejam $Y = A = \mathbb{R}^n$ e seja a penalidade $l(y, a) = \left(\|y-a\|_2\right)^2$. Então, a função $\mathcal{M}(\vec{y})$ é a média aritmética vetorial de $\vec{y}$, ou seja,

$$\mathcal{M}(\vec{y}) = (1/n)\ (y_1 + y_2 + ... + y_n).$$

Prova: Seja $\vec{y} = (y_1, y_2, ..., y_n)$. Queremos achar $a \in \mathbb{R}^n$ que minimiza a seguinte função:

$$f(a) = \left(\|y_1 - a\|_2\right)^2 + \cdots + \left(\|y_n - a\|_2\right)^2 = \sum_{i=1}^{n}\sum_{j=1}^{d}\left(y_i[j] - a[j]\right)^2 = \sum_{j=1}^{d}\left(\sum_{i=1}^{n}\left(y_i[j] - a[j]\right)^2\right)$$

Isto é, para cada j, deve achar $a[j]$ que minimize $\sum_{i=1}^{n}\left(y_i[j] - a[j]\right)^2$. Mas pelo exemplo da média aritmética, sabemos que $a[j] = \frac{1}{n}\sum_{i=1}^{n} y_i[j]$ minimiza a expressão acima, para todo $j \in [1 \ldots d]$. ■

Exemplo (mediana vetorial): Sejam $Y = A = \mathbb{R}^n$ e seja a penalidade $l(y, a) = \|y-a\|_1$. Então, a função $\mathcal{M}(\vec{y})$ é a mediana de $\vec{y}$ coordenada a coordenada, ou seja,

$$\mathcal{M}(\vec{y})[j] = \text{mediana}(y_1[j] + y_2[j] + ... + y_d[j]),\ 1 \le j \le d.$$

Prova: Queremos minimizar a expressão abaixo:

$$\sum_{i=1}^{n} l(y_i, a) = \sum_{i=1}^{n}\|y_i - a\|_1 = \sum_{i=1}^{n}\sum_{j=1}^{d}|y_i[j] - a[j]| = \sum_{j=1}^{d}\sum_{i=1}^{n}|y_i[j] - a[j]|$$

Para cada j, a somatória $\sum_{i=1}^{n}|y_i[j] - a[j]|$ é minimizada pela $a[j]$=mediana($y_1[j] + y_2[j] + ... + y_n[j]$), conforme mostramos no exemplo da mediana. ■

Exemplo (moda vetorial): Sejam $Y = A = \mathbb{R}^n$ e seja a penalidade tipo "acerto ou erro"

$$l(y,a) = \begin{cases} 0 & \text{se } y = a \\ 1 & \text{cc} \end{cases}$$

Então, a função $\mathcal{M}(\vec{y})$ é a moda dos $y \in \vec{y}$. A demonstração deste resultado é imediata.

Uma vez definida a função minimizadora de penalidade, estamos em condições de generalizar a consistência.

Definição (consistência generalizada): Diremos que um algoritmo $\mathcal{A}^+$: $\bigcup_{m\geq 1} Z^m \to \mathcal{H}$ é *consistente* para um problema de aprendizagem generalizada (ou de otimização generalizada) sse, para uma amostra de qualquer tamanho $\vec{z}$:

$$C_{\vec{z}}(x) \text{ é uma seqüência não-nula } \Rightarrow \ \mathcal{A}^+(\vec{z})(x) = \mathcal{M}(C^y_{\vec{z}}(x)).$$

Os dois resultados abaixo demonstram que um algoritmo consistente é uma solução PAC do problema de aprendizagem generalizado toda vez que o espaço $\mathcal{H}$ for finito.

Teorema 3.4 (algoritmo consistente ⇒ otimização PAC): Todo algoritmo de aprendizagem consistente $\mathcal{A}^+ : \bigcup_{m\geq 1} Z^m \to \mathcal{H}$ com um problema de otimização generalizada (X, Y, A, $\mathcal{H}$, $\mathcal{P}$, $\{\tilde{L}_\varepsilon\}$) é uma solução PAC.

Prova: Seja $\mathcal{A}^+$ um algoritmo consistente e seja dada uma amostra de treinamento $\vec{z}$. Seja $\tilde{h}^*(\vec{z})$ a regra de decisão ótima. Para demonstrar o que se pede, basta provar que

$$\tilde{\boldsymbol{r}}_{x_o}(A^+(\vec{z})(x_o)) = \tilde{\boldsymbol{r}}_{x_o}(\tilde{h}^*(\vec{z})(x_o)), \quad \forall x_o \in \vec{x}.$$

Pois, nesse caso, para todo ε>0,

$$\tilde{R}_{\tilde{L}_\varepsilon,\mathcal{A},m}(P) = \sum_{\vec{z}\in Z^m} \tilde{L}_\varepsilon(\vec{z},\mathcal{A}^+(\vec{z}))\, P^m(\vec{z}) \ = \ P^m\left\{\vec{z} \in Z^m : d\left(\tilde{r}_{A^+(\vec{z})}(\vec{z}), \tilde{r}_{\tilde{h}^*(\vec{z})}(\vec{z})\right) \geq \varepsilon\right\} = 0,$$

e $\mathcal{A}^+$ será PAC. Para isso, observe que, $\forall x_o \in \vec{x}$, a seguinte igualdade vale:

$$\tilde{\boldsymbol{r}}_{x_o}(\tilde{h}^*(\vec{z})(x_o)) = \min_{a\in A}\left\{\tilde{\boldsymbol{r}}_{x_o}(a)\right\} = \min_{a\in A}\left\{\sum_{y\in C^y_{\vec{z}}(x_o)} l(a,y) \Big/ \left|C_{\vec{z}}(x_o)\right|\right\}$$

Por outro lado,

$$\tilde{\boldsymbol{r}}_{x_o}(\mathcal{A}^+(\vec{z})(x_o)) = \tilde{\boldsymbol{r}}_{x_o}(\mathcal{M}(C_{\vec{z}}^{y}(x_o))) = \sum_{y \in C_{\vec{z}}^{y}(x_o)} l(\mathcal{M}(C_{\vec{z}}^{y}(x_o)), y) \Big/ \left| C_{\vec{z}}(x_o) \right|.$$

Pela definição da função minimizadora de penalidade, quando $a = \mathcal{M}(C_{\vec{z}}^{y}(x_o))$, a expressão abaixo torna-se mínima:

$$\sum_{y \in C_{\vec{z}}^{y}(x_o)} l(a, y).$$

Conseqüentemente, $\tilde{\boldsymbol{r}}_{x_o}(A^+(\vec{z})(x_o)) = \tilde{\boldsymbol{r}}_{x_o}(\tilde{h}^*(\vec{z})(x_o))$. ■

Corolário (aprendizagem consistente ⇒ PAC): Todo algoritmo $\mathcal{A}^+ : \bigcup_{m \geq 1} Z^m \rightarrow \mathcal{H}$ consistente com um problema de aprendizagem generalizada (X, Y, A, $\mathcal{H}$, $\mathcal{P}$, $\{ L_\varepsilon \}$) é uma solução PAC, quando $\mathcal{H}$ for finito.

Prova: Sabemos que um algoritmo consistente $\mathcal{A}^+$ é uma solução PAC do problema de otimização (١). Também sabemos, pelo Corolário do ١, que toda solução PAC de um problema de otimização generalizada também é uma solução PAC do problema de aprendizagem associado, sempre que $\mathcal{H}$ for finito. ■

Corolário (complexidade de aprendizagem consistente): Um algoritmo de aprendizagem consistente $\mathcal{A}^+$ tem a seguinte complexidade de amostra para $\mathcal{H}$ finito.

$$m_0(\varepsilon, \delta) = O\left(\frac{M^2}{\varepsilon^2} \left(\log|\mathcal{H}| + \log\frac{1}{\delta} \right) \right)$$

Prova: No ١, provamos que:

$$\tilde{r}_{\mathcal{A}^+}(\vec{z}) = \tilde{r}^*(\vec{z})$$

Portanto, utilizando as idéias do ١, temos:

$$2|\mathcal{H}| \exp\left[-2m_0\varepsilon^2 / M^2 \right] \leq \delta$$

$$\frac{-2m_o\varepsilon^2}{M^2} \leq \ln\left(\frac{\delta}{2|\mathcal{H}|} \right)$$

$$m_o \leq \frac{M^2}{2\varepsilon^2}\left(\ln\left(2|\mathcal{H}|\right) - \ln(\delta) \right)$$

Com isso concluimos:

$$m_0(\varepsilon,\delta) = O\left(\frac{M^2}{\varepsilon^2}\left(\log|\mathcal{H}| + \log\frac{1}{\delta}\right)\right) \quad \blacksquare$$

Quando o conceito de consistência generalizada é aplicada ao modelo de Valiant, obtemos o conceito de consistência clássica. Pois, para que um algoritmo $\mathcal{A}^+$ seja consistente com um problema de aprendizagem generalizado exigimos:

$$C_{\vec{z}}(x) \text{ é uma seqüência não-nula } \Rightarrow \ \mathcal{A}^+(\vec{z})(x) = \mathcal{M}(C_{\vec{z}}^y(x)).$$

Ora, no modelo de Valiant, quando $C_{\vec{z}}(x)$ é uma seqüência não-nula, todos os elementos de $C_{\vec{z}}^y(x)$ são iguais, digamos *y*. Isto faz com que $\mathcal{M}(C_{\vec{z}}^y(x))$ seja obrigatoriamente *y*. Isto mostra que a consistência generalizada, conforme definimos, de fato generaliza a consistência clássica.

3.3 Conclusão

Neste capítulo, apresentamos o modelo de aprendizagem computacional PAC generalizado (modelo de Haussler) que servirá de suporte teórico para a aprendizagem do operador morfológico. Em muitos pontos, detalhamos a exposição mais que no artigo original, para facilitar a compreensão das idéias. Introduzimos a função minimizadora de penalidade e estendemos o conceito de consistência do modelo de Valiant para o de Haussler. Utilizaremos nos capítulos seguintes a consistência para mostrar a convergência do processo de aprendizagem de operador.

4 Aprendizagem de Operador

4.1 Introdução

Apresentamos a morfologia matemática no capítulo 2 e a aprendizagem computacional no capítulo 3. No capítulo presente, utilizando como base teórica a morfologia matemática e a aprendizagem computacional, descreveremos a construção automática de operadores morfológicos. Como vimos no primeiro capítulo, a morfologia matemática apresenta duas dificuldades práticas quando procuramos aplicá-la num problema real de processamento de imagem.

A primeira dificuldade é o projeto do operador morfológico, isto é, a escolha do operador mais adequado entre um número muito grande de candidatos para resolver um problema determinado. Como vimos no capítulo 1, alguns trabalhos propõem utilizar as técnicas de inteligência artificial ([Schmitt, 1989a], [Schmitt, 1989b]) e outro um sistema especialista nebuloso ([Kim et al., 1997]) para facilitar o trabalho do usuário.

Nesta tese, estamos interessados numa outra abordagem: expressar o conhecimento do usuário através de exemplos da tarefa que se deseja executar ([Dougherty, 1992a], [Dougherty, 1992b], [Loce, 1993], [Barrera et al., 1995], [Tomita, 1996] e [Harvey and Marshall, 1996]). Para que o projeto automático do operador possa ser utilizado amplamente é necessário que os algoritmos de aprendizagem sejam eficientes, tanto do ponto de vista do tempo como do espaço. Neste sentido, nenhum dos trabalhos anteriores é plenamente satisfatório, pois todos apresentam o problema de "explosão combinatória". Isto é, os algoritmos têm complexidade de tempo proporcionais à k^d, onde k é a quanti-

dade de níveis de cinza da imagem e d é o tamanho da janela utilizada (aqui, tamanho da janela, tamanho do elemento estruturante e dimensão do problema são todos sinônimos entre si).

Os trabalhos [Dougherty, 1992a] e [Dougherty, 1992b] não apresentam explicitamente o algoritmo utilizado. Descrevem métodos de estimação estatística de operadores morfológicos (binário e em níveis de cinza) que minimizam o erro quadrático médio. Nos exemplos explicados mais extensamente, a distribuição de probabilidade é suposta conhecida "a priori", o que permite projetar o operador ótimo. Isto é, num exemplo cujo objetivo é eliminar o ruído, o modelo de corrupção da imagem pelo ruído é considerado conhecido. E, nos exemplos onde a distribuição de probabilidade é aparentemente considerada desconhecida, o operador projetado é aplicado na própria imagem utilizada para o treinamento. Evidentemente esta abordagem não pode ser utilizada na prática. A seguinte afirmação não deixa dúvidas sobre a natureza exponencial do método: "Embora a teoria estatística que forma a base seja similar, a seleção dos elementos estruturantes ótimos é muito mais delicada (no caso de níveis de cinza). No caso binário, o número de possíveis elementos estruturantes é 2^d e cada um desses deve ser examinado. No caso em níveis de cinza (com k níveis de cinza) a busca pelos elementos estruturantes ótimos deve ser considerado cuidadosamente para achar uma coleção lógica mínima sobre a qual efetuar a busca".

A tese [Loce, 1993, pg. 141] afirma sobre o seu trabalho: "Embora os teoremas de erro médio absoluto forneçam um meio rápido para examinar o espaço de projeto do filtro, a natureza combinatorial deste espaço é, em geral, demasiadamente grande para uma busca exaustiva". Esse trabalho tentou minimizar o tempo de busca restringindo o espaço de busca, porém sem conseguir fugir à explosão combinatória. Esta tese afirma na introdução: "A presente tese (...) desenvolve metodologias de projeto para filtros baseados na morfologia com erro médio absoluto ótimo que são adequados para processar imagens que são binárias ou com poucos níveis de cinza (2 ou 3 bits/pixel)". A metodologia

apresentada por Loce é adequada somente para imagens com poucos níveis de cinza justamente por causa da explosão combinatória.

Os trabalhos [Barrera et al., 1995] e [Tomita, 1996] apresentam um algoritmo denominado ISI-2 que demorou 56.858 segundos (aproximadamente 16 horas) para projetar um operador binário a partir de 13.966 exemplos sem repetição (Tabela 4.1) com janela tamanho 49. Enquanto isso, o mesmo algoritmo levou apenas 5 segundos para resolver um outro problema com janela tamanho 12. Certamente, este súbito aumento de tempo de processamento deve ser atribuído à explosão combinatória.

Algoritmo	Tamanho da janela	Tamanho da imagem amostra em pixels	Tamanho da amostra eliminando exemplos idênticos	Tamanho do operador em bytes	Tempo de treinamento em segundos	Tempo de aplicação em segundos
1. Cortes	12	65.341	1.918	15.444	1	2*
2. ISI-2	12	?	1.386	55.672	5	?
3. Cortes	49	65.341	27.486	220.284	4	5*
4. ISI-2	49	?	13.966	620.352	56.858	?

Tabela 4.1: Comparação de desempenho do algoritmo de cortes com ISI-2. Os dados da linha 1 foram obtidos na aplicação da אי.g. Os da linha 3 foram obtidos na aplicação da אי.h. Os dados de desempenho do algoritmo ISI-2, das linhas 2 e 4, foram transcritos do artigo [Barrera et al., 1995].

Como não dispomos nem do programa nem das imagens utilizadas em [Barrera et al., 1995], as aplicações comparadas não são as mesmas. O algoritmo ISI-2 foi executado num computador Sparc Station 2 e o algoritmo de cortes num Pentium 100 MHz.

Observe que escolhemos aplicações com tamanho de amostra maior que aquela utilizada para testar o ISI-2. Mesmo assim, para uma dimensão grande e um tamanho de amostra relativamente grande (linhas 3 e 4), o algoritmo de cortes foi mais de 10.000 vezes mais rápido e gastou 3 vezes menos memória.

? Indica que não dispomos destes dados.

* Tempo gasto para aplicar o operador numa imagem 480 × 512.

O artigo [Harvey and Marshall, 1996], que utiliza um algoritmo genético para projeto de operador, não faz referências à eficiência do seu algoritmo mas analisando-o pode-se chegar à conclusão de que também apresenta o problema da explosão combinatória. Além disso, o operador obtido por este método tem "otimalidade restrita", isto é, o método converge para um operador ótimo dentro de uma classe restrita de operadores, por exemplo, as combinações de duas erosões e duas dilatações 5×5. Não se conhece o quanto esta otimalidade restrita está próxima da otimalidade global.

O problema da explosão combinatória não ocorre no algoritmo de cortes, proposto neste capítulo e publicado resumidamente em [Kim, 1997]. O algoritmo de cortes gasta tempo $O(dm \log m)$, onde d é a dimensão do espaço das instâncias e m é a quantidade de exemplos de treinamento. Em alguns testes práticos, utilizando inclusive uma amostra de tamanho maior, o algoritmo de cortes foi cerca de 10.000 vezes mais rápido que o algoritmo ISI-2 descrito nos trabalhos [Barrera et al., 1995] e [Tomita, 1996] e gastou 3 vezes menos memória (Tabela 4.1). Observe que o teste não foi feito com aplicações idênticas, pois não dispomos nem do programa nem das imagens utilizadas nos trabalhos citados.

O algoritmo de cortes representa um operador como uma árvore binária que denominamos de árvore de cortes. Esta estrutura de dados é bastante semelhante à kd-árvore publicada nos artigos [Bentley, 1975] e [Friedman et al., 1977] e apresentada nesta tese no capítulo 5 e no anexo E. A kd-árvore fornece um operador mais preciso gastando, porém, mais tempo de processamento que a árvore de cortes.

Como já afirmamos, esta tese utiliza como base teórica a morfologia matemática para reticulados completos. Isto faz com que os algoritmos descritos nesta tese possam ser aplicados para praticamente todos os tipos de imagens: binárias, em níveis de cinza, coloridas, multi-espectrais, animações, tridimensionais, etc. Isto não ocorre com os trabalhos anteriores, pois [Dougherty, 1992a], [Dougherty, 1992b] e [Loce, 1993] trabalham com as teorias específicas para operadores binários e em níveis de cinza. O algoritmo descrito em [Barrera et al., 1995] e [Tomita, 1996] só projeta operadores binários.

O artigo [Harvey e Marshall, 1996] trabalha com base teórica em níveis de cinza, apesar de que aparentemente não haveria grande dificuldade em estender a técnica por eles apresentada para reticulados completos.

Um operador projetado manualmente dificilmente é composto por mais de algumas dezenas de operadores elementares. Projetando automaticamente um operador morfológico, muitas vezes o operador resultante é composto por centenas de milhares de operadores elementares. Evidentemente, está fora de questão tentar projetar manualmente um operador desse tamanho. Assim, esses operadores gigantes só são possíveis de serem gerados por um processo automático. Por outro lado, a criação automática faz aparecer um outro problema, inexistente quando o projeto é manual. Aplicar um operador composto por centenas de milhares de operadores numa imagem requer um tempo computacional proibitivo. Esta é a segunda dificuldade para utilizar a morfologia matemática: o tempo necessário para aplicar um operador gigante numa imagem.

Suponha, como costuma acontecer num computador Pentium 100 MHz, que leve 4 segundos para aplicar um operador elementar, como uma erosão ou uma dilatação com elemento estruturante (tamanho da janela) 2×2, a uma imagem colorida com, por exemplo, 480 × 621 pixels. Então, técnicas convencionais gastariam pelo menos 400.000 segundos, ou aproximadamente quatro dias, para aplicar 100.000 erosões. Pois as técnicas convencionais têm complexidade O(d*tn*), onde d é número de bandas da imagem vezes o tamanho do elemento estruturante, *t* é o número de operadores elementares e *n* é o número de pixels da imagem a ser processada. Enquanto isso, aplicar um operador composto de mais de 100.000 operadores elementares representado como uma árvore de cortes gastou apenas 8 segundos (ה), pois este algoritmo tem complexidade $O(n \log t)$. Todos os tempos de processamento desta tese foram medidos num computador Pentium 100 MHz.

O artigo [Jones and Svalbe, 1994a] utiliza “look-up table” para aumentar a velocidade dos operadores morfológicos. Porém, a idéia deles só pode ser utilizada no processa-

mento de imagens binárias. A árvore de cortes proposta e a kd-árvore podem ser utilizadas para acelerar processamento de imagens não necessariamente binárias.

Os trabalhos [Barrera et al., 1995] e [Tomita, 1996] utilizam o modelo PAC clássico para formalizar a aprendizagem computacional. Isso restringe o campo de trabalho a operadores binários apenas. Além disso, não permite lidar com exemplos que possuam conflitos. Nesta tese, a construção do operador pela aprendizagem está formalizada utilizando o modelo PAC generalizado (modelo de Haussler). O uso do modelo de Haussler dá uma sustentação teórica sólida que permite formalizar a aprendizagem de operadores não-binários assim como lidar com conflitos.

Ao contrário dos capítulos anteriores, daremos neste uma ênfase aos aspectos algorítmicos, de estruturas de dados e de análise da complexidade computacional.

4.2 Problema de Aprendizagem de Operador

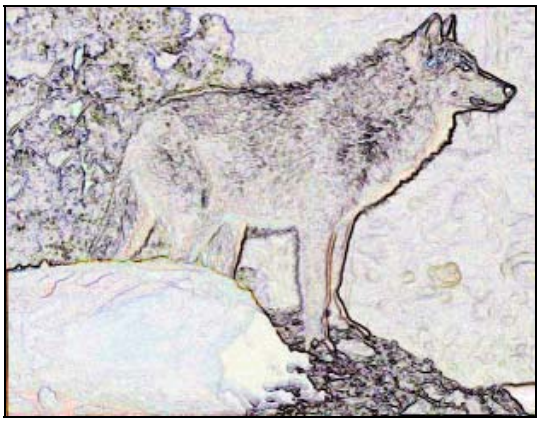
4.2.1 Descrição Informal

Nesta subseção, vamos descrever informalmente o processo de construção de um operador morfológico pela aprendizagem computacional. A ו ilustra este processo. Para simplificar a notação, suporemos que a coleção de imagens de treinamento consiste num único par de imagens entrada-saída, denotados $A^{\mathbf{x}}$ (amostra de entrada) e $A^{\mathbf{y}}$ (amostra de saída). Observe que quando a amostra consiste de mais de um par de imagens, podemos imaginar que "grudamos" todos eles para obter um único par de treinamento, com um envoltório de alguns pixels de cor de fundo.

(4.1.a) Exemplo de entrada $A^{\mathbf{x}}$.
297 × 442 pixels, 3 bytes por pixel.

(4.1.b) Exemplo de saída $A^{\mathbf{y}}$.
297 × 442 pixels, 3 bytes por pixel.

(4.1.c) Imagem a ser processada $Q^{\mathbf{x}}$.
480 × 621 pixels, 3 bytes por pixel.



(4.1.d) Imagem ideal $Q^{\mathbf{y}}$
480 × 621 pixels, 3 bytes por pixel.

(4.1.e) Imagem processada $Q^{\mathbf{p}}$ utilizando janela 2×2.
Tempo de treinamento: 22 s. Tempo de aplicação: 8 s.
Tamanho do operador: 1.201.022 bytes. Número de int-primitivos: 100.073.

Figura 4.1: Existe um filtro digital supostamente desconhecido ("find edge" do Adobe Photoshop 3.0) que, processando a imagem 4.1.a, fornece a imagem 4.1.b. Utilizando as imagens 4.1.a e 4.1.b como exemplos de treinamento, o algoritmo de cortes constrói o operador que emula o filtro desconhecido. Este operador, quando aplicado à imagem 4.1.c, fornece a imagem 4.1.e. A imagem 4.1.d é o resultado da aplicação do filtro "find edge" original na imagem 4.1.c.

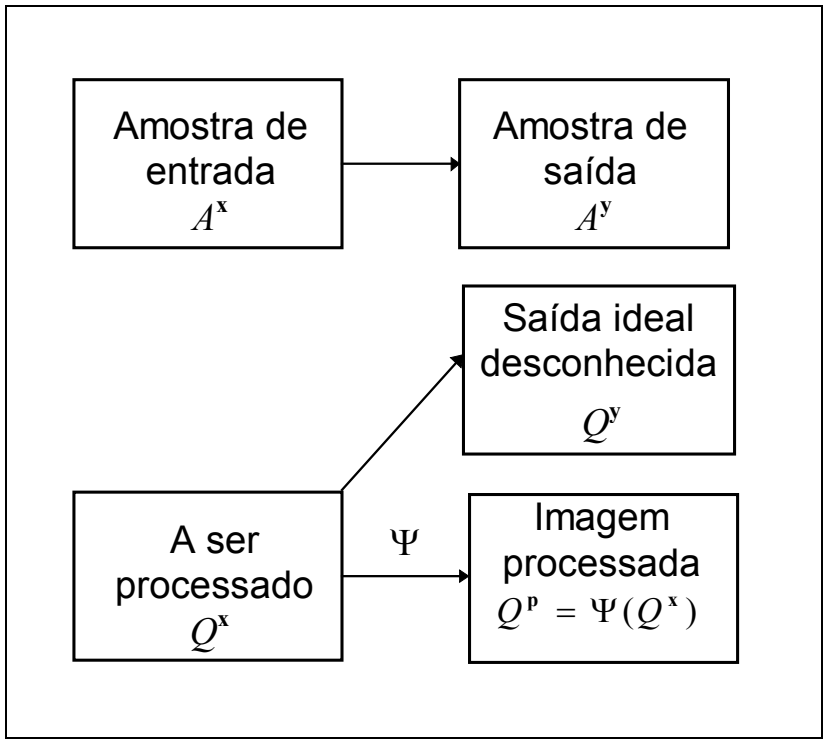
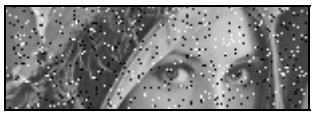
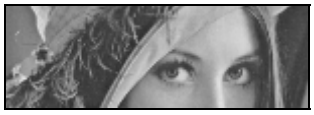


Figura 4.2: Esquema de construção de operador pela aprendizagem



(4.3.a) Amostra de entrada $A^{\mathbf{x}}$.
51 × 151 pixels, 1 byte por pixel.



(4.3.b) Amostra de saída $A^{\mathbf{y}}$.
51 × 151 pixels, 1 byte por pixel.

(4.3.c) Imagem a ser processado $Q^{\mathbf{x}}$.
256 × 256 pixels, 1 byte por pixel.

(4.3.d) Imagem processada $Q^{\mathbf{p}}$ pelo algoritmo de cortes usando janela 2×2.

(4.3.e) Saída ideal desconhecida $Q^{\mathbf{y}}$.

Figura 4.3: A imagem 4.3.c está corrompida pelo ruído conhecido como "sal e pimenta". Usando as imagens 4.3.a e 4.3.b como amostras de treinamento, o algoritmo de cortes aprende o comportamento do ruído e constrói um operador morfológico que o elimina. Este operador, quando aplicado na imagem ruidosa 4.3.c, gera a imagem filtrada correspondente 4.3.d. Esta imagem é semelhantes à saída ideal desconhecida 4.3.e. No capítulo 5 esta mesma aplicação é explicada mais detalhadamente.

Supõe-se existir uma distribuição de probabilidade desconhecida que gera as imagens $A^{\mathbf{x}}$ e $A^{\mathbf{y}}$. A mesma distribuição gera o par de imagens $Q^{\mathbf{x}}$ (imagem a ser processada) e $Q^{\mathbf{y}}$ (saída ideal). A imagem $Q^{\mathbf{y}}$ é considerada desconhecida, isto é, não temos acesso a ela. A partir das amostras $A^{\mathbf{x}}$ e $A^{\mathbf{y}}$, queremos achar um operador morfológico Ψ tal que, dada a imagem a ser processada $Q^{\mathbf{x}}$, gere a imagem processada $Q^{\mathbf{p}} = \Psi(Q^{\mathbf{x}})$ de modo que $Q^{\mathbf{p}}$ seja "provavelmente semelhante" à imagem ideal $Q^{\mathbf{y}}$ desconhecida.

Para facilitar a compreensão deste processo, veja um exemplo de aplicação concreta na ı. Queremos eliminar os ruídos da imagem em níveis de cinza $Q^{\mathbf{x}}$ (c). Este processo pode ser dividido em duas etapas: *treinamento* e *aplicação*. Na etapa de treinamento, recorta-se uma sub-imagem $A^{\mathbf{x}}$ (a) de $Q^{\mathbf{x}}$ (c). Suponha que dispomos de alguma maneira desta mesma sub-imagem sem ruído ou que eliminamos o ruído manualmente obtendo a imagem $A^{\mathbf{y}}$ (b). Um algoritmo de aprendizagem irá construir automaticamente um operador que transforma a sub-imagem ruidosa $A^{\mathbf{x}}$ (a) na sub-imagem sem ruídos $A^{\mathbf{y}}$ (b). A etapa de aplicação consiste em aplicar o operador obtido na fase de treinamento à imagem ruidosa $Q^{\mathbf{x}}$ (c), obtendo a imagem filtrada $Q^{\mathbf{p}}$ (d). Compare esta com a imagem ideal sem ruídos $Q^{\mathbf{y}}$ (e) considerada desconhecida. Podemos verificar que a eliminação de ruídos foi bastante boa. No capítulo 5, esta mesma aplicação pode ser vista utilizando janelas de tamanhos diferentes, assim como comparada com a imagem obtida pelo algoritmo kd-árvore.

4.2.2 Formalização

Uma vez que tivemos, na subseção anterior, uma visão informal do problema que queremos resolver, vamos formalizá-lo nesta subseção. No anexo D, a aprendizagem operacional está colocada no contexto da teoria da estimação estatística. As definições da imagem digital, cor de fundo e suporte já foram apresentadas no capítulo 2. Sejam (E, +) um grupo abeliano e $\mathcal{L}^{\mathbf{x}}$ e $\mathcal{L}^{\mathbf{y}}$ dois reticulados arbitrários. Então as cinco imagens digitais envolvidas no processo de aprendizagem de operador são definidas matematicamente como cinco funções:

$$A^{\mathbf{x}}, Q^{\mathbf{x}} \in (\mathcal{L}^{\mathbf{x}})^E \quad \text{e} \quad A^{\mathbf{y}}, Q^{\mathbf{p}}, Q^{\mathbf{y}} \in (\mathcal{L}^{\mathbf{y}})^E.$$

Consideraremos que $\gamma(A^{\mathbf{x}}) = \gamma(Q^{\mathbf{x}})$ e denotaremos esta cor, a cor de fundo das imagens de entrada, por $\gamma^{\mathbf{x}}$. Da mesma forma, consideraremos $\gamma(A^{\mathbf{y}}) = \gamma(Q^{\mathbf{p}}) = \gamma(Q^{\mathbf{y}})$ e denotaremos esta cor, a cor de fundo das imagens de saída, por $\gamma^{\mathbf{y}}$. Consideraremos o suporte das imagens $A^{\mathbf{x}}$ e $A^{\mathbf{y}}$ iguais e o denotaremos como $S(A) = S(A^{\mathbf{x}}) = S(A^{\mathbf{y}})$. Além disso, denotaremos a sua cardinalidade por $m = |S(A)|$. Da mesma forma, $S(Q) = S(Q^{\mathbf{x}}) = S(Q^{\mathbf{p}}) = S(Q^{\mathbf{y}})$ e $n = |S(Q)|$.

Para continuar a formalização do problema, faremos mais uma série de suposições. Primeiro, para não sobrecarregar desnecessariamente a notação, suporemos $\mathcal{L}^{\mathbf{x}} = \mathcal{L}^{\mathbf{y}}$ e denotaremos esse reticulado como $\mathcal{L}$. Observe que esta suposição tem como única finalidade simplificar a notação, pois os resultados continuam idênticos sem ela. Segundo, supomos que $\mathcal{L}$ é um produto de b cadeias limitadas, pois isto é verdade em praticamente todas as imagens digitais reais de modo que não estamos restringindo o problema na prática. Também suporemos que todas as b cadeias limitadas são iguais e assim podemos escrever $\mathcal{L} = \mathcal{K}^b$. Esta suposição tem por finalidade não criar, sem necessidade, índices em excesso. Como qualquer cadeia $\mathcal{K}$ possui um isomorfismo com qualquer intervalo de números inteiros com $k = |\mathcal{K}|$ elementos, suporemos que $\mathcal{K} = [0 \ldots k\text{-}1]$. Com estas suposições, podemos escrever: $A^{\mathbf{x}}, Q^{\mathbf{x}}, A^{\mathbf{y}}, Q^{\mathbf{p}}, Q^{\mathbf{y}} \in (\mathcal{K}^b)^E$.

Como observamos no capítulo 2, o objetivo deste trabalho é o projeto automático de W-operadores. Suponha que uma janela $\vec{W} = (W_1, W_2, ..., W_w) \in E^w$ foi escolhida de alguma forma. A escolha da janela apropriada leva mais rapidamente a um resultado melhor. No capítulo 5, discutimos a escolha da janela conveniente.

Uma vez fixada uma janela $\vec{W}$ e restringindo o espaço dos operadores a W-operadores, a aprendizagem do operador pode ser encarada como aprendizagem da função caracte-

rística do W-operador, pois um W-operador é univocamente determinado pela sua função característica.

Cada pixel p de $A^{\mathbf{x}}$, juntamente com os seus pixels vizinhos que pertencem à janela $\vec{W} + p$, isto é, $\vec{W}$ transladada a p, formam um ponto $a^{\mathbf{x}}$ no espaço $\mathcal{K}^{d}$, onde $d = w$ é a dimensão do domínio da função característica:

$$a^{\mathbf{x}} = (A^{\mathbf{x}}(W_1 + p), A^{\mathbf{x}}(W_2 + p), \ldots, A^{\mathbf{x}}(W_w + p)).$$

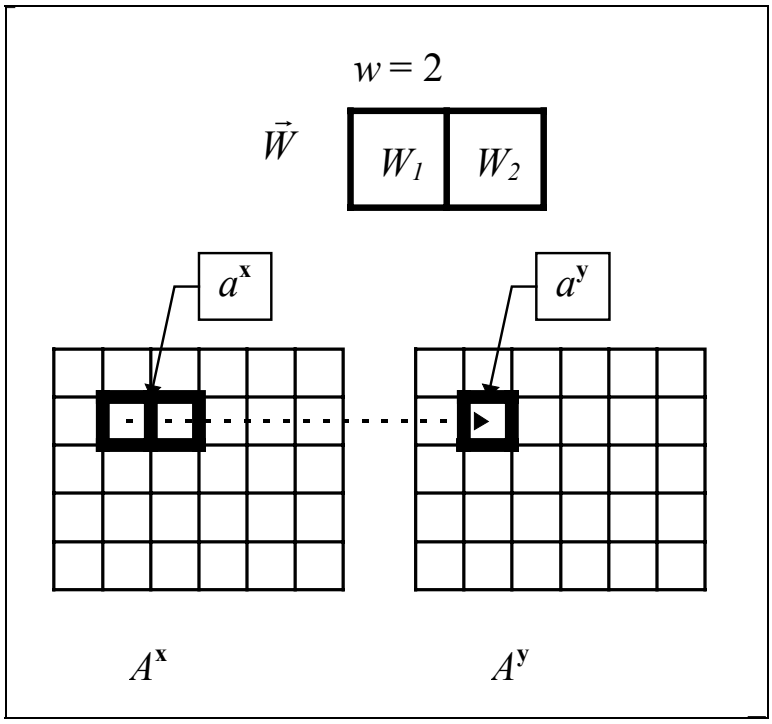


Figura 4.4: Geração da amostra de treinamento $\vec{a}$ a partir de imagens em níveis de cinza $A^{\mathbf{x}}$ e $A^{\mathbf{y}}$ usando uma janela 1 × 2.

