Universidade de São Paulo
Instituto de Física de São Carlos
Departamento de Física e Informática

João Paulo Lemos Escola

Reconhecimento de assinaturas baseado em seus ruídos caligráficos

São Carlos - SP
2013

João Paulo Lemos Escola

Reconhecimento de assinaturas baseado em seus ruídos caligráficos

Dissertação apresentada ao Programa de Pós-graduação em Física da Universidade de São Paulo para obtenção do título de Mestre em Ciências.

Área de concentração: Física Aplicada - Opção Computacional.
Orientador: Prof. Dr. Rodrigo Capobianco Guido

Versão Original

São Carlos - SP
2013

Ficha catalográfica elaborada pelo Serviço de Biblioteca e Informação do IFSC,
com os dados fornecidos pelo(a) autor(a)

Escola, João Paulo Lemos
Reconhecimento de assinaturas baseado em seus ruídos caligráficos. / João Paulo Lemos Escola; orientador Rodrigo Capobianco Guido São Carlos, 2013.
108p.

Dissertação (Mestrado - Programa de Pós-Graduação em Física Aplicada Computacional) Instituto de Física de São Carlos, Universidade de São Paulo, 2013.

1. Processamento de sinais. 2. Biometria. 3. Máquinas de vetores de suporte. 4. Inteligência artificial. 5. *Wavelets*. I. Capobianco Guido, Rodrigo, orient. II. Título.

FOLHA DE APROVAÇÃO

João Paulo Lemos Escola

Dissertação apresentada ao Instituto de Fisica de São Carlos da Universidade de São Paulo para obtenção do título de Mestre em Ciências. Área de Concentração: Física Aplicada - Opção: Física Computacional.

Aprovado(a) em: 04/02/2014

Comissão Julgadora

Prof(a). Dr(a). Rodrigo Capobianco Guido
Instituição: (UNESP/São José Rio Preto)

Prof(a). Dr(a). Ivan Nunes da Silva
Instituição: (EESC/USP)

Prof(a). Dr(a). Ellen Francine Barbosa
Instituição: (ICMC/USP)

Dedico a presente dissertação a todos que acreditaram em mim e me apoiaram durante essa longa caminhada.

Agradecimentos

A Deus, por me ter permitido existir e ter sido sempre presença contante em minha vida dando-me motivação, força e sabedoria em todos os momentos. E a Jesus pela amizade e presença certa em todas as horas.

Aos meus pais, José (*in memorian*) e Maria da Conceição, e à minha esposa Vanessa, por terem incentivado meus estudos e se esforçado para que eles continuassem.

Ao meu orientador Rodrigo, pelo apoio e presteza ao me atender sempre que precisei.

Aos meus amigos e companheiros de mestrado, Sylvio, Lucimar, Luciene, Fabrício, Paulo Fantinato, Leonardo, Márcio, Regiane, Ricardo, William e todos os demais.

"O sucesso é ir de fracasso em fracasso sem perder entusiasmo."

Winston Churchill

Resumo

ESCOLA, J.P.L. *Reconhecimento de assinaturas baseado em seus ruídos caligráficos.* 2013. 108p. Dissertação (Mestrado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2013.

A biometria é o processo de reconhecimento dos seres vivos baseado em suas características fisiológicas ou comportamentais. Existem atualmente diversos métodos biométricos e a assinatura em papel é uma das técnicas de mensuração comportamental mais antigas. Por meio do processamento de sinais de áudio, é possível realizar o reconhecimento de padrões dos ruídos emitidos pela caneta ao assinar. Com o objetivo de aumentar o grau de sucesso ao validar a assinatura realizada por uma pessoa, este trabalho propõe uma técnica baseada em um algoritmo que combina Máquinas de Vetores de Suporte (SVMs), treinadas com o uso de um procedimento de aprendizado semi-supervisionado, alimentadas por um conjunto de parâmetros obtidos com o uso da Transformada *Wavelet* Discreta do sinal de áudio do ruído emitido pela caneta ao assinar sobre uma superfície rígida. Os testes realizados com uma base de dados de assinaturas reais, testando diversos filtros wavelet, demonstram a eficácia da técnica proposta.

Palavras-chave: Processamento de sinais. Biometria. Máquinas de vetores de suporte. Inteligência artificial. *Wavelets*.

Abstract

ESCOLA, J.P.L. *Recognition of signatures based on their calligraphic noise.* 2013. 108p. Dissertação (Mestrado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2013.

Biometrics is the process of recognition of human beings based on their physiological or behavioral characteristics. There are, currently, several methods and biometric signatures on papers is one of the oldest techniques for measuring behavioral characteristics. By digital processing the audio signals, it is possible to recognize noise emitted by pens when signing. To increase the degree of success, this work propõe a technique based on an algorithm that combines two Support Vector Machines (SVMs), trained using a semi-supervised learning procedure, fed by a set of parameters obtained using the Discrete Wavelet Transform of the audio signals produced by the noise emitted by the pen when signing on a hard surface. Tests conducted with a database of signatures, trying many wavelet filters, demonstrate the real effectiveness of the proposed approach.

Key-words: Signal processing. Biometrics. Support vector machines. Artificial intelligence. *Wavelets*.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

DSP Digital Signal Processor

DWT Discrete Wavelet Transform

FIR Finite Impulse Response

FPGA Field-Programmable Gate Array

IA Inteligência Artificial

MRA Multi-Resolution Analysis

PDS Processamento Digital de Sinais

PRFB Perfect Reconstruction Filter Bank

QMF Quadrature Mirror Filters

RNA Redes Neurais Artificiais

SVM Support Vector Machines

WAV Waveform Audio Format

Sumário

Capítulo 1

Introdução e Motivação

A biometria, um processo que possibilita a validação de um ser humano para fins de segurança e vem, segundo Bravo (1), substituindo mecanismos clássicos de autenticação pessoal, tais como o uso de senhas ou cartões magnéticos. Dos mais diversos tipos de medição biométrica, o da assinatura é o mais comum, utilizado nos mais diferentes segmentos, como administradoras, bancos e cartórios.

Por meio do processamento digital de sinais PDS é possível executar diversos tipos de análise em dados de áudio específicos. Esses dados podem ser a fala de uma pessoa, uma canção ou o ruído emitido pela caneta ao assinar, que podem ser utilizados para aumentar o nível de garantia da autenticação por assinatura.

Atualmente, muitos são os esforços na busca de formas de autenticação pessoal confiáveis. Como muitas dessas técnicas - o exame da retina com *laser*, por exemplo - são invasivas, causando desconforto, a análise do ruído emitido pela caneta ao assinar mostra-se uma forma adicional de autenticá-lo sem que seja necessário acrescentar esforço em situações de validação pessoal que demandem assinatura manuscrita.