Evidentemente, se para algum j, $W_j + p$ não pertence ao suporte de $A^{\mathbf{x}}$, isto é, $W_j + p \notin S(A^{\mathbf{x}})$ então o valor $A^{\mathbf{x}}(W_j + p)$ deve ser considerado como a cor de fundo das imagens de entrada $\gamma^{\mathbf{x}}$. Para cada ponto $a^{\mathbf{x}}$ assim obtido, existe um valor de saída associado $a^{\mathbf{y}} \in \mathcal{K}^{6}$, a cor correspondente na imagem de saída $A^{\mathbf{y}}$. A ⊓ ilustra este pro-

cesso. Denominaremos estas informações como amostra de treinamento e denotaremos como:

$$\vec{a} = (a_1,\ldots,a_m) = ((a_1^{\mathbf{x}},a_1^{\mathbf{y}}),\ldots,(a_m^{\mathbf{x}},a_m^{\mathbf{y}})), \quad \vec{a} \in (\mathcal{K}^d \times \mathcal{K}^b)^m .$$

Também denotaremos:

$$\vec{a}^{\mathbf{x}} = (a_1^{\mathbf{x}},\ldots,a_m^{\mathbf{x}}), \quad \vec{a}^{\mathbf{x}} \in (\mathcal{K}^d)^m \quad \text{e}$$

$$\vec{a}^{\mathbf{y}} = (a_1^{\mathbf{y}},\ldots,a_m^{\mathbf{y}}), \quad \vec{a}^{\mathbf{y}} \in (\mathcal{K}^b)^m .$$

A seqüência $\vec{a}$ será usada como a amostra de treinamento do algoritmo de aprendizagem.

De forma análoga, as imagens $Q^{\mathbf{x}}$ e $Q^{\mathbf{y}}$ formam *n* pontos no espaço $\mathcal{K}^d \times \mathcal{K}^b$ que denotaremos como:

$$\vec{q} = (q_1,\ldots,q_n) = ((q_1^{\mathbf{x}},q_1^{\mathbf{y}}),\ldots,(q_n^{\mathbf{x}},q_n^{\mathbf{y}}))$$

Também definimos analogamente $\vec{q}^{\mathbf{x}}$ (seqüência a ser processada) e $\vec{q}^{\mathbf{y}}$ (saída ideal desconhecida).

Suporemos que existe uma distribuição de probabilidade conjunta *P* em $\mathcal{K}^d \times \mathcal{K}^b$ que gera independentemente cada elemento das seqüências $\vec{a}$ e $\vec{q}$. Com isso, podemos reescrever o processo como:

a) **Etapa de treinamento:** São dadas as imagens de treinamento $A^{\mathbf{x}}$, $A^{\mathbf{y}}$ e uma janela $\vec{W}$. Estes dados podem ser vistos como uma amostra de treinamento $\vec{a}$. Cada elemento $(a^{\mathbf{x}},a^{\mathbf{y}}) \in \vec{a}$ é imaginado como tendo sido gerado independentemente por uma distribuição de probabilidade conjunta *P* desconhecida. Um algoritmo de aprendizagem $\mathcal{A}$ recebe a amostra de treinamento $\vec{a}$ e constrói a função característica $\psi = \mathcal{A}(\vec{a})$. A função ψ e a janela $\vec{W}$ juntos representam um W-operador Ψ.

b) **Etapa de aplicação:** São dadas a janela $\vec{W}$ (a mesma da etapa anterior) e a imagem a processar $Q^{\mathbf{x}}$. Além disso, existe uma imagem de saída ideal $Q^{\mathbf{y}}$ desconhecida. Estes dados ($Q^{\mathbf{x}}$, $\vec{W}$ e $Q^{\mathbf{y}}$) podem ser vistos como uma seqüência $\vec{q}$. Suporemos que a

mesma distribuição P da etapa de treinamento é responsável pela geração independente de cada elemento $(q^{\mathbf{x}}, q^{\mathbf{y}}) \in \vec{q}$. O algoritmo de aplicação calcula $q^{\mathbf{p}} = \psi(q^{\mathbf{x}})$ para cada $q^{\mathbf{x}}$, gerando a imagem processada $Q^{\mathbf{p}}$. Se o processo de aprendizagem foi "bom", cada valor $q^{\mathbf{p}}$ será "próximo" ao valor de saída desconhecida $q^{\mathbf{y}}$.

Com as idéias expostas acima, estamos em condições de escrever o problema de aprendizagem do operador como um problema de aprendizagem do modelo de Haussler:

Definição (problema de aprendizagem de W-operador): Um *problema de aprendizagem de W-Operador* é um problema de aprendizagem generalizado do modelo de Haussler onde

$$(X, Y, A, \mathcal{H}, \mathcal{P}, \mathrel{\llcorner}) = (\mathcal{K}^{d}, \mathcal{K}^{b}, \mathcal{K}^{b}, \mathcal{K}^{d} \to \mathcal{K}^{b}, \mathcal{P}, \mathrel{\llcorner}).$$

Com esta definição, todos os resultados que obtivemos no capítulo 3 para problema de aprendizagem generalizado poderão ser utilizados aqui.

4.2.3 Complexidade de Amostra

No capítulo 3, demonstramos que a complexidade de amostra de um algoritmo de aprendizagem consistente é:

$$m_o(\varepsilon, \delta) \leq \frac{M^2}{2\varepsilon^2}\left(\ln\left(2\left|\mathcal{H}\right|\right) - \ln(\delta)\right)$$

A complexidade de aprendizagem acima é útil para demonstrar a convergência teórica do processo de aprendizagem. Porém, convém observar que esse tamanho de amostra m_0 está muito além de qualquer amostra possível de ser obter na prática. Para nos certificarmos disso, vamos considerar um problema prático, por exemplo o da ŕ, onde queremos aprender um operador com 256 níveis de cinzas com janela de tamanho 4. Então,

$$\ln(|\mathcal{H}|) = \ln(|Y|^{|X|}) = |\mathcal{K}^{d}|\ln(|\mathcal{K}^{b}|) = (256^4)\ln(256) = 2{,}3816 \times 10^{10}.$$

Supondo, ε=0,1, δ=0,1, M=1, obtemos $m_0 \leq 1{,}1908\times10^{12}$. Isto significa que precisamos alimentar o algoritmo de aprendizagem com $3{,}876\times10^{6}$ pares de imagens 480×640! Ora, está fora de questão tentar obter tal quantidade de imagens de treinamento.

Podemos afirmar que não existe nenhum estudo teórico que forneça uma complexidade de amostra que possa ser utilizado na prática tanto para o problema que estamos tratando assim como para um problema de aprendizagem de Haussler em geral. Assim, infelizmente, devemos nos contentar com os resultados teóricos que mostram a convergência para um número astronômico de amostras e as experiências práticas que mostram que a convergência se dá muito antes de atingir esse número.

Um fator que torna a convergência prática da aprendizagem do operador mais rápida que a convergência teórica é o seguinte: duas cores que são muito "semelhantes" entre si são dificilmente distinguíveis assim como dois pedaços de imagens "parecidos" entre si parecem ser iguais. Assim, qualquer operador útil deve levar duas cores semelhantes (ou duas configurações de pixels semelhantes numa janela) a duas saídas também semelhantes.

Vamos fazer algumas suposições e contas para que esta idéia fique mais clara. Considere que estamos trabalhando com imagens com 256 níveis de cinza mas que somente 16 níveis de cinza são distinguíveis à visão de um ser humano. Trabalhando com janela de tamanho 4, temos:

$$\ln(|\mathcal{H}|) = \ln(|Y|^{|X|}) = |\mathcal{K}^d|\ln(|\mathcal{K}^b|) = (16^4)\ln(16) = 1{,}817\times10^5$$

Supondo, como no exemplo anterior, ε=0,1, δ=0,1, M=1, obtemos $m_0 \leq 9{,}085\times10^{6}$. Isto significa que precisamos alimentar o algoritmo de aprendizagem com 30 pares de imagens 480×640. Ora, isto já é algo mais factível.

Outro fator que reduz, na prática, a complexidade de amostra é o fato de grande parte do espaço das instâncias ter probabilidade próxima a zero de aparecer. Falando informalmente, através de exemplos, se a nossa aplicação visa reconhecer letras romanas, é de se esperar que nunca apareçam caracteres árabes ou chineses nas imagens de entrada. Ali-

ás, se isto ocorresse, qualquer saída gerada pelo sistema seria satisfatória. Ou então, se a aplicação é reconhecer falhas em circuitos impressos, podemos desprezar o comportamento do sistema quando a imagem de entrada é uma micro-fotografia do músculo cardíaco. Isto reduz na prática a cardinalidade do espaço das instâncias $\mathcal{K}^d$ o que conseqüentemente reduz a cardinalidade do espaço dos operadores $\mathcal{H}$. Levando este fator em consideração, pode-se esperar que a complexidade de amostra diminua ainda mais.

Em resumo, a complexidade de amostra teórica só serve como uma demonstração da convergência do processo de aprendizagem para amostras de tamanho muito grande. Numa aplicação prática, a complexidade de amostra deve ser estimada empiricamente, pois não há nenhum resultado teórico que permita estimá-lo com a devida acuidade. É de se esperar que não exista estimativa precisa, uma vez que não conhecemos a distribuição P. À medida em que restringimos o espaço das distribuições de probabilidade $\mathcal{P}$, é possível estimar mais precisamente a complexidade de amostra. Observe que os dois fatores que diminuem a complexidade, citados acima, no fundo restringem o espaço $\mathcal{P}$.

4.2.4 Análise de Complexidade de Tempo

Conforme descrevemos na subseção anterior, uma análise de complexidade de aprendizagem em função de ε e δ é útil para demonstrar a convergência teórica, mas não permite afirmar se um algoritmo de aprendizagem é mais rápido ou lento que um outro, pois dois algoritmos podem gastar diferentes tempos de processamento para amostras de treinamento de tamanhos iguais.

Portanto, analisaremos a complexidade computacional dos algoritmos em função do tamanho de amostra de treinamento m e tamanho da seqüência a ser processada n, num espaço das instâncias de dimensão d. Quando necessário também utilizaremos o parâmetro b, a dimensão do espaço de saída.

4.3 Algoritmos de Aprendizagem

Na última seção definimos o problema de aprendizagem de W-operador, a complexidade de amostra da aprendizagem na teoria e na prática e o critério para analisar a complexidade de tempo dos algoritmos. O objetivo desta seção é achar algoritmos de aprendizagem eficientes que tenham a garantia de convergência. Vimos no capítulo 3 que PAC é uma garantia de convergência. Também vimos que todo algoritmo consistente é PAC. Relembrando a definição de consistência do modelo de Haussler, dizemos que um algoritmo $\mathcal{A}^+$ é *consistente* para um problema de aprendizagem de W-operador se,

$$C_{\vec{a}}(q^{\mathbf{x}}) \text{ é uma seqüência não-nula } \Rightarrow \ \mathcal{A}^+(\vec{a})(q^{\mathbf{x}}) = \mathcal{M}(C^{\mathbf{y}}_{\vec{a}}(q^{\mathbf{x}})).$$

A função minimizadora de penalidade $\mathcal{M}$ e a subseqüência coincidente C estão definidas no capítulo 3.

Os tempos de processamento dos algoritmos foram medidos num computador Pentium 100 MHz com 32 MBytes de memória. Outros detalhes de implementação são apresentados no capítulo 6.

4.3.1 Algoritmo Força Bruta Consistente

Existe um algoritmo consistente óbvio que denominamos de "força bruta". Este algoritmo procura exaustivamente, para cada elemento $q^{\mathbf{x}}$ da seqüência a ser processada $\vec{q}^{\,\mathbf{x}}$, a amostra de treinamento $(a^{\mathbf{x}}, a^{\mathbf{y}}) \in \vec{a}$ tal que $q^{\mathbf{x}} = a^{\mathbf{x}}$. Se existir tal elemento, a saída do operador será $a^{\mathbf{y}}$. Senão, define-se um valor arbitrário, por exemplo **0**, como saída.

Abaixo, descrevemos esse algoritmo numa linguagem semelhante a Pascal. As estruturas de dados que utilizaremos são:

```
1.   type
2.     Kd = array [1..d] of byte;
3.     Kb = array [1..b] of byte;
```

```
4.      exemplo = record
5.        x:Kd;
6.        y:Kb;
7.      end;
```

Isto é, denotaremos por `Kd` o produto de cadeias $\mathcal{K}^d$, por `Kb` o produto de cadeias $\mathcal{K}^b$ e por `exemplo` um ponto do espaço $\mathcal{K}^d \times \mathcal{K}^b$. O algoritmo é descrito abaixo, onde o vetor `a` indica a amostra de treinamento $\vec{a}$ e os vetores `qx` e `qp` indicam respectivamente as seqüências $\vec{q}^{\mathbf{x}}$ e $\vec{q}^{\mathbf{p}}$.

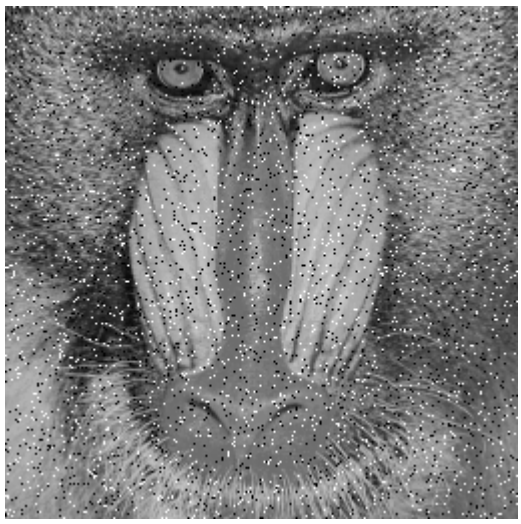
```
1.    procedure FBConsistente(
2.      var a:array [1..m] of exemplo;
3.      var qx:array [1..n] of Kd;
4.      var qp:array [1..n] of Kb);
5.    var coincidente:seqüência de Kb; i,j:integer;
6.    begin
7.      for i:=1 to n do begin
8.        coincidente:=seqüência_vazia;
9.        for j:=1 to m do
10.         if a[j].x=qx[i] then
11.           Insira a[j].y no final de coincidente;
12.       if coincidente<>seqüência_vazia
13.         then qp[i]:=minimizadora(coincidente)
14.         else qp[i]:=0;
15.     end;
16.   end;
```

A função `minimizadora(coincidente)` calcula a função minimizadora de penalidade da seqüência `coincidente`. O valor `0` indica o elemento mínimo de `Kb`.

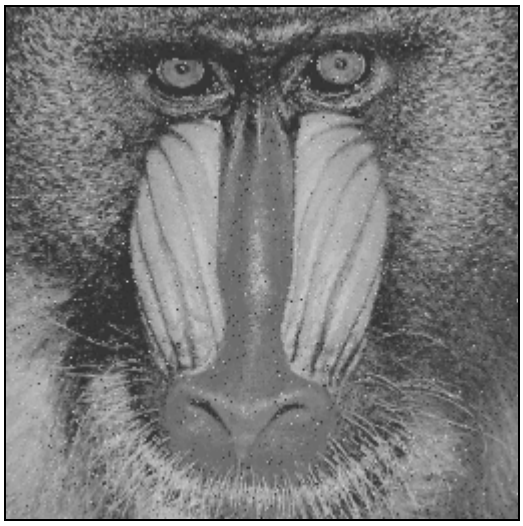
Se as amostras de treinamento são geradas por um W-operador Ψ e se temos exemplo(s) de cada ponto do domínio então o procedimento acima irá gerar uma representação por intervalos exata do Ψ. Vamos analisar agora a complexidade do algoritmo força bruta consistente:

(4.5.a) Imagem obtida ($Q^{\mathbf{p}}$) processando a imagem 4.3.c pelo algoritmo força bruta com exemplos de treinamento 4.3.a e 4.3.b (janela 2×2). Compare-a com a imagem 4.3.d.

Tempo de processamento: 2.806 s.

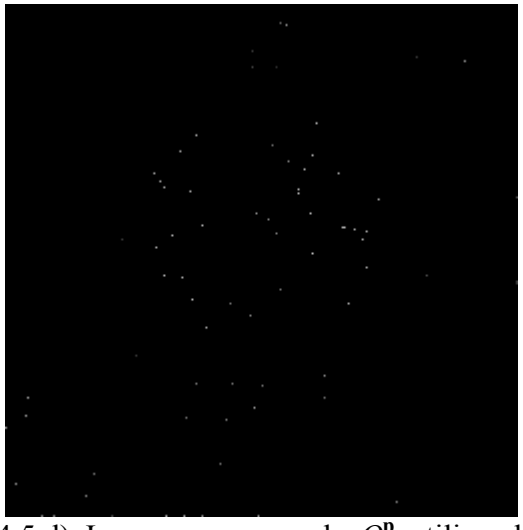


(4.5.b) Imagem ruidosa $Q^{\mathbf{x}}$ a ser processada



(4.5.c) Imagem processada $Q^{\mathbf{p}}$ utilizando o operador 2×2 (algoritmo de cortes) obtido das amostras 4.3.a e 4.3.b.

Tempo de processamento: 1s.



(4.5.d) Imagem processada $Q^{\mathbf{p}}$ utilizando as imagens 4.3.a e 4.3.b como exemplos de treinamento (força bruta, janela 2×2).

Tempo de processamento: 2.806 s.

Figura 4.5: Comparação do algoritmo de cortes com algoritmo força bruta consistente. Observe que o algoritmo força bruta, apesar de possuir a garantia de convergência teórica (PAC), não pode ser utilizado na prática, pois a convergência é muito lenta. Isto pode ser comprovada pela baixíssima qualidade das imagens processadas.

Proposição 4.1 (`FBConsistente`): O algoritmo `FBConsistente` acima tem complexidade de tempo $O(mn\alpha)$.

Prova: O teste de igualdade da linha 10 leva tempo $O(\alpha)$. Essa linha repete-se O(mn) vezes. Logo, todo o algoritmo demora $O(mn\alpha)$. ■

Observe que a saída de uma instância $q^{\mathbf{x}}$ do algoritmo `FBConsistente` é **0** quando não existem exemplos de treinamentos idênticos ao $q^{\mathbf{x}}$. Isto faz com que a convergência do algoritmo seja muito lenta, impossível de ser utilizado na prática (veja a ט). Num caso desses, o algoritmo de cortes da próxima subseção irá aproximar a saída da instância $q^{\mathbf{x}}$ para a saída de uma instância conhecida "mais ou menos próxima" de $q^{\mathbf{x}}$ e o algoritmo do vizinho mais próximo irá aproximá-lo para a saída do verdadeiro exemplo mais próximo.

4.3.2 Árvore de Cortes

Na subseção anterior, descrevemos o algoritmo força bruta onde as etapas de treinamento e aplicação são feitas ao mesmo tempo gastando tempo $O(\alpha nm)$. Nesta seção apresentaremos um outro algoritmo consistente, denominado *algoritmo de cortes*, cujo tempo de treinamento é $O(\alpha m \log m)$ e o tempo de aplicação é $O(n \log m)$ resultando na complexidade total $O((\alpha m + n) \log m)$. Observe que este resultado teórico indica que o algoritmo de cortes é muito mais rápido que a força bruta. De fato, o algoritmo de cortes gastou apenas 1 segundo de treinamento e 1 segundo de aplicação para resolver um problema onde a força bruta gastou 2.806 segundos (ט). As imagens envolvidas nesse problema eram pequenas e as diferenças de desempenho entre os dois algoritmos tendem a aumentar ainda mais à medida que as imagens tornam-se maiores.

Uma árvore de cortes só pode ser construída para mapeamentos entre produtos de cadeias. Mas, como já observamos no capítulo 2, isto não constitui uma limitação na prática.

O algoritmo de cortes utiliza a int-partição definida no capítulo 2 para representar um operador (a int-partição está definida no capítulo 2). Para acelerar o algoritmo, a int-partição é representada como uma árvore binária que denominamos de *árvore de cortes*. Nem toda int-partição pode ser representada como uma árvore binária mas qualquer mapeamento entre dois produtos de cadeias possui uma int-partição que pode ser armazenada numa árvore.

Do ponto de vista algorítmico, esta função foi originalmente inspirada no *Median Cut Algorithm* utilizada para escolher uma paleta para uma imagem colorida descrita em [Heckbert, 1982]. No problema de escolha de paleta, dada uma imagem colorida com, por exemplo, 16 milhões de cores (3 bytes por pixel) deve-se escolher algumas cores representantes, por exemplo 256 cores, de forma que quando cada cor de pixel da imagem é aproximada para uma das 256 cores representantes, tenhamos a menor erro médio provocado pela aproximação. Heckbert dá uma solução aproximada, porém rápida para este problema, cortando recursivamente o espaço das cores por planos situados na mediana. Depois de ter implementado a árvore de cortes, verificamos que esta é muito semelhante à kd-árvore ([Bentley, 1975], [Friedman et al., 1977] e [Preparata and Shamos, 1985]). A kd-árvore é utilizada para resolver a busca do vizinho mais próximo. Neste trabalho, a kd-árvore está descrita no capítulo 5 e no anexo E. No problema da busca do vizinho mais próximo, são dados m pontos num espaço de dimensão d. Deve-se construir alguma estrutura de dados tal que, quando n pontos adicionais forem dados, possamos buscar rapidamente, para cada um deles, o seu vizinho mais próximo dentre os m primeiros pontos.

No algoritmo de cortes, um hiper-plano paralelo aos eixos do sistema de coordenadas particiona o hiper-paralelepípedo $\mathcal{K}^d$. Este hiper-plano é escolhido de modo que os pontos da amostra sejam divididos em dois conjuntos com aproximadamente mesma cardinalidade. Para cada um dos dois hiper-paralelepípedos assim obtidos, o processo de partição continua recursivamente, gerando mais e mais hiper-paralelepípedos. Este processo pára quando cada hiper-paralelepípedo contém somente pontos com o mesmo

valor de saída ou pontos com a mesma coordenada em $\mathcal{K}^d$ (neste caso, temos um conflito).

A idéia de construir hiper-paralelepípedos contendo exemplos com saídas semelhantes (podemos dizer, "clusters" de exemplos) é interessante do ponto de vista prático, pois permitiria construir mais rapidamente operadores menores. Isto poderia ser conseguido parando o processo de recursão quando todos os exemplos de um paralelepípedo tiverem saídas "próximas" entre si. Porém, utilizando esta heurística, do ponto de vista teórico, o algoritmo de aprendizagem deixaria de ser PAC.

Cada hiper-plano obtido corresponde a um nó interno da árvore e cada hiper-paralelepípedo corresponde a uma folha (veja). Uma árvore de cortes possui as seguintes propriedades:

a) Cada hiper-paralelepípedo $[a,b]$ representa um int-primitivo $\lambda_{[a,b],v}$ e é armazenado numa folha da árvore de cortes. O valor v é o valor da minimizadora de penalidade dos pontos exemplos que pertencem ao intervalo $[a,b]$.
b) Denotando o conjunto de todos os int-primitivos de uma árvore de cortes como $\mathcal{B} = \{\lambda_1, \lambda_2, \ldots, \lambda_t\}$, $\vee\mathcal{B}$ é uma int-partição de uma função característica ψ.
c) Dado um ponto $x \in \mathcal{K}^d$, $\psi(x)$ é igual à saída de um único int-primitivo, digamos $\lambda_e(x)$, porque para todo $j \in [1, t]$, $j \neq e$, $\lambda_j(x) = \mathbf{0}$.
d) Dado um ponto $x \in \mathcal{K}^d$, a estrutura de árvore binária permite busca rápida do λ_e e conseqüentemente um cálculo rápido de $\psi(x)$.

Além das estruturas de dados mencionadas na subseção anterior, utilizaremos também as seguintes:

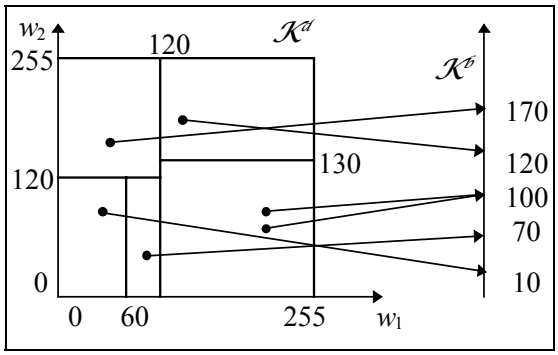
```
1.   type
2.     NodePt = ^Node;
3.     Node = record
4.       case leaf:Boolean of
5.         false:(
```

```
6.          c:integer; {valor de corte, número no intervalo [0..255]}
7.          p:integer; {plano de corte, número no intervalo [1..d]}
8.          left:NodePt; {apontador para filho à esquerda}
9.          right:NodePt; {apontador para filho à direita}
10.       );
11.     true:(
12.       a:Kd; {limite inferior do intervalo}
13.       b:Kd; {limite superior do intervalo}
14.       v:Kb; {valor de saída}
15.     );
16.   end;
```

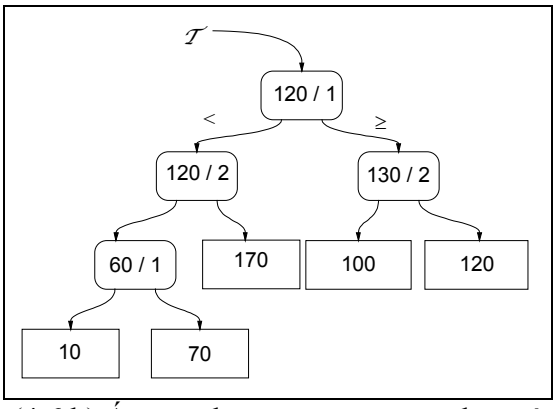
Quando uma árvore de cortes estiver completamente construída, os valores `a` e `b` das folhas tornam-se redundantes, pois os nós internos da árvore armazenam a mesma informação. A função recursiva abaixo constrói uma árvore de cortes. Inicialmente, os parâmetros `l` e `r` devem ser 1 e `m`, respectivamente.

```
1.  function ConstroiCortes(
2.    var a:array [1..m] of exemplo;
3.    l,r:integer): NodePt;
4.  var t,tl,tr:NodePt; j,k,p,c:integer;
5.  begin
6.    if (a[l].y=a[l+1].y=...=a[r].y) then begin
7.      new(T); T^.leaf:=true; T^.v:=a[l].y;
8.    end else if (a[l].x=a[l+1].x=...=a[r].x) then begin
9.      new(T); T^.leaf:=true;
10.     T^.v:=minimizadora(a[l].y,a[l+1].y,...,a[r].y);
11.   end else begin
12.     p:=MaxEspalhamento(a,l,r);
13.     particiona(a,l,r,p,k,c);
14.     tl:=ConstroiCortes(a,l,k);
15.     tr:=ConstroiCortes(a,k+1,r);
16.     new(t); t^.leaf:=false; t^.p:=p; t^.c:=c;
17.     t^.left:=tl; t^.right:=tr;
18.   end;
19.   ConstroiCortes:=t;
20. end;
```

A função `minimizadora(a[l].y,a[l+1].y,...,a[r].y)` calcula a função minimizadora de penalidade. A função `MaxEspalhamento` abaixo calcula o eixo no qual os pontos estão mais dispersos.



(4.6.a) Algoritmo de cortes particiona o espaço $\mathcal{K}^d$ para construir uma int-partição.



(4.6.b) Árvore de cortes correspondente à partição da figura (4.6.a).

Figura 4.6: Sejam $\mathcal{L}^x = [0, 255]^2$, $\mathcal{L}^y = [0, 255]$ e ψ: $\mathcal{L}^x \rightarrow \mathcal{L}^y$ conhecida parcialmente através da seguinte amostra de treinamento:

$\vec{a}$ =([(30, 100), 10], [(40, 140), 170], [(90, 40), 70], [(150, 160), 120], [(200, 60), 100], [(200, 100), 100]).

Então, calcula-se o "espalhamento" nos eixos w_1 e w_2 obtendo 200-30=170 e 160-40=120, respectivamente. Assim, o eixo w_1 é escolhido como o plano de corte, pois 170>120. Achando a mediana das coordenadas dos pontos no eixo w_1, obtemos (150+90)/2=120. Este primeiro corte é indicado na figura 4.6.b como 120 / 1, onde 120 indica o valor de corte (a mediana) e 1 indica o eixo de corte (w_1).

Considerando somente os pontos à esquerda da primeira corte, obtemos o espalhamento nos eixos w_1 e w_2 respectivamente 90-30=60 e 140-40=100. O eixo w_2 é escolhido, pois 60<100. Achando a mediana em w_2 obtemos (140+100)/2=120. Este corte é indicado como 120 / 2.

Considerando somente os pontos (30, 100) e (90, 40), obtemos os espalhamentos nos eixos w_1 e w_2 ambos iguais a 60. Assim sendo, poderíamos particionar em qualquer dos dois eixos. Escolhendo w_1, obtemos a corte 60 / 1.

Considerando os pontos (150, 160), (200, 60) e (200, 100), concluímos que devemos cortar no eixo w_2 e obtemos a corte 130 / 2.

Os pontos (200, 60) e (200, 100) não são cortados pois o valor de saída de ambos é igual a 100.

```
1.   function MaxEspalhamento(
2.     var a:array [1..m] of exemplo;
3.     l,r:integer):integer;
4.   var max,e,p,i:integer;
5.   begin
6.     p:=l;
7.     max:=espalhamento(a,l,r,p);
8.     for i:=l+1 to r do begin
9.       e:=espalhamento(a,l,r,i);
10.      if e>max then begin
11.        p:=i; max:=e;
12.      end;
13.    end;
14.    MaxEspalhamento:=p;
15.  end;
```

A função `espalhamento(a,l,r,p)` abaixo calcula o tamanho do menor intervalo ao qual os pontos `a[l].x[p], a[l+1].x[p], ..., a[r].x[p]` pertencem.

```
1.   function espalhamento(
2.     var a:array [1..m] of exemplo;
3.     l,r,p:integer):integer;
4.   var max,min,i:integer;
5.   begin
6.     min:=a[l].x[p]; max:=min;
7.     for i:=l+1 to r do begin
8.       if a[i].x[p]<min then min:=a[i].x[p];
9.       if a[i].x[p]>max then max:=a[i].x[p];
10.    end;
11.    espalhamento:=max-min;
12.  end;
```

O procedimento `particiona` abaixo divide os exemplos `a[l], a[l+1], ..., a[r]` em dois subconjuntos.

```
1.   procedure particiona(
2.     var a:array [1..m] of exemplo;
3.     l,r,p:integer;
4.     var k,c:integer);
5.   begin
6.     Escolha inteiros k (l≤k<r) e c tais que
7.     para todo i∈[l..k], temos a[i].x[p]≤c e
8.     para todo i∈[k+1..r], temos a[i].x[p]>c.
9.   end;
```

Nem todo valor `k (l≤k<r)` permite particionar o vetor `a` de modo a satisfazer as condições das linhas 7 e 8. Porém, existe pelo menos um valor `k` que permite particionar o vetor. Se tivéssemos

```
a[l].x[p]=a[l+1].x[p]=...=a[r].x[p]                    (1)
```

então nenhum valor `k` particionaria o vetor. Porém, isto não pode acontecer, pois o eixo de corte `p` escolhido é aquele onde os pontos estão mais espalhados e assim (1) só poderia ocorrer se tivéssemos

```
a[l].x=a[l+1].x=...=a[r].x                             (2)
```

Mas, neste caso, a condição da linha 8 da `ConstroiCortes` seria verdadeira e nem haveria tentativa de particionar o vetor `a`. Entre todos os valores `k (l≤k<r)` que permitem particionar o vetor `a` satisfazendo as condições das linhas 7 e 8 do procedimento `particiona`, deve-se escolher aquele valor mais próximo de `(l+r) div 2`. Pois caso contrário, pode-se gerar árvore degenerada o que inclusive impediria de provar a complexidade desejada. O procedimento `particiona` pode ser executado em tempo esperado linear adaptando o algoritmo para calcular a mediana de um vetor, descrito em [Wirth, 1976, pg. 82], derivado do algoritmo Quick Sort.

Proposição 4.2 (`ConstroiCortes`): A função `ConstroiCortes` gasta tempo esperado $O(dm \log m)$ e a árvore gerada ocupa espaço $O(Lm)$.

Prova: Se a função `ConstroiCortes` é chamada com parâmetros `r` e `l`, considere u=`r-l`+1. Gastam-se os seguintes tempos nas linhas principais da função, não considerando a recursão das linhas 14 e 15:

- O teste da linha 6 gasta tempo $O(ud)$.
- O teste da linha 8 gasta tempo $O(ud)$.
- O cálculo da função minimizadora de penalidade gasta tempo $O(ud)$.
- A escolha do plano de corte na linha 12 gasta tempo $O(ud)$.
- A partição do vetor (linha 13) gasta tempo $O(ud)$ em média.

Portanto, a função toda gasta tempo $O(ud)$ se não considerarmos a recursão. Por outro lado, observe que a função será chamada uma vez para o vetor $\vec{a}$ inteiro (l=1 e r=m e,

portanto, u=m), depois $\vec{a}$ é dividido em duas metades aproximadamente iguais para cada qual a função é chamada recursivamente e cada metade volta a ser dividida e assim por diante. Este processo pára quando a condição da linha 6 ou a da linha 8 é satisfeita. Observe que, quando l=r essas condições são obrigatoriamente satisfeitas, o que termina a recursão. Supondo que o vetor é sempre quebrado em duas metades iguais, o tempo gasto é:

$$m\mathcal{d} + 2 \times \frac{m}{2}\mathcal{d} + 4 \times \frac{m}{4}\mathcal{d} + \cdots + m \times \frac{m}{m}\mathcal{d} =$$

$$\mathcal{d}(\underbrace{m + m + \cdots + m}_{\log_2 m \text{ vezes}}) = \mathcal{d} m \log_2 m$$

Portanto, a complexidade total de tempo é $O(\mathcal{d} m \log m)$.

A árvore de cortes gerada pela função `ConstroiCortes` pode ter no máximo m folhas, e uma árvore binária balanceada (supondo que as divisões são feitas sempre no meio do vetor) com m folhas tem m-1 nós internos. Portanto, o espaço gasto por uma árvore de cortes gerada pela função `ConstroiCortes` é $O(\mathcal{b} m)$, já que em cada folha deve-se armazenar um elemento de $\mathcal{K}^{\mathcal{b}}$. ■

Vamos considerar agora a aplicação de um operador numa imagem. Dada uma função característica ψ: $\mathcal{K}^{\mathcal{d}} \rightarrow \mathcal{K}^{\mathcal{b}}$, seja $\mathcal{B} = \{\lambda_1, \lambda_2, \ldots, \lambda_t\} \subset \mathcal{I}$ tal que $\vee\mathcal{B}_\psi = \psi$. Seja $x \in \mathcal{K}^{\mathcal{d}}$. Para avaliar $\vee\mathcal{B}_\psi(x)$, devemos calcular a seguinte expressão:

$$\lambda_1(x) \vee \lambda_2(x) \vee \ldots \vee \lambda_t(x).$$

Isto gastaria um tempo $O(t\mathcal{d})$ utilizando técnicas convencionais, pois para avaliarmos t int-primitivos gasta-se $O(t\mathcal{d})$ e para calcular t-1 supremos gasta-se $O(t\mathcal{b})$. Observando que $\mathcal{b} \leq \mathcal{d}$, obtemos a complexidade total $O(t\mathcal{d})$.

Com a árvore de corte, conseguiremos um tempo $O(\mathcal{b}+\log t)$, claramente melhor que a complexidade anterior, conforma a demonstração abaixo. Dada uma função característica ψ, representada como uma árvore de cortes `t`, e uma instância $x \in \mathcal{K}^{\mathcal{d}}$, a função `ProcuraCortes` abaixo calcula o valor de $\psi(x)$.

```
1.   function ProcuraCortes(t:NodePt; var x:Kd):Kb;
2.   var p:integer;
3.   begin
4.     If (t^.leaf=true) then
5.       ProcuraCortes:=a^.v
6.     else begin
7.       p:=t^.p;
8.       if (x[p]<=t^.c)
9.         then ProcuraCortes:=ProcuraCortes(t^.left,x)
10.        else ProcuraCortes:=ProcuraCortes(t^.right,x);
11.    end;
12.  end;
```

Proposição 4.3 (`ProcuraCortes`): A função `ProcuraCortes` gasta tempo $O(\log m)$.

Prova: Vamos supor que a árvore seja balanceada. Sabemos que percorrer uma árvore de busca binária balanceada com t folhas leva um tempo $O(\log_2 t)$ e copiar um vetor de dimensão $\mathcal{b}$ (linha 5) gasta tempo $O(\mathcal{b})$, o que resulta na complexidade $O(\mathcal{b} + \log t)$. Considerando que $t \leq m$ e que normalmente $\mathcal{b}$ é um número pequeno fixo (1 para imagens em níveis de cinza, 3 para imagens coloridas) obtemos a complexidade proposta. ■

A função `AplCortes` abaixo aplica a função característica ψ, representada como uma árvore de cortes, em toda seqüência a ser processada `qx`.

```
1.   procedure AplCortes(
2.     t:NodePt;
3.     var qx:array [1..n] of Kd;
4.     var qp:array [1..n] of Kb);
5.   var i:integer;
6.   begin
7.     for i:=1 to n do
8.       qp[i]:=ProcuraCortes(t,qx[i]);
9.   end;
```

Proposição 4.4 (`AplCortes`): A função `AplCortes` gasta tempo $O(n \log m)$.

Prova: Sabemos que a função `ProcuraCortes` gasta tempo $O(\log m)$. Esta função é chamada n vezes, resultando $O(n \log m)$. ■

Observe que algoritmo de cortes, conforme descrito nesta subseção, é mais que um algoritmo consistente. Pois, dada uma instância x que não aparece na amostra de treinamento, o algoritmo de cotes aproxima a saída de x para a saída y' de uma

mento, o algoritmo de cotes aproxima a saída de x para a saída y' de uma instância x' "mais ou menos próxima" de x, o que acelera substancialmente a convergência de aprendizagem. Os algoritmos descritos na próxima seção irão achar o vizinho mais próximo verdadeiro. Infelizmente, não dispomos ainda da análise que caracteriza o quanto "mais ou menos próximo" dista do "o mais próximo". Porém as aplicações do capítulo 5 mostram que esta distância não é muito grande, em média. Como veremos, o algoritmo de cortes é mais rápido que os algoritmos de vizinho mais próximo conhecidos, tornando a sua aplicação interessante em circunstâncias em que um vizinho "mais ou menos próximo" já resolva o problema, não necessitando achar o vizinho mais próximo verdadeiro.

4.3.3 Função Penalidade ou Medida de Erro

Como medir a qualidade de um filtro projetado pela aprendizagem computacional? Se tivermos acesso à saída ideal, é possível medir a função penalidade pixel a pixel e calcular a penalidade média ou o erro médio cometido, que é dado pela expressão:

$$\frac{1}{n}\sum_{i=1}^{n} l(q_i^p - q_i^y)\,.$$

Nos exemplos do capítulo 3 citamos três funções penalidade que poderiam ser utilizadas para avaliar o erro cometido pelos operadores em níveis de cinza e binários:

1. Média do quadrado da diferença entre os pixels das imagens processada e ideal, denominada erro quadrático médio e abreviada na literatura como MSE, do inglês *mean square error*.
2. Média do módulo da diferença entre pixels das imagens processada e ideal, denominada erro absoluto médio e abreviada na literatura como MAE, do inglês *mean absolute error*.
3. Média da função "acerto ou erro" entre os pixels das imagens processada e ideal.

Vimos que a esperança das funções penalidade acima são minimizadas calculando respectivamente, no caso de conflito, média aritmética, mediana e moda (arredondadas para o inteiro mais próximo).

No caso do operador binário, onde cada pixel pode assumir somente os valores 0 ou 1, os resultados das três funções penalidade acima são exatamente iguais fornecendo um número entre 0 e 1. Para que a medida de erro se torne mais intuitiva, convertemos o erro para porcentagem. Por exemplo, para operadores binários, ter erro de 6% significa que 6% dos pixels da imagem processada são diferentes daqueles da imagem ideal.

No caso do operador em níveis de cinza, utilizamos a média aritmética para resolver os conflitos, o que minimiza MSE. Porém, MSE não possui uma interpretação intuitiva, pois a unidade da medida seria o nível de cinza elevado ao quadrado. Assim, resolvemos utilizar MAE como medida do erro. Esta decisão faz sentido pois na prática o operador ótimo é único para uma variedade grande de funções penalidade consideradas (veja [Sage and Melsa, 1971, pg. 180]). Como representamos uma imagem em níveis de cinza como uma matriz de bytes, o erro absoluto poderia ir de 0 a 255. Dividimos este erro por 255 para apresentá-lo como uma porcentagem.

Nos capítulo 3 citamos três funções penalidade que poderiam ser utilizadas para avaliar o erro cometido pelos operadores coloridos:

1. Média do quadrado da 2-norma (distância euclidiana) da diferença entre os pixels das imagens processada e ideal, que denominamos erro euclidiano quadrático médio.
2. Média da 1-norma (soma dos lados) entre pixels das imagens processada e ideal.
3. Média da função “acerto ou erro” entre os pixels das imagens processada e ideal.

Vimos que a esperança das funções penalidades acima são minimizadas calculando respectivamente, no caso de conflito, média aritmética vetorial, mediana vetorial e moda. No projeto dos operadores coloridos, utilizamos a média aritmética vetorial para resolver os conflitos, o que minimiza erro euclidiano quadrático médio. Porém, a unidade

desta medida (distância entre duas cores ao quadrado) não possui uma interpretação intuitiva. Assim, resolvemos apresentar o erro pelo erro euclidiano (não elevado ao quadrado) médio (que abreviaremos como MEE, do inglês mean euclidean error). Tomamos o erro absoluto médio pixel a pixel entre a imagem processada e a ideal, com a distância entre pixels avaliado por métrica euclidiana no espaço tridimensional das cores. Como representamos uma imagem colorida como uma matriz onde cada elemento é um vetor de 3 bytes, o erro absoluto poderia ir de 0 a $\sqrt{3\times 255^2}$ =441,67. Dividimos este erro por 441,67 para apresentá-lo como uma porcentagem.

4.3.4 Escolha da Janela Adequada

Um ponto a ser considerado na aprendizagem de W-operador é a escolha da janela $\vec{W}$ apropriada. É interessante que a janela seja tão pequena quanto possível, uma vez que, do ponto de vista prático, janelas excessivamente grandes não aumentam a qualidade do operador e aumentam o tempo de processamento. Por outro lado, uma janela pequena demais pode perder algumas informações a respeito da distribuição alvo, disponíveis somente em janelas maiores. Assim, existe uma janela de tamanho e forma ideal para cada aplicação.

Do ponto de vista teórico, quanto maior a janela, maior a cardinalidade do conjunto $\mathcal{X}^{\mathcal{A}}$ o que aumenta a complexidade da amostra e conseqüentemente aumenta o tempo de processamento. Porém, constata-se, na prática, que uma janela um pouco maior do que a ideal continua gerando resultados bastante satisfatórios com uma degradação pequena no tempo de processamento (א, Figura 5.1 e Figura 5.2). Por outra parte, uma janela menor do que a ideal costuma produzir um filtro de qualidade bem inferior ao ótimo. Assim, é melhor errar escolhendo a janela um pouco maior do que a ideal do que escolher janela excessivamente pequena.

Na א.e, utilizando janela 2×2, obtivemos 6,09% dos pixels diferentes em relação à saída ideal, enquanto que utilizando janela 3×3, 3×4 ou 7×7 (figuras 4.7.f-h) obtivemos zero

pixels diferentes. O tempo de processamento aumentou de 2 segundos com janela 2×2 para 3 segundos nos casos 3×3 e 3×4 e 9 segundos usando janela 7×7. Assim sendo, a melhor escolha da janela seria 3×3. O operador aprendido na אי foi aplicado numa imagem diferente na בי. Utilizando o operador 2×2, obtivemos 2,44% dos pixels diferentes em relação à saída ideal, enquanto que utilizando a janela 3×3 obtivemos 0% dos pixels diferentes, o que confirma que a janela 3×3 é a ideal. O erro zero obtido indica que a aprendizagem foi efetiva e que certos filtros binários podem ser emulados facilmente. Nas Figuras 5.1 e 5.2, temos exemplos em que a precisão do operador atinge um máximo, sendo que para janelas maiores ou menores a precisão diminui.

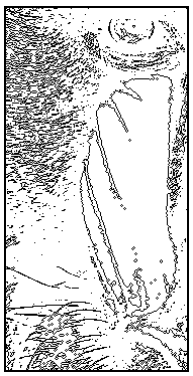
A escolha da janela tem sido sempre uma tarefa empírica na morfologia matemática. Utilizando os algoritmos eficientes apresentados nesta tese é possível testar diversas janelas e escolher empiricamente a que dá melhores resultados.

4.4 Conclusão

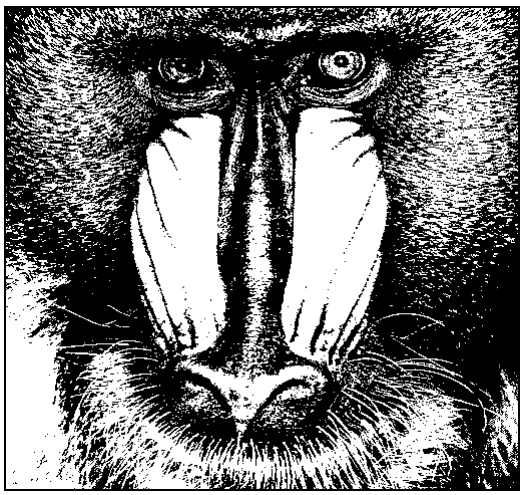
Neste capítulo, definimos o problema de aprendizagem de W-operador como um caso particular do problema de aprendizagem generalizado. Conseqüentemente, pudemos utilizar toda a teoria de aprendizagem que desenvolvemos no capítulo 3. Expusemos os motivos que tornam a complexidade de amostra prática de um problema de aprendizagem de operador bem menor do que a complexidade teórica. Defendemos a metodologia utilizada para analisar a complexidade dos algoritmos. Listamos dois algoritmos de aprendizagem consistentes: algoritmo força bruta e algoritmo de cortes. Analisamos os dois algoritmos e demonstramos que o algoritmo de cortes tem complexidade de tempo menor que o primeiro e argumentamos por que a convergência do algoritmo de cortes é mais rápida do que a do algoritmo força bruta. Definimos um critério para julgar a qualidade do operador aprendido. Fizemos algumas considerações a respeito da escolha adequada da janela.

Figura 4.7: Existe um filtro digital supostamente desconhecido ("trace contour" do Adobe Photoshop 3.0) que, processando a imagem 4.7.a, fornece a imagem 4.7.b. Pode-se emular o comportamento deste filtro desconhecido utilizando a aprendizagem computacional. Utilizando as imagens 4.7.a e 4.7.b como exemplos de treinamento, o algoritmo de cortes constrói o operador morfológico que emula o filtro desconhecido. Este operador, quando aplicado à imagem 4.7.c, fornece as imagens 4.7.e-h, com janelas respectivamente 2×2, 3×3, 3×4 e 7×7. Entre a imagem processada 4.7.e e a imagem ideal 4.7.d houve 6,09% dos pixels diferentes. Enquanto isso, a porcentagem de pixels diferentes entre as imagens processadas 4.7.f-h e a imagem ideal 4.7.d foi zero, ou seja, não há nenhum pixel diferente. Isto mostra que certos filtros binários podem ser emulados facilmente. Também podemos deduzir que a janela 2×2 é menor do que a ideal.

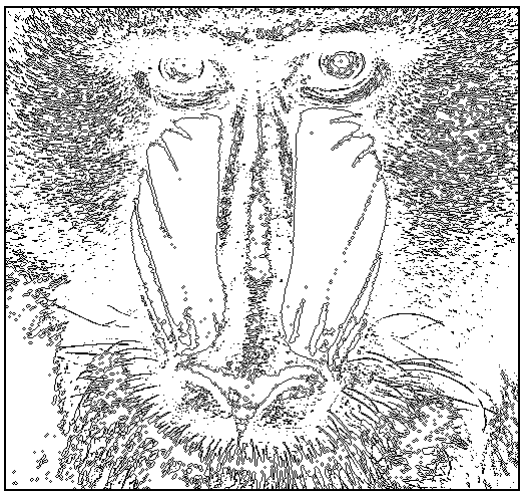
(4.7.a) Exemplo de entrada $A^{\mathbf{x}}$.
361 × 181 pixels, 1 bit por pixel.



(4.7.b) Exemplo de saída $A^{\mathbf{y}}$.
361 × 181 pixels, 1 bit por pixel.

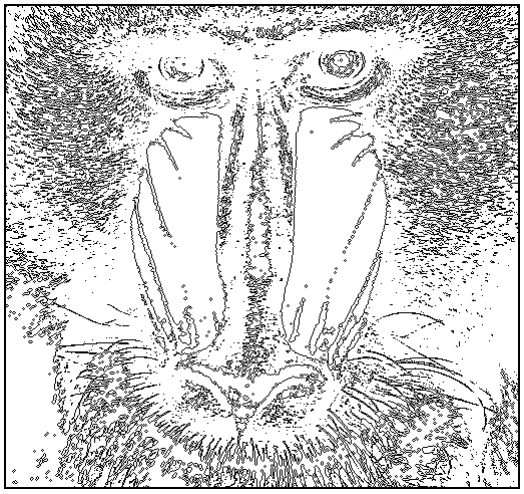


(4.7.c) Imagem a ser processada $Q^{\mathbf{x}}$.
480 × 512 pixels, 1 bit por pixel.



(4.7.d) Imagem ideal $Q^{\mathbf{y}}$

(4.7.e) Imagem processada $Q^{\mathbf{p}}$ utilizando janela 2×2.
Tempo de treinamento: 1 s.
Tempo de aplicação: 1 s.
Tamanho do operador: 156 bytes.
Número de int-primitivos: 15.
6,09% dos pixels diferentes.



(4.7.f) Imagem processada $Q^{\mathbf{p}}$ utilizando janela 3×3.
Tempo de treinamento: 1 s.
Tempo de aplicação: 2 s.
Tamanho do operador: 2.972 bytes.
Número de int-primitivos: 362.
0% dos pixels diferentes.

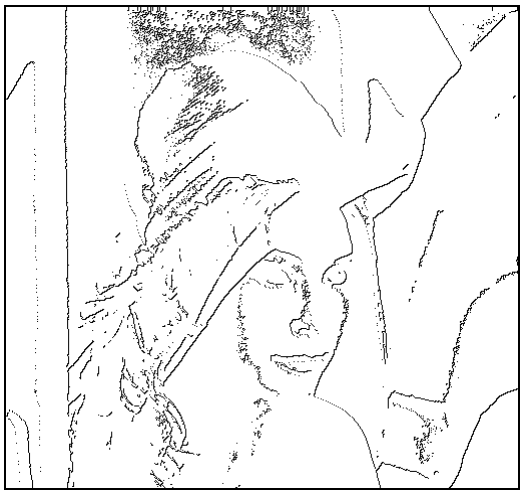
(4.7.g) Imagem processada $Q^{\mathbf{p}}$ utilizando janela 3×4.
Tempo de treinamento: 1 s.
Tempo de aplicação: 2 s.
Tamanho do operador: 15.444 bytes.
Número de int-primitivos: 1.918.
0% dos pixels diferentes.

(4.7.h) Imagem processada $Q^{\mathbf{p}}$ utilizando janela 7×7.
Tempo de treinamento: 4 s.
Tempo de aplicação: 5 s.
Tamanho do operador: 220.284 bytes.
Número de int-primitivos: 27.486.
0% dos pixels diferentes.

Figura 4.8: Os operadores construídos na aplicação da א foram aplicados numa imagem diferente (4.8.a). Aplicando o filtro "trace contour" original do Adobe Photoshop, obtemos a imagem ideal 4.8.b. Aplicando o operador morfológico 2×2 obtido na א, obtemos a imagem 4.8.c que possui 2,44% dos pixels diferentes em relação à saída ideal. Utilizando o operador com janela 3×3, obtemos erro zero na imagem 4.8.d.

(4.8.a) Imagem a ser processada Q^{x}.
480 × 512 pixels, 1 bit por pixel.

(4.8.b) Saída ideal Q^{y}



(4.8.c) Imagem processada Q^{p} utilizando janela 2×2.
Tempo de aplicação: 1 s.
2,44% dos pixels diferentes.

(4.8.d) Imagem processada Q^{p} utilizando janela 3×3.
Tempo de aplicação: 2 s.
0% dos pixels diferentes.

5 Vizinho Mais Próximo

5.1 Introdução

No capítulo 4 apresentamos o algoritmo de cortes, utilizado para construir automaticamente operadores morfológicos pelo treinamento. Observando este algoritmo, surge naturalmente a seguinte pergunta: "Por que não utilizar alguma outra função aproximadora, mesmo que com isso nos afastemos das técnicas da morfologia matemática?"

Em termos de velocidade, a árvore de cortes tem um desempenho melhor que outras técnicas.

É um fato bem conhecido que o treinamento de uma rede neural é bastante lento. Conseqüentemente, a construção do operador utilizando esta técnica é um tanto "incômoda" para o usuário. Assim, esta abordagem parece-nos pouco promissora.

O artigo [Harvey and Marshall, 1996] utilizou algoritmo genético para o projeto automático de operadores e resultou num algoritmo lento, que tem o problema de explosão combinatória. Além disso, a técnica descrita em [Harvey and Marshall, 1996] converge para um operador ótimo dentro de uma classe restrita de operadores e não se conhece o quanto esse operador ótimo restrito é próximo do operador ótimo global.

O vizinho mais próximo parece-nos ser a alternativa mais promissora em termos de velocidade, pois se conhece um algoritmo, a kd-árvore, que resolve a busca do vizinho mais próximo em tempo razoável para dimensões não muito grandes. Mesmo assim, a

árvore de cortes torna-se milhares de vezes mais rápida que a kd-árvore quando a dimensão cresce. Experimentalmente, a árvore de cortes foi mais de 2000 vezes mais rápida que a kd-árvore na dimensão 6 (ו e ז).

Em termos de memória, tanto a árvore de cortes como a kd-árvore utilizam espaço linear na quantidade de dados de treinamento. Isto caracteriza um bom desempenho, que acreditamos ser impossível de melhorar sem perder a característica PAC.

Para as aplicações testadas, o vizinho mais próximo gerou resultados com melhor qualidade que a árvore de cortes. Ao contrário de uma rede neural ou algoritmo genético cujo comportamento não é intuitivo, o vizinho mais próximo apresenta um comportamento previsível. Acreditamos que o vizinho mais próximo e o κ-vizinhos mais próximos (uma pequena variação da estratégia anterior) sejam as melhores estratégias em termos de qualidade para os problemas que aparecem na prática, embora não tenhamos como demonstrar esta afirmação.

O modelo de aprendizagem vizinho mais próximo é tradicionalmente abreviado como NN (do inglês nearest neighbor) e um dos primeiros artigos que o descreve é [Cover and Hart, 1967]. Para aplicar este modelo ao nosso problema, devemos ter um critério para medir o quanto um pedaço da imagem (isto é, a imagem restrita à janela) é "semelhante" a um outro pedaço, ou seja, necessitamos de uma métrica no espaço das instâncias X. Esta métrica não foi definida nem na morfologia nem na aprendizagem computacional. Pois a morfologia trata fundamentalmente dos conjuntos com uma relação de ordem parcial, sem que esteja definida uma métrica. Por outro lado, o modelo de Haussler trabalha com conjuntos arbitrários e uma função penalidade definida no espaço $A \times Y$. Como vimos, no nosso problema A e Y são dois conjuntos iguais e a função penalidade é normalmente uma métrica nesses espaços. Assim fica definida uma métrica no espaço das saídas Y, mas não em X. Portanto, é necessário acrescentar no modelo de aprendizagem uma métrica no espaço das instâncias. Relembrando a noção de métrica, temos:

Definição (métrica): Uma função ρ: $X \times X \rightarrow \mathbb{R}$ é uma métrica sse para todo a, b, $c \in X$:

1. $\rho(a, b) \geq 0$
2. $\rho(a, b) = \rho(b, a)$
3. $\rho(a, b) = 0 \Leftrightarrow a = b$
4. $\rho(a, c) \leq \rho(a, b) + \rho(b, c)$ (axioma triangular)

O número real $\rho(a, b)$ é chamado distância entre a e b e o conjunto X é denominado espaço métrico.

A kd-árvore, a estrutura de dados que acelera a busca do vizinho mais próximo, está descrita nos artigos [Bentley, 1975] e [Friedman et al., 1977], e no livro [Preparata and Shamos, 1985]. Implementamos a kd-árvore para compará-la com a árvore de cortes e descrevemos neste capítulo os resultados obtidos.

5.2 Aprendizagem NN

5.2.1 Introdução

No algoritmo de cortes (capítulo 4), sem a definição formal de uma métrica, aproximamos a saída de uma instância desconhecida pela saída de uma instância "mais ou menos próxima". Vimos que sem esta aproximação, a convergência do processo de aprendizagem torna-se muitíssimo mais lenta (basta observar no capítulo 4 os resultados gerados pelo algoritmo força bruta). Ora, quando uma instância x é desconhecida, nada parece ser mais razoável do que aproximar $\mathcal{A}^+(\vec{z})(x)$ para $\mathcal{A}^+(\vec{z})(x')$, onde x' é o "vizinho mais próximo" de x. O modelo NN utiliza esta idéia.

O modelo NN tem sido amplamente estudado e aplicado em diversas áreas. A formalização deste modelo e os resultados teóricos de convergência estatística podem ser encontrado em [Cover and Hart, 1967]. Outros artigos teóricos que tratam do modelo NN

são [Short and Fukunaga, 1981], [Fukunaga and Flick, 1985], [Loizou and Maybank, 1987], entre outros.

O modelo NN pode ser encarado simplesmente como o modelo de Haussler que utiliza um algoritmo de aprendizagem particular, o algoritmo do vizinho mais próximo, e a exposição desta seção seguirá esta abordagem.

Consideraremos que está definida uma métrica ρ no espaço das instâncias. O algoritmo de aprendizagem NN, conforme definido em [Cover and Hart, 1967], decide que, dadas uma amostra de treinamento $\vec{z} = ((x_1, y_1), (x_2, y_2), ..., (x_m, y_m))$ e um exemplo (x, y) do qual só é mostrado ao aprendiz a instância x, o aprendiz escolhe y' como a saída de x, onde $(x', y') \in \vec{z}$ e x' é a instância de $\vec{z}$ mais próxima de x.

Em caso de empate (isto é, se existirem dois ou mais vizinhos mais próximos todos com a mesma distância de x), o artigo [Cover and Hart, 1967] sugere escolher a saída mais freqüente, ou seja, a moda. Porém, afirma que esta suposição é desnecessária para os teoremas que obtiveram. De fato, nas demonstrações subseqüentes do artigo, os autores assumem implicitamente que, no caso de empate, qualquer um dos vizinhos mais próximos é escolhido de modo equiprovável. Denominamos este algoritmo de aprendizagem de *NN não-consistente*, para diferenciá-lo daquele *consistente* que definiremos mais adiante.

5.2.2 NN Não-Consistente

Como afirmamos acima, o modelo NN é um caso particular do modelo de Haussler. Para obtermos o modelo NN, conforme definido em [Cover and Hart, 1967], além de supor que o espaço das instâncias é métrico, é necessário fazer mais algumas restrições ao modelo PAC generalizado. Em primeiro lugar, o modelo NN supõe que o espaço das saídas Y e o espaço das ações A são iguais. Além disso, [Cover and Hart, 1967] utiliza a seguinte função como penalidade:

$$l(y_1, y_2) = \begin{cases} 1 & \text{se } y_1 \neq y_2 \\ 0 & \text{cc} \end{cases}$$

Vamos definir o vizinho mais próximo:

Definição (vizinhos mais próximos): Dados $x \in X$ e uma seqüência $\vec{z} = ((x_1, y_1), (x_2, y_2), ..., (x_m, y_m)) \in Z^m$, chamaremos de *seqüência dos vizinhos mais próximos* de x em $\vec{z}$ (todos com uma mesma distância de x) e denotaremos por $V_{\vec{z}}(x)$ a subseqüência $((x'_1, y'_1), (x'_2, y'_2), ..., (x'_n, y'_n))$ de $\vec{z}$ onde $\rho(x'_i, x) \leq \rho(x_j, x)$, $\forall i \in [1\ldots n]$ e $\forall j \in [1\ldots m]$. Além disso, denotaremos a seqüência $(x'_1, x'_2, ..., x'_n)$ por $V^x_{\vec{z}}(x)$ e a seqüência $(y'_1, y'_2, ..., y'_n)$ por $V^y_{\vec{z}}(x)$.

Note que a seqüência coincidente $C_{\vec{z}}(x)$ é uma subseqüência de $V_{\vec{z}}(x)$, pois toda vez que $C_{\vec{z}}(x)$ é uma seqüência não-vazia, $C_{\vec{z}}(x) = V_{\vec{z}}(x)$. Com isso, podemos definir o algoritmo NN não-consistente.

Definição (algoritmo NN não-consistente): Seja $(X, Y, Y, \mathcal{H}, \mathcal{P}, \{ L_\varepsilon \})$ um problema de aprendizagem generalizado onde X possui uma métrica ρ. O *algoritmo do vizinho mais próximo não-consistente* NN: $\bigcup_{m \geq 1} Z^m \rightarrow \mathcal{H}$ é definido $\mathrm{NN}(\vec{z})(x) = y$, onde $y \in V^y_{\vec{z}}(x)$. Se $V^y_{\vec{z}}(x)$ tiver comprimento maior que um, um dos seus elementos é escolhido de forma equiprovável como y.

O risco, ou seja, a esperança de penalidade, é definida da mesma forma que no modelo de Haussler. O risco ótimo será denotado por $R^* = r^*(P) = \wedge \{r_h(P) : h \in H\}$. A regra de decisão que gera o risco otimal será denotada como h^*. Vamos denotar o risco esperado ao utilizar o algoritmo de aprendizagem NN com uma amostra de tamanho m por $R(m)$:

$$R(m) = \sum_{\vec{z} \in Z^m} r_{NN(\vec{z})}(P) P(\vec{z})$$

Além disso, denotaremos por R o limite de $R(m)$, isto é, $R = \lim_{m\to\infty}\big(R(m)\big)$.

O principal teorema do artigo [Cover and Hart, 1967] pode ser resumido na seguinte inequação:

$$R^* \le R \le R^*\left(2 - \frac{|Y|R^*}{|Y|-1}\right)$$

Não demonstraremos este teorema, pois estamos interessados num resultado mais forte. Os leitores interessados na demonstração podem consultar o artigo original. Considerando que a conseqüência direta da inequação acima é $R^* \le R \le 2R^*$, podemos dizer que o algoritmo NN irá errar no máximo duas vezes mais do que a regra de decisão ótima quando o tamanho da amostra tende ao infinito. Além disso, [Cover and Hart, 1967] mostra que este limite é o mais estreito possível, pois mostra que existem casos em que $R^* = R$ e assim como casos em que

$$R = R^*\left(2 - \frac{|Y|R^*}{|Y|-1}\right).$$

Este resultado não é tão interessante como pode parecer à primeira vista. Pois ele está afirmando que o algoritmo NN não é uma solução PAC! Se NN fosse PAC, o risco deveria tender para o risco ótimo quando o número de amostras tender ao infinito, isto é, deveríamos ter a seguinte igualdade:

$$R = \lim_{m\to\infty}\big(R(m)\big) = R^*.$$

Onde está o problema? O problema é que o artigo [Cover and Hart, 1967] está trabalhando com algoritmo NN que não é consistente.

5.2.3 NN Consistente

Na última subseção, vimos que um algoritmo NN não-consistente não é uma solução PAC de um problema de aprendizagem. Como devemos alterar o algoritmo NN para que ele se torne consistente? Basta colocar um bom critério de desempate para quando

houver mais de um vizinho mais próximo. Esse "bom critério" é nada mais do que a função minimizadora de penalidade que definimos no capítulo 3.

Definição (NN$^+$): Seja (X, Y, Y, $\mathcal{H}$, $\mathcal{P}$, $\{ L_\varepsilon \}$) um problema de aprendizagem onde X possui uma métrica ρ. O *algoritmo do vizinho mais próximo consistente* NN$^+$: $\bigcup_{m \geq l} Z^m \to \mathcal{H}$ define-se pontualmente $\mathrm{NN}^+(\vec{z})(x) = \mathcal{M}(V_{\vec{z}}^{y}(x))$.

Utilizando a função penalidade escolhida para o modelo NN,

$$l(y_1, y_2) = \begin{cases} 1 & \text{se } y_1 \neq y_2 \\ 0 & \text{cc} \end{cases},$$

a função minimizadora de penalidade é a moda, isto é, o valor mais freqüente (capítulo 3).

Evidentemente, todo algoritmo do vizinho mais próximo consistente (NN$^+$) é um algoritmo consistente. Assim, o algoritmo NN$^+$ é uma solução PAC do problema de aprendizagem generalizado, como todos outros algoritmos consistentes.

5.2.4 Aprendizagem Consistente e NN Consistente

Qual a vantagem do algoritmo vizinho mais próximo consistente sobre algum outro algoritmo consistente genérico, uma vez que ambos são soluções PAC? Para uma distribuição alvo caótica, não é possível afirmar que um seja melhor que outro. A distribuição tem que ser "bem comportada", isto é, a esperança de erro tem que diminuir à medida em que instâncias cada vez mais próximas são escolhidas para estimar a saída de uma instância. Denominamos este "bom comportamento" de "suavidade", que pode ser descrita matematicamente como segue:

Definição (distribuição suave): Diremos que uma distribuição P é *suave* sse, quando três exemplos $z = (x, y)$, $z' = (x', y')$ e $z'' = (x'', y'')$ são gerados pela P,

$$\rho(x, x') \leq \rho(x, x'') \Rightarrow E(l(y, y')) \leq E(l(y, y'')).$$

Certamente, na maioria dos casos práticos de processamento de imagem, a distribuição alvo é suave. Falando informalmente, é muito natural que dois padrões de cores semelhantes na entrada devam ser levados a saídas próximas. É razoável supor que os operadores que não têm esse comportamento não sejam úteis na prática, pois neste caso dois pedaços de imagem dificilmente distinguíveis a um ser humano estariam sendo levados a duas cores de saída completamente diferentes. Deste modo, aproximar a saída *y* desconhecida (de um padrão de cores *x* dado) para a saída y' (de um padrão x' semelhante a *x*) deve ser melhor na prática que aproximar *y* para a saída y'' (de um padrão x'' muito diferente de *x*).

Também podemos dizer que se a distribuição alvo for suave então a convergência do algoritmo NN^+ será mais rápida (em relação à quantidade de amostras) que a de algum outro algoritmo consistente (que escolhe como saída algum elemento de $\vec{z}^{\,y}$ quando $C_{\vec{z}}(x)$ é nula). Isto é demonstrado no próximo teorema.

Teorema 5.1 (NN^+ é melhor que consistente): Seja (X, Y, Y, $\mathcal{H}$, $\mathcal{P}$, $\{ L_\varepsilon \}$) um problema de aprendizagem generalizado onde X possui uma métrica ρ e seja $P \in \mathcal{P}$ uma distribuição alvo suave. Sejam NN^+ e $\mathcal{A}^+$ respectivamente o algoritmo do vizinho mais próximo consistente e um algoritmo consistente cuja saída $\mathcal{A}^+(\vec{z})(x)$ é escolhida dentre a seqüência $\vec{z}^{\,y}$ quando $C_{\vec{z}}(x)$ é uma seqüência nula. Então, para toda amostra $\vec{z}$ de comprimento $m \geq 1$,

$$r_{\mathrm{NN}^+(\vec{z})}(P) \leq r_{\mathcal{A}^+(\vec{z})}(P)$$

Prova: Aplicando a definição de risco, a desigualdade que queremos demonstrar pode ser reescrita como:

$$E\Big(l(y, \mathrm{NN}^+(\vec{z})(x))\Big) \leq E\Big(l(y, \mathcal{A}^+(\vec{z})(x))\Big)$$

Ora, a desigualdade acima é verdadeira, pois:

a) Se a seqüência $C_{\vec{z}}(x)$ é não-nula então $\mathrm{NN}^+(\vec{z})(x) = \mathcal{A}^+(\vec{z})(x) = \mathcal{M}(C^y_{\vec{z}}(x))$ e as os dois algoritmos sofrem penalidades iguais.

b) Se $C_{\vec{z}}(x)$ é nula então $\mathrm{NN}^{+}(\vec{z})(x) = y' = \mathcal{M}(V_{\vec{z}}^{y}(x))$. Observe que, como no modelo de aprendizagem NN a função minimizadora de penalidade é sempre a moda, y' é um dos elementos da seqüência $V_{\vec{z}}^{y}(x)$. Enquanto isso, $\mathcal{A}^{+}(\vec{z})(x) = y''$, onde (x'', y'') é um exemplo da seqüência $\vec{z}$. Como a distribuição alvo é suave e $\rho(x,x') \leq \rho(x,x'')$, temos $E(\, l(y,y')\,) \leq E(\, l(y,y'')\,)$. ■

5.2.5 Aprendizagem κ-NN

Nas subseções anteriores, o modelo vizinho mais próximo foi descrito utilizando um único vizinho mais próximo (exceto quando existirem dois ou mais pontos de distância mínima). Por que não se escolhem dois ou mais vizinhos próximos e se tira uma média (ou mediana, ou moda) entre os seus valores de saída? Esta estratégia é abreviada na literatura como κ-NN, onde κ é a quantidade de vizinhos mais próximos que se deve escolher.

O artigo [Cover and Hart, 1967] demonstra que, assumindo certas hipóteses, a estratégia 1-NN é estritamente melhor que qualquer estratégia κ-NN, κ≠1. Para esta demonstração, é necessário assumir que, dadas instâncias x, x' e x'', com saídas respectivamente y, y' e y'', se $y = y'$ e $y \neq y''$ então $\rho(x,x') < \rho(x,x'')$. Isto é, uma distância intra-classe é sempre menor que qualquer distância inter-classe. Ora, no nosso caso, esta suposição não é verdadeira e conseqüentemente desaparece o argumento que desautoriza o uso da estratégia κ-NN, κ≠1.

Verifica-se na prática que o valor de κ que minimiza o erro depende da aplicação (veja a figura 6.1).

5.3 Kd-Árvore

Para utilizar o modelo NN na prática, necessita-se de um algoritmo eficiente para buscar os vizinhos próximos. O problema de busca dos κ vizinhos mais próximos pode ser enunciado como segue: "Dados *m* pontos de treinamento, cada um composto por d números reais, queremos fazer um pré-processamento de forma que, dados outros *n* novos pontos de dimensão d seja possível localizar rapidamente, para cada um deles, os κ pontos mais próximos dentre os pontos de treinamento." Evidentemente, quando κ=1, obtemos a busca do vizinho mais próximo.

A busca do vizinho mais próximo possui uma ampla gama de aplicações que vão desde busca de resposta às "queries" num banco de dados até reconhecimento de caracteres manuscritos (veja, por exemplo, [Smith et al., 1994]) e reconhecimento de voz. Conseqüentemente, é importante ter meios de resolver este problema eficientemente.

Existe uma solução trivial com complexidade $O(\kappa dm)$ para o problema. Basta calcular as distâncias entre o ponto dado e os *m* pontos de treinamento e escolher κ pontos com as menores distâncias. Evidentemente, aplicar este algoritmo para *n* pontos a serem processados levaria tempo $O(\kappa dmn)$, o que é demasiado lento para *m* e *n* grandes. Assim, diversas técnicas alternativas foram desenvolvidas para acelerar a busca: [Bentley, 1975], [Friedman et al., 1977], [Sproull, 1991], [Djouadi and Bouktache, 1997], entre outros. Também é possível utilizar o diagrama de Voronoi ([Preparata and Shamos, 1985]), porém a construção do diagrama de Voronoi para dimensões altas é inviável na prática. O algoritmo amplamente utilizado para resolver este problema é a kd-árvore.

O artigo [Bentley, 1975] introduziu a kd-árvore (abreviação de árvore k-dimensional). Depois, o artigo [Friedman et al., 1977] otimizou-a. A kd-árvore é uma estrutura de dados que permite achar rapidamente os κ vizinhos mais próximos verdadeiros. A sua descrição algorítmica encontra-se no anexo E, pois julgamos que a compreensão do seu funcionamento não é essencial para o entendimento do resto do trabalho.

O algoritmo otimizado apresentado em [Friedman et al., 1977] tem, de acordo com a análise apresentada no artigo, complexidade de tempo de treinamento $O(d m \log m)$ e a de aplicação $O(n \log m)$, isto é, exatamente igual à árvore de cortes. Aplicações práticas mostram que, de fato, o tempo de treinamento da árvore de cortes e da kd-árvore são praticamente idênticos. Porém, fazer uma busca em kd-árvore é centenas ou milhares de vezes mais lento do que utilizando árvore de corte (veja ı e ĭ) e esta diferença cresce à medida em que a dimensão aumenta. Isto nos leva a concluir que a análise de complexidade de [Friedman et al., 1977] deve estar incorreta. De fato, o artigo [Sproull, 1991] diz: "É errado assumir que a kd-árvore garanta a busca do vizinho mais próximo em tempo esperado logarítmico. (...) Para d=16, uma busca na kd-árvore com 76000 pontos examina praticamente todos os pontos."