1.1 Objetivos e Contribuições do trabalho

Com base nas considerações anteriroes, o objetivo específico deste trabalho é o de apresentar uma técnica não invasiva de autenticação pessoal por meio a análise do ruído emitido pela caneta durante o processo de assinatura manuscrita. Uma estrutura composta por Máquinas de Vetores de Suporte (*Support Vector Machines* - SVMs), treinadas com um procedimento semi-supervisionado e recebendo parâmetros obtidos com base na Transformada *Wavelet* Discreta (*Discrete Wavelet Transform* - DWT) do sinal de áudio capturado por meio de um microfone embutido em uma caneta esferográfica, constitui a base para o desenvolvimento do projeto. Em particular, acredita-se que o sistema desenvolvido possa ser adotado por instituições bancárias para autenticação de clientes em tempo real. No que tange às contribuições originais do presente trabalho, destacam-se:

- a obtenção de um novo algoritmo de baixa complexidade para autenticação biométrica em tempo real;
- a especificação de um sistema binário de classificação com base em uma estrutura composta por SVMs que se comportam de forma a distinguir entre as características da assinatura de um usuário real e de N impostores;
- a caracterização do melhor conjunto de parâmetros extraídos dos sinais de áudio para uso com o sistema de classificação proposto, produzindo os resultados mais eficientes na distinção entre o ruído emitido pelo usuário autêntico e um usuário impostor.

1.2 Organização do Trabalho

Este trabalho está organizado da seguinte forma: o capítulo 2 apresenta uma revisão da literatura, envolvendo os princípios básicos da biometria, das SVMs e da DWT, além de outros conceitos importantes. O capítulo 3 apresenta, com detalhes, o algoritmo proposto; já os resultados obtidos com as diversas combinações de parâmetros extraídos por meio da transformada *wavelet* estão descritos no capítulo 4. Tendo em vista os resultados, e apoiado nos conceitos e

características estudados durante a revisão de literatura, o capítulo 5 apresenta as conclusões. Por fim, logo após as referências bibliográficas, três apêndices apresentam, respectivamente, uma lista dos coeficientes dos filtros *wavelet* utilizados nas experiências, o código-fonte da implementação em linguagem de programação de alto nível, assim como as publicações obtidas durante o curso de mestrado do autor.

Capítulo 2

Revisão Bibliográfica e Direcionamento do Trabalho

Neste capítulo, apresenta-se uma revisão da literatura, abordando basicamente os seguintes tópicos: biometria e processos de autenticação pessoal, já que caracterizam o presente trabalho; as SVMs, que constituem os classificadores inteligentes adotados como ponto de partida para formulação do sistema proposto; os conceitos de digitalização de sinais com base no formato WAV de arquivos; energia e escala Bark, que caracterizam os parâmetros utilizados para alimentar o classificador proposto e; por fim, a DWT, que é a principal ferramenta utilizada para o cálculo dos parâmetros retrocitados. Todos esses conceitos são necessários para a perfeita compreensão do sistema proposto na presente dissertação.

2.1 Identificação Pessoal

De acordo com Bravo (1), o ser humano está acostumado a comprovar sua identidade a fim de poder adentrar estabelecimentos ou utilizar serviços. Dentre os meios de se indentificar um ser humano podemos citar:

- utilização de senhas e códigos;
- crachás ou pulseiras de identificação;

- documentos em papel, cartões magnéticos e de chip;
- identificação de retina, íris, contorno da face, voz, entre outros.

O propósito da identificação pessoal é proteger um sistema de acessos sem autorização que podem provocar prejuízos a uma organização. Segundo Ratha e Bolle (2), o processo de identificação consiste em selecionar a identidade correta de um indivíduo desconhecido a partir de um banco de dados de possíveis identidades. Este processo visa à proteção de usuários registrados, negando a possibilidade de personificação daqueles por intrusos mal intencionados.

Existem basicamente três técnicas de autenticação pessoal em um sistema (1):

- autenticação com base no conhecimento do indivíduo: senhas, números de identificação como registros de entidades de classe (Registro Geral e Cadastro de Pessoa Física), número de passaporte, *login* para acesso ao computador ou sistema de rede, entre outros;
- autenticação com base em dispositivo físico em posse do indivíduo: cartão magnético ou de *chip*, carteira de identidade e chaves;
- autenticação com base em característica física do indivíduo: biometria da íris, impressão digital, formato do rosto, voz, entre outros.

Por serem de baixo custo, as duas primeiras técnicas citadas acima são as mais comuns, também porque as pessoas já estão acostumadas a utilizá-las. Apesar disso, graças ao declínio do custo dos equipamentos computacionais, nos últimos anos a biometria vem conquistando o espaço antes ocupado pelos métodos clássicos de autenticação pessoal.

2.2 Biometria

Consoante Vigliazzi (3), a Biometria é o ramo da ciência que trata da mensuração dos seres vivos, significando, na área de sistemas de segurança de informação, a verificação da identidade de um indivíduo por meio de uma característica única, isto é, por meio de processos automatizados. A utilização da biometria em oposição ao emprego de senhas ou posses (cartões ou

chaves), no controle de acessos, mostra-se vantajosa pois permite que o usuário seja autenticado, ainda que perca a sua senha ou esqueça seu cartão de acesso, por exemplo.

Atualmente, muitos são os métodos biométricos utilizados no ramo da tecnologia da informação, entre eles: termogramas de face, mãos e veias da mão (4), dinâmica do andar (5), reconhecimento de impressão digital (2) e reconhecimento da assinatura manuscrita (3), sendo que, neste último, o reconhecimento é normalmente realizado a partir da imagem da assinatura e não do ruído emitido por ela.

A grande vantagem da biometria para fins de autenticação pessoal em relação aos métodos clássicos está no fato de que é baseada em características fisiológicas do indivíduo, dessa forma, já que estas não podem ser perdidas, roubadas ou esquecidas, a confiabilidade do sistema aumenta subtancialmente.

Segundo Bravo (1), as técnicas biométricas podem ser divididas em biometria fisiológica, que engloba características fisiológicas como impressão digital e contorno da face, e biometria de comportamento, que implica características relativas ao comportamento como a escrita, assinatura, voz, entre outras.

A escolha do método biométrico de comportamento sustenta-se nas vantagens citadas anteriormente: trata-se de um processo não invasivo comumente utilizado em diversos segmentos.

Por último, pode-se afirmar que dentre as diversas características da escrita como o estilo da caligrafia (formas das letras), também devem ser levadas em conta a dinâmica e a postura ao escrever, como a força, velocidade e a aceleração, que podem servir de subsídio para a distinção no processo de identificação do indivíduo.