O livro [Preparata and Shamos, 1985] atribui uma outra complexidade para a busca em kd-árvore, que nos parece ser a análise correta. O tempo de busca de um ponto seria $O(d m^{1-1/d} + \kappa)$, o que dá o tempo de aplicação em n pontos $O(n(d m^{1-1/d} + \kappa))$ ou, desprezando κ, $O(n d m^{1-1/d})$. Ora, isto indica que para dimensões d grandes, a kd-árvore é um algoritmo $O(n d m)$, o que é muito pior que a complexidade $O(n \log m)$ do algoritmo de cortes. Isto concorda com os resultados experimentais que obtivemos.

A aplicação da ı o tempo de processamento da kd-árvore pulou de 9 segundos usando janela 1×2 para 2028 segundos com janela 2×3. Enquanto isso, a árvore de cortes gastou 2 segundos utilizando qualquer uma das duas janelas. Assim sendo, acreditamos que em muitas aplicações, vale a pena utilizar a árvore de cortes pela sua velocidade, apesar de gerar um resultado um pouco mais pobre (veja ı e ĭ).

5.4 Conclusão

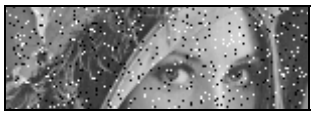
Neste capítulo, utilizamos a aprendizagem NN para obter um operador com qualidade superior àquele obtido pelo algoritmo de cortes. Para isso, passamos a trabalhar com

espaços métricos. Apresentamos o modelo teórico de aprendizagem NN e distinguimos a aprendizagem NN consistente da não-consistente. Argumentamos que, para que o algoritmo NN consistente seja preferível a um algoritmo consistente genérico, é suficiente que a distribuição alvo possua uma característica que denominamos de suavidade. Descrevemos a kd-árvore, uma estrutura de dados que permite uma busca eficiente do vizinho mais próximo. Argumentamos que a árvore de cortes é computacionalmente mais rápida que a kd-árvore. Por outro lado, na prática, a kd-árvore gera um operador mais acurado que a árvore de cortes. Por fim, mostramos alguns exemplos de aplicação juntamente com os dados de desempenho.

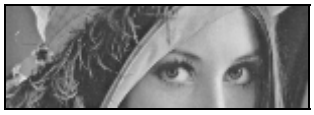
Figura 5.1: A imagem 5.1.c está corrompida pelo ruído conhecido como "sal e pimenta". Este ruído é utilizado em muitos exemplos de filtragem não-linear em níveis de cinza como, por exemplo, [Dougherty and Astola, 1994, pg. 115]. Com probabilidade 1/15, um pixel torna-se preto e com a mesma probabilidade um pixel torna-se branco.

Usando as imagens 5.1.a e 5.1.b como amostras de treinamento, o algoritmo de aprendizagem aprende o comportamento do ruído e constrói um operador morfológico que o elimina, utilizando árvore de cortes ou kd-árvore. Este operador, quando aplicado a uma imagem ruidosa 5.1.c, gera as imagens filtradas (5.1d-i). Estas imagens são semelhantes à saída ideal desconhecida 5.1.j.

Observe como a boa escolha da janela (1×2) diminui o erro absoluto médio. Também observe como o tempo de processamento da kd-árvore aumenta com o aumento do tamanho da janela, enquanto que o tempo de processamento da árvore de cortes permanece praticamente inalterado. Podemos reparar que, de uma forma geral, a saída da kd-árvore é melhor do que a saída da árvore de cortes. Como exceção, nas saídas 5.1.f e 5.1.g ocorre uma inversão na qualidade (0,56% contra 0,58%).



(5.1.a) Amostra de entrada $A^{\mathbf{x}}$.
51 × 151 pixels, 1 byte por pixel.



(5.1.b) Amostra de saída $A^{\mathbf{y}}$.
51 × 151 pixels, 1 byte por pixel.

(5.1.c) Imagem a ser processada $Q^{\mathbf{x}}$.
256 × 256 pixels, 1 byte por pixel.

(5.1.d) Imagem processada $Q^{\mathbf{p}}$ pelo algoritmo de cortes usando janela 1×1.
MAE: 0,95%.
Tempo de treinamento: 1 s.
Tempo de aplicação: 1 s.

(5.1.e) Imagem processada $Q^{\mathbf{p}}$ usando kd-árvore e janela 1×1.
MAE: 0,90%.
Tempo de treinamento: 1 s.
Tempo de aplicação: 12 s.

(5.1.f) Imagem processada $Q^{\mathbf{p}}$ pelo algoritmo de cortes usando janela 1×2.
MAE: 0,56%.
Tempo de treinamento: 1 s.
Tempo de aplicação: 1 s.

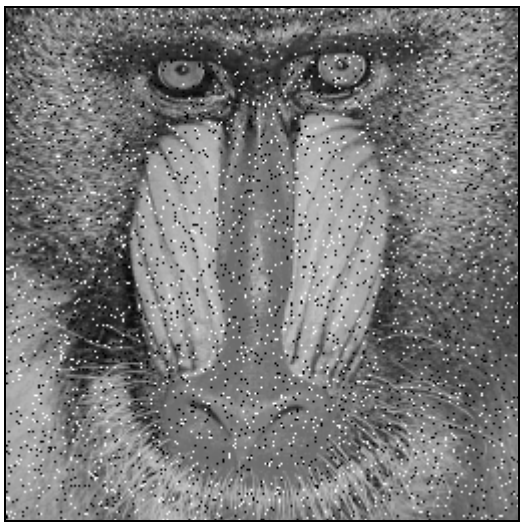
(5.1.g) Imagem processada $Q^{\mathbf{p}}$ usando kd-árvore e janela 1×2.
MAE: 0,58%.
Tempo de treinamento: 1 s.
Tempo de aplicação: 8 s.

(5.1.h) Imagem processada $Q^{\mathbf{p}}$ pelo algoritmo de cortes usando janela 2×3.
MAE: 1,81%.
Tempo de treinamento: 1 s.
Tempo de aplicação: 1 s.

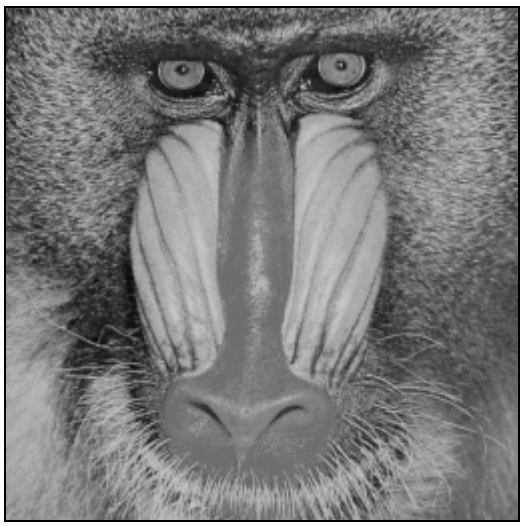
(5.1.i) Imagem processada $Q^{\mathbf{p}}$ usando kd-árvore e janela 2×3.
MAE: 1,45%.
Tempo de treinamento: 1 s.
Tempo de aplicação: 2.027 s.

(5.1.j) Saída ideal desconhecida $Q^{\mathbf{y}}$.

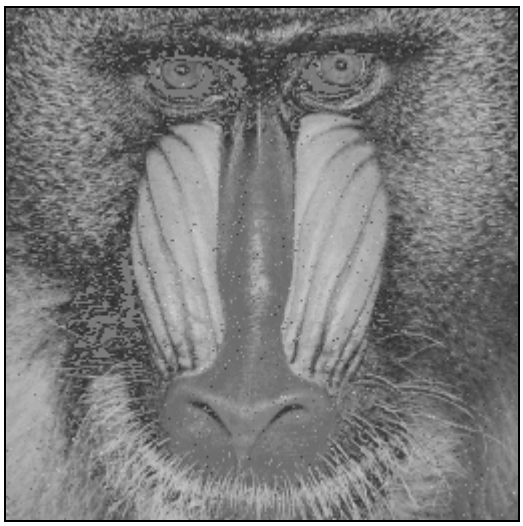
Figura 5.2: Neste exemplo, os filtros construídos na ו são utilizados para eliminar ruído de uma imagem diferente mas corrompida pelo mesmo ruído "sal e pimenta". Esses filtros, aplicados à imagem ruidosa 5.2.a gerou as imagens 5.2.c-h.



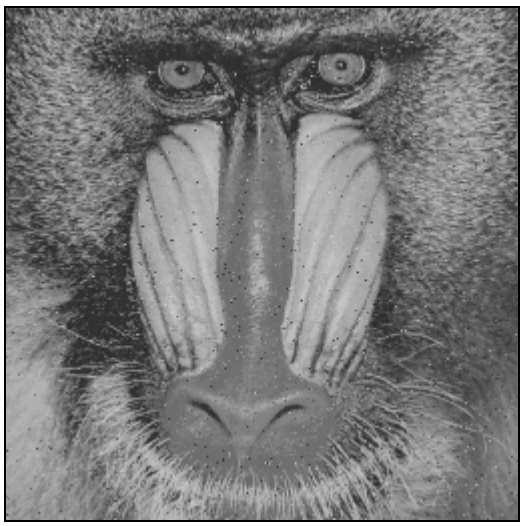
(5.2.a) Imagem a ser processada $Q^{\mathbf{x}}$.
256 × 256 pixels, 1 byte por pixel.



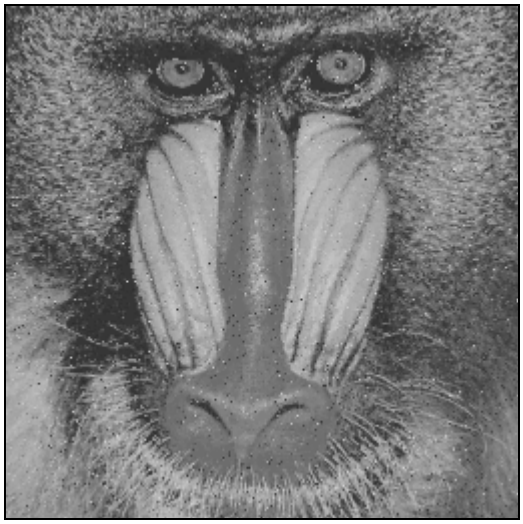
(5.2.b) Saída ideal desconhecida $Q^{\mathbf{y}}$.



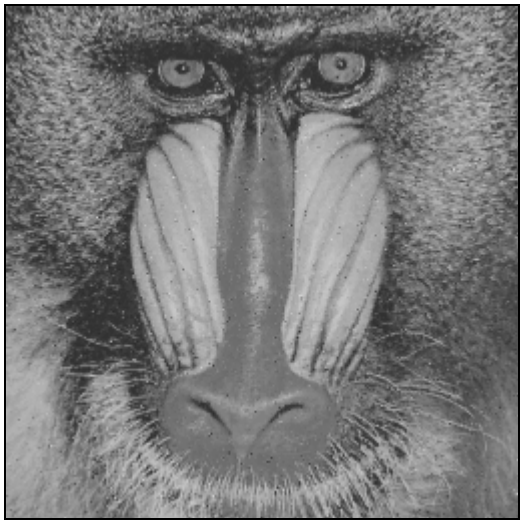
(5.2.c) Imagem processada $Q^{\mathbf{p}}$ pelo operador obtido na ו.f utilizando algoritmo de cortes com janela 1×2.
MAE: 1,77%.
Tempo de aplicação: 1 s.



(5.2.d) Imagem processada $Q^{\mathbf{p}}$ pelo operador obtido na ו.g utilizando algoritmo kd-árvore com janela 1×2.
MAE: 0,96%.
Tempo de aplicação: 12 s.



(5.2.e) Imagem processada $Q^{\mathbf{p}}$ pelo operador obtido na ı.h utilizando algoritmo de cortes com janela 2×3.
MAE: 3,47%.
Tempo de aplicação: 1 s.



(5.2.f) Imagem processada $Q^{\mathbf{p}}$ pelo operador obtido na ı.i utilizando algoritmo kd-árvore com janela 2×3.
MAE: 2,97%.
Tempo de aplicação: 2.573 s.

6 Aplicações

Apresentaremos, neste capítulo, algumas aplicações das técnicas descritas nesta tese. Também descrevemos certos detalhes de implementação omitidos nos capítulos anteriores.

6.1 Sistema de Processamento de Imagem Utilizado

Para gerar as aplicações mostradas neste trabalho, criamos uma biblioteca de rotinas para lidar com as imagens. Estas rotinas, que denominamos de "MM" (iniciais de Morfologia Matemática), foram escritas utilizando DJGPP versão 2.6.3, um compilador 32 bits da linguagem C++ para DOS. Escolhemos este compilador pois há versões deste compilador para diferentes sistemas operacionais, o que permite uma fácil migração para outros sistemas. O compilador gerencia toda a memória física disponível no computador, até 128 Mbytes. Além disso, caso o programa necessite de mais memória do que o disponível fisicamente, o código gerado por este compilador cria automaticamente a memória virtual com paginação no disco rígido.

Tomamos a decisão de não utilizar um ambiente semi-pronto de processamento de imagens, como Khoros, AVS (Application Visualization System) ou o 3DViewnix, pois a incorporação de novas funções nestes ambientes não é um procedimento muito simples e também porque o custo de alguns desses sistemas é proibitivo.

As duas classes que constituem o núcleo da biblioteca MM são `IMAGEM` e `IMGCOR`, destinadas respectivamente a armazenar uma imagem em níveis de cinza e colorida (uma imagem binária é armazenada como uma imagem em níveis de cinza).

Foram criadas as rotinas `letga` e `imptga` para leitura e escrita de imagens no formato targa ([Murray and vanRyper, 1994, pp. 644-662]). Adotamos este formato pois é simples de manipular e amplamente difundido. Outras rotinas podem ser acrescentadas na biblioteca MM para ler e escrever outros formatos de imagem. O sub-formato para imagens em níveis de cinzas é targa 8 bits, não compactado, sem mapa de cores. Para imagens coloridas foi utilizado o formato 24 bits, "true color", não compactado.

Na biblioteca MM, muitos operadores da linguagem C++ receberam a "sobrecarga" para trabalhar com variáveis do tipo `IMAGEM` e `IMGCOR`. Por exemplo, o operador parênteses foi utilizado para acessar cada pixel da imagem. O operador `-` indica uma erosão por um elemento estruturante plano e o operador `+` indica uma dilatação. O operador `||` indica o supremo entre duas imagens e o operador `&&` indica o ínfimo.

A partir da biblioteca de rotinas, criamos uma série de programas simples, por exemplo, para calcular erosão, dilatação, abertura, fechamento, para extrair uma sub-imagem, para "grudar" duas imagens, para corromper uma imagem com ruído "sal e pimenta", para calcular a média da diferença pixel a pixel entre duas imagens, etc. Como exemplo, copiamos abaixo o programa que calcula o supremo entre duas imagens em níveis de cinza:

```
#include <stdio.h>
#include <mm.h>

int main(int argc, char **argv)
{ IMAGEM a1,a2,s;

  if (argc!=4)
    erro("MaximoG arqent1.tga arqent2.tga arqsai.tga");
  letga(a1,argv[1]);
  letga(a2,argv[2]);
  if (a1.nl!=a2.nl || a1.nc!=a2.nc)
    erro("Imagens tamanhos diferentes");

  s = a1 || a2;
```

```
  imptga(s,argv[3]);
  return 0;
}
```

O arquivo `mm.h` contém o "header" das rotinas implementadas. A função `erro` imprime na tela uma mensagem de erro e aborta o programa. Os campos `nl` e `nc` de uma `IMAGEM` contém, respectivamente, o número de linhas e colunas da imagem. Observe que o operador `||` que calcula o supremo entre duas imagens está definido na biblioteca MM, que deve ser "unida" (linked) juntamente com as bibliotecas padrões da linguagem C++.

Esta forma de trabalhar permite criar fácil e rapidamente novos operadores para processamento de imagem, com todas as facilidades que a linguagem C++ oferece.

Além da programação explícita, utilizamos uma série de programas disponíveis comercialmente ou como "shareware" para tarefas auxiliares. Para fazer a conversão entre os diferentes formatos de imagens, utilizamos o programa comercial "Image Alchemy" (conhecido como "alchemy") versão 1.10. Para visualizar as imagens, utilizamos o programa "CompuShow" (conhecido como "cshow") versão 9.04a em DOS, e "WinJpeg" versão 2.65 ou "PSP browser" versão 3.0 em Windows 95.

Para editar manualmente as imagens, para ajustar brilho e contraste, ou para aplicar alguns filtros pré-definidos utilizamos três programas que tem funcionalidades semelhantes: "Paint Shop Pro" versão 3.0, "Adobe Photoshop" versão 3.0 e "Corel Photo-Paint" versão 5.0.

Todos os programas foram executados num computador Pentium 100 MHz com 32 MBytes de memória. O sistema operacional utilizado foi Windows 95. O disco rígido está compactado utilizando o programa DrvSpace 3, o que pode aumentar um pouco o tempo de leitura-escrita do disco rígido. As aplicações foram executadas numa janela DOS do sistema operacional Windows 95. O tempo de processamento foi medido utilizando a função `time` da linguagem C++. A precisão desta função chega somente aos

segundos. Assim, as medidas dos pequenos tempos de processamento, como um segundo, têm pouca precisão.

6.2 Implementação dos Algoritmos Apresentados

Os algoritmos apresentados neste trabalho (força bruta consistente, algoritmo de cortes e kd-árvore) foram implementados em C++ utilizando a biblioteca MM, descrita na seção anterior. Foram utilizadas "estruturas" (structures) com "campo de bits" (fields) para diminuir o uso de memória. Pela mesma finalidade, foram utilizadas "campos sobrepostos" (unions). As árvores foram construídas num grande vetor alocado dinamicamente numa única chamada de alocação de memória. Isto diminui o tempo gasto pelas funções de alocação dinâmica assim como o espaço perdido pela fragmentação de memória.

Foram escritos dois programas, `ConsG` e `ConsC`, que implementam a força bruta consistente, respectivamente para imagens em níveis de cinzas e coloridas. A entrada do programa são três imagens ($A^{\mathbf{x}}$, $A^{\mathbf{y}}$ e $Q^{\mathbf{x}}$) e um arquivo que representa a janela. A saída do programa é a imagem processada $Q^{\mathbf{p}}$.

Foram escritos seis programas que implementam a árvore de cortes.

- `CutB`, `CutG` e `CutC`: Constrói a árvore de cortes para as imagens respectivamente binárias, em níveis de cinza e coloridas. A entrada do programa são duas imagens (amostra de entrada $A^{\mathbf{x}}$ e amostra de saída $A^{\mathbf{y}}$) e um arquivo que especifica a janela. A saída é um operador morfológico representado como uma árvore de cortes.
- `AppB`, `AppG` e `AppC`: Aplica um operador representado como árvore de cortes numa imagem binária, em níveis de cinza ou colorida. A entrada do programa consiste numa imagem a ser processada $Q^{\mathbf{x}}$ e num operador criado por um dos programas do item anterior. A saída é a imagem processada $Q^{\mathbf{p}}$.

Foram escritos dois programas (para imagens em níveis de cinza e coloridas) que implementam a kd-árvore: `KdG` e `KdC`. A entrada do programa são três imagens ($A^{\mathbf{x}}$, $A^{\mathbf{y}}$ e $Q^{\mathbf{x}}$) e um arquivo que representa a janela. A saída do programa é a imagem processada $Q^{\mathbf{p}}$. Estes programas criam a kd-árvore, processam a imagem $Q^{\mathbf{x}}$ e apaga a kd-árvore no final do processamento. Optamos por não armazenar no disco rígido a kd-árvore criada pois o tempo de construção da árvore é muito pequeno em comparação com o tempo de aplicação do operador, o que permite criar a kd-árvore em cada processamento e destruí-la em seguida.

6.3 Eliminação de Ruído

6.3.1 Ruído "Sal e Pimenta" com κ-NN

Nesta subseção, vamos testar a restauração de imagens muito corrompidas pelo ruído "sal e pimenta" utilizando a estratégia κ-NN. Este ruído é utilizado em muitos exemplos de filtragem não-linear como, por exemplo, [Dougherty and Astola, 1994, pg. 115] e [Harvey and Marshall, 1994]. Com probabilidade 1/4, um pixel torna-se preto e com a mesma probabilidade um pixel torna-se branco. Assim, aproximadamente a metade dos pixels está corrompida pelo ruído.

Usando as imagens 6.1.a e 6.1.b como amostras de treinamento, o algoritmo de aprendizagem (árvore de cortes ou kd-árvore) aprende o comportamento do ruído e constrói um filtro que o elimina. Este operador, quando aplicado à imagem ruidosa 6.1.c, gera as imagens filtradas. Queremos que estas imagens filtradas sejam semelhantes à saída ideal desconhecida 6.1.d.

Nas figuras 6.1.e-h estão as imagens obtidas com janela 1×2 e utilizando respectivamente a árvore de cortes, 1-NN, 9-NN e 20-NN. Entre as três imagens obtidas pelo método vizinho mais próximo, a melhor foi 20-NN com MAE 4,74% (terceira linha da tabela 6.1 e figura 6.1.e). Observe como o resultado obtido pela árvore de cortes tem

praticamente a mesma qualidade que utilizando o vizinho mais próximo e o tempo de processamento é muito menor (terceira linha da tabela 6.1).

Utilizando uma janela um pouco maior, que denominamos de janela "gama" $\begin{bmatrix} \times & \times \\ \times & \end{bmatrix}$, os operadores melhoraram de qualidade, como podemos ver nas figuras 6.1.i-l. Os erros diminuíram conforme pode-se ver na tabela 6.1 linha 5. Com o aumento da dimensão, o tempo de processamento de árvore de cortes permaneceu praticamente inalterado (2 segundos) mas os tempos de processamento da kd-árvore aumentaram (linhas 2 e 4 da tabela 6.1). Uma exceção ocorreu com a estratégia 1-NN, pois o tempo de processamento diminuiu de 37 segundos com janela 1×2 para 23 segundos com janela gama. Desta vez, o melhor resultado foi obtido com a estratégia 9-NN (MAE 3,45%).

Para permitir uma comparação, mostramos nas figuras 6.1.m-p as imagens obtidas utilizando o filtro "mediano", descrito, por exemplo, em [Dougherty and Astola, 1994]. Filtros medianos e seus derivados são amplamente utilizados no processamento de imagem e sinal. As imagens 6.1.o e 6.1.p, obtidas com filtros medianos 5×5 e 7×7 apresentam um erro absoluto médio menor que qualquer imagem obtida utilizando a aprendizagem (3,14% e 3,02%, respectivamente). Porém, todos os detalhes (por exemplo, nos cabelos) foram perdidos e a imagem está muito "borrada", o que não ocorre com os filtros obtidos pela aprendizagem.

As imagens 6.1.q-z mostram a aplicação dos filtros obtidos anteriormente numa outra imagem, porém corrompida com um ruído com a mesma distribuição estatística.

Aplicando os operadores (janela 1×2) obtidos pela aprendizagem a partir das figuras 6.1.a e 6.1.b na figura 6.1.q resultaram as imagens 6.1.s e 6.1.t, usando respectivamente árvore de cortes e a estratégia 20-NN. A tabela 6.2, linhas 2 e 3, mostra os tempos e os erros obtidos, inclusive com as estratégias 1-NN e 9-NN cujas figuras correspondentes foram omitidas.

Utilizando a janela "gama", obtivemos as imagens 6.1.u e 6.1.v, respectivamente pela árvore de cortes e a estratégia 9-NN. Na tabela 6.2 linhas 4 e 5 mostram os tempos de processamento e os erros obtidos com a janela gama e diferentes estratégias.

Mostramos nas imagens 6.1.w-6.1.z as imagens obtidas utilizando o filtro "mediano".

		cortes	1-NN	9-NN	20-NN
janela	tempo (s)	2	37	83	151
1×2	MAE	4,96%	6,42%	4,79%	4,74%
janela	tempo (s)	2	23	104	278
"gama"	MAE	4,03%	4,75%	3,45%	3,55%

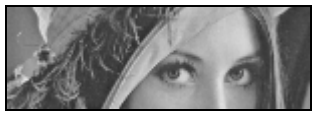
Tabela 6.1: Resumo de desempenho da eliminação de ruído "sal e pimenta" para imagem "Lenna" em níveis de cinza.

		cortes	1-NN	9-NN	20-NN
janela	tempo (s)	2	50	82	158
1×2	EMA	5,61%	6,99%	5,40%	5,18%
janela	tempo (s)	2	34	131	283
"gama"	EMA	5,82%	6,35%	4,54%	4,56%

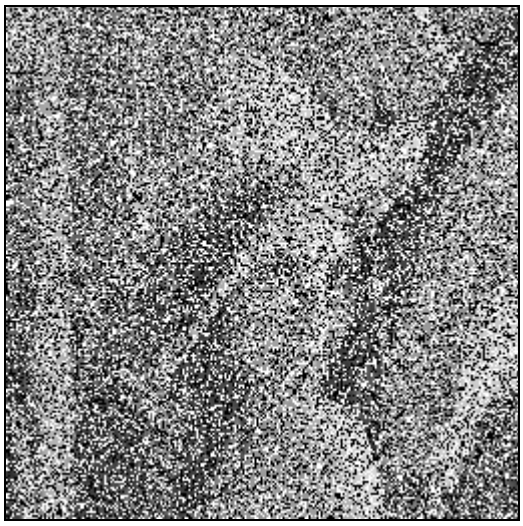
Tabela 6.2: Resumo de desempenho da eliminação de ruído "sal e pimenta" para imagem "Mandril" em níveis de cinza.

Figura 6.1: Eliminação do ruído "sal e pimenta" em imagens em níveis de cinza.

(6.1.a) Amostra de entrada $A^{\mathbf{x}}$.
51 × 151 pixels, 1 byte por pixel.



(6.1.b) Amostra de saída $A^{\mathbf{y}}$.
51 × 151 pixels, 1 byte por pixel.



(6.1.c) Imagem a ser processada $Q^{\mathbf{x}}$.
256 × 256 pixels, 1 byte por pixel.

(6.1.d) Saída ideal desconhecida $Q^{\mathbf{y}}$.
256 × 256 pixels, 1 byte por pixel.

(6.1.e) Saída do algoritmo de cortes.
Janela 1×2. MAE: 4,96%.
Treinamento: 1 s. Aplicação: 1 s.

(6.1.f) Saída da aprendizagem 1-NN.
Janela 1×2. MAE: 6,42%.
Treinamento: 1 s. Aplicação: 36 s.

(6.1.g) Saída da aprendizagem 9-NN.
Janela 1×2. MAE: 4,79%.
Treinamento: 1 s. Aplicação: 82 s.

(6.1.h) Saída da aprendizagem 20-NN.
Janela 1×2. MAE: 4,74%.
Treinamento: 1 s. Aplicação: 150 s.

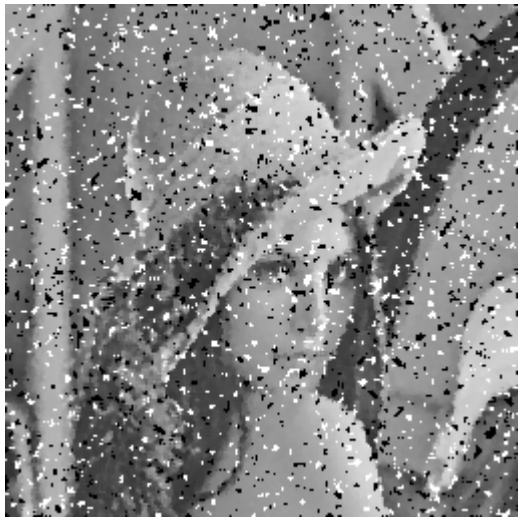
(6.1.i) Saída do algoritmo de cortes.
Janela "gama". MAE: 4,03%.
Treinamento: 1 s. Aplicação: 1 s.

(6.1.j) Saída da aprendizagem 1-NN.
Janela "gama". MAE: 4,75%.
Treinamento: 1 s. Aplicação: 22 s.

(6.1.k) Saída da aprendizagem 9-NN. Janela "gama". MAE: 3,45%. Treinamento: 1 s. Aplicação: 103 s.

(6.1.l) Saída da aprendizagem 20-NN. Janela "gama". MAE: 3,55%. Treinamento: 1 s. Aplicação: 277 s.

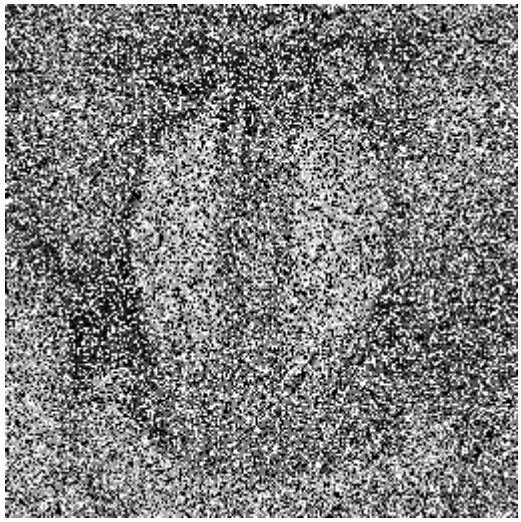
(6.1.m) Filtro "mediano". Janela "gama". MAE: 17,15%. Processamento: 2 s.



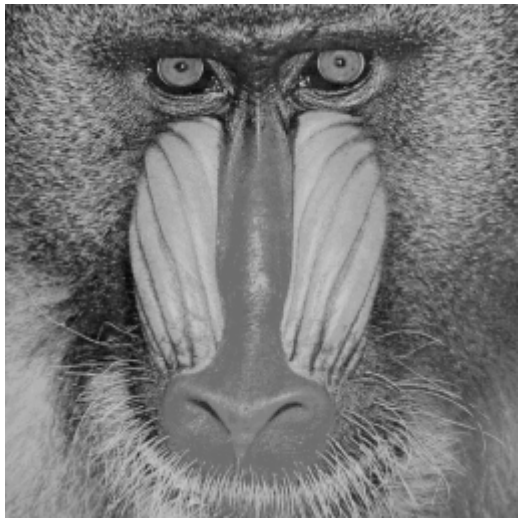
(6.1.n) Filtro "mediano". Janela 3×3. MAE: 7,18%. Processamento: 3 s.

(6.1.o) Filtro “mediano” com janela 5×5.
MAE: 3,14%.
Processamento: 4 s.

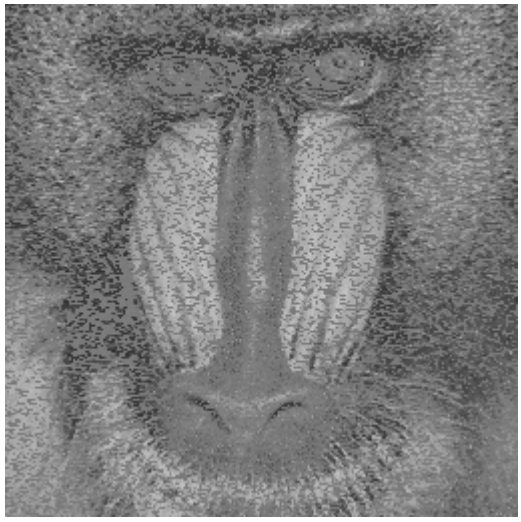
(6.1.p) Filtro “mediano” com janela 7×7.
MAE: 3,02%.
Processamento: 5 s.



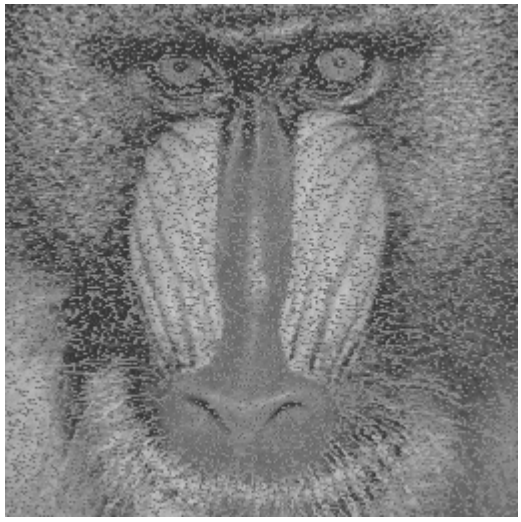
(6.1.q) Imagem a ser processada $Q^{\mathbf{x}}$.
256 × 256 pixels, 1 byte por pixel.



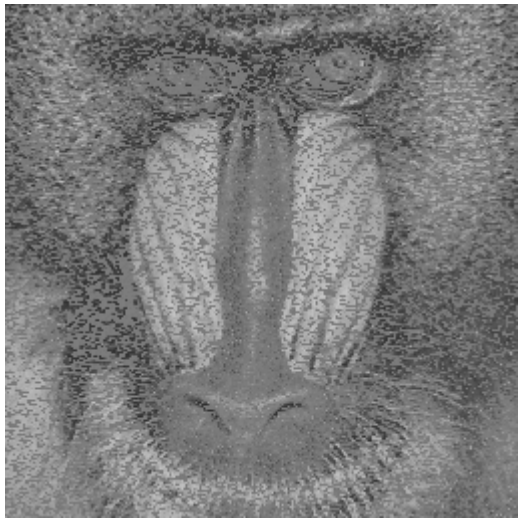
(6.1.r) Saída ideal desconhecida $Q^{\mathbf{y}}$.
256 × 256 pixels, 1 byte por pixel.



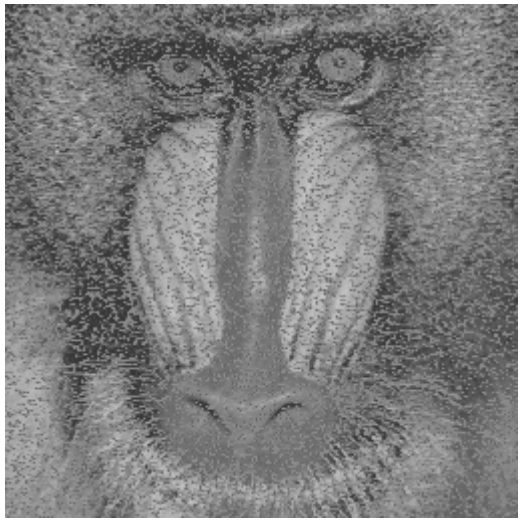
(6.1.s) Saída do algoritmo de cortes.
Janela 1×2. MAE: 5,61%.
Treinamento: 1 s. Aplicação: 1 s.



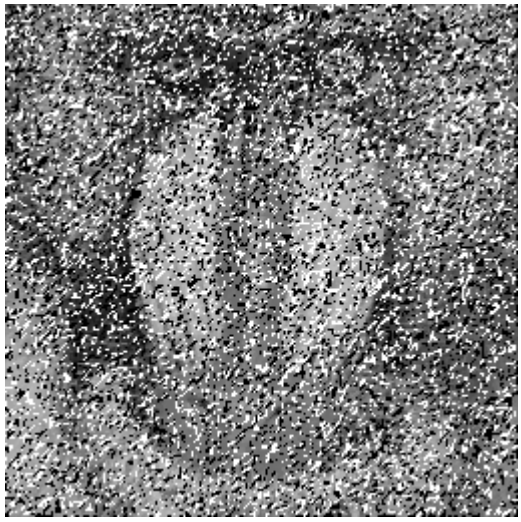
(6.1.t) Saída da aprendizagem 20-NN.
Janela 1×2. MAE: 5,18%.
Treinamento: 1 s. Aplicação: 157 s.