2.3 Classificadores inteligentes

De acordo com Silva et al. (8), as RNAs (Redes Neurais Artificiais) são “modelos computacionais inspirados no sistema nervoso de seres vivos”. Assim como nas redes neurais do

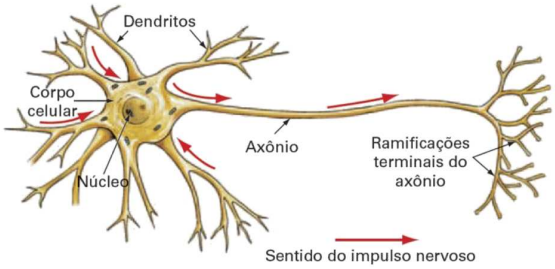

Figura 2.1 – Neurônio Biológico (extraído de (6))

cérebro humano, as RNAs "tem capacidade de aquisição e manutenção do conhecimento", possuem interconexões como os neurônios biológicos, como na Figura 2.1. Essas interconexões são representadas no computador por meio de vetores e matrizes multiplicadas por seus respectivos pesos sinápticos.

O sistema de classificação de padrões empregado no presente trabalho foi elaborado a partir de uma ferramenta de Inteligência Artificial (IA). Segundo Souza (9), esta técnica é utilizada para classificação binária e é uma evolução das RNAs, pois pode trazer resultados muitas vezes superiores a estas. Ainda de acordo com Souza (9), as SVMs foram desenvolvidas por Vladimir Vapnik em 1998 e se baseiam no princípio da "Minimização do Risco Estrutural", que visa minimizar o risco empírico no conjunto de treinamento e controlar sua função de decisão. Esta última é utilizada para obter o referido valor de risco. Assim como as RNAs, as SVMs oferecem capacidade de generalização de acordo com o conjunto de treinamento atribuído.

Segundo Baranoski (10), a idéia da minimização do risco estrutural busca minimizar o erro de generalização, ou seja, os erros verdadeiros em amostras não descobertas.

Dado um conjunto de vetores de treinamento $\{(\vec{x_1}, y_1); (\vec{x_2}, y_2); ...; (\vec{x_n}, y_n)\}$ pertencente a duas classes linearmente separáveis, W_1 representada pela saída $y_i = +1$ e W_2 representada pela saída $y_i = -1$, a SVM encontra o hiperplano com a máxima distância do conjunto de treinamento. De acordo com esse princípio, existirá somente um hiperplano com a margem máxima δ, definida como sendo a soma das distâncias do hiperplano até o ponto mais próximo das classes, sendo que esse limiar do classificador é a separação ótima, conforme se pode observar na figura 2.2.

Podem existir casos, como no presente trabalho, nos quais os conjuntos de treinamento não são linearmente separáveis. Neste caso, associa-se ao i-ésimo ponto x_i, $(0 \leqslant i \leqslant N)$, uma variável σ_i, que representa a amplitude do erro de classificação. A função de penalidade (função 2.1) equivale ao somatório dos erros de classificação. A proposta da SVM é buscar a

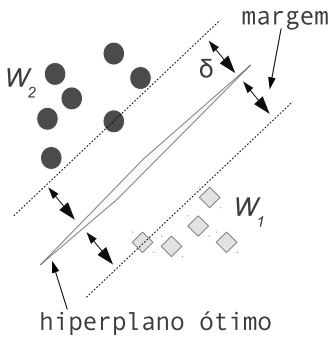

Figura 2.2 - Separação linear das classes W_1 e W_2 pelo hiperplano. Os vetores de suporte definem os limites do hiperplano (extraído de (9)).

minimização dos erros de treinamento de (função 2.2), onde $\vec{w}$ é o vetor de pesos e $C \in \Re$.

$$F(\sigma) = \sum_{i=0}^{N-1} \sigma_i \tag{2.1}$$

$$\frac{1}{2}\vec{w}^T\vec{w} + C\sum_{i=0}^{N-1} \sigma_i \tag{2.2}$$

Conforme Souza (9), em uma operação que compreende duas etapas, é possível verter o equacionamento anterior (Figura 2.3):

1. com a aplicação do Teorema de Cover, convertendo o vetor de entrada não-linear em um espaço de características de alta dimensão. Essa função de mapeamento é denominada *kernel* ou núcleo da aplicação, geralmente representada por $\phi(\cdot)$.

2. construção do hiperplano que irá separar as características dos vetores da etapa anterior. A função desse hiperplano á a de separar os padrões de entrada mapeados no passo anterior.

De acordo com Baranoski (10), "a fim de encontrar a superfície de decisão ótima, o algoritmo da SVM tenta separar da melhor forma possível os pontos dos dados de ambas as classes". Nesse caso, são selecionados os pontos mais próximos da fronteira entre as classes, pelo fato de serem mais importantes para a solução do problema, em comparação com os pontos mais distantes, pois, por estarem mais próximos, ajudam a definir a melhor superfície de decisão em relação aos pontos menos próximos do limite.

Na Figura 2.4, apresenta-se a arquitetura básica da SVM, no qual há três camadas de elementos:

1. camada de entrada, que recebe os elementos de teste;

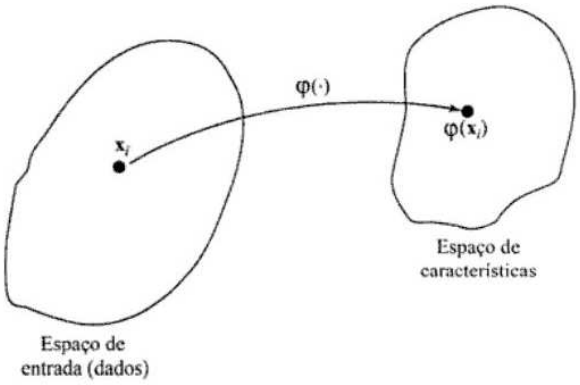

Figura 2.3 – Mapeamento não-linear do espaço de entrada para o espaço de características (extraído de (9))

2. camada intermediária, ou camada oculta, que é responsável pelo mapeamento não-linear dos dados de entrada recebidos na camada anterior;

3. camada de saída, que é composta pelo elemento de processamento associado a um vetor de pesos $\vec{w}$, responsável por separar linearmente as características do mapeamento anterior.

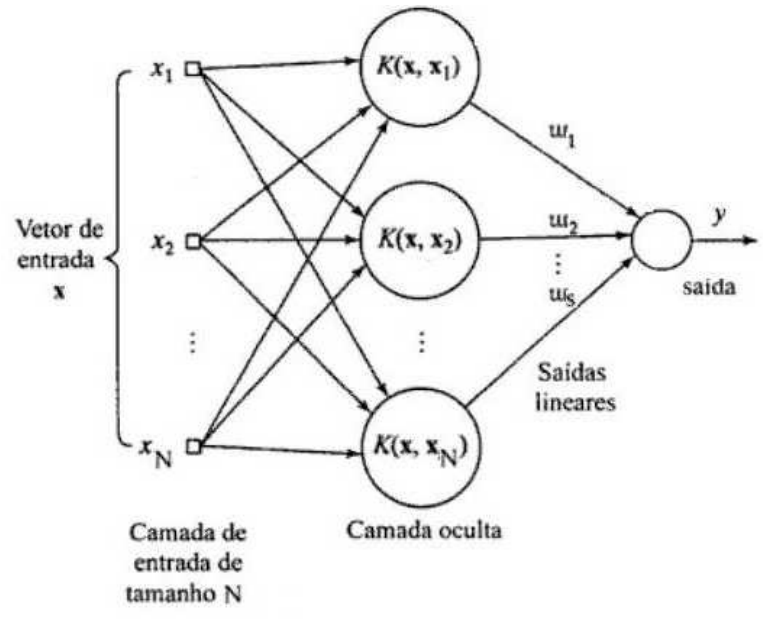

Figura 2.4 – Arquitetura da SVM (extraído de (9))

2.4 Conceitos elementares e principais parâmetros oriundos dos sinais de áudio

A seguir, são apresentados alguns conceitos elementares utilizados durante o projeto e a implementação do presente trabalho. Além disso, a considerável experiência anterior do grupo de pesquisas no qual se insere o presente trabalho (9, 15–18), tem mostrado quais os parâmetros que mais traduzem as características acústicas de interesse para a aplicação em questão. Tais parâmetros também são brevemente revistos.