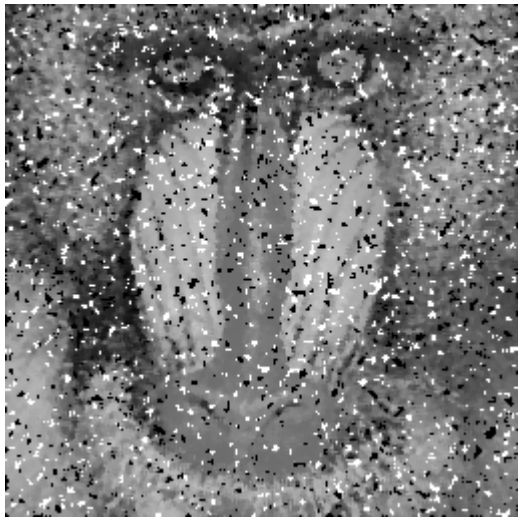
(6.1.u) Saída do algoritmo de cortes.
Janela "gama". MAE: 5,82%.
Treinamento: 1 s. Aplicação: 1 s.



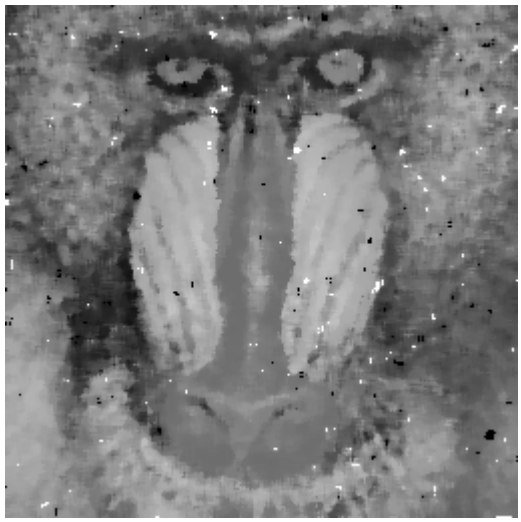
(6.1.v) Saída da aprendizagem 9-NN.
Janela "gama". MAE: 4,54%.
Treinamento: 1 s. Aplicação: 130 s.



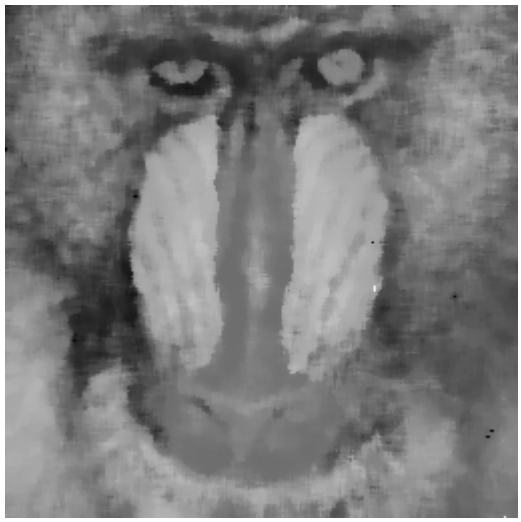
(6.1.w) Filtro “mediano”. Janela “gama”.
MAE: 18,28%. Processamento: 2 s.



(6.1.x) Filtro “mediano”. Janela 3×3.
MAE: 9,38%. Processamento: 3 s.



(6.1.y) Filtro “mediano”. Janela 5×5.
MAE: 5,77%.
Processamento: 4 s.



(6.1.z) Filtro “mediano”. Janela 7×7.
MAE: 5,61%.
Processamento: 5 s.

6.3.2 Ruído "Sal e Pimenta" Colorido

Nesta subseção, vamos testar a restauração de imagens coloridas corrompidas pelo ruído "sal e pimenta". Com probabilidade 1/4, um pixel torna-se preto e com a mesma probabilidade um pixel torna-se branco (figuras 6.2.a-d).

Usando as imagens 6.2.a e 6.2.b como amostras de treinamento, um algoritmo de aprendizagem (árvore de cortes ou kd-árvore) aprende o comportamento do ruído e constrói um filtro que o elimina. Este operador, quando aplicado à imagem ruidosa 6.2.c, gera as imagens filtradas mostradas abaixo. Queremos que estas imagens processadas sejam semelhantes à saída ideal desconhecida 6.2.d.

Utilizando os algoritmos de corte e 9-NN, na janela "gama" $\begin{bmatrix} \times & \times \\ \times & \end{bmatrix}$, obtivemos as saídas 6.2.e e 6.2.f. Para servir de comparação, incluímos as imagens (g) e (h), geradas pelo filtro mediano.

As imagens 6.2.i-6.2.u mostram a aplicação dos filtros obtidos anteriormente numa outra imagem, porém corrompida por um ruído com a mesma distribuição estatística. A imagem 6.2.i é a imagem ruidosa e a imagem 6.2.j é a saída ideal. Aplicando na imagem 6.2.i os operadores, aprendidos respectivamente pelos algoritmo de cortes e 9-NN, obtivemos as imagens 6.2.k e 6.2.l.

Observe que as imagens resultantes contém somente as tonalidades de cor-de-rosa e roxo, pois são as únicas cores presentes na saída da imagem de treinamento. Assim, o erro euclidiano médio foi muito grande (20,64% e 16,54%). Este problema, provocado pelo treinamento insuficiente, pode ser minimizado utilizando alguma informação "a priori" sobre o operador que desejamos projetar. Para a aplicação em questão, sabemos "a priori" que o comportamento do filtro é idêntico para as três bandas (vermelho, verde e azul) permitindo-nos projetar um filtro em níveis de cinza para ser aplicado independentemente nas três bandas da imagem. Denominamos este processo de

"decomposição em bandas". Para projetar este filtro em níveis de cinza, convertemos cada uma das três bandas da amostra de entrada 6.2.a para níveis de cinza e os "grudamos" entre si, obtendo a imagem 6.2.m. Utilizamos o mesmo procedimento para obter a imagem 6.2.n a partir da amostra de saída 6.2.b. A parte superior da imagem corresponde à banda vermelha, a parte central é a banda verde e a parte inferior é a banda azul. Utilizando as imagens 6.2.m e 6.2.n como amostras de treinamento, obtivemos um filtro (árvore de cortes) para níveis de cinza. O projeto deste filtro levou 3 segundos e o filtro ocupou 212.776 bytes.

Separando as três bandas da imagem a ser processada 6.2.i, obtivemos as imagens 6.2.o, 6.2.p e 6.2.q que correspondem respectivamente aos espectros vermelho, verde e azul da imagem 6.2.i. Aplicamos o filtro obtido a partir das imagens 6.2.m e 6.2.n nas imagens 6.2.o-p obtendo as imagens 6.2.r-t. A aplicação do filtro para cada banda levou aproximadamente 1 segundo.

Juntando as imagens das três bandas filtradas 6.2.r-t, obtivemos a imagem 6.2.u, que possui o erro euclidiano médio 7,56%, muito inferior ao erro 20,64% obtido anteriormente (imagem 6.2.k). Assim, podemos concluir que a "decomposição em bandas" pode ser muito eficaz quando sabemos "a priori" que um único filtro em níveis de cinza deve ser aplicado para cada uma das três bandas.

Utilizando esta mesma técnica para processar a imagem 6.2.c, obtivemos a imagem 6.2.v com erro euclidiano médio 5,00%, pouco menor que 5,99% obtido anteriormente na imagem 6.2.e.

Figura 6.2: Eliminação do ruído "sal e pimenta" em imagens coloridas.

(6.2.a) Amostra de entrada A^x.
51 × 151 pixels, 3 bytes por pixel.

(6.2.b) Amostra de saída A^y.
51 × 151 pixels, 3 bytes por pixel.

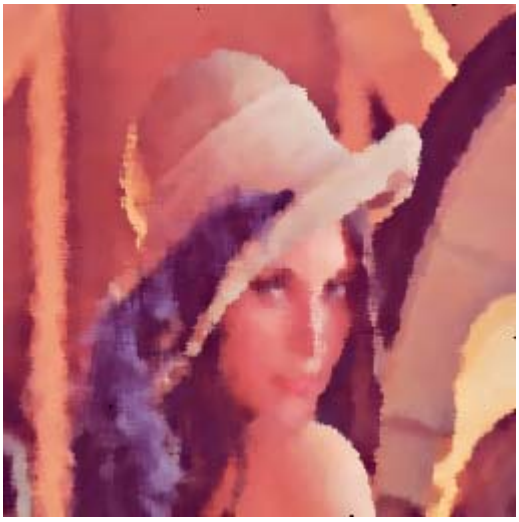
(6.2.c) Imagem a ser processada Q^x.
256 × 256 pixels, 3 bytes por pixel.

(6.2.d) Saída ideal desconhecida Q^y.
256 × 256 pixels, 3 bytes por pixel.

(6.2.e) Saída do algoritmo de cortes.
Janela "gama". MEE: 5,99%.
Treinamento: 1 s. Aplicação: 2 s.

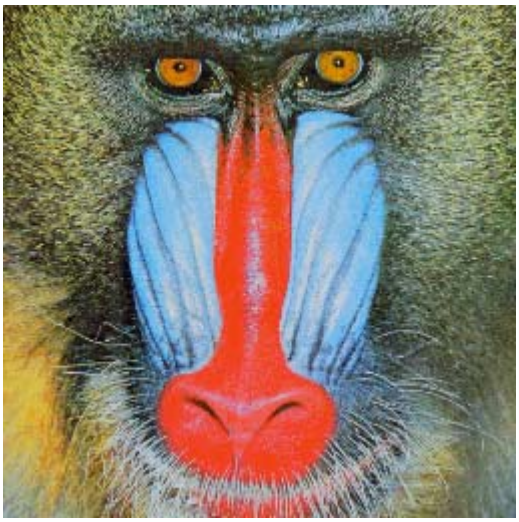
(6.2.f) Saída da aprendizagem 9-NN.
Janela "gama". MEE: 4,61%.
Treinamento: 2 s. Aplicação: 1.008 s.

(6.2.g) Filtro “mediano”. Janela 3×3. MEE: 7,33%. Processamento: 6 s.



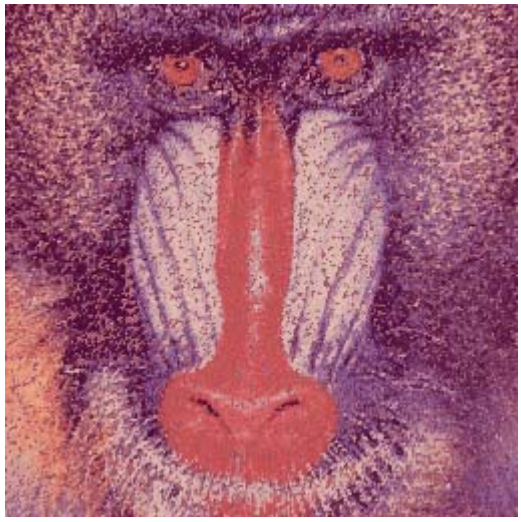
(6.2.h) Filtro “mediano” com janela 7×7. MEE: 3,20%. Processamento: 15 s.

(6.2.i) Imagem a ser processada Q^{x}. 256 × 256 pixels, 3 bytes por pixel.



(6.2.j) Saída ideal desconhecida Q^{y}. 256 × 256 pixels, 3 bytes por pixel.

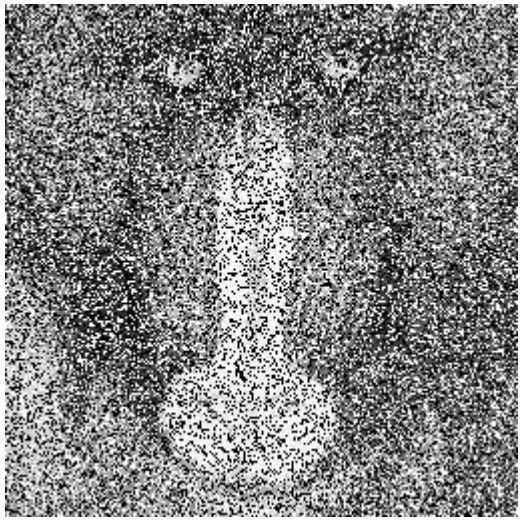
(6.2.k) Saída do algoritmo de cortes. Janela "gama". MEE: 20,64%. Aplicação: 2 s.



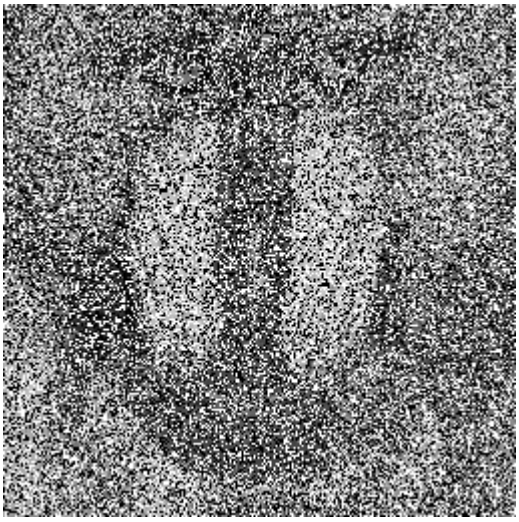
(6.2.l) Saída da aprendizagem 9-NN. Janela "gama". MEE: 16,54%. Aplicação: 6.379 s.

(6.2.m) Amostra de entrada $A^{\mathbf{x}}$ obtida separando as três bandas da imagem 6.2.a. 153 × 151 pixels, 1 byte por pixel.

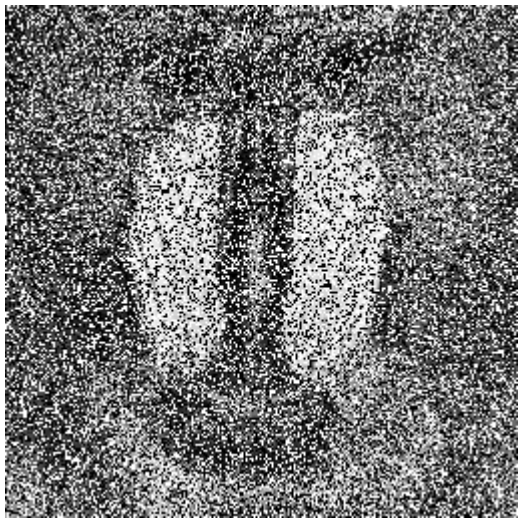
(6.2.n) Amostra de saída $A^{\mathbf{y}}$ obtida separando as três bandas da imagem 6.2.b. 153 × 151 pixels, 1 byte por pixel.



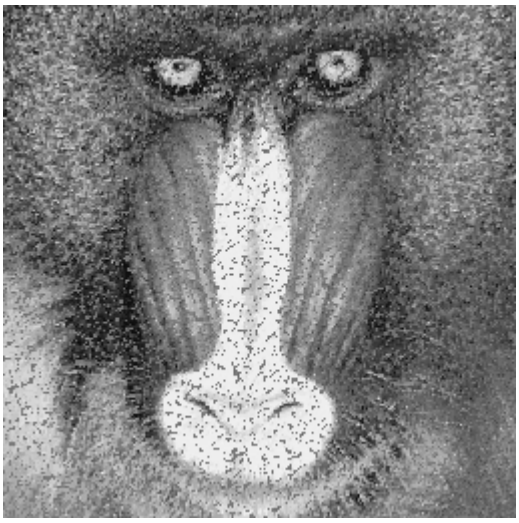
(6.2.o) Espectro vermelho da imagem 6.2.i.



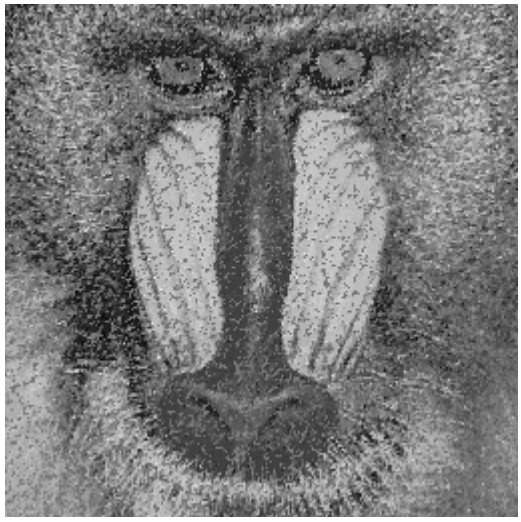
(6.2.p) Espectro verde da imagem 6.2.i.



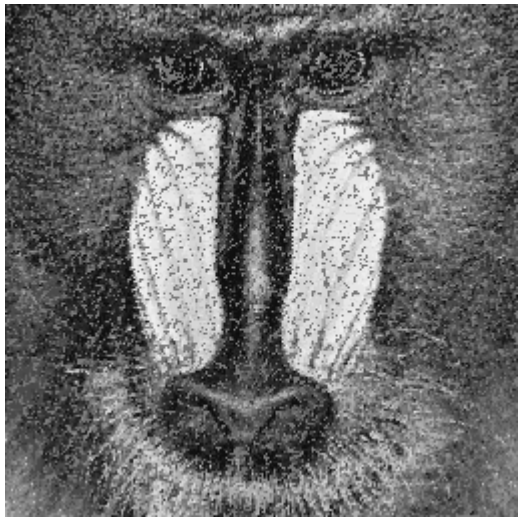
(6.2.q) Espectro azul da imagem 6.2.i.



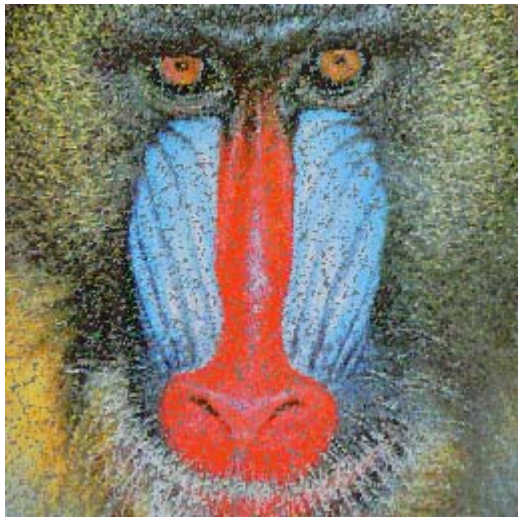
(6.2.r) Imagem 6.2.o filtrada (vermelho).



(6.2.s) Imagem 6.2.p filtrada (verde).



(6.2.t) Imagem 6.2.q filtrada (azul).



(6.2.u) Imagem processada final, juntando as imagens 6.2.r, 6.2.s e 6.2.t.
Janela "gama". MEE: 7,56%.

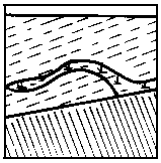
(6.2.v) Imagem processada final.
Janela "gama". MEE: 5,00%.

6.4 Reconhecimento de Textura

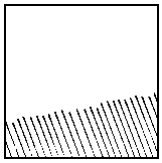
6.4.1 Reconhecimento de Textura Binária

Nesta subseção, vamos ilustrar o reconhecimento de textura binária. A construção manual do operador morfológico para reconhecer uma textura, mesmo que esta seja binária, não é uma tarefa muito simples. O algoritmo de cortes pode construir automaticamente o operador a partir das imagens amostras, facilitando o trabalho do usuário. Fornecendo a amostra de entrada (figura 6.3.a) e de saída (figura 6.3.b), o algoritmo de cortes constrói um operador morfológico. Neste caso, utilizamos uma janela 3×3 e o algoritmo de cortes levou 1 segundo para projetar o operador. Este operador, quando aplicado à imagem 6.3.c, reconhece a textura desejada, gerando a imagem 6.3.d. Esta imagem ainda não está "limpa". Para eliminar as "sujeiras", aplicamos uma série de erosões e dilatações. Primeiro, fizemos com que a região com a textura desejada se tornasse um bloco, o que permitiu eliminar em seguida todas as "sujeiras" que estão longe desse bloco. Para isso, aplicamos em 6.3.d uma erosão 6×7 o que gerou a imagem 6.3.e, seguida por uma dilatação 6×7 em 6.3.e que gerou 6.3.f. Observe que em 6.3.f a região reconhecida forma um grande bloco. Resta eliminar todas as "sujeiras" que não fazem parte deste bloco. Para isso, aplicamos uma dilatação 10×10 na imagem 6.3.f. Isto elimina todas as "sujeiras" (6.3.g) mas, ao mesmo tempo, diminui o tamanho da região reconhecida. Aplicando uma erosão 18×18, obtemos a região final (6.3.h). A erosão final aplicada (18×18) foi maior que a dilatação anterior (10×10) pois em muitos pontos a textura a ser reconhecida não chegava até às bordas. Sobrepondo a região reconhecida à imagem original, obtemos a imagem 6.3.i.

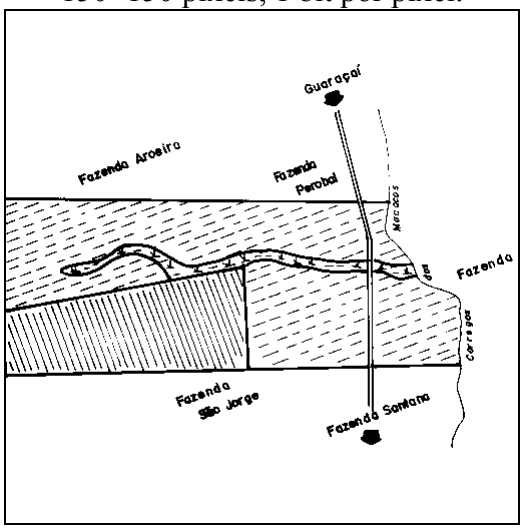
Figura 6.3: Reconhecimento de textura binária.



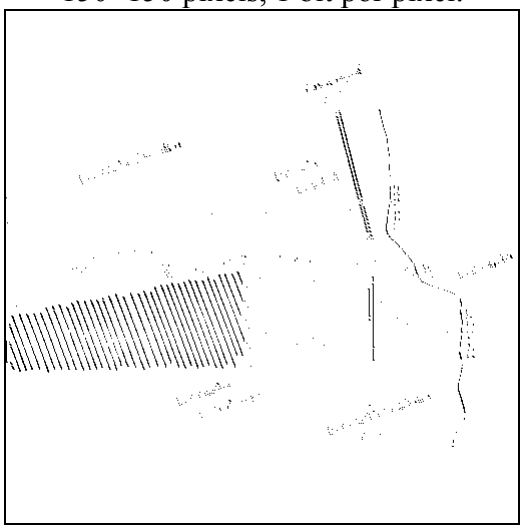
(6.3.a) Amostra de entrada.
150×150 pixels, 1 bit por pixel.



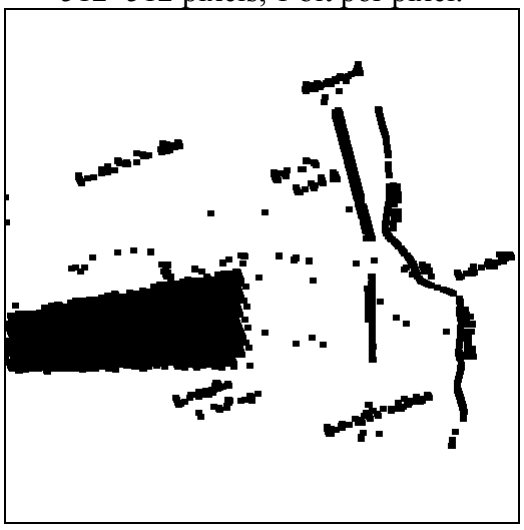
(6.3.b) Amostra de saída.
150×150 pixels, 1 bit por pixel.



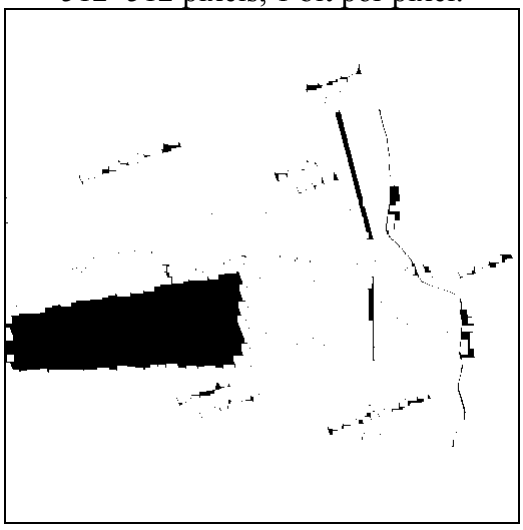
(6.3.c) Imagem a ser processada.
512×512 pixels, 1 bit por pixel.



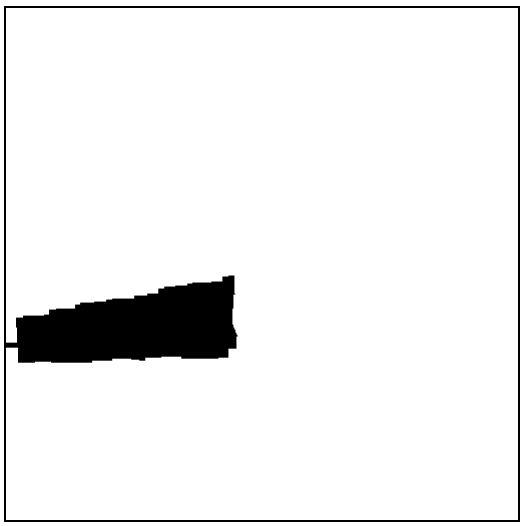
(6.3.d) Imagem processada.
512×512 pixels, 1 bit por pixel.



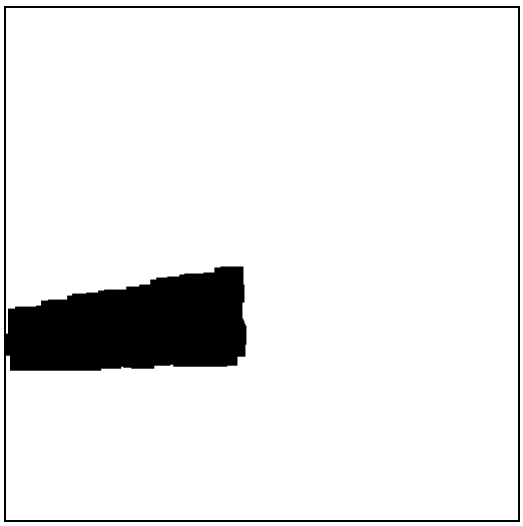
(6.3.e) Aplicação da erosão 6×7 em 6.3.d.



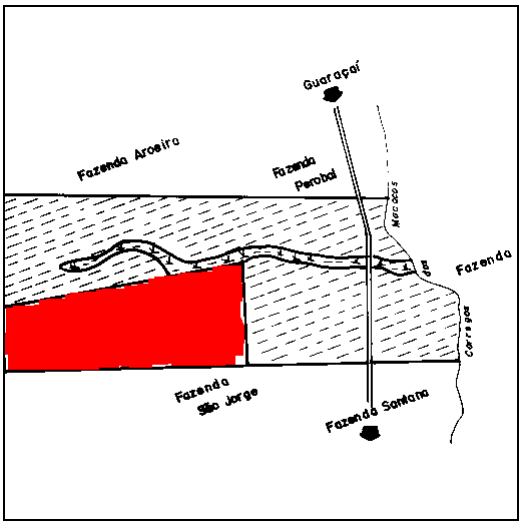
(6.3.f) Aplicação da dilatação 6×7 em 6.3.e.



(6.3.g) Aplicação da dilatação 10×10 em 6.3.f.



(6.3.h) Aplicação da erosão 18×18 em 6.3.g.



(6.3.i) Sobreposição das imagens 6.3.c e 6.3.h.

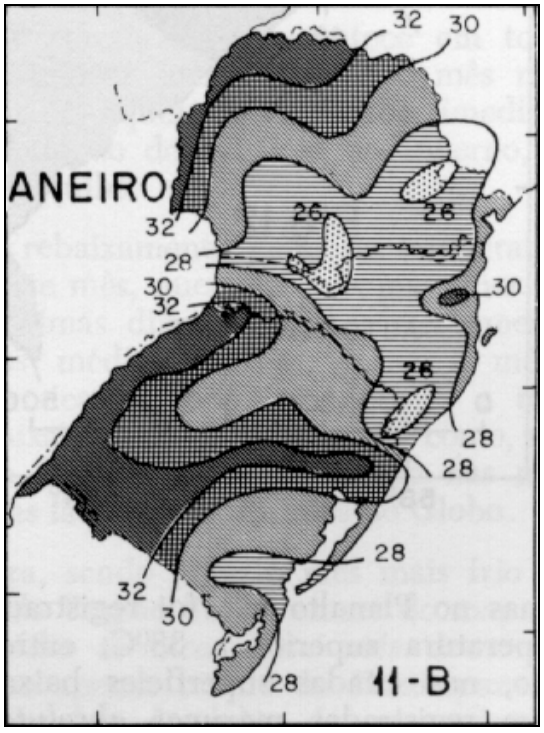
6.4.2 Reconhecimento de Textura em Níveis de Cinza

Nesta subseção, vamos ilustrar o reconhecimento de textura em níveis de cinza. As ilustrações desta seção referem-se aos mapas que mostram a média das temperaturas máximas diárias da região sul do Brasil, "scanneados" em 256 níveis de cinza do livro [IBGE, 1977, pp. 55, 61]. As figuras 6.4.a e 6.4.b mostram a temperatura do mês de janeiro, que serviram como amostras de treinamento para reconhecer a região de temperatura de 30 a 32 graus, com textura "xadrez". Foi utilizada a janela 3×3 e o algoritmo de cortes. Neste caso, não seria boa escolha utilizar a kd-árvore, pois o tempo de processamento seria demasiado longo devido ao tamanho da janela e da amostra de treinamento. A árvore de cortes levou apenas 15 segundos de treinamento. Aplicando o operador assim obtido no mapa do mês de junho, o que levou 3 segundos, obtivemos a imagem 6.4.d, que está excessivamente ruidosa. Aplicando na figura 6.4.d o filtro mediano 6×6 seguido por um outro filtro mediano 9×9, obtivemos a imagem 6.4.f, que corresponde à região procurada. Sobrepondo esta imagem com a imagem original 6.4.c, obtivemos a imagem 6.4.g.

O mesmo procedimento (árvore de cortes seguida por dois filtros medianos) foi aplicado aos mapas dos outros meses, obtendo as imagens 6.4.h-o.

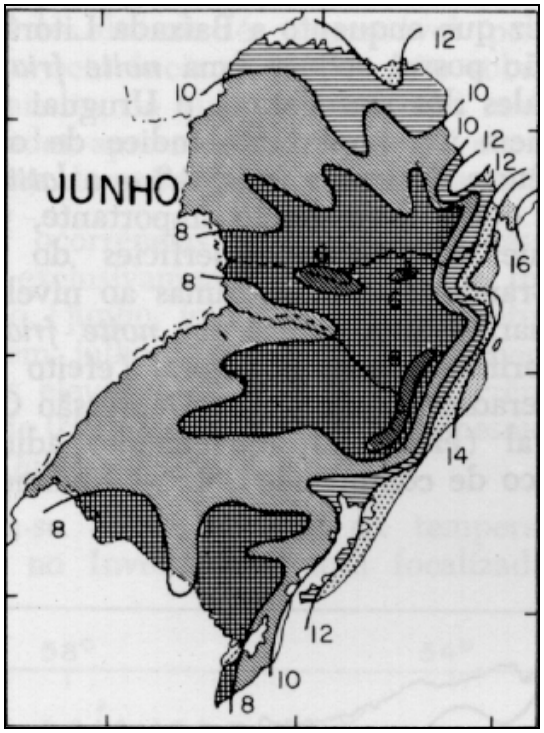
Utilizando o procedimento exatamente igual, também foi possível reconhecer uma outra textura, o "hachurado escuro", que corresponde às temperaturas acima de 32 graus. Apesar de existir nos mapas também uma textura "hachurado claro" o que aparentemente poderia dificultar o reconhecimento, podemos observar que o reconhecimento teve ótima qualidade. As imagens 6.4.p e 6.4.q são de treinamento. As imagens 6.4.r-v mostram os resultados obtidos.

Figura 6.4: Reconhecimento de Textura em Níveis de Cinza



(6.4.a) Amostra de entrada.
422×312 pixels, 8 bits por pixel.

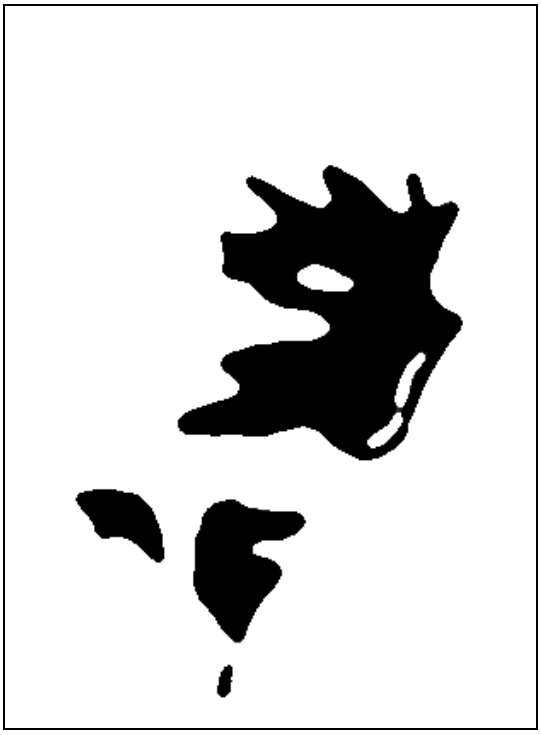
(6.4.b) Amostra de saída. Janela 3×3.
Treinamento: 15s.
Operador: 2.105.336 bytes



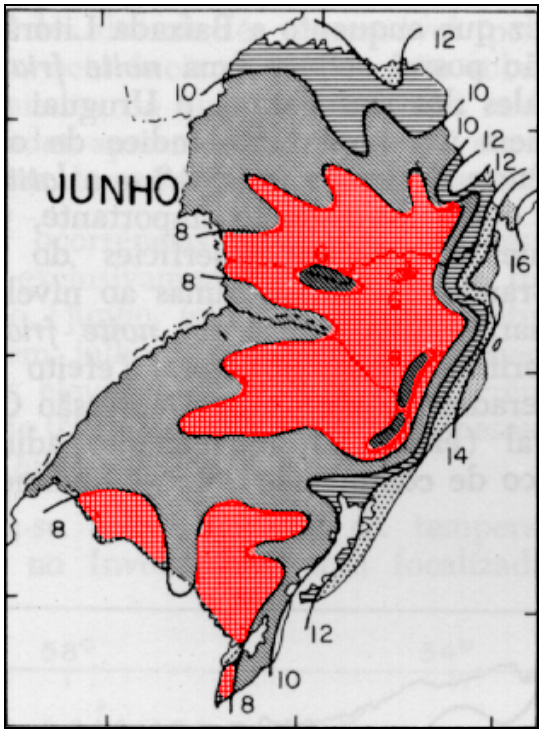
(6.4.c) Imagem a ser processada.
424×312 pixels, 8 bits por pixel.

(6.4.d) Saída do algoritmo de cortes.
Aplicação 3s.

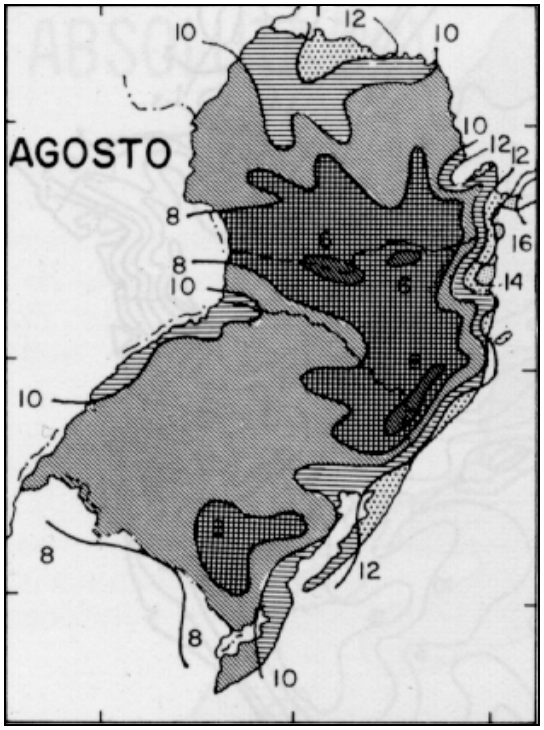
(6.4.e) Saída do filtro mediano 6×6 aplicado em 6.4.d.