2.4.1 Digitalização de sinais e o formato WAV de arquivos de áudio

No presente trabalho, os arquivos utilizados com a finalidade de armazenar os sinais de áudio para o treinamento do classificador apresentado foram gravados no formato WAV (12), tendo em vista que ele é o mais simples existente para fins de armazenamento das amostras digitalizadas de tais sinais. Basicamente, um arquivo de áudio no formato WAV é composto por um cabeçalho sub-dividido em duas partes chamadas *chunk 1* e *chunk 2*, seguidas pelos dados brutos, que fazem parte de uma porção identificada como *chunk* de dados. De acordo com a especificação do formato, no caso de sinais amostrados com uma quantização de 16 bits, que são os utilizados aqui, a faixa de valores para cada amostra vai de -32768 até 32767, e ainda, no padrão *little endian*, que se refere à forma com que os valores são armazenados no arquivo. Adicionalmente, os arquivos utilizados nas experiências do presente trabalho foram gravados em um único canal (mono) sem compressão. A especificação de cada um dos *chunks* mencionados consta nas tabelas 2.1, 2.2 e 2.3. Um trecho de código-fonte escrito em linguagem Java, que se encontra na parte final do presente trabalho, foi elaborado para extrair os “dados brutos” dos arquivos WAV e permitir seu uso nos experimentos.

Tabela 2.1 – *Chunk* 1 do formato WAV (9)

Bytes	Descrição
0 a 3	*string* ascii “RIFF”
4 a 7	comprimento do *chunk*
8 a 12	*string* ascii “WAV”

Tabela 2.2 – *Chunk* 2 do formato WAV (9)

Bytes	Descrição
0 a 3	*string* ascii “FMT”
4 a 7	comprimento do *chunk*
8 a 9	0 para mono (um canal) e 1 para estéreo (dois canais)
10 a 11	número de canais (1 para mono e 2 para estéreo)
12 a 15	taxa de amostragem em Hz
16 a 19	bytes por segundo
20 a 21	bytes por amostra
22 a 23	bits por amostra

2.4.2 O conceito de energia

O tamanho de um sinal em tempo discreto pode ser medido por meio de sua energia (14). A energia de um sinal x_i de N amostras discretas é um escalar definido como ilustrado na função 2.3. No presente trabalho, as energias de várias sub-bandas de frequências de um sinal serão utilizadas para alimentar o sistema de classificação adotado.

$$E = \sum_{i=0}^{N-1} {x_i}^2 \tag{2.3}$$

Tabela 2.3 – *Chunk* de dados do formato WAV(9)

Bytes	**Descrição**
0 a 3	*string* ascii “data”
4 a 7	comprimento do *chunk*
8 até o fim	“dados brutos”

Tabela 2.4 - As 25 bandas críticas do sistema auditivo humano que são aproveitadas no presente trabalho para obtenção das energias predominantes do sinal.

Bark (z)	*Hertz (Hz)*	*Bark (z)*	*Hertz (Hz)*
0	0 - 100	12	1720 - 2000
1	100 - 200	13	2000 - 2320
2	200 - 300	14	2320 - 2700
3	300 - 400	15	2700 - 3150
4	400 - 510	16	3150 - 3700
5	510 - 630	17	3700 - 4400
6	630 - 770	18	4400 - 5300
7	770 - 920	19	5300 - 6400
8	920 - 1080	20	6400 - 7700
9	1080 - 1270	21	7700 - 9500
10	1270 - 1480	22	9500 - 12000
11	1480 - 1720	23	12000 - 15500
		24	15500 - 22050

2.4.3 A Escala Bark

Segundo Bosi (12) e Vieira (17), a escala Bark permite sub-dividir a faixa de frequências audíveis pelo ouvido humano em 25 intervalos chamados de bandas críticas. Quando dois ou mais sons, de frequências diferentes mas pertencentes à mesma banda crítica, estão sendo captados simultaneamente, o de maior amplitude mascara o(s) outro(s), portanto, o de menor amplitude não é percebida. A tabela 2.4 mostra as bandas críticas, na escala de frequências em Hertz (Hz) e na escala Bark (z), que é uma escala adotada para linearizar o intervalo entre as bandas. A conversão entre Hertz e Bark é dada pela equação 2.4 onde f representa a frequência em Hertz. Nos casos onde $z < 2$ então aplica-se a correção $z \leftarrow z + 0.15(2 - z)$ e onde $z > 20$ então faz-se $z \leftarrow z + 0.22(z - 20.1)$.

$$z = \frac{26.81f}{1960 + f} - 0.53 \tag{2.4}$$

2.4.4 A Transformada *Wavelet* Discreta (DWT)

De acordo com Souza (9) e Barbon Junior. (16), a transformada *wavelet* discreta (19–21) é uma alternativa mais eficiente para realizar a análise tempo-frequência de um sinal, bem como produzir filtragens e separações de um sinal em sub-bandas do seu espectro de frequências, em comparação com a Transformada de Fourier de Tempo Reduzido (22). Existem duas abordagens básicas em relação à DWT. A primeira é baseada em conceitos da álgebra linear e a segunda, relacionada aos conceitos mais práticos de processamento de sinais. A segunda abordagem é adotada neste trabalho, devido a sua praticidade e aplicabilidade. De acordo Barbon Junior. (16), a DWT funciona como um par de filtros, sendo um deles passa-baixas ($h[\cdot]$) e o outro, passa-altas ($g[\cdot]$), de modo que, em geral, sua frequência de corte está na metade da máxima frequência presente no sinal de entrada. Aplicando a DWT em um sinal discreto, este é submetido a ambos os filtros via convolução(16, 22). "Cada vez que esse processo é aplicado, diz-se que se tem um nível de decomposição e obtêm-se dois novos sinais, um dos quais contém as frequências abaixo da metade da máxima frequência original do sinal, e o outro contém as frequências acima desse limiar" (16). Nesse processo, quando da passagem do sinal original pelo filtro passa-altas, os termos obtidos são chamados coeficientes de *detalhamento*, e quando da passagem do sinal original pelo filtro passa-baixas, os termos chamados coeficientes de *aproximação* designam o sinal obtido. Após cada nível de decomposição, somente o sinal obtido pela aplicação do filtro passa-baixas é aproveitado para continuidade do processo recursivo de decomposição.

Há um detalhe fundamental a ser notado: cada vez que um nível da transformação é realizado, os dois novos sinais obtidos são sub-amostrados por 2, pois eles contêm apenas metade da faixa de frequências do sinal original, de acordo com o Teorema da Amostragem (22) e como ilustra a figura 2.5. Um sinal de n amostras tem a sua transformada *wavelet* com a mesma quantidade de amostras, compostas por uma sequência de coeficientes, iniciando-se com aqueles provenientes da aplicação do filtro passa-baixas no último nível, seguidos pelos coeficientes resultantes da aplicação dos filtros passa-altas nos níveis intermediários, e terminando com os coeficientes resultantes da aplicação do filtro passa-altas do primeiro nível de decomposição.

Duas considerações importantes para se aplicar a transformada *wavelet* (16, 19, 21):

1. para realizar a decomposição até o último nível possível, é necessário que o sinal discreto tenha comprimento equivalente a uma potência de 2, sendo possível realizar $\frac{log(n)}{log(2)}$ decomposições para um sinal de comprimento n.

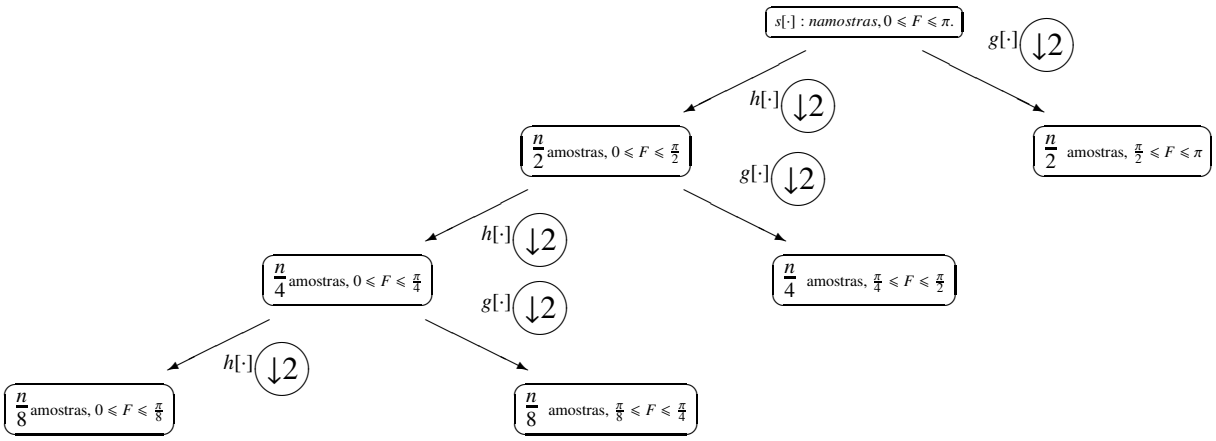

Figura 2.5 - DWT: sinal $s[\cdot]$ de n amostras discretas e máxima frequência π, decomposto até o terceiro nível (extraído de (9))

2. para que um filtro digital seja considerado um filtro *wavelet* é necessário que a resposta em frequência do filtro passa-baixas seja 0 em $\omega = \pi$.

Segundo Barbon Junior (16), o processo de filtragem e sub-amostragem por 2, realizado conjuntamente nos sinais da transformada *wavelet* em cada nível, pode ser representado por uma "convolução modificada" da seguinte forma:

$$y[\cdot] = x[\cdot] * t[\cdot] = \sum_{k=0}^{n-1} t_k x_{2n-k} \qquad , \tag{2.5}$$

ou, mais especificamente:

$$y_{passa-baixas}[\cdot] = x[\cdot] * h[\cdot] = \sum_{k=0}^{n-1} h_k x_{2n-k} \tag{2.6}$$

e

$$y_{passa-altas}[\cdot] = x[\cdot] * g[\cdot] = \sum_{k=0}^{n-1} g_k x_{2n-k} \qquad . \tag{2.7}$$

De acordo com Barbon Junior (16), a DWT está diretamente relacionada à análise de multi-resolução (*Multi-Resolution Analysis* - MRA), proposta por Mallat, Meyer, Stromberg e outros (23, 24). Esta técnica consiste em decompor o sinal sob análise $\vec{f}$, que é um vetor, em uma soma de outros vetores, os quais são pertencentes a uma sequência de sub-espaços vetoriais (25). Em outras palavras, isso significa representar um sinal em vários níveis de resolução. Assim, de acordo com a MRA, no caso de um vetor $\vec{f}$ de n pontos:

$$\vec{f} = \vec{A} + \vec{D} \tag{2.8}$$

onde

$$\vec{A} = \sum_{k=0}^{\frac{n}{2}-1} < \vec{f}, \vec{v_k} > \vec{v_k} \qquad ,$$

e

$$\vec{D} = \sum_{k=0}^{\frac{n}{2}-1} < \vec{f}, \vec{w_k} > \vec{w_k} \qquad ,$$

ou seja:

- $\vec{A}$ é a projeção de $\vec{f}$ num sub-espaço V, com uma base de $\frac{n}{2}$ vetores;
- $\vec{D}$ é a projeção de $\vec{f}$ num sub-espaço W, com uma base de $\frac{n}{2}$ vetores;
- $V \perp W \leftrightarrow \vec{A} \perp \vec{D}$;
- $\vec{v_i} \perp \vec{w_i} \leftrightarrow < \vec{v_i}, \vec{w_i} >= 0$.

O processo acima consiste em uma única decomposição do sinal, ou seja, decomposição de nível 1. Numa transformada *wavelet* de nível 2, o vetor $\vec{A}$ (aproximação) é novamente decomposto na soma de dois outros vetores ortogonais, e pode este processo, como já foi mencionado, ser repetido $\frac{log(n)}{log(2)}$ vezes. Quando ambos os vetores $\vec{A}$ e $\vec{D}$ são re-decompostos, para fins de aumento na resolução frequencial, tem-se o que se chama de *wavelet-packet* ou DWT-*packet*. Assim, generalizando, para uma decomposição de nível j, temos:

$$\vec{f} = \vec{A_j} + \sum_{i=1}^{j} \vec{D_i}. \tag{2.9}$$

sendo que:

- $\vec{A_j}$ é a projeção de $\vec{f}$ num sub-espaço V_j, com uma base contendo $\frac{n}{2^j}$ vetores;

- $\vec{D_i}$ é a projeção de $\vec{f}$ num sub-espaço W_i, com uma base contendo $\frac{n}{2^i}$ vetores;
- $V_j \perp W_j \leftrightarrow \vec{A_j} \perp \vec{D_j}$;
- $\vec{v_{i,j}} \perp \vec{w_{i,j}} \leftrightarrow < \vec{v_{i,j}}, \vec{w_{i,j}} >= 0$.