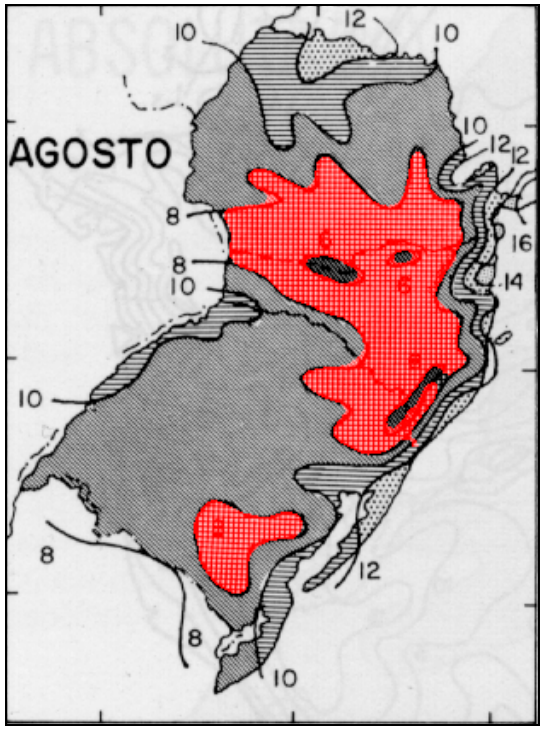
(6.4.f) Saída do filtro mediano 9×9 aplicado em 6.4.e.



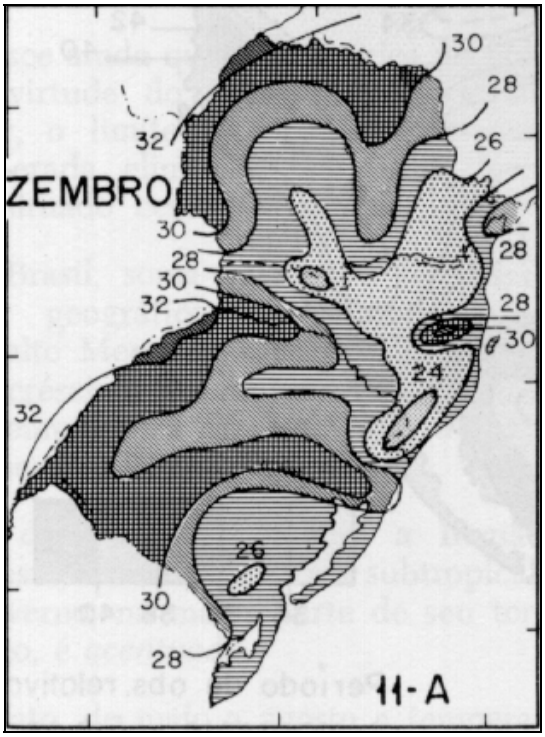
(6.4.g) Sobreposição das imagens 6.4.c e 6.4.f. A imagem 6.4.f está em vermelho.



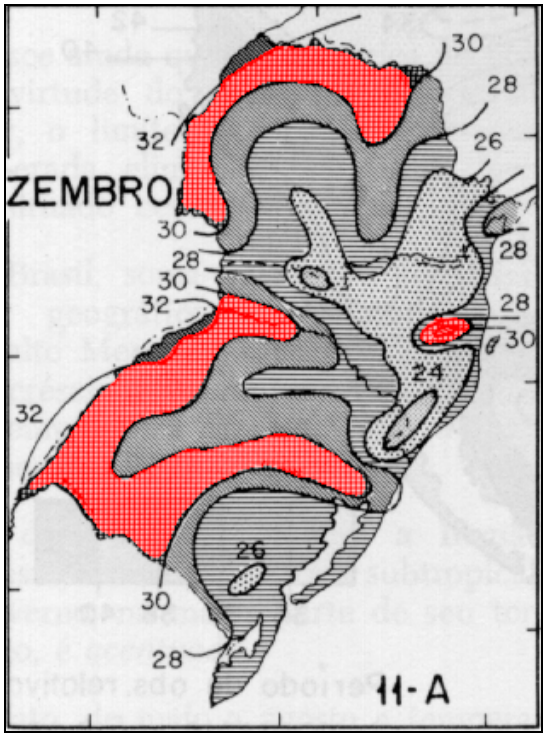
(6.4.h) Mapa do mês agosto.



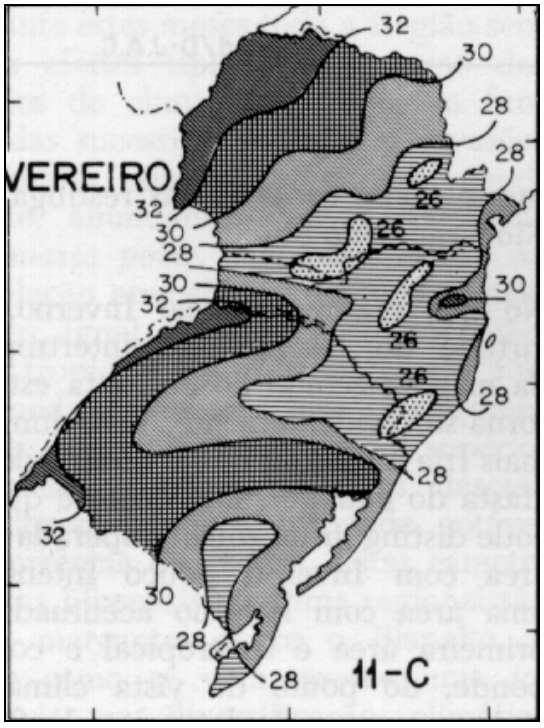
(6.4.i) Mês agosto processado.



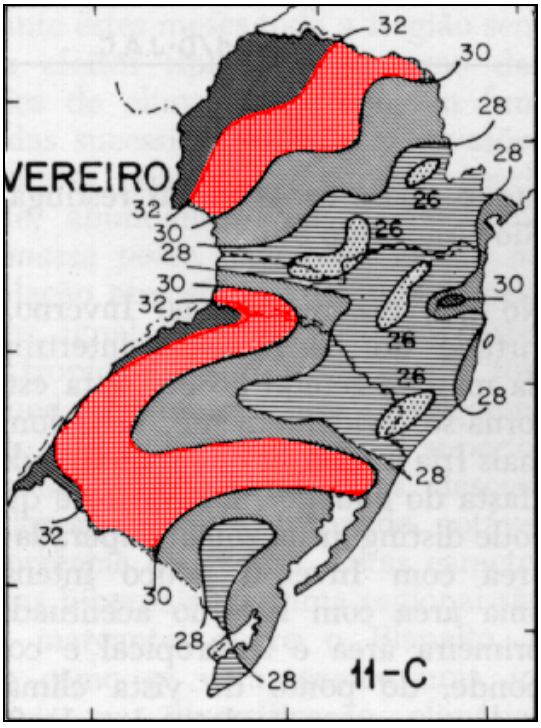
(6.4.j) Mapa do mês dezembro.



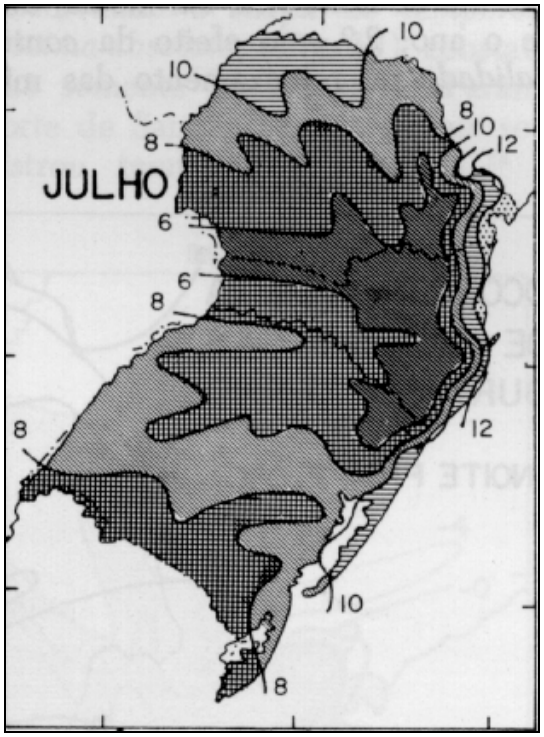
(6.4.k) Mês dezembro processado.



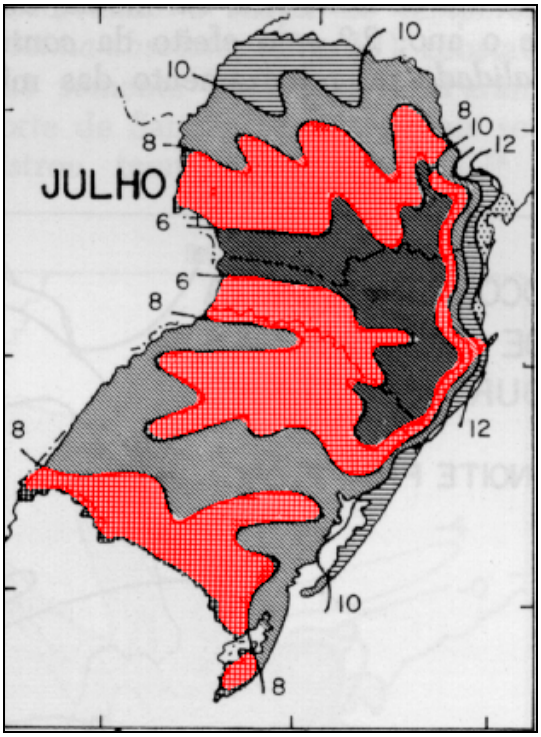
(6.4.l) Mapa do mês fevereiro.



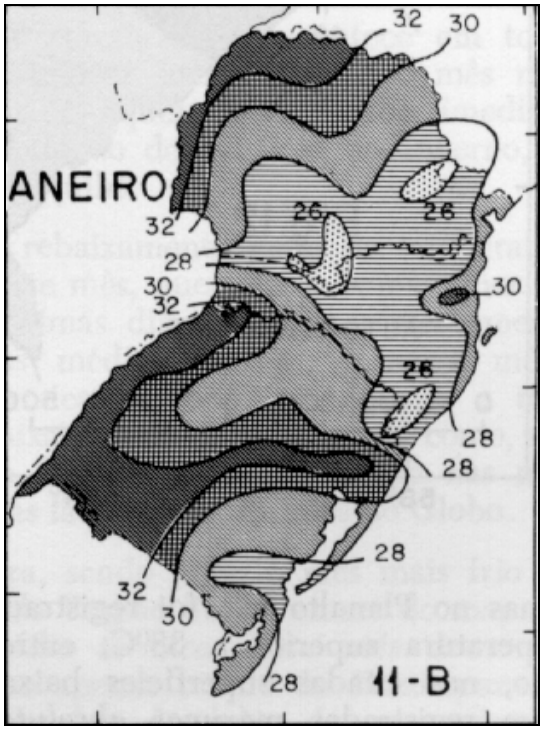
(6.4.m) Mês fevereiro processado.



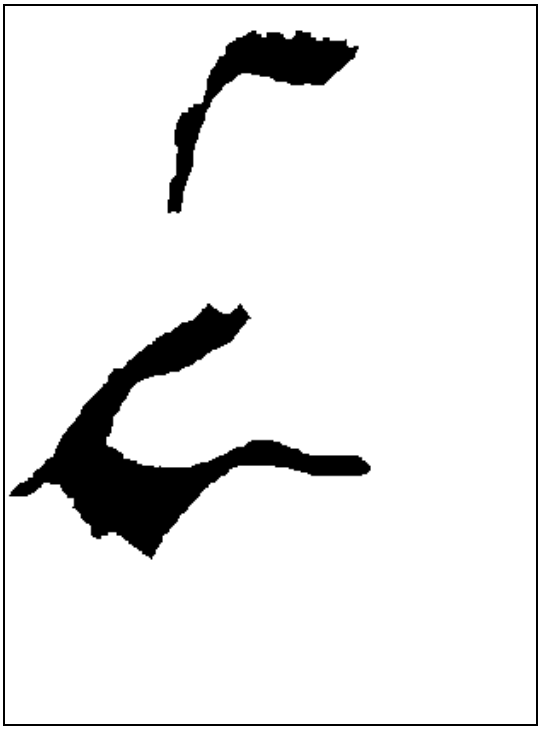
(6.4.n) Mapa do mês julho.



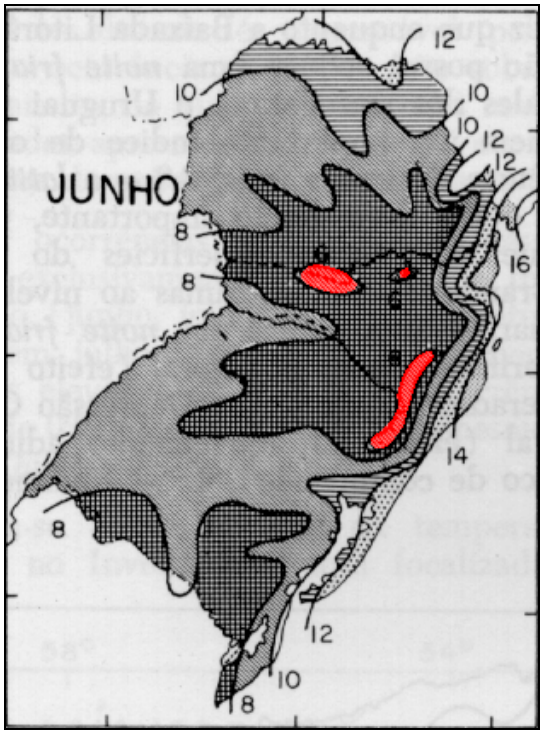
(6.4.o) Mês julho processado.



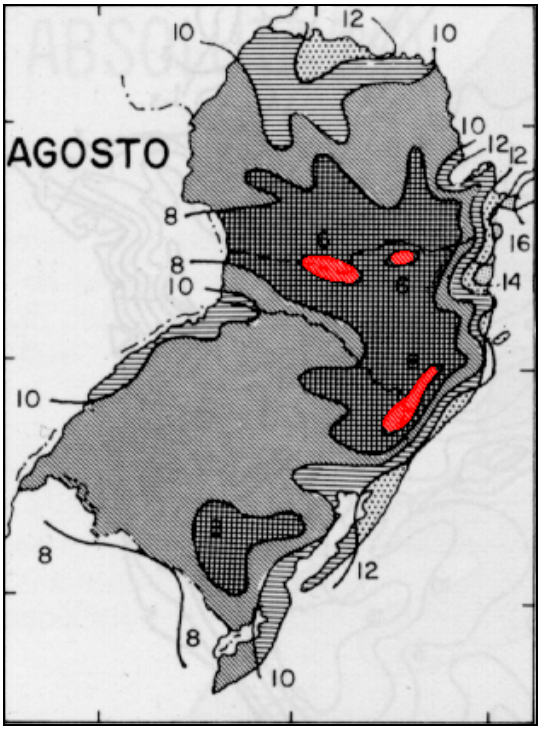
(6.4.p) Amostra de entrada.



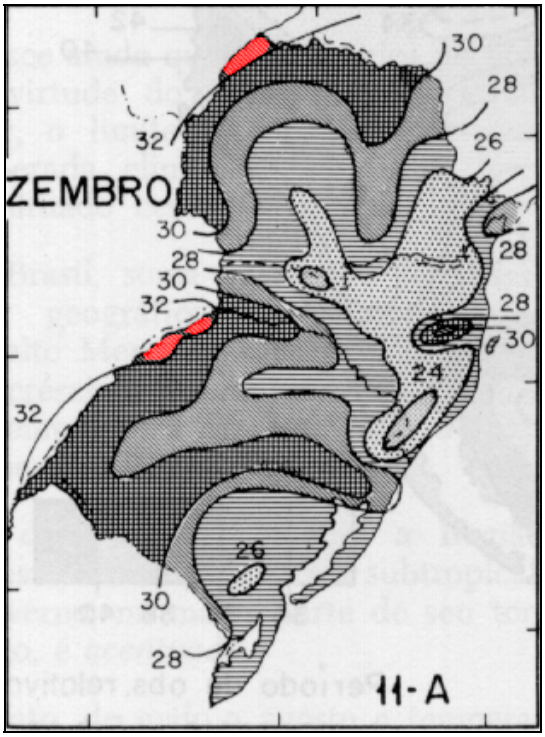
(6.4.q) Amostra de saída para reconhecer a textura “hachurado escuro”.



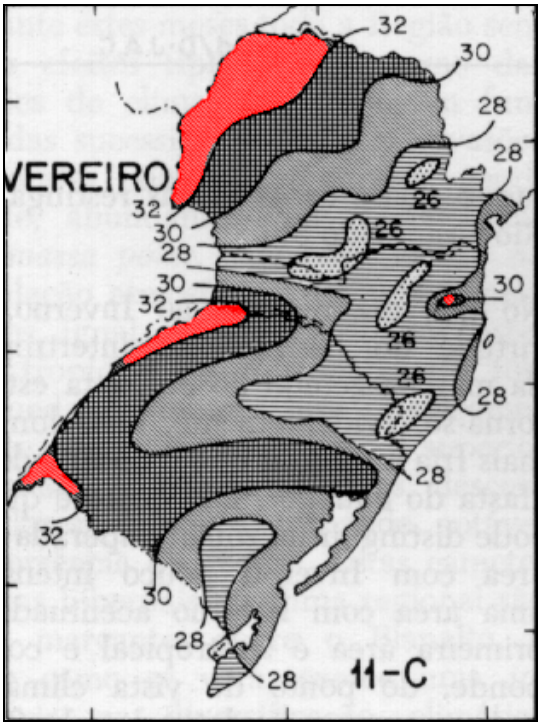
(6.4.r) Mapa de junho processado.



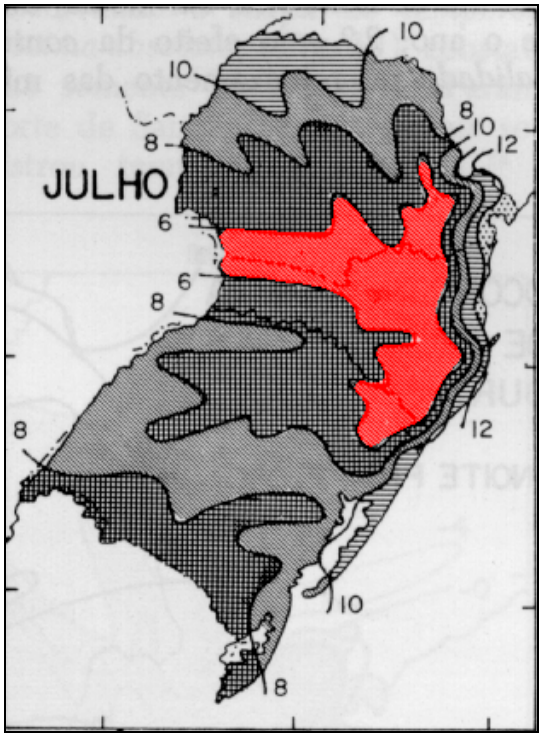
(6.4.s) Mapa de agosto processado.



(6.4.t) Mapa de dezembro processado.



(6.4.u) Mapa de fevereiro processado.



(6.4.v) Mapa de julho processado.

6.5 Emulação de Filtros Desconhecidos

Nesta seção, iremos ilustrar a emulação de um filtro digital "caixa preta" (isto é, do qual desconhecemos o mecanismo interno de funcionamento) através da aprendizagem computacional. O comportamento deste filtro digital é conhecido apenas através de algumas imagens com respectivas imagens filtradas.

6.5.1 Filtro "Solarize", Decomponível em Bandas

Nesta subseção, iremos emular o filtro "solarize" do Adobe Photoshop 3.0. A figura 6.5.a é a imagem a ser processada e a figura 6.5.b é a amostra de entrada. A amostra de saída 6.5.c foi obtida aplicando o filtro "solarize" na imagem 6.5.b. Fornecendo as imagens 6.5.b e 6.5.c como amostras de treinamento, o algoritmo de cortes constrói um operador morfológico. A janela utilizada foi 2×2. Aplicando este operador na imagem 6.5.a obtemos a imagem 6.5.d. Compare-a com a saída ideal 6.5.e, obtida aplicando o próprio filtro "solarize" na imagem 6.5.a. O erro euclidiano médio foi de apenas 0,98%. De fato, é muito difícil achar alguma diferença entre as imagens 6.5.d e 6.5.e. O tempo de treinamento foi 23 segundos e o de aplicação 9 segundos. Observe que nesta aplicação fica difícil utilizar o algoritmo kd-árvore pois a dimensão 12 (janela 2×2 e quantidade de bandas 3), relativamente grande, torna este algoritmo extremamente lento.

Aplicando o operador morfológico que emula o filtro "solarize" obtido anteriormente na imagem 6.5.f, completamente diferente da amostra de treinamento, resulta a imagem 6.5.g. Podemos verificar visualmente que o resultado está muito distante da saída ideal mostrada na figura 6.5.h. De fato, o erro euclidiano médio é 14,89%.

Como melhorar a qualidade do operador obtido? A técnica mais óbvia seria aumentar o tamanho de amostras de treinamento. Mas a qualidade pode ser melhorada sem a necessidade de obter mais amostras recorrendo à técnica que denominamos "decomposição em bandas". Uma imagem colorida pode ser convertida em três imagens em níveis de

cinza correspondentes às intensidades das três bandas. Assim, a imagem a ser processada 6.5.f foi convertida na imagem 6.5.i, a amostra de entrada 6.5.b foi convertida na imagem 6.5.j e a amostra de saída 6.5.c na imagem 6.5.k. Utilizando as imagens 6.5.j e 6.5.k como amostras de entrada e de saída, é possível construir um operador em níveis de cinza pelo algoritmo de cortes. Foi utilizada a janela 2×2 e o treinamento levou 50 segundos. Aplicando esse operador na imagem 6.5.i obtivemos a imagem processada 6.5.l. Este processo levou 11 segundos. Convertendo a imagem 6.5.l numa imagem colorida, obtivemos a imagem 6.5.m. O erro médio euclidiano desta imagem é 0,81%, muito inferior ao erro 14,89% obtido sem utilizar a técnica de decomposição em bandas.

Processando a imagem 6.5.a pela decomposição em bandas, obtemos uma saída com erro médio euclidiano 0,14%, bem inferior ao erro 0,98% obtido sem utilizar esta técnica.

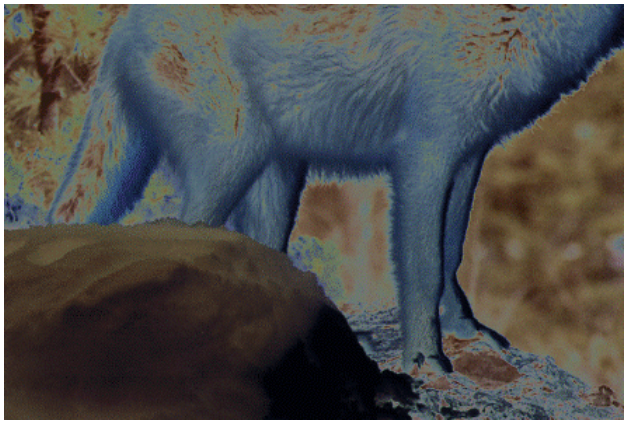
Evidentemente, a decomposição em bandas não pode ser utilizada para emular qualquer filtro digital. Para que esta técnica possa ser utilizada, o nível de saída de cada banda deve depender exclusivamente da mesma banda da imagem de entrada. Isto é, o nível de saída de vermelho deve depender somente dos níveis de vermelho da entrada, o mesmo ocorrendo para verde e azul. Além disso, um mesmo operador deve ser responsável pelo processamento das três bandas.

A próxima subseção mostra um exemplo em que a decomposição em bandas não pode ser utilizada.

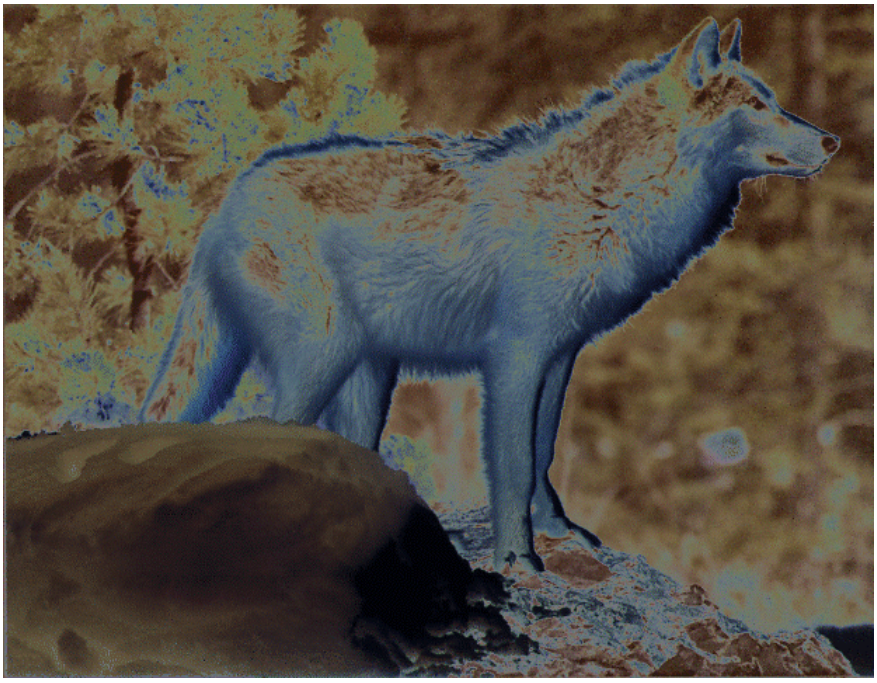
Figura 6.5: Emulação do filtro “Solarize” do Adobe Photoshop

(6.5.a) Imagem a ser processada. 480×621 pixels, 3 bytes por pixel.

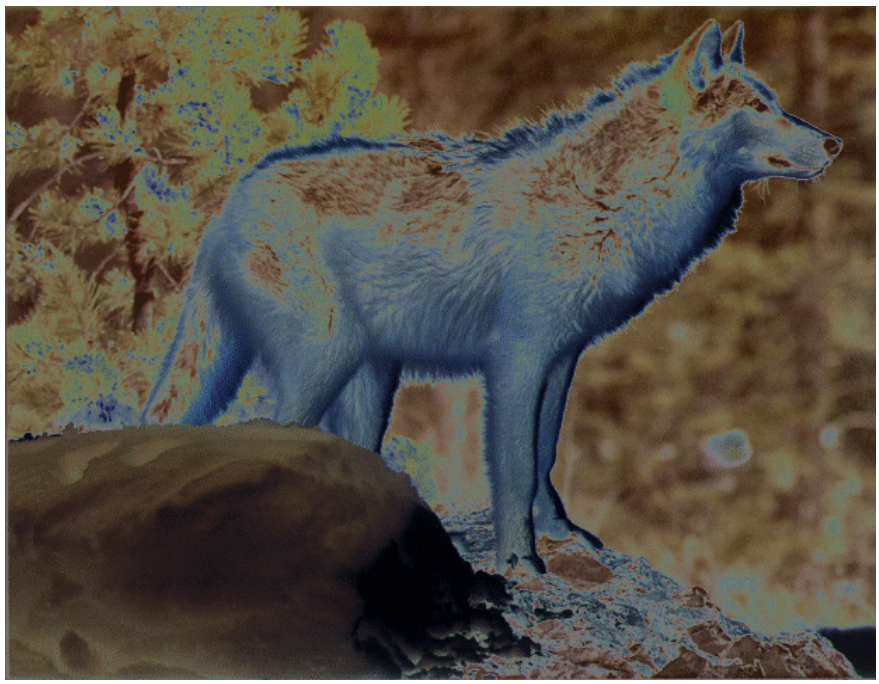
(6.5.b) Amostra de entrada. 297×442 pixels, 3 bytes por pixel.



(6.5.c) Amostra de saída, obtida aplicando “solarize” em 6.5.b.



(6.5.d) Imagem processada pela árvore de cortes. MEE: 0,98%. Janela: 2×2. Treinamento: 23 s. Aplicação: 9 s.

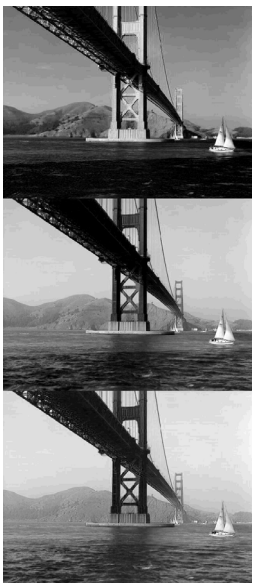


(6.5.e) Saída ideal obtida aplicando o filtro “solarize” na imagem 6.5.a.

(6.5.f) Imagem a ser processada. 480×621 pixels, 3 bytes por pixel.

(6.5.g) Imagem processada utilizando o mesmo operador utilizado para gerar 6.5.d. MEE: 14,89%.

(6.5.h) Saída ideal obtida aplicando o filtro original “solarize” na imagem 6.5.f.



(6.5.i) Decomposição de 6.5.f. Imagem a ser processada.

(6.5.j) Decomposição de 6.5.b. Amostra de entrada.

(6.5.k) Decomposição de 6.5.c. Amostra de saída.

(6.5.l) Imagem processada.

Decompondo as imagens 6.5.f, 6.5.b e 6.5.c em bandas obtemos as imagens em níveis de cinza. Projetando um operador a partir das imagens 6.5.j e 6.5.k (janela 2×2) e aplicando à imagem 6.5.i, obtemos 6.5.l.

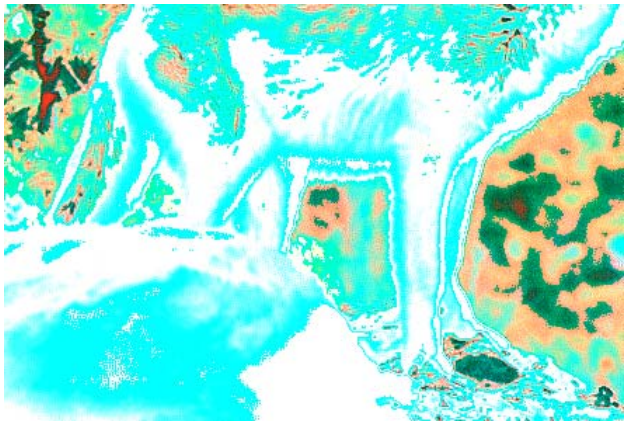
(6.5.m) Imagem recomposta a partir da imagem 6.5.l.
MEE: 0,81%. Janela 2×2.

6.5.2 Filtro “Psychedelic”, Indecomponível em Bandas.

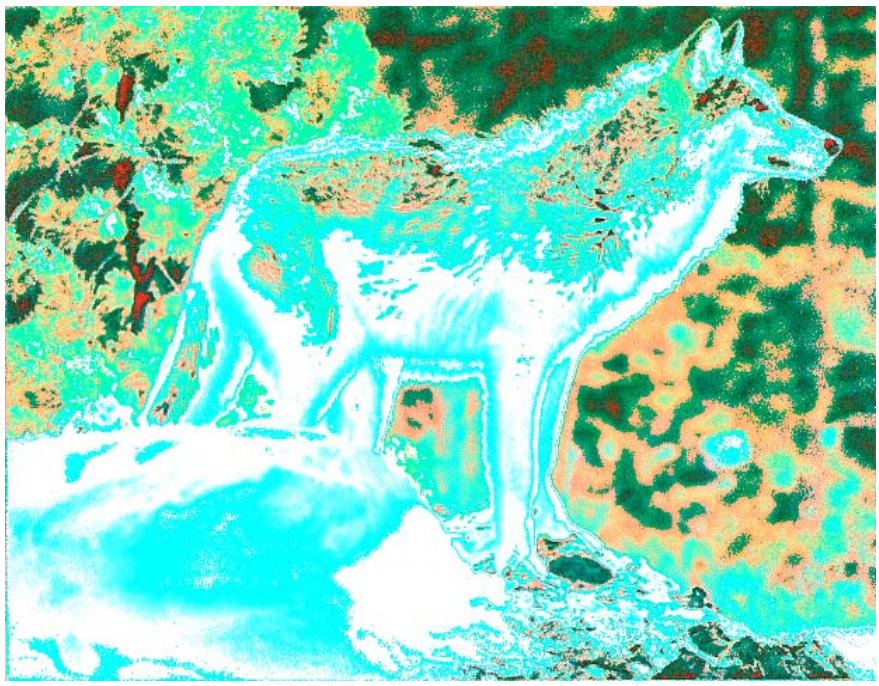
Nesta subseção, iremos emular o filtro “psychedelic” do Corel Photo-Paint 5.00. Processando a imagem 6.5.b com esse filtro obtemos a imagem 6.6.a. O parâmetro “level” controla o comportamento deste filtro e utilizamos o valor 127. A partir destas duas imagens, foi construído um operador morfológico que, processando a imagem 6.5.a fornece a imagem 6.6.b. O erro médio da imagem 6.6.b, em relação à saída ideal 6.6.c é 6,89%. Utilizando a decomposição em bandas, obtemos a imagem 6.6.d cuja qualidade visual é bem inferior à imagem 6.6.b, o que é confirmado pelo erro 19,85%.

Este resultado mostra que no filtro “psychedelic” existe uma interdependência entre as três bandas, isto é, o nível de saída vermelho depende não somente dos níveis de entrada vermelho mas também das intensidades das outras bandas. Isto torna este filtro “indecomponível em bandas”.

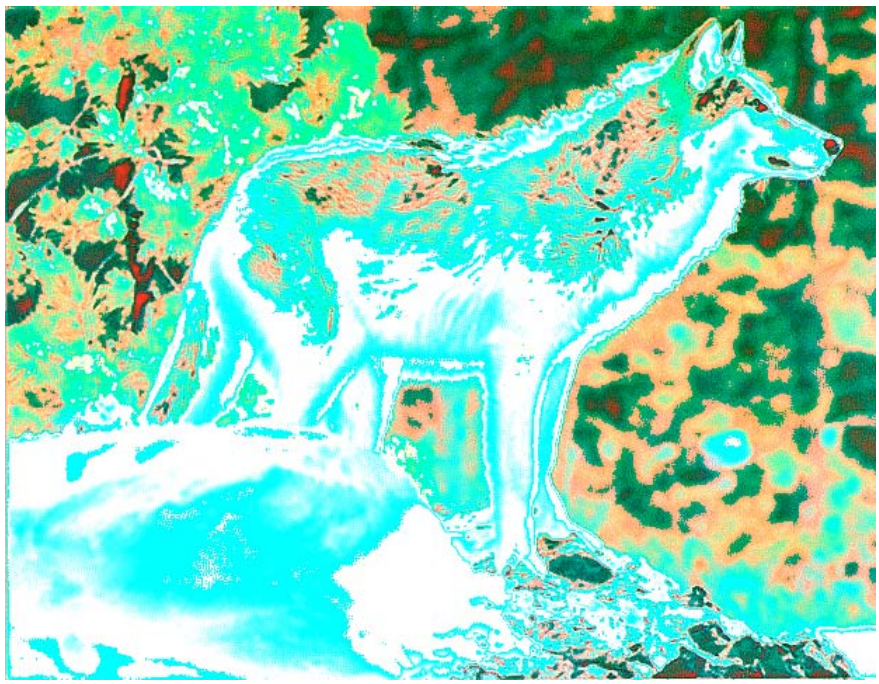
Figura 6.6: Emulação do filtro “Psychedelic” do Corel Photo-Paint



(6.6.a) Amostra de saída



(6.6.b) Imagem processada pelo algoritmo de cortes.
Janela 2×2. MEE: 6,89%.



(6.6.c) Saída ideal.

(6.6.d) Processada pela decomposição em bandas, janela 2×2. Observe que a qualidade desta imagem é bem inferior àquela obtida sem utilizar a decomposição em bandas (6.6.b). MEE: 19,85%.

7 Conclusão

Neste trabalho, apresentamos novas técnicas para projetar operadores morfológicos pela aprendizagem computacional, mais eficientes que aquelas conhecidas na literatura. Também estendemos a possibilidade de uso dessas técnicas de imagens binárias e níveis de cinza para imagens digitais em geral. Apesar de termos implementado estas técnicas somente para imagens binárias, em níveis de cinza e coloridas, a teoria prevê a possibilidade de seu uso para animações, imagens tridimensionais, etc.

Para atingir este objetivo, no capítulo 2 resumimos a morfologia matemática para reticulados e, em especial, descrevemos a teoria de representação de operadores. Introduzimos a representação por intervalos, uma forma alternativa para representar operadores. A teoria da morfologia binária foi apresentada no anexo A.