O processo anterior equivale a escrever (26):

$$f[n] = \sum_{k=0}^{\frac{n}{2^j}-1} H_{j,k}[n]\phi_{j,k}[n] + \sum_{t=1}^{j} \sum_{k=0}^{\frac{n}{2^j}-1} G_{t,k}[n]\psi_{t,k}[n] \tag{2.10}$$

onde

- $\phi[n]$ e $\psi[n]$ formam uma base de Riesz (26) para escrever $\vec{f}$;
- $\phi[n] = \sum_k h_n \phi[2n-k]$;
- $\psi[n] = \sum_k g_n \phi[2n-k]$;
- $H_{j,k}[n] =< f, \phi_{j,k}[n] >$;
- $G_{t,k}[n] =< f, \psi_{t,k}[n] >$;
- $\{0\} \leftarrow ... \subset V_{-1} \subset V_0 \subset V_1 \subset ... \rightarrow L^2$;
- se $f[n] \in V_j \rightarrow f[2n] \in V_{j+1}$;
- $V_{j+1} = V_j \oplus W_j$;
- os coeficientes h_k correspondem ao filtro passa-baixas (aproximação);
- os coeficientes g_k correspondem ao filtro passa-altas (detalhe);
- $h[\cdot]$ e $g[\cdot]$, que são chamados filtros de análise, formam um par de *Quadrature Mirror Filters* - QMF;
- um filtro com k coeficientes é dito filtro de suporte k.

A função ϕ é definida recursivamente, é chamada função *wavelet* e é ortogonal a função *scaling*, a qual é definida recursivamente por dilatações e translações de si mesma(26);

Cada par de filtros de análise, $h[\cdot]$ e $g[\cdot]$, possui uma única função *scaling* e uma única função *wavelet* associadas. A forma de obtenção dessas funções é detalhada por (21) a partir

dos filtros, e vice-versa.

Existem outros filtros associados com $h[\cdot]$ e $g[\cdot]$ chamados *filtros de síntese*. Representados por $\bar{h}[\cdot]$ e $\bar{g}[\cdot]$, são utilizados para inverter a transformada, permitindo recuperar o sinal original a partir de um sinal transformado (16). Tais filtros obedecem às relações das equações 2.11, 2.12 e 2.13, para $k = 0, ..., n-1$, que ficam mais claras através do exemplo na figura 2.6, para filtros de suporte 4.

$$g_k = (-1)^k h_{N-k-1} \quad , \tag{2.11}$$

$$\bar{h}_k = h_{N-k-1} \quad , \tag{2.12}$$

$$\bar{g}_k = (-1)^{k+1} h_k \quad . \tag{2.13}$$

No presente trabalho, a inversão da DWT não se faz necessária, entretanto, é importante que o algoritmo proposto utilize apenas filtros de reconstrução perfeita (*Perfect Reconstruction Filter Bank* - PRFB) (23) (24). Um filtro pode ser considerado PRFB quando $h[\cdot]$, $g[\cdot]$, $\bar{h}[\cdot]$, e $\bar{g}[\cdot]$ mantém as relações descritas nas equações 2.11, 2.12 e 2.13 acima, ou seja, as condições de *anti-aliasing* e *no-distortion*, no domínio Z, representadas nas equações 2.14 e 2.15, respectivamente, são satisfeitas.

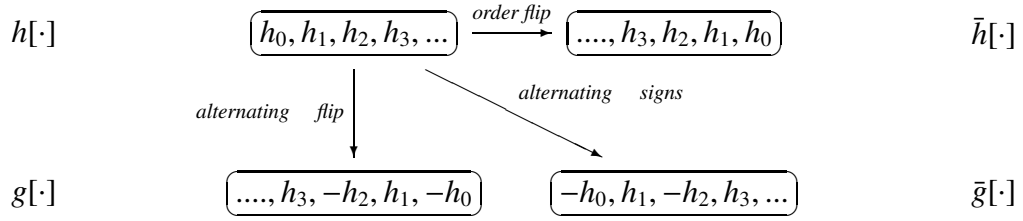

Figura 2.6 – Relação entre os filtros de análise e síntese (extraído de (16))

$$\bar{H}[z] = G[-z] \quad , \quad \bar{G}[z] = -H[-z] \quad . \tag{2.14}$$

$$\bar{H}[z]H[z] + \bar{G}[z]G[z] = 2z^{-N+1} \quad . \tag{2.15}$$

Cálculo da DWT:

Segundo Barbon Junior (16), para calcular da DWT de um sinal é possível empregar, sobre os filtros $h[\cdot]$ e $g[\cdot]$, o algoritmo de Mallat (21). O procedimento de cálculo envolve apenas a multiplicação de duas matrizes para cada nível de transformação. Se $A[\cdot][\cdot]$ é a matriz de coeficientes dos filtros e $B[\cdot]$ é o sinal original, então $C[\cdot] = A[\cdot][\cdot]B[\cdot]$ corresponde ao sinal

transformado, e a disposição dos coeficientes nas matrizes é a seguinte:

$$A[\cdot][\cdot] = \begin{pmatrix} h_0 & h_1 & h_2 & \dots & \dots & \dots & \dots & h_{n-1} & 0 & 0 & 0 & 0 & \dots & \dots & 0 & 0 \\ g_0 & g_1 & g_2 & \dots & \dots & \dots & \dots & g_{n-1} & 0 & 0 & 0 & 0 & \dots & \dots & 0 & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & \dots & \dots & \dots & h_{n-1} & 0 & 0 & 0 & \dots & \dots & 0 & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & \dots & \dots & \dots & g_{n-1} & 0 & 0 & 0 & \dots & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \cdot & & & \cdot & & & \cdot & & & \cdot & & & \cdot & & & \cdot \\ \cdot & & & \cdot & & & \cdot & & & \cdot & & & \cdot & & & \cdot \\ \cdot & & & \cdot & & & \cdot & & & \cdot & & & \cdot & & & \cdot \\ h_{n-1} & 0 & 0 & 0 & \dots & \dots & \dots & 0 & 0 & h_0 & h_1 & \dots & \dots & \dots & h_{n-3} & h_{n-2} \\ g_{n-1} & 0 & 0 & 0 & \dots & \dots & \dots & 0 & 0 & g_0 & g_1 & \dots & \dots & \dots & g_{n-3} & g_{n-2} \end{pmatrix},$$

$$B[\cdot] = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \dots \\ \dots \\ \cdot \\ \cdot \\ \cdot \\ b_{n-2} \\ b_{n-1} \end{pmatrix} \quad e \quad C[\cdot] = \begin{pmatrix} c_0 \\ c_{\frac{n}{2}} \\ c_1 \\ c_{\frac{n}{2}+1} \\ \dots \\ \dots \\ \cdot \\ \cdot \\ \cdot \\ c_{n-1} \\ c_{\frac{n}{2}-1} \end{pmatrix}.$$

Pode-se notar na matriz $A[\cdot][\cdot]$ que dois procedimentos estão embutidos no algoritmo de Mallat: *downsampling* e *wrap-around* (21). Este consiste em fazer com que os coeficientes precedentes dos filtros ocupem as posições iniciais de cada linha, fazendo com que a DWT mantenha sempre o mesmo número de elementos do sinal original.