No capítulo 3, descrevemos o modelo de aprendizagem de Haussler. Introduzimos a função minimizadora de penalidade e, utilizando-a, generalizamos a consistência. Depois demonstramos que todo algoritmo de aprendizagem consistente é PAC. No anexo B, a aprendizagem PAC clássica (modelo de Valiant) foi descrita. No anexo C foi feito um pequeno estudo comparativo da aprendizagem computacional com a teoria do conhecimento humana. No anexo D o problema de aprendizagem do operador foi descrito numa linguagem de teoria da estimação.

No capítulo 4 formalizamos a construção automática de operador como um problema de aprendizagem de Haussler e mostramos que, em certos casos, uma representação por intervalos pode ser armazenada numa árvore de busca binária, o que acelera o processamento computacional. Utilizando esta idéia construímos o algoritmo de cortes. Calculamos a complexidade computacional deste algoritmo concluindo que é

lamos a complexidade computacional deste algoritmo concluindo que é teoricamente muito mais veloz que os algoritmos anteriormente conhecidos. Mostramos algumas aplicações que confirmam o alto desempenho. Apresentamos algumas razões pelas quais a complexidade de amostra real de um problema de aprendizagem do operador é muito menor que a teórica e discutimos o critério para escolha da janela conveniente.

No capítulo 5, utilizamos o algoritmo de busca do vizinho mais próximo como algoritmo de aprendizagem. Distinguimos algoritmo de vizinho mais próximo consistente do não-consistente e demonstramos que somente o primeiro é uma solução PAC do problema de aprendizagem de operador. Utilizamos o algoritmo kd-árvore como algoritmo de busca do vizinho mais próximo. Comprovamos que o algoritmo de cortes é mais rápido que kd-árvore, embora o último forneça normalmente um resultado melhor que o primeiro. O algoritmo kd-árvore foi descrito computacionalmente no anexo E.

No capítulo 6 descrevemos alguns detalhes da implementação dos programas e mostramos diversas aplicações das técnicas estudadas.

A. Morfologia Matemática Binária

Este anexo apresenta resumidamente a morfologia binária, principalmente para os leitores não familiarizados com este assunto. Historicamente, toda a teoria da morfologia começou a partir do tratamento de imagens binárias. Os livros [Matheron, 1975] e [Serra, 1982] contêm a base desta teoria e os seus autores são considerados os fundadores da morfologia matemática. O artigo [Serra, 1986] e [Haralick et al., 1987] e o livro [Dougherty and Astola, 1994] também expõem a teoria da morfologia binária. Algumas aplicações da morfologia binária podem ser encontradas em, por exemplo, [Maragos and Schafer, 1986], [Schmitt, 1989b] e [Pai and Hansen, 1994]. Algoritmos para acelerar o processamento de operadores binários encontram-se em [Vincent, 1990] e [Jones and Svalbe, 1994a].

Uma abordagem completamente diferente para morfologia binária encontra-se nos trabalhos [Ghosh, 1990], [Ghosh, 1991] e [Ghosh, 1993]. Estes trabalhos descrevem os algoritmos fundamentais da morfologia para imagens binárias vetoriais. Uma imagem binária vetorial considera todos os objetos da imagem como poligonais e cada polígono é descrito armazenando as coordenadas dos seus vértices. Uma convenção de ordem dos vértices (por exemplo, anti-horária) indica o interior do polígono.

Para poder definir a imagem binária necessitamos do grupo abeliano. Portanto, vamos relembrar abaixo a definição do grupo abeliano (maiores detalhes veja, por exemplo, [Birkhoff and Bartee, 1970]).

Definição (grupo abeliano): Um conjunto E com uma operação + é um grupo abeliano se para todo $a, b, c \in E$ valem as seguintes propriedades:

1. $a + b = b + a$
2. $(a + b) + c = a + (b + c)$
3. $\exists\, o \in E$ tal que $a + o = o + a = a, \forall a \in E$
4. $\exists\, -a \in E$ tal que $a + (-a) = (-a) + a = o, \forall a \in E$

O elemento o é chamado de *identidade do grupo* ou *elemento neutro* e $-a$ é denominado inverso do elemento a. Costuma-se escrever $a+(-b)$ como $a-b$.

Uma imagem binária é tradicionalmente definida sobre um grupo abeliano como segue:

Definição (imagem binária): Uma *imagem binária* é um subconjunto de um grupo abeliano E, isto é, uma imagem binária A é um elemento de $\mathbb{P}(E)$, o espaço de todos os subconjuntos de E.

Normalmente, trabalha-se com $E=\mathbb{Z}^2$. Também observe que, como $\mathbb{P}(E)$ é isomorfo ao espaço das funções $E \rightarrow \{0,1\}$, podemos definir uma imagem binária como uma função de E em $\{0,1\}$. Uma transformação de uma imagem binária, isto é, um filtro digital sobre uma imagem binária, é denominada de operador.

Definição (operador): Um *operador morfológico binário* ou simplesmente *operador binário* é uma função $\mathbb{P}(E) \rightarrow \mathbb{P}(E)$, onde $\mathbb{P}(E)$ é o espaço de todos os subconjuntos de E.

Definição (translação): Dada uma imagem binária B, denota-se por B_p a *translação* de B por um ponto $p \in E$, isto é $B_p = \{\, b + p \mid b \in B \,\}$.

Definição (soma Minkowski): A *soma de Minkowski* entre duas imagem binárias A e B é:

$$A \oplus B = \{a + b \mid a \in A, b \in B\} = \bigcup_{a \in A} B_a = \bigcup_{b \in B} A_b$$

O operador morfológico denotado $\delta_B(A)$ que leva A em $A \oplus B$ é denominado de *dilatação* binária e a imagem B é denominada elemento estruturante. Segundo [Heijmans and Ronse, 1990], esta operação foi definida originalmente em [Minkowski, 1903]. Segundo [Heijmans and Ronse, 1990], o dual da soma de Minkowski foi definido mais tarde pelo Hadwiger [Hadwiger, 1950] e recebe o nome de *subtração de Minkowski*:

Definição (subtração Minkowski): A *subtração de Minkowski* entre duas imagem binárias A e B é:

$$A \ominus B = \{z \in E \mid B_z \subseteq A\} = \bigcap_{b \in B} A_{-b}$$

O operador morfológico denotado $\varepsilon_B(A)$ que leva A em $A \ominus B$ recebe o nome de *erosão* binária e a imagem B é denominada de elemento estruturante. Na א, temos exemplos de erosão e dilatação binárias.

Definição (crescente): Dizemos que um operador binário ψ é *crescente* se $X \subseteq X' \Rightarrow \psi(X) \subseteq \psi(X')$ $(\forall X, X' \in \mathbb{P}(E))$. Um operador decrescente define-se de forma análoga.

Definição (τ-operador): Dizemos que um operador binário ψ é *invariante por translação* ou τ-*operador* se para todo $X \in \mathbb{P}(E)$ e para todo $p \in E$ vale $\psi(X_p) = (\psi(X))_p$.

Definição (núcleo): Dado um τ-operador ψ, o seu *núcleo* é definido como segue:

$$\mathrm{Ker}(\psi) = \{ B \subset E \mid o \in \psi(B) \}.$$

Matheron em [Matheron, 1975, pp. 218-219] provou o seguinte teorema:

Teorema A.1 (Matheron): Se ψ é um τ-operador crescente então para todo $X \subset E$ temos:

$$\psi(X) = \bigcup_{B \in \mathrm{Ker}(\psi)} X \ominus B$$

Pelo princípio de dualidade, o teorema acima possui um resultado dual que diz que todo τ-operador crescente pode ser representado como uma intersecção de dilatações.

Definição (intervalo): Dado um par de imagens binárias A e B tais que $A \subset B$, o intervalo fechado de imagens com extremidades A e B, denotado por $[A,B]$, é o seguinte conjunto de imagens:

$$[A,B] = \{X \subset E : A \subset X \subset B\}$$

O artigo [Banon and Barrera, 1991] estendeu o teorema de Matheron para operador não necessariamente crescente utilizando como operador elementar o *sup-gerador* definido abaixo:

Definição (sup-gerador): Um sup-gerador binário é caracterizado por um intervalo fechado $[A,B]$ e é definido como:

$$\lambda_{[A,B]}(X) = \{x \in E : A_x \subset X \subset B_x\}, \quad X \subset E$$

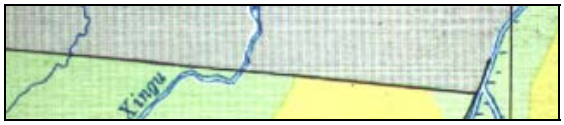
O inf-gerador pode ser definido de forma dual. O trabalho [Banon and Barrera, 1991] demonstra que qualquer τ-operador pode ser representado como uma união de sup-geradores.

Teorema A.2 (sup-gerador): Dado um τ-operador ψ (não necessariamente crescente) vale a seguinte igualdade:

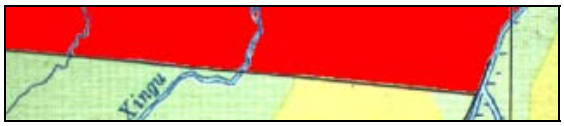
$$\psi(X) = \bigcup \{\lambda_{[A,B]}(X) : [A,B] \subset \mathrm{Ker}(\psi)\}, \quad X \subset E$$

Pelo princípio da dualidade, pode-se mostrar que todo τ-operador pode ser representado como intersecção de inf-geradores.

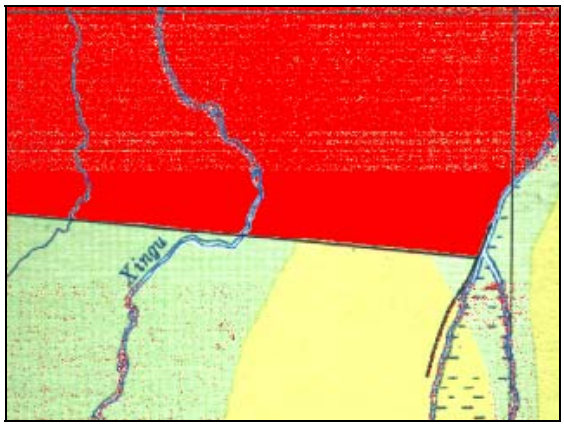
(A.1.a) Imagem original a ser processada.



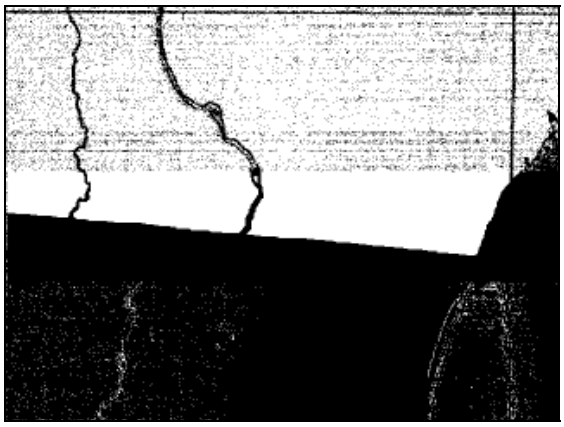
(A.1.b) Exemplo de entrada.



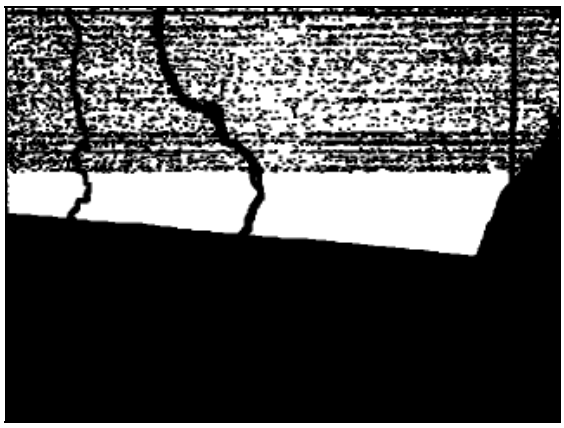
(A.1.c) Exemplo de saída.



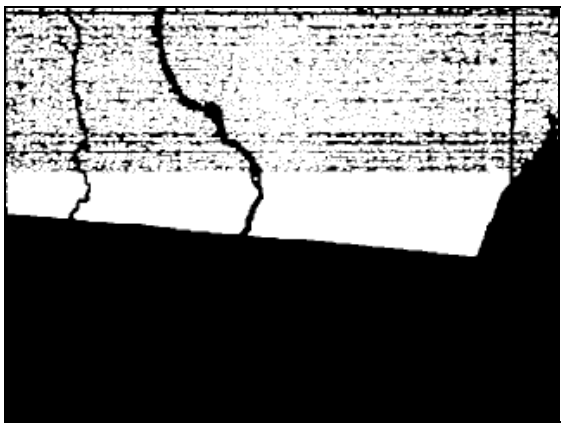
(A.1.d) Imagem processada.



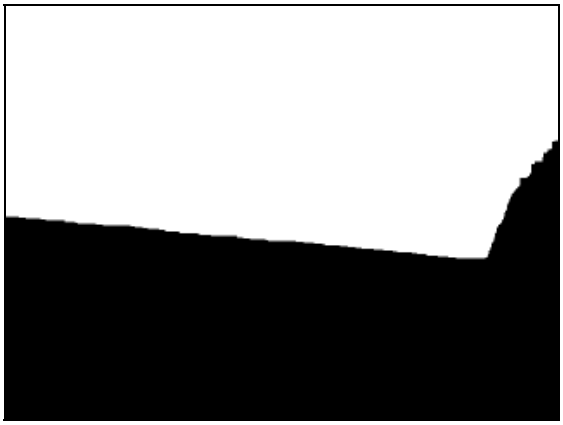
(A.1.e) Região de interesse binário.



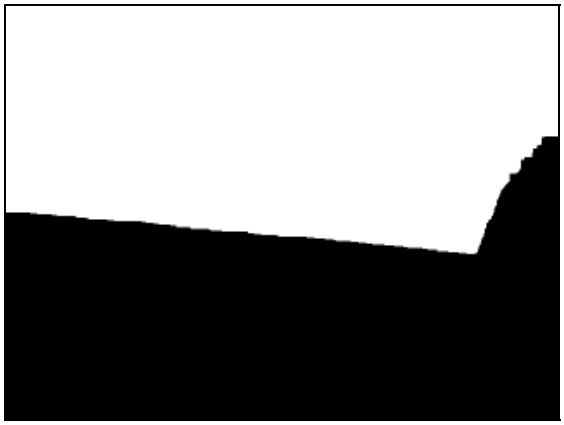
(A.1.f) Erosão 3 × 4.



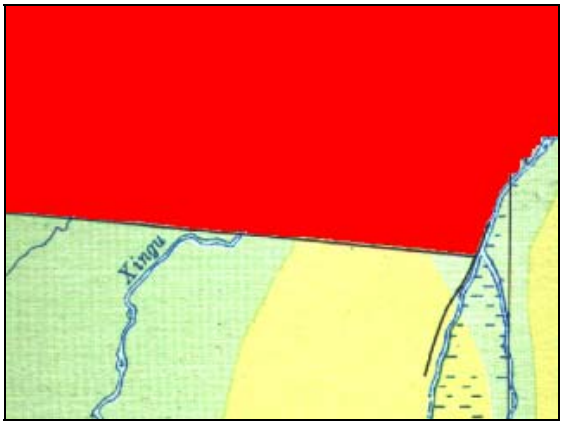
(A.1.g) Dilatação 3 × 4.



(A.1.h) Dilatação 10 × 25.



(A.1.i) Erosão 10 × 25.



(A.1.j) Sobreposição da imagem original e a imagem A.1.i.

Figura A.1: Esta figura ilustra o comportamento da erosão e dilatação binárias. Na figura A.1.a, vemos um mapa do centro-oeste do Brasil digitalizada por um "scanner". Uma sub-imagem deste mapa foi recortada A.1.b e a região cuja textura queremos reconhecer foi pintada de vermelho manualmente A.1.c. Utilizando as imagens A.1.b e A.1.c como amostras de treinamento, um algoritmo de aprendizagem (algoritmo de cortes) criou um operador morfológico para reconhecer a região de interesse. Aplicando esse operador à imagem original A.1.a obtemos a imagem processada A.1.d. A partir desta imagem, foi criada uma imagem binária A.1.e com área branca correspondendo ao região de interesse achada pelo operador. Esta figura é ruidosa e gostaríamos de limpá-la. Aplicando uma erosão binária 3 × 4 em A.1.e, obtemos A.1.f. Aplicar erosão equivale a "espalhar" pontos pretos o que elimina ruídos brancos. Aplicando uma dilatação 3 × 4 na imagem A.1.f, obtemos A.1.g. A dilatação "espalha" os pontos brancos. Aplicando uma dilatação 10 × 25 em A.1.g obtemos A.1.h e com uma erosão 10 × 25 em A.1.h obtemos A.1.i. Compondo a imagem A.1.i e a imagem original A.1.a, obtemos a imagem A.1.j, que reconhece a textura alvo sem ruídos.

B. Aprendizagem PAC Clássica

O modelo de aprendizagem provavelmente aproximadamente correta (PAC), foi descrito originariamente no artigo [Valiant, 1984] e por isso é também conhecido como modelo de Valiant. O livro [Anthony and Biggs, 1992] expõe-no didaticamente. O anexo presente baseia-se neste livro, com algumas alterações para que as notações e definições sejam coerentes com o modelo PAC generalizado que descrevemos no capítulo 3. Este anexo destina-se principalmente àqueles leitores não familiarizados com a teoria da aprendizagem computacional.

Além das obras citadas no parágrafo anterior, o trabalho [Angluin, 1988] descreve a aprendizagem de conceito para responder diferentes tipos de perguntas: pertinência, equivalência, inclusão de um conjunto em outro, verificar se os conjuntos são disjuntos, etc. [Angluin and Laird, 1988] apresenta o modelo de aprendizagem PAC clássico com ruído, isto é, quando podem haver conflitos nos exemplos. Porém, o modelo de Haussler descrito no capítulo 3 é muito mais abrangente e também consegue tratar o problema de conflito. O artigo [Bergadano and Cutello, 1995] utiliza o modelo de aprendizagem de Valiant para construir um sistema de classificação nebuloso.

A aprendizagem computacional tem sido utilizada freqüentemente no reconhecimento de padrões. O trabalho [Saitta and Bergadano, 1993] é um "survey" teórico da aplicação de aprendizagem computacional para essa finalidade. Na morfologia, o trabalho [Barrera et al., 1995] modela a aprendizagem de operadores morfológicos binários utilizando o modelo de Valiant.

Neste anexo, iremos trabalhar somente com conjuntos finitos, pois esta suposição não coloca restrições a nenhuma aplicação prática além de simplificar muito a teoria e as demonstrações de muitos teoremas.

Definição (instância, saída): Seja X um conjunto finito arbitrário que chamaremos de *espaço das instâncias* e seja o conjunto $Y = \{0,1\}$ que denominaremos de *espaço das saídas*. Seus membros são denominados de *instância* e *saída*, respectivamente.

Definição (amostra, exemplo, conceito, conflito): Chamaremos o conjunto $Z = X \times Y$ de *espaço das amostras*. Uma função $c: X \to Y$ é chamado de *conceito* ou *regra de decisão*. Um *exemplo do conceito c* é uma dupla ordenada $(x, y) \in Z$, onde $y = c(x)$. Um exemplo onde $y=1$ é conhecido como um *exemplo positivo*, e um exemplo com $y=0$ é denominado de *exemplo negativo*. Uma *amostra do conceito c* é uma seqüência exemplos, isto é, $\vec{z} = ((x_1 , y_1), (x_2 , y_2), ..., (x_m , y_m))$, onde $y_i = c(x_i)$. Como c é uma função, não podem haver *conflitos* nos exemplos de uma amostra. Dizemos que há *conflito* quando numa amostra $x_i=x_j$ mas $y_i \neq y_j$,.

Definição (distribuição de probabilidade): O espaço das *distribuições de probabilidade* $\mathcal{P}$ é uma família de distribuições de probabilidade em X.

As instâncias de uma amostra são escolhidas aleatoriamente de acordo com uma distribuição de probabilidade $P \in \mathcal{P}$, suposta desconhecida.

Definição (espaço das hipóteses): O *espaço das hipóteses* ou o *espaço das regras de decisão* $\mathcal{H}$ é um subconjunto do espaço das funções $X \to \{0,1\}$.

Definição (conceito alvo, hipótese): Um conceito desconhecido $t \in \mathcal{H}$ a ser aprendido é chamado de *conceito alvo*.

O problema de aprendizagem clássico consiste em achar um conceito $h \in \mathcal{H}$, chamado *hipótese*, que seja uma boa aproximação de *t*.

Definição (algoritmo de aprendizagem): Um algoritmo de aprendizagem é uma função $\mathcal{A} : \bigcup_{m \geq 1} Z^m \rightarrow \mathcal{H}$.

Isto é, um algoritmo de aprendizagem associa, a cada amostra $\vec{z} \in Z^m$ de um conceito alvo $t \in \mathcal{H}$, uma hipótese $h \in \mathcal{H}$. Este processo será denotado por $h = \mathcal{A}(\vec{z})$.

Definição (erro de hipótese): Seja *P* uma distribuição de probabilidade em *X*. Definimos o erro de um hipótese $h \in \mathcal{H}$ em relação a um conceito alvo *t*, como a probabilidade do evento $h(x) \neq t(x)$, isto é, $er_P(h, t) = P\{ x \in X : h(x) \neq t(x) \}$.

Podemos dizer, informalmente, que o objetivo de um problema de aprendizagem clássico é, dadas uma amostra gerada por uma distribuição de probabilidade $P \in \mathcal{P}$ e um conceito alvo $t \in \mathcal{H}$, achar uma hipótese $h \in \mathcal{H}$ tal que $er_P(h, t)$ seja "pequeno".

Definição (problema de aprendizagem): Vamos definir um *problema de aprendizagem clássico* como uma tripla ordenada $(X, \mathcal{H}, \mathcal{P})$. O primeiro componente, *X*, é o espaço das instâncias que é um conjunto finito arbitrário. O segundo, $\mathcal{H}$, é o espaço das regras de decisão, uma família de funções $X \rightarrow \{0,1\}$. O terceiro componente, $\mathcal{P}$, é uma família de distribuições de probabilidades em *X* que governa a geração dos exemplos.

Definição (algoritmo PAC): Dizemos que um algoritmo de aprendizagem $\mathcal{A}$: $\bigcup_{m \geq 1} Z^m \rightarrow \mathcal{H}$ é uma solução *provavelmente aproximadamente correta* (abreviada como *PAC*) de um problema de aprendizagem clássico $(X, \mathcal{H}, \mathcal{P})$ sse, dados dois números reais ε e δ ($0 < \varepsilon, \delta < 1$) existe um tamanho finito de amostra $m_0 = m_0(\varepsilon, \delta)$ tal que para toda distribuição de probabilidade $P \in \mathcal{P}$ e para todo conceito alvo $t \in \mathcal{H}$, sempre que tivermos $m \geq m_0$,

$$P^m \{ \vec{z} \in Z^m : er_P(\mathcal{A}(\vec{z}), t) < \varepsilon \} > 1 - \delta$$

A função $m_0(\varepsilon, \delta)$ é chamada de *complexidade de amostra*.

Definição (algoritmo consistente): Dizemos que um algoritmo de aprendizagem $\mathcal{A}$: $\bigcup_{m \geq 1} Z^m \rightarrow \mathcal{H}$ é consistente com um problema de aprendizagem clássico (X, $\mathcal{H}$, $\mathcal{P}$) sse, para qualquer conceito alvo $t \in \mathcal{H}$ e para amostra de qualquer tamanho $\vec{z}$, temos:

$$(x, y) \in \vec{z} \Rightarrow \mathcal{A}(\vec{z})(x) = y.$$

Os teoremas 4.1.1 e 4.2.1 do [Anthony and Biggs, 1992] juntos demonstram o enunciado abaixo. Unimos essas duas demonstrações num único teorema.

Teorema B.1 (consistência): Sejam (X, $\mathcal{H}$, $\mathcal{P}$) um problema de aprendizagem clássico e $\mathcal{A}$ um algoritmo consistente. Se $\mathcal{H}$ é finito, então $\mathcal{A}$ é PAC.

Prova: Pela definição, $\mathcal{A}$ é PAC sse dados $0 < \varepsilon, \delta < 1$, $\forall t \in \mathcal{H}$ e $\forall P \in \mathcal{P}$, existir $m_0 = m_0(\varepsilon, \delta)$ tal que $\forall m \geq m_0$,

$$P^m \{ \vec{z} \in Z^m : er_P(\mathcal{A}(\vec{z}), t) < \varepsilon \} > 1 - \delta$$

Considere, portanto, dados ε, δ, t, P e um algoritmo consistente $\mathcal{A}$. Seja $h \in \mathcal{H}$ tal que:

$$er_P(h, t) = P\{ x \in X : h(x) \neq t(x) \} \geq \varepsilon$$

Então, $P\{x \in X : h(x) = t(x)\} < 1-\varepsilon$. Portanto, $P^m\{ \vec{x} \in X^m : h(x_i) = t(x_i),\ 1 \leq i \leq m \} < (1-\varepsilon)^m$. Esta é a probabilidade de que uma regra de decisão h que tem a propriedade $er_P(h,t) \geq \varepsilon$ seja consistente com uma amostra de tamanho m. Portanto, a probabilidade de que exista pelo menos uma regra de decisão $h \in \mathcal{H}$, que satisfaz $er_P(h,t) \geq \varepsilon$ e que simultaneamente seja consistente com uma amostra de m exemplos é menor que $|\mathcal{H}|(1-\varepsilon)^m$. Ora, $|\mathcal{H}|(1-\varepsilon)^m \leq \delta$ quando:

$$m \geq m_0 = \left\lceil \frac{1}{\varepsilon} \ln\left(\frac{|\mathcal{H}|}{\delta} \right) \right\rceil$$

Pois nesse caso,

$$|\mathcal{H}|(1-\varepsilon)^m \quad < \quad |\mathcal{H}|\exp(-\varepsilon m) \quad \leq \quad |\mathcal{H}|\exp\left(\ln\left(\frac{\delta}{|\mathcal{H}|}\right)\right) \quad = \quad \delta$$

Logo, quando m satisfaz a desigualdade acima, $P^m\{\vec{z} \in Z^m : er_P(\mathcal{A}(\vec{z}), t) < \varepsilon\} > 1 - \delta$. ■

Como neste trabalho estamos lidando somente com conjuntos finitos, um algoritmo consistente será sempre PAC.

Corolário (complexidade): A complexidade de amostra de um algoritmo consistente com um problema de aprendizagem clássico com $\mathcal{H}$ finito é:

$$m_0(\varepsilon, \delta) = \left\lceil \frac{1}{\varepsilon} \ln\left(\frac{|\mathcal{H}|}{\delta}\right) \right\rceil.$$

Prova: É uma conseqüência direta da demonstração do Teorema acima. ■

C. Aprendizagem Computacional e Humana

Apesar deste trabalho não ter a pretensão de realizar um estudo filosófico profundo sobre as diferenças e as semelhanças entre a aprendizagem computacional e humana, julgamos interessante escrever algumas considerações a este respeito, pois este questionamento surge naturalmente ao estudarmos a aprendizagem computacional.

A aprendizagem computacional está matemática e estatisticamente definida nos trabalhos que citamos no capítulo 3 e no anexo B, o que permite conhecer exatamente o seu mecanismo de funcionamento. Porém, não existe uma opinião unânime sobre o mecanismo de funcionamento do processo de aprendizagem humana. Ao compararmos a aprendizagem humana com a computacional, é possível assumir duas atitudes: ou pensar que os dois mecanismos são essencialmente diferentes apesar de apresentarem algumas semelhanças ou pensar que no fundo são a mesma coisa.

A primeira corrente é representada pela filosofia clássica ocidental que se encontra sintetizada na obra "Suma Teológica" de Tomás de Aquino. Uma edição em português desta obra é [Aquino, 1980]. Adotando as idéias da filosofia clássica, chegaremos necessariamente à conclusão de que a aprendizagem humana e computacional são radicalmente diferentes, apesar das semelhanças aparentes.

A segunda corrente de pensamento é defendida, por exemplo, no artigo [Valiant, 1984] que introduziu a aprendizagem computacional. Este artigo, apesar de ter caráter matemático, também faz incursões no terreno gnoseológico, ao procurar tirar conclusões filosóficas a partir dos resultados matemáticos que obteve.

Como Tomás de Aquino pode ser desconhecido para muitos leitores da área de exatas, transcrevemos abaixo, da apresentação da obra [Aquino, 1980], o contexto histórico e cultural desse filósofo: "No auge do desenvolvimento intelectual do século XIII, no maior centro cultural da Idade Média, em Paris, Tomás de Aquino iniciou suas atividades docentes, atividades estas que se desenvolveram posteriormente também em Colônia e na Itália. (...) Em seus estudos universitários em Nápoles, Paris e Colônia, entrou em contato com a tradição cristã. Defrontou-se, também, com os clássicos da antigüidade, principalmente obras de Platão e Aristóteles. Foi-lhe importante igualmente o confronto com o pensamento árabe-judaico que, através da Espanha, havia atingido o Ocidente. Todo esse material, que desde o início do século XIII vinha sendo assimilado pelos mestres universitários, encontraria em Tomás quem melhor o sintetizou." Na teoria do conhecimento, também chamada gnoseologia, Tomás segue fielmente Aristóteles, segundo reconhece a apresentação de [Aquino, 1980]: "Não se limitou ele a citar Aristóteles, mas reconhecendo-lhe superioridade da síntese filosófica sobre a então dominante, e percebendo a irredutibilidade entre ambas, desfez-se dos esquemas gnoseológicos e metafísicos neoplatônicos" para adotar o pensamento aristotélico.

Para compararmos os dois processos de aprendizagem considere o reconhecimento de um objeto, por exemplo, "cadeira". Para ensinar o conceito "cadeira" a um computador (utilizando o modelo de Valiant, de Haussler ou de vizinho mais próximo) é necessário fornecer-lhe uma certa quantidade de imagens de objetos, cada qual classificada corretamente (ou com uma taxa de acerto pelo menos maior que 50%, dependendo do modelo adotado) como "cadeira" ou "não-cadeira". Quando um novo objeto é mostrado ao computador assim treinado, ele o classificará como cadeira ou não, buscando na memória um objeto semelhante, para dar ao objeto desconhecido a mesma classificação. Quando uma criança aprende a reconhecer uma cadeira, aparentemente ela procede do mesmo modo que o computador.

As duas aprendizagens possuem semelhanças, pois os dois processos têm início nos sentidos. No caso do computador, o processo de aprendizagem parte das imagens

captadas por uma câmara digitalizadora ou um "scanner". Também no homem, o processo de conhecimento parte dos sentidos, como diz Agostinho citado em [Aquino, 1980, parte I, questão 84, artigo2]: "A mente conhece as coisas corpóreas pelos sentidos do corpo". Aristóteles, citado em [Aquino, 1980, parte I, questão 84, artigo3] diz: "O intelecto é como uma tábua na qual nada está escrito." Isto é, se possuímos algum conhecimento, não é porque nascemos com ele, nem porque esse conhecimento foi implantado na nossa mente de alguma forma, mas porque o aprendemos a partir das informações fornecidas pelos sentidos. Tomás, em [Aquino, 1980, parte I, questão 84, artigo 6], explica o funcionamento dos sentidos do seguinte modo: "Aristóteles concorda com Demócrito em que as operações da parte sensitiva são causadas pela impressão dos sensíveis no sentido (...) pois Demócrito também ensinava que todo sentir se dá por influição dos átomos". Em linguagem moderna diríamos que vemos porque os fótons chegam aos nossos olhos ou que ouvimos porque as ondas sonoras chegam aos nossos ouvidos.

Tanto o computador como a criança são capazes de abstrair algumas características simples da imagem do objeto apresentado, como a cor, o número de pernas, a textura, o tamanho, se possui ou não encosto, se é estofado, etc. Também ambos são capazes de memorizar as imagens dos objetos, juntamente com as características que conseguiram abstrair das imagens, e a sua classificação (é ou não uma cadeira).

Porém, após alguns exemplos, a criança entende que se chama de cadeira um móvel projetado para sentar em cima, mesmo que o seu aspecto externo, a cor, o material possa variar imensamente de uma para outra. Ela consegue fazer esta abstração mesmo que o seu conhecimento de linguagem não lhe permita expressar o que aprendeu com palavras. O computador, pelo menos nos modelos de aprendizagem que estamos considerando, é incapaz de fazer esta abstração, por si e a partir dos exemplos. Uma vez captada a essência do conceito, a criança é capaz de classificar facilmente um objeto desconhecido em cadeira ou não-cadeira, verificando se ele foi projetado para sentar ou não. Podemos mostrar cadeiras com três, cinco, seis pernas, com ou sem braço, com ou sem encosto, em forma de coelho, de elefante, etc. e a criança as classificará corretamente,

mesmo sem nunca ter visto anteriormente exemplos semelhantes. Por outro lado, se apresentarmos ao computador uma cadeira em forma de coelho, que não faz parte da sua base de conhecimento, ele tentará classificá-lo baseado unicamente na semelhança externa deste com os exemplos previamente conhecidos (nos modelos considerados), o que possivelmente levaria a uma classificação incorreta.

O artigo [Valiant, 1984] tem uma opinião diferente para a aprendizagem do conceito, pois este afirma: "Considere um mundo contendo robôs e elefantes. Suponha que um dos robôs tenha descoberto um algoritmo de reconhecimento para elefantes (...). Nosso teorema implica que este robô pode comunicar seu algoritmo ao resto da população de robôs simplesmente exclamando 'elefante' toda vez que um elefante aparecer." Também afirma: "Um aspecto importante da nossa abordagem (...) é que nós exigimos que o professor e aprendiz concordem somente numa pequena fração de exemplos naturais. O comportamento deles para exemplos não-naturais é irrelevante, e portanto a descrição de todos possíveis mundos não é necessária". Portanto, para Valiant, uma cadeira em forma de coelho seria um exemplo não-natural e portanto irrelevante. Valiant completa o seu raciocínio afirmando: "Se levarmos avante as conclusões, esta idéia tem consideráveis implicações filosóficas: Um conceito aprendível é nada mais que um pequeno programa que distingue algum exemplo natural de alguns outros. Se tal conceito é difundido numa população, variações substanciais no significado podem aparecer. Mais importante, o consenso irá existir somente para entradas naturais. O comportamento de um programa individual para exemplos não naturais não tem relevância".

Na realidade, a capacidade de abstração do ser humano vai muito além de descobrir a funcionalidade de um objeto (que a cadeira é um objeto feito para sentar), o que o computador já é incapaz de fazer por si, pelo menos nos modelos de aprendizagem considerados. O ser humano é capaz de aprender conceitos abstratos, é capaz de abstrair leis universais a partir das observações, é capaz de inventar algoritmos inéditos, é capaz de formular e demonstrar teoremas, etc. Tomás de Aquino em [Aquino, 1980, parte I, questão 84, artigo 6] explica esta capacidade humana de abstração dizendo: "Quanto ao intelecto, Aristóteles ensina que (...) aquele agente mais nobre e superior, a que chamou

intelecto agente, torna os sensíveis recebidos nos sentidos em inteligíveis atuais, por meio da abstração", em outras palavras, existe uma faculdade humana que, partindo de uma série de exemplos de cadeiras, é capaz de abstrair o que realmente significa o conceito cadeira ou permite abstrair a lei da gravidade a partir da trajetória da lua e dos planetas. Depois acrescenta: "Como não bastam os sensíveis para inteligir, mas é necessário o intelecto agente, não se pode dizer que o conhecimento sensível seja a causa perfeita e total do conhecimento, mas a sua matéria causadora." Tirando conclusões a partir das idéias da filosofia clássica, a aprendizagem computacional nunca poderia ultrapassar o conhecimento sensível, pois falta ao computador o intelecto agente, a capacidade para extrair o inteligível do sensível. O computador não conhece o conceito universal "cadeira" mas apenas uma quantidade enorme de objetos singulares classificados como "cadeira" ou "não-cadeira". Aristóteles, citado em [Aquino, 1980, parte I, questão 86, artigo 1] diz: "O universal é conhecido pela razão e o singular pelo sentido." E conhecer uma quantidade enorme de singulares nunca é conhecer o universal.