Famílias de Transformadas *Wavelet*:

Existem diversas famílias de filtros *wavelet*(21), conforme ilustrado na tabela 2.5 e nas figuras 2.7, 2.8 e 2.9. Estas se diferenciam no suporte dos filtros, assim como nas características de resposta em frequência e fase dos mesmos, o que faz com que as funções ϕ e ψ também sofram reflexo de tais diferenças.

Foram utilizadas neste trabalho, para fins de separação dos sinais de áudio em sub-bandas de frequências, as *wavelets* de Haar, Daubechies, Symmlets, Coiflets, Vaidyanathan e Beylkin com diversos suportes, a exemplo de (9, 16).

É importante salientar que todas as famílias aqui empregadas constituem-se de filtros FIR , ou seja, com respostas finitas ao impulso (*Finite Impulse Response* - FIR), isto é, limitadas (22), sendo que as respostas em frequência se aproximam das ideais à medida que o suporte cresce (9).

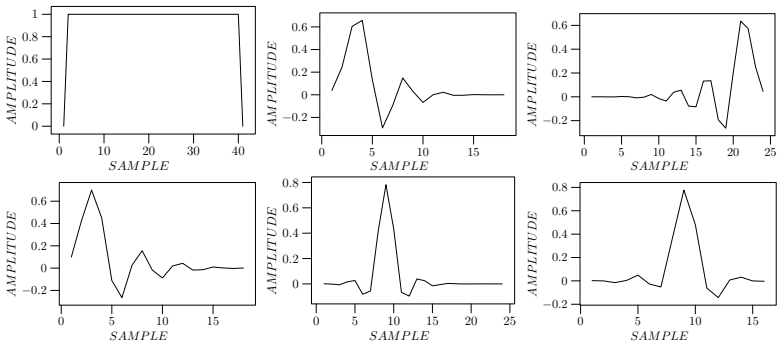

Figura 2.7 - Formato das respostas ao impulso dos filtros *wavelet* de Haar, Daubechies, Vaidyanathan, Beylkin, Coiflet e Symmlet, respectivamente (extraído de (16))

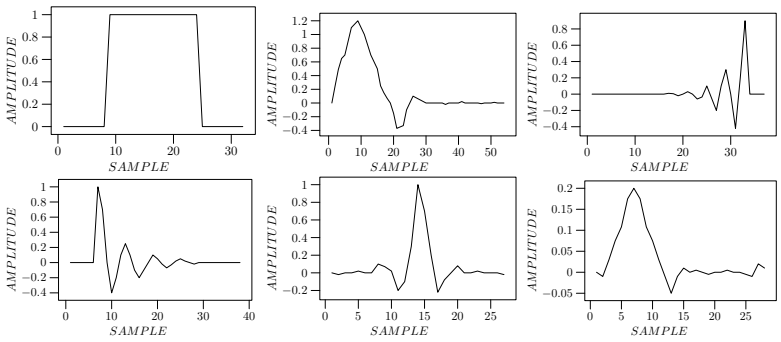

Figura 2.8 - Formatos das funções *scaling* dos filtros *wavelet* de Haar, Daubechies, Vaidyanathan, Beylkin, Coiflet e Symmlet, respectivamente (extraído de (16))

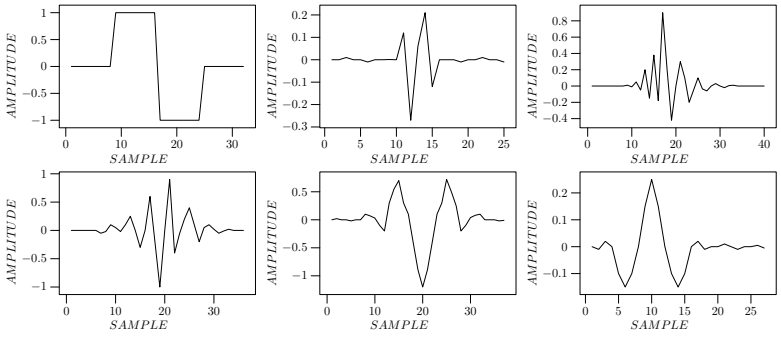

Figura 2.9 - Formatos das funções *wavelet* dos filtros *wavelet* de Haar, Daubechies, Vaidyanathan, Beylkin, Coiflet e Symmlet, respectivamente (extraído de (16))

Tabela 2.5 - Características das famílias de *wavelets* utilizadas no presente trabalho, incluindo a quantidade de momentos da função *wavelet* (9)

Família	Suporte(n)	Fase	Observação	Momentos
Haar	2	linear	é a mais simples das *wavelets*, criada por Alfred Haar (20)(21)	1
Daubechies	par, maior que 4	não linear	resposta ao impulso *maximally flat*, criada por Ingrid Daubechies (20)(21)	$\frac{n}{2}$
Symmlets	par, múltiplo de 8	não linear	resposta ao impulso mais simétrica(20)(21)	$\frac{n}{2} - 2$
Coiflets	par, múltiplo de 6	quase linear	resposta ao impulso quase simétrica, criada por Ronald Coifman (20)(21)	$\frac{n}{2} - 1$
Vaidyanathan	24	não linear	otimizada para voz, criada por P. P. Vaidyanathan (20)(21)	–
Beylkin	18	não linear	otimizada para áudio em geral (20)(21)	$\frac{n}{2} - 2$

Capítulo 3

Descrição da Técnica Proposta

No presente capítulo apresenta-se o sistema proposto, sua estrutura e cada passo dos algoritmos associados para treinamento e execução.

3.1 A Estrutura do Sistema Proposto

O sistema proposto foi dividido em dois módulos distintos, descritos ao final do capítulo. O primeiro módulo é responsável pelo treinamento do sistema de classificação com base em um banco de assinaturas coletadas por meio de um dispositivo com microfone embutido, que está detalhado na seção de Materiais e Métodos. Já o segundo, corresponde ao módulo de aplicação, que funciona com base no sistema previamente treinado, no qual sinais de assinaturas aleatórias foram submetidos a fim de efetuar os testes de reconhecimento biométrico.

Conforme se pode observar na figura 3.1, o módulo de treinamento é responsável por treinar, de maneira isolada e com base numa abordagem semi-supervisionada, diversas SVMs, sendo uma para cada indivíduo pertencente à base de dados de assinaturas. Este é um diferencial do presente trabalho em relação aos esquemas tradicionais encontrados na literatura. Assim, cada SVM comporta-se como uma função que recebe dados de entrada, maximizando um valor de saída para determinados padrões e minimizando-o para outros. Isoladamente, cada SVM pode ser vista como um sistema discriminativo de reconhecimento de padrões.