Um ser humano consegue aprender um novo conceito a partir de um número relativamente pequeno de exemplos, devido à sua capacidade de abstração. Por outro lado, a impossibilidade do computador captar o conceito universal faz com que haja a necessidade de treiná-lo fornecendo uma quantidade imensa de exemplos para atingir uma taxa de erro tolerável, nos modelos de aprendizagem considerados. Muitas vezes, esta quantidade está muito além do realizável na prática. Apesar desta dificuldade, a aprendizagem computacional tem sido utilizada com sucesso em diversas áreas, como no reconhecimento ótico de caracteres (OCR), reconhecimento de voz, reconhecimento de padrões, etc.

Na opinião pessoal do autor deste trabalho, as idéias da filosofia clássica sobre a aprendizagem são mais plausíveis que aquelas que consideram que a aprendizagem humana e computacional são no fundo a mesma coisa.

D. Teoria da Estimação e Aprendizagem de Operador

Neste anexo, colocamos o problema de aprendizagem de operador no contexto da *teoria da estimação* ([Spiegel, 1974], [Spiegel, 1979], [Sage and Melsa, 1971], [Sorenson, 1980], [Dudewicz and Mishra, 1988]). Os artigos [Dougherty, 1992a] e [Dougherty, 1992b] utilizam a teoria da estimação para estabelecer uma metodologia para o projeto de operadores ótimos binários e em níveis de cinza, respectivamente.

Para que as analogias entre a aprendizagem computacional e a teoria da estimação fiquem mais claras, utilizamos as mesmas notações utilizadas no resto do trabalho, que não são as mais usuais na teoria da estimação. Denotaremos as variáveis aleatórias em negrito não-itálico.

A *teoria da amostragem* estuda as relações existentes entre uma *população* e as *amostras* dela extraídas. A *inferência estatística* tem por objetivo fazer generalizações sobre uma população com base em dados de uma amostra. Os dois problemas básicos neste processo são a *estimação de parâmetros* e a *teste de hipóteses* sobre parâmetros. A estimação de parâmetros procura deduzir parâmetros populacionais da estatística amostral correspondente. A estimação de parâmetros pode ser subdividida em *estimação por intervalo* e *estimação pontual*.

A estimação por intervalo expressa a estimativa de um parâmetro por um intervalo no qual o parâmetro deve estar situado. Uma indicação da precisão dessa estimativa cha-

ma-se *confiabilidade*. Esta técnica é amplamente utilizada, por exemplo, nas pesquisas eleitorais, pesquisa de audiência de televisão, etc.

A situação básica da estimação pontual é a seguinte. Observamos variáveis aleatórias $\mathbf{z}_1$, ..., $\mathbf{z}_m$. A função de distribuição de $\mathbf{z}_1$, ..., $\mathbf{z}_m$ depende de um parâmetro desconhecido h^* (por exemplo, a média populacional de uma distribuição normal) que conhecemos pertencer a um conjunto $\mathcal{H}$. Estimamos h^* por uma função apropriada $\mathcal{A}(\mathbf{z}_1, ..., \mathbf{z}_m)$. Tal função é denominada *estatística* ou *estimador*. Como $\mathbf{z}_1$, ..., $\mathbf{z}_m$ são variáveis aleatórias, $\mathcal{A}(\mathbf{z}_1, ..., \mathbf{z}_m)$ também é uma variável aleatória. Um valor particular do estimador, digamos $h=\mathcal{A}(z_1, ..., z_m)$ é chamado estimativa de h^*. Dizemos que $\mathcal{A}(\mathbf{z}_1, ..., \mathbf{z}_m)$ é um estimador consistente de h^* se

$$\lim_{m\to\infty}\left[P\left(\left|\mathcal{A}\left(\mathbf{z}_1,\ldots,\mathbf{z}_m\right) - h^*\right| \geq \varepsilon\right)\right] = 0\,.$$

Para colocar a aprendizagem de operador restrito à janela no contexto de teoria da estimação, não podemos mais trabalhar com variáveis aleatórias simples, como num caso em que se deseja estimar a média, a variância, etc. As observações $\mathbf{z}_i$ serão variáveis aleatórias vetoriais e o parâmetro a ser estimado h^* será uma função. Considere os conjuntos $X=\mathbb{R}^d$, $Y=\mathbb{R}^b$ e $Z=X\times Y$. Suponha que exista uma distribuição de probabilidade conjunta P em $Z=X\times Y$ e seja $(\mathbf{x}, \mathbf{y})$ o par de variáveis aleatórias vetoriais cuja função densidade de probabilidade conjunta é P.

Considere uma função l: $Y \times Y \to [0, M]$, onde M é um número real positivo, que denominaremos de penalidade. Também considere um conjunto $\mathcal{H}$, o espaço das regras de decisão, um subconjunto do espaço de funções $X \to Y$. O risco de uma regra de decisão h é definida como a sua esperança de penalidade:

$$r_h(P) = E\left(l\left(\mathbf{y}, h(\mathbf{x})\right)\right) = \sum_{(x,y)\in Z} l(y, h(x))\; P(x,y)\,.$$

A regra de decisão ótima, denotada por h^*, é o elemento de $\mathcal{H}$ que minimiza o risco, isto é:

$$r_{h^*}(P) = \wedge\left\{r_h(P) : h \in \mathcal{H}\right\} = E\left(l\left(\mathbf{y}, h^*(\mathbf{x})\right)\right).$$

O parâmetro a ser estimado, no problema de aprendizagem de operador, é a regra de decisão ótima.

O conceito de algoritmo de aprendizagem PAC equivale à noção do estimador consistente no caso presente. Como vimos no capítulo 3, um algoritmo de aprendizagem é PAC se:

$$\lim_{m\to\infty} P^m\left\{\vec{z} \in Z^m : \left|r_{\mathcal{A}(\vec{z})}(P) - r_{h^*}(P)\right| \geq \varepsilon\right\} = 0.$$

Na teoria da estimação, diríamos que o estimador $\mathcal{A}$ de h^* é consistente. Por simplicidade estamos considerando a métrica *d* como o módulo da diferença.

Para seguir adiante, adotaremos uma penalidade concreta, o erro quadrático, muito utilizado na teoria da estimação: $l(\, y_1, y_2\,) = \left(\left\|y_1 - y_2\right\|_2\right)^2$. Com o que expusemos até agora, surgem dois problemas:

1. Quem é a regra de decisão ótima h^*?
2. Como construir o estimador consistente $\mathcal{A}$ do parâmetro h^*?

A solução do primeiro problema equivale a conhecer o valor de $h^*(x)$ para cada $x \in X$. Portanto, considere um $x \in X$ dado e seja (**x**, **y**) o par de variáveis aleatórias cuja função densidade de probabilidade conjunta é *P*. Então, $h^*(x)$ deve minimizar o risco condicional da regra de decisão *h*:

$$\begin{aligned} r_h\left(P \mid \mathbf{x} = x\right) &= E\left(\left(\left\|h(x) - \mathbf{y}\right\|_2\right)^2 \mid \mathbf{x} = x\right) \\ &= \sum_{y \in Y}\left(\left\|h(x) - \mathbf{y}\right\|_2\right)^2 P\left(\mathbf{y} = y \mid \mathbf{x} = x\right) \end{aligned}$$

Afirmamos que $E\left(\mathbf{y} \mid \mathbf{x} = x\right)$ minimiza o risco condicional acima. Esta afirmação pode ser demonstrada fazendo pequenas alterações no exemplo média aritmética vetorial do capítulo 3. Assim, o estimador ótimo pode ser construído, para cada $x \in X$, fazendo:

$$h^*(x) = E\left(\mathbf{y} \mid \mathbf{x} = x\right) = \sum_{y \in Y} y \; P\left(\mathbf{y} = y \mid \mathbf{x} = x\right).$$

Na prática, não se conhece a probabilidade $P\left(\mathbf{y} = y \mid \mathbf{x} = x\right)$, o que impossibilita calcular $h^*(x)$. Assim, necessitamos de um estimador do parâmetro h^*. É muito conhecido que a média amostral é um estimador consistente da média populacional (veja, por exemplo, [Spiegel, 1979, pg. 275]) e portanto,

$$h(x) \; = \text{média}\left(C_{\vec{z}}^{y}(x)\right)$$

é um estimador consistente de $h^*(x)$. Onde, $\vec{z}$ = ((x_1, y_1), (x_2, y_2), ..., (x_m, y_m)) é a seqüência de *m* realizações da variável aleatória (**x**, **y**), $C_{\vec{z}}(x)$ é a subseqüência ((x_1', y_1'), (x_2', y_2'), ..., (x_n', y_n')) de $\vec{z}$ composta por todos os elementos $(x_i, y_i) \in \vec{z}$ tais que $x_i = x$ e $C_{\vec{z}}^{y}(x)$ é (y_1', y_2', ..., y_n'). Portanto, o estimador $\mathcal{A}$ de h^* deve ser definido como segue:

$$\mathcal{A}(\vec{z}\,)(x) = \text{média}\left(C_{\vec{z}}^{y}(x)\right)$$

Assim, chegamos ao estimador consistente da regra de decisão ótima.

E. Kd-Árvore

Neste anexo, descrevemos algoritmicamente a construção da kd-árvore e a busca nessa estrutura. Como dissemos no capítulo 5, a kd-árvore é utilizada para acelerar o problema de busca dos κ vizinhos mais próximos, que foi enunciado como segue: "Dados *m* pontos de treinamento, cada um composto por d números, queremos fazer um pré-processamento de forma que, dados outros *n* pontos de dimensão d seja possível localizar rapidamente, para cada novo ponto, os κ pontos mais próximos dentre os pontos de treinamento."

A descrição do algoritmo segue a do artigo [Friedman et al., 1977]. Adaptamos somente a notação e a nomenclatura para serem coerentes com o resto deste trabalho. Observe que uma kd-árvore é praticamente idêntica a uma árvore de cortes, tanto na estrutura de dados quanto no algoritmo de construção. A diferença maior consiste no algoritmo de busca. Descreveremos o algoritmo supondo que os pontos são vetores de números inteiros (o artigo original supõe que os pontos são vetores de números reais).

A kd-árvore é uma generalização da árvore binária simples utilizada na ordenação e busca. Uma kd-árvore é uma árvore binária na qual cada nó representa um subconjunto dos *m* pontos de treinamento dados. Cada ponto de treinamento é um vetor inteiro de dimensão d. Para que o algoritmo possa ser utilizado para gerar o filtro de processamento de imagem, junto com cada ponto armazenamos o valor de saída, isto é, um vetor inteiro de dimensão b.

A raiz da árvore representa o conjunto de todos os pontos de treinamento. Cada nó interno possui dois filhos. Estes dois filhos representam dois subconjuntos de pontos definidos pela partição. As folhas representam pequenos subconjuntos de pontos mutuamente exclusivos, que coletivamente formam uma partição do espaço das instâncias.
No caso da busca unidimensional, um ponto é representado por um número inteiro e uma partição é definida por um valor inteiro. Todos os pontos com valor menor que o valor da partição pertencem ao filho esquerdo, enquanto que aqueles com um valor maior ou igual pertencem ao filho direito.

No caso de dimensão $d > 1$, um ponto é representado por um vetor com d números inteiros. Qualquer um desses números pode servir como chave para particionar o subconjunto representado por um nó particular da árvore. Isto é, a chave da partição pode ir de 1 a d.

O algoritmo de busca é mais facilmente descrito como um procedimento recursivo. O argumento do procedimento é o nó sob investigação. A primeira chamada passa a raiz da árvore como o seu argumento. O domínio daquele nó, isto é, os limites geométricos que delimitam o subconjunto de pontos representado pelo nó, é disponível como uma variável global. O domínio do nó raiz é definido como mais e menos infinito para todas coordenadas. Os limites geométricos de um nó são determinados pelas partições definidas nos nós superiores. Em cada nó, a partição não somente divide o subconjunto corrente de pontos, mas também define um limite inferior e superior para os pontos dos dois subconjuntos. Estes limites definem um hiper-paralelepípedo no espaço das instâncias. O volume deste hiper-paralelepípedo é menor para subconjuntos definidos pelos nós mais profundos da árvore. Se o nó sob investigação é terminal, então todos os pontos do nó são investigados. A lista de κ pontos mais próximos encontrados até então, juntamente com as distâncias do ponto de busca, é sempre mantida como uma fila de prioridade durante a busca. Toda vez que um ponto é examinado, se este é mais próximo que o membro mais distante da lista de prioridade, a lista é atualizada.

Se o nó sob investigação é não-terminal, o procedimento recursivo é chamado para nós representando os subconjuntos no mesmo lado da partição que o ponto de busca. Quando o controle retorna, é feito um teste para determinar se é necessário considerar os pontos no lado da partição oposta ao ponto de busca. É necessário considerar esse subconjunto somente se os limites geométricos que delimitam esses pontos sobrepõem a bola centrada no ponto de busca com raio igual à distância do κ-ésimo ponto encontrado até então. Isto é referido como teste "bounds-overlap-ball". Se este teste falha, então nenhum dos pontos no lado oposto da partição pode estar entre os κ pontos mais próximos ao ponto de busca. Se os limites sobrepõem a bola, então os pontos do lado oposto devem ser considerados e o procedimento é chamado recursivamente para o nó oposto. O teste chamado "ball-within-bounds" é feito antes de retornar para determinar se é necessário continuar a busca. Este teste determina se a bola está inteiramente dentro do domínio geométrico do nó. Neste caso, a lista corrente de κ pontos mais próximos é correto para conjunto inteiro de pontos e não é necessário examinar mais pontos.

O artigo [Friedman et al., 1977] faz algumas recomendações para obter a kd-árvore otimizada. Primeiro, pede que a partição seja feita na mediana da distribuição marginal do valor da chave. Segundo, pede que escolha a coordenada onde os pontos estão mais espalhados como chave da partição.

A alteração da estrutura de dados em relação ao algoritmo de cortes se dá principalmente nos nós terminais. Um nó terminal irá armazenar `pf` exemplos de treinamento (`pf` é abreviação de Pontos por Folha). Ao contrário da árvore de cortes, onde não havia necessidade de armazenar explicitamente o valor de espaço das instâncias, aqui os exemplos devem ser realmente armazenados, pois caso contrário não é possível calcular a distância real entre duas instâncias. Assim, a estrutura de dados torna-se:

```
1.   type
2.     NodePt = ^Node;
3.     Node = record
4.       case leaf:Boolean of
5.         false:(
6.           c:integer; {valor de corte, número no intervalo [0..255]}
7.           p:integer; {plano de corte, número no intervalo [1..d]}
```

```
8.          left:NodePt; {apontador para filho à esquerda}
9.          right:NodePt; {apontador para filho à direita}
10.       );
11.     true:(
12.       x:array [1..pf] of Kd; {vetor de instâncias}
13.       y:array [1..pf] of Kb; {valores de saída}
14.     );
15.   end;
```

A função `KDConstroi` abaixo é praticamente idêntico à função `CortesConstroi`, visto no capítulo 4.

```
1.   function KDConstroi(
2.     var a:array [1..m] of exemplo;
3.     l,r:integer): NodePt;
4.   var t,tl,tr:NodePt; j,k,p,c:integer;
5.   begin
6.     if (r-l+1<=pf) then begin
7.       MakeTerminal(a,l,r);
8.     end else begin
9.       p:=MaxEspalhamento(a,l,r);
10.      particiona(a,l,r,p,k,c);
11.      tl:=KDConstroi(a,l,k);
12.      tr:=KDConstroi(a,k+1,r);
13.      new(t); t^.leaf:=false; t^.p:=p; t^.c:=c;
14.      t^.left:=tl; t^.right:=tr;
15.    end;
16.    KDConstroi:=t;
17.  end;
```

As funções `MaxEspalhamento` e `particiona` são iguais àquelas para construção de árvore de cortes e a função `MakeTerminal` cria uma folha com os exemplos `a[l]` a `a[r]`.

O procedimento para buscar os vizinhos mais próximos de um ponto `x` numa kd-árvore é mais complexo que a busca numa árvore de cortes. As seguintes variáveis globais serão utilizadas:

```
1.   Global
2.     x:Kd;
3.     PQD:array [1..capa] of real;
4.     PQR:array [1..capa] of exemplo;
5.     bl:Kd;
6.     bu:Kd;
```

No caso, `x` é o ponto do qual queremos achar κ (`capa`) vizinhos mais próximos. O vetor `PQD` é a lista de prioridade das distâncias dos κ vizinhos mais próximos, sendo `PQD[1]` a distância do κ-ésimo ponto mais próximo. O vetor `PQR` é a lista dos κ vizinhos mais próximos de `x`. Antes de se iniciar a busca, é necessário inicializar as variáveis globais da seguinte forma:

```
1.   procedure KDInicializa;
2.   begin
3.     PQD[1:capa]:=infinito;
4.     bu[1:d]:=infinito;
5.     bl[1:d]:=-infinito;
6.     x:=instância a ser buscada;
7.   end;
```

A função de busca propriamente dita pode ser escrita como segue:

```
1.   procedure KDBusca(t:NodePt);
2.   var p,c,temp:integer;
3.   begin
4.     if (t^.leaf=true) then begin
5.       <Examine t^.x e atualize PQD e PQR>
6.       if BallWithinBounds then done else return;
7.     end;
8.     p:=t^.p; c:=t^.c;
9.
10.    if x[p]<=c then begin
11.      temp:=bu[p]; bu[p]:=c;
12.      KDBusca(t^.left);
13.      bu[p]:=temp;
14.    end else begin
15.      temp:=bl[p]; bl[p]:=c;
16.      KDBusca(t^.right);
17.      bl[p]:=temp;
18.    end;
19.
20.    if x[p]<=c then begin
21.      temp:=bl[p]; bl[p]:=c;
22.      if BoundsOverlapBall then KDBusca(t^.right);
23.      bl[p]:=temp;
24.    end else begin
25.      temp:=bu[p]; bu[p]:=c;
26.      if BoundsOverlapBall then KDBusca(t^.left);
27.      bu[p]:=temp;
28.    end;
29.    if BallWithinBounds then done else return;
30.  end;
```

Após o final da busca, os κ pontos mais próximos de `x` estarão armazenados no vetor `PQR` e as suas distâncias no vetor `PQD`. Observe que `PQR[capa]` é o ponto mais próximo de `x`. As subfunções que `KDBusca` utiliza são:

```
1.   function BallWithinBounds:Boolean;
2.   var i:integer;
3.   begin
4.     for i:=1 to d do begin
5.       if CoordDist(x[i],bl[i])<=PQD[1]
6.       or CoordDist(x[i],bu[i])<=PQD[1]
7.         then return false;
8.     end;
9.     return true;
10.  end;
```

```
1.   function BoundsOverlapBall:Boolean;
2.   var i:integer; sum:real;
3.   begin
4.     sum:=0;
5.     for i:=1 to d do begin
6.       if x[i]<bl[i] then begin
7.         sum:=acrescenta(sum,CoordDist(x[i],bl[i]));
8.         if dissim(sum)<PQD[1] then return true;
9.       end else if x[i]>bu[i] then begin
10.        sum:=acrescenta(sum,CoordDist(x[i],bl[i]));
11.        if dissim(sum)<PQD[1] then return true;
12.      end;
13.    end;
14.    return false;
15.  end;
```

Utilizando a distância euclidiana como a métrica, devemos utilizar as seguintes funções:

```
1.   function CoordDist(a,b:integer):real;
2.   begin
3.     CoordDist:=abs(a-b);
4.   end;
```

A função acima calcula a distância numa coordenada. A função seguinte adiciona a distância numa coordenada à soma dos quadrados:

```
1.   function acrescenta(sum,c:real):real;
2.   begin
3.     acrescenta:=sum+c*c;
4.   end;
```

A distância euclidiana verdadeira entre dois pontos é dada pela função abaixo, quando é invocada com a variável `sum` contendo a soma dos quadrados das distâncias de todas as coordenadas.

```
1.   function dissim(sum:real):real;
2.   begin
3.     dissim:=sqrt(sum);
4.   end;
```

Se alterarmos as três funções acima, podemos obter a distância "maior lado" ou a distância "soma dos lados".

Como todos os outros algoritmos apresentados nesta tese, a kd-árvore foi implementada na linguagem C++. Para evitar repetidas alocações dinâmicas, o que diminuiria o desempenho tanto do ponto de vista de tempo como de espaço, a árvore foi construída num grande vetor construído numa única chamada de alocação dinâmica de memória. Para evitar desperdício de memória, nas folhas das árvores não foram armazenados os exemplos, porém índices que apontam para pixels das imagens de treinamento.

Referências Bibliográficas

[Angluin and Laird, 1988] D. Angluin and P. Laird, "Learning from Noisy Examples," *Mach. Learning*, vol. 2, pp. 343-370, 1988.

[Angluin, 1988] D. Angluin, "Queries and Concept Learning," *Mach. Learning*, vol. 2, pp. 319-342, 1988.

[Anthony and Biggs, 1992] M. Anthony and N. Biggs, *Computational Learning Theory - An Introduction*, Cambridge University Press, 1992.

[Aquino, 1980] Tomás de Aquino, *Suma Teológica*, co-edição Esc. Sup. de Teologia de S. Lourenço de Brindes, Livraria Sulina Ed., Univ. Caxias do Sul, 1980.

[Banon and Barrera, 1991] G. J. F. Banon and J. Barrera, "Minimal Representations for Translation-Invariant Set Mappings by Mathematical Morphology," *SIAM J. Appl. Math.*, vol. 51, no. 6, pp. 1782-1798, 1991.

[Banon and Barrera, 1993] G. J. F. Banon and J. Barrera, "Decomposition of Mappings between Complete Lattices by Mathematical Morphology, Part I. General Lattices," *Signal Processing*, vol. 30, no. 3, pp. 299-327, 1993.

[Barrera et al., 1995] J. Barrera, N. S. Tomita, F. S. C. Silva and R. Terada, "Automatic Programming of Binary Morphological Machines by PAC Learning," *SPIE, Proceedings Neural, Morphological and Stochastic Methods in Image and Signal Processing*, vol. 2568, pp. 233-244, San Diego, 1995.

[Bentley, 1975] J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Comm. ACM*, vol. 18, no. 9, pp. 509-517, 1975.

[Bergadano and Cutello, 1995] F. Bergadano and V. Cutello, "Probably Approximately Correct Learning in Fuzzy Classification Systems," *IEEE T. Fuzzy Systems,* vol. 3, no. 4, pp. 473-478, 1995.

[Birkhoff and Bartee, 1970] G. Birkhoff and T. C. Bartee, *Modern Applied Algebra*, McGraw-Hill, 1970.

[Bleau et al., 1992a] A. Bleau, J. Guise and A. R. LeBlanc, "A New Set of Fast Algorithms for Mathematical Morphology, I. Idempotent Geodesic Transforms," *CVGIP: Image Understanding,* vol. 56, no. 2, pp. 178-209, 1992.

[Bleau et al., 1992b] A. Bleau, J. Guise and A. R. LeBlanc, "A New Set of Fast Algorithms for Mathematical Morphology, II. Identification of Topographic Features on Grayscale Images," *CVGIP: Image Understanding,* vol. 56, no. 2, pp. 178-209, 1992.

[Boomgaard and Smeulders, 1994] R. Boomgaard and A. Smeulders, "The Morphological Structure of Images: The Differential Equations of Morphological Scale-Space," *IEEE T. Pattern Analysis Machine Intell.,* vol. 16, no. 11, pp 1101-1113, 1994.

[Breen and Jones, 1996] E. J. Breen and R. Jones, "Attribute Openings, Thinnings, and Granulometries," *Comp. Vision Image Understanding,* vol. 64, no. 3, pp. 377-389, 1996.

[Cover and Hart, 1967] T. M. Cover and P. E. Hart, "Nearest Neighbor Pattern Classification," *IEEE T. Information Theory*, vol. IT-13, no. 1, pp. 21-27, 1967.

[Djouadi and Bouktache, 1997] A. Djouadi and E. Bouktache, "A Fast Algorithm for the Nearest-Neighbor Classifier," *IEEE T. Pattern Anal. and Machine Intell.,* vol. 19, no. 3, pp. 277- 282, 1997.

[Dougherty and Astola, 1994] E. R. Dougherty and J. Astola, *An Introduction to Nonlinear Image Processing*, SPIE Optical Engineering Press, 1994.

[Dougherty and Sinha, 1995a] E. R. Dougherty and D. Sinha, "Computational Gray-scale Mathematical Morphology on Lattices (A Comparator-based Image Algebra, Part I: Architecture)," *Real-Time Imaging,* vol. 1, pp. 69-85, 1995.

[Dougherty and Sinha, 1995b] E. R. Dougherty and D. Sinha, "Computational Gray-scale Mathematical Morphology on Lattices (A Comparator-based Image Algebra, Part II: Image Operators)," *Real-Time Imaging,* vol. 1, pp. 283-295, 1995.

[Dougherty, 1992a] E. R. Dougherty, "Optimal Mean-Square N-Observation Digital Morphological Filters, Part I - Optimal Binary Filters," *CVGIP: Image Understanding*, vol. 55, no. 1, pp. 36-54, 1992.

[Dougherty, 1992b] E. R. Dougherty, "Optimal Mean-Square N-Observation Digital Morphological Filters, Part II - Optimal Gray-Scale Filters," *CVGIP: Image Understanding*, vol. 55, no. 1, pp. 55-72, 1992.

[Dudewicz and Mishra, 1988] E. J. Dudewicz and , S. N. Mishra, *Modern Mathematical Statistics*, John Wiley, New York, 1988.

[Friedman et al., 1977] J. H. Friedman, J. L. Bentley and R. A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM T. Mathematical Software*, vol. 3, no. 3, pp. 209-226, 1977.

[Fukunaga and Flick, 1985] K. Fukunaga and T. E. Flick, "The 2-NN Rule for More Accurate NN Risk Estimation," *IEEE T. Pattern Anal. Machine Intell.Machine Intell.*, vol. PAMI-7, no. 1, pp. 107-112, 1985.

[Ghosh, 1990] P. K. Ghosh, "A Solution of Polygon Containment, Spatial Planning, and Other Related Problems Using Minkowski Operations," *Comput. Vision Graphics Image Proc.*, vol. 49, pp. 1-35, 1990.

[Ghosh, 1991] P. K. Ghosh, "An Algebra of Polygons through the Notion of Negative Shapes," *CVGIP: Image Understanding*, vol. 54, no. 1, pp. 119-144, 1991.

[Ghosh, 1993] P. K. Ghosh, "A Unified Computational Framework for Minkowski Operations," *Comput. & Graphics*, vol. 17, no. 4, pp. 357-378, 1993.

[Golub and Van Loan, 1989] G. H. Golub and C. F. Van Loan, *Matrix Computations - Second Edition*, The Johns Hopkins University Press, 1989.

[Gonzalez and Woods, 1992] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

[Hadwiger, 1950] H. Hadwiger, "Minkowskische Addition und Subtraktion beliebiger Punktmengen und die Theoreme von Erhard Schmidt," *Math. Z.,* vol. 53, pp. 210-218, 1950.

[Haralick et al., 1987] R. M. Haralick, S. R. Sternberg, X. Zhuang, "Image Analysis Using Mathematical Morphology," *IEEE T. Pattern Analysis and Machine Intell.,* vol. PAMI-9, no. 4, pp. 532-550, 1987.

[Harvey and Marshall, 1996] N. R. Harvey and S. Marshall, "The Use of Genetic Algorithms in Morphological Filter Design," *Signal Processing: Image Communication*, vol. 8, pp. 55-71, 1996.

[Haussler, 1992] D. Haussler, *Decision Theoretic Generalizations of the PAC Model for Neural Net and Other Learning Applications;* Information and Computation, vol. 100, pp. 78-150, 1992.

[Heckbert, 1982] P. Heckbert, "Color Image Quantization for Frame Buffer Display," Computer Graphics, vol. 16, no. 3, pp. 297-307, 1982.

[Heijmans and Ronse, 1990] H. J. A. M. Heijmans and C. Ronse, "The Algebraic Basis of Mathematical Morphology - Part I Dilations and Erosions," *Comput. Vision Graphics Image Processing*, vol. 50, pp. 245-295, 1990.

[Heijmans, 1991] H. J. A. M. Heijmans, "Theoretical Aspects of Gray-Level Morphology," *IEEE T. Pattern Anal. and Machine Intell.*, vol. 13, no. 6, pp. 568-582, 1991.

[Heijmans, 1994] H. J. A. M. Heijmans, *Morphological Image Operators*, Academic Press, 1994.

[IBGE, 1977] Fundação Instituto Brasileiro de Geografia e Estatística, *Geografia do Brasil - Região Sul*, 1977.

[Jones and Svalbe, 1994a] R. Jones and I. Svalbe, "Morphological Filtering as Template Matching," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 16, no. 4, pp. 438-443, 1994.

[Jones and Svalbe, 1994b] R. Jones and I. Svalbe, "Algorithms for the Decomposition of Gray-Scale Morphological Operations," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 16, no. 6, pp. 581-588, 1994.

[Kim et al., 1997] H. Y. Kim, F. A. M. Cipparrone and M. T. C. Andrade, "Technique to Construct Grey-Scale Morphological Operators Using Fuzzy Expert System," *Electronics Letters,* vol. 33, no. 22, pp. 1859-1861, 1997.

[Kim, 1997] H. Y. Kim, "Quick Construction of Efficient Morphological Operators by Computational Learning," *Electronics Letters,* vol. 33, no. 4, pp. 286-287, 13th Feb. 1997.

[Li and Chen, 1991] D. Li and X. Y. Chen, "Automatically Generating Triangulated Irregular Digital Terrain Model Networks by Mathematical Morphology," *ISPRS J. Photogrammetry and Remote Sensing,* vol. 46, pp. 283-295, 1991.

[Loce, 1993] R. P. Loce, *Morphological Filter Mean-Absolute-Error Representation Theorems and Their Application to Optimal Morphological Filter Design*, Ph. D. Thesis, Center for Imaging Science, Rochester Institute of Technology, May 1993.

[Loizou and Maybank, 1987] G. Loizou and S. J. Maybank, "The Nearest Neighbor and the Bayes Error Rates", *IEEE T. Pattern Anal. Machine Intell.*, vol. PAMI-9, no. 2, pp. 254-262, 1987.

[MacLane and Birkhoff, 1967] S. MacLane and G. Birkhoff, *Algebra*, MacMillan, 1967.

[Maragos and Schafer, 1986] P. A. Maragos and R. W. Schafer, "Morphological Skeleton Representation and Coding of Binary Images," *IEEE T. Acoustics Speech Signal Processing,* vol. ASSP-34, no. 5, pp. 1228-1244, 1986.

[Matheron, 1975] G. Matheron, *Random sets and integral geometry*, Wiley, New York, 1975.

[Minkowski, 1903] H. Minkowski, "Volumen und Oberfläche," *Math. Ann.,* vol. 57, pp. 447-495, 1903.

[Murray and vanRyper, 1994] J. D. Murray and W. vanRyper, *Encyclopedia of Graphics File Formats*, O'Reilly & Associates, 1994.

[Overturf et al., 1995] L. A. Overturf, M. L. Comer and E J. Delp, "Color Image Coding Using Morphological Pyramid Decomposition," *IEEE T. Image Processing,* vol. 4, no. 2, pp. 177-185, 1995.

[Pai and Hansen, 1994] T. W. Pai and J. H. L. Hansen, "Boundary-Constrained Morphological Skeleton Minimization and Skeleton Reconstruction," *IEEE T. Pattern Analysis Machine Intell.,* vol. 16, no. 2, pp. 201-208, 1994.

[Pavlidis, 1982] T. Pavlidis, *Algorithms for Graphics and Image Processing*, Computer Science Press, 1982.

[Pollard, 1984] D. Pollard, *Convergence of Stochastic Processes*, Springer-Verlag, New York, 1984.

[Pratt, 1978] W. K. Pratt, *Digital Image Processing*, John Willey & Sons, 1978.

[Preparata and Shamos, 1985] F. P. Preparata and M. I. Shamos, *Computational Geometry, an Introduction*, Springer-Verlag, New York, 1985.

[Ronse and Heijmans, 1990] C. Ronse and H. J. A. M. Heijmans, "The Algebraic Basis of Mathematical Morphology - Part II Openings and Closings," *CVGIP: Image Understanding*, vol. 54, no. 1, pp. 74-97, 1991.

[Ronse, 1996] C. Ronse, "A Lattice-Theoretical Morphological View on Template Extraction in Images," *Journal of Visual Comm. and Image Representation*, vol. 7, no. 3, pp. 273-295, 1996.

[Sage and Melsa, 1971] A. P. Sage and J. L. Melsa, *Estimation Theory with Applications to Communications and Control,* McGraw-Hill Book Company, 1971.

[Saitta and Bergadano, 1993] L. Saitta and F. Bergadano, "Pattern Recognition and Valiant's Learning Framework," *IEEE T. Pattern Anal. and Machine Intell.*, vol. 15, no. 2, pp. 145-155, 1993.

[Schmitt, 1989a] M. Schmitt, *Des Algorithmes Morphologiques a l'Intelligence Artificielle*, thèse, Ecole Nationale Supérieure des Mines de Paris, Fév. 1989.

[Schmitt, 1989b] M. Schmitt, "Mathematical Morphology and Artificial Intelligence: an Automatic Programming System," *Signal Processing*, vol. 16, no. 4, pp. 389-401, 1989.

[Serra and Vincent, 1992] J. Serra and L. Vincent, "An Overview of Morphological Filtering," *Circuits Systems Signal Process,* vol. 11, no. 1, pp. 47-108, 1992.

[Serra, 1982] J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, New York, 1982.

[Serra, 1986] J. Serra, "Introduction to Mathematical Morphology," *Comput. Vision Graphics and Image Processing,* vol. 35, pp. 283-305, 1986.

[Serra, 1988] J. Serra (ed), *Image Analysis and Mathematical Morphology, part II*, Academic Press, 1988.

[Short and Fukunaga, 1981] R. D. Short and K. Fukunaga, "The Optimal Distance Measure for Nearest Neighbor Classification," *IEEE T. Information Theory*, vol. IT-27, no. 5, pp. 622-627, 1981.

[Smith et al., 1994] S. J. Smith, M. O. Bourgoin, K. Sims and H. L Voorhees, "Handwritten Character Classification Using Nearest Neighbor in Large Databases," *IEEE T. Pattern Anal. Machine Intell.*, vol. 16, no. 9, pp. 915-919, 1994.

[Song and Delp, 1990] J. Song and E. J. Delp, "The Analysis of Morphological Filters with Multiple Structuring Elements," *Comp. Vision Graphics Image Proc.,* vol. 50, pp. 308-328, 1990.

[Sorenson, 1980] H. W. Sorenson, *Parameter Estimation*, Marcel Dekker Inc., 1980.

[Spiegel, 1974] M. R. Spiegel, *Estatística*, Coleção Schaum, Ed. McGraw-Hill do Brasil, 1974.

[Spiegel, 1979] M. R. Spiegel, *Probabilidade e Estatística*, Coleção Schaum, Ed. McGraw-Hill do Brasil, 1979.

[Sproull, 1991] R. F. Sproull, "Refinements to Nearest-Neighbor Searching in k-Dimensional Trees," *Algorithmica*, vol. 6, pp. 579-589, 1991.

[Sternberg, 1986] S. R. Sternberg, "Grayscale Morphology," *Comput. Vision Graphics and Image Processing*, vol. 35, pp. 333-355, 1986.

[Tomita, 1996] N. S. Tomita, *Programação Automática de Máquinas Morfológicas Binárias Baseada em Aprendizagem PAC*, Dissertação de Mestrado, Instituto de Matemática e Estatística da Universidade de São Paulo, fev. 1996.

[Valiant, 1984] L. G. Valiant, "A Theory of Learnable," *Comm. ACM*, vol. 27, no. 11, pp. 1134-1142.

[Vincent, 1990] L. Vincent, *Algorithmes Morphologiques a Base de Files d'Attente et de Lacets, Extension aux Graphes*, thèse, Ecole Nationale Supérieure des Mines de Paris, Mai 1990.

[Wirth, 1976] N. Wirth; *Algorithms + Data Structures = Programs*, Prentice Hall Englewood Cliffs, 1976.