O módulo de treinamento recebe os arquivos de assinatura e o nome do indivíduo ao qual as referidas assinaturas pertencem. Dessa forma, este módulo executa o treinamento da SVM vinculada ao indivíduo, fixando o valor 1 para a saída de cada assinatura pertencente ao indivíduo atual, do qual, por convenção, é considerada uma assinatura *verdadeira*, ou seja, pertencente ao referido indivíduo. A metodologia de treinamento proposta para cada SVM consiste, conforme relatado, apenas em maximizar as suas saídas, aproximando-as do valor 1, e não em criar um plano separador, como ocorre tradicionalmente. Esta técnica, referida como treinamento generativo (7), visa não representar os indivíduos impostores, isto é, os indivíduos que não são o de interesse, por meio de uma quantidade diminuta de exemplos, tendo em vista que o universo inteiro de indivíduos, exceto o de interesse, é considerado como impostor.

Conforme mencionado anteriormente, o esquema adotado para o treinamento das SVMs é semi-supervisionado. Isso significa que, num primeiro momento, apenas os parâmetros de entrada são utilizados, sem qualquer relação com o valor de saída desejado. Em particular, cada uma das SVMs eleva a dimensão do sinal de entrada de P para T, que representam, respectivamente, o número de parâmetros utilizados para cada exemplo de treinamento, e o número de exemplos de treinamento. Com o uso desse artifício, é sabido que existirá sempre uma solução direta para a respectiva formulação. A função *kernel* utilizada nessa etapa é a Gaussiana (equação 3.1) e cada um dos T-ésimos elementos de processamento da camada intermediária é ajustado para apresentar saída máxima (1) somente para o T-ésimo exemplo de treinamento. O leitor pode recorrer à figura 2.4 do capítulo anterior para acompanhar essa explicação.

$$\phi(v) = e^{\frac{-\mathrm{dist}(\vec{x},\vec{y})}{2}} \quad , \tag{3.1}$$

sendo *dist* a distância Euclidiana (7) entre os vetores $\vec{x}$ e $\vec{y}$ de teste e de treinamento.

O módulo de aplicação, ilustrado na figura 3.2, lê o arquivo de assinatura desconhecido, extrai os parâmetros correspondentes, que são os mesmos extraídos no módulo de treinamento. Finalmente, é medida a diferença entre o valor de saída apresentado por cada SVM e o valor 1 (*verdadeiro*). A SVM que apresenta a menor diferença corresponde à saída 1 desejada, retorna o nome do indivíduo vinculado a ela, sendo esta a classificação emitida como resposta do sistema, isto é, a referida assinatura pertence ao indivíduo cujo nome está vinculado à SVM.

A exemplo de Souza (9), o sistema de classificação planejado treina cada uma das SVMs levando em conta apenas os elementos de uma classe particular a ser identificada, ou seja, a classe pertencente ao indivíduo em questão, e distingue-se dos esquemas tradicionais de classificação. Em particular, cada uma das SVMs, isoladamente, em vez de criar uma margem de separação entre elementos de classes distintas, minimiza as distâncias de cada ponto do espaço

até o ponto 1. A rotina EMR do módulo de aplicação forma, então, em associação com o conjunto total de SVMs, um sistema de classificação que leva em conta um procedimento global de minimização, um para cada SVM, escolhendo a melhor solução. A combinação das SVMs com a estrutura de decisão da rotina EMR é uma das chaves do presente trabalho, e permitiu a obtenção de resultados promissores, conforme documentado no próximo capítulo.

As próximas sub-seções detalham os algoritmos utilizados nos módulos de treinamento e de aplicação, respectivamente. Já os parâmetros mencionados acima, tanto para o treinamento quanto para a aplicação do classificador, serão discutidos no próximo capítulo.

3.1.1 Detalhamento do algoritmo proposto para o módulo de treinamento

Como complemento às explicações apresentadas anteriormente, segue a descrição detalhada das etapas do algoritmo proposto para o módulo de treinamento do sistema.

- **INÍCIO**
- **PASSO 1**: (rotina CIT - Consulta Indivíduos de Treinamento): Para cada indivíduo da base de dados:
  - **Repita**:
    - **PASSO 2**: (rotina LDB - Leitura dos Dados Brutos): extrai os "dados brutos" (amostras digitalizadas) do arquivo atual;
    - **PASSO 3**: (rotina CNI - Consulta nome do indivíduo): procurar na base de dados de treinamento a indicação textual de identificação do indivíduo representado pelo arquivo WAV de assinatura em questão;
      - **INÍCIO**
      - **PASSO 3.1.A**: extraem-se as 25 energias de cada banda Bark a serem utilizadas para o treinamento, utilizando-se uma determinada DWT-*packet*, segundo descrito no próximo Capítulo. Em função do nível de decomposição utilizado na DWT-*packet*, conforme descrito no próximo Capítulo, escolheram-se as folhas da decomposição que ofereciam os limites de frequências referentes à banda de interesse;

· **PASSO 3.1.B**: normaliza-se o conjunto de 25 energias, para evitar variações devidas às amplitudes, concatenando-as em um vetor único que representará os parâmetros de entrada do indivíduo atual para a respectiva SVM;

· **FIM;**

– **enquanto** ainda existem arquivos na base de treinamento (rotina UAD - Último Arquivo de Dados);

- **PASSO 4**: (rotina SVM): utiliza os parâmetros para treinar a SVM do indivíduo atual, conforme descrito anteriormente, objetivando o valor 1 na saída;

- **FIM.**

3.1.2 Detalhamento do algoritmo proposto para o módulo de aplicação

- **INÍCIO**

- **PASSO 1**: (rotina LDB): lê o arquivo WAV a ser classificado, extraindo os seus “dados brutos";

- **PASSO 2**: (rotina EPE - Extração dos Parâmetros de Energia): extrai-se, conforme realizado na etapa de treinamento, as 25 energias do sinal, as quais serão consideradas como os parâmetros de entrada do teste;

  – **Para cada SVM treinada:**

    * **PASSO 2.1**: (rotina SVM): injetar os parâmetros extraídos no passo anterior, previamente normalizados, obtendo o valor de saída da SVM atual e adicionando-o ao vetor de resultados VR);

- **PASSO 3**: (rotina EMR - Escolha da Máxima Resposta): a partir do cálculo do elemento que obteve a menor diferença em relação ao valor 1 esperado, ou seja, $max(VR)$, retornando a identificação do usuário.

- **FIM.**

3.2 Implementação

Os módulos de treinamento e de aplicação, os trechos de código referentes à extração dos parâmetros do sinal de assinatura e as etapas de processamento foram implementados em linguagem Java sob ambiente Linux (27). Os códigos-fonte respectivos encontram-se no apêndice II.

O próximo capítulo descreve os testes e resultados obtidos com o algoritmo proposto.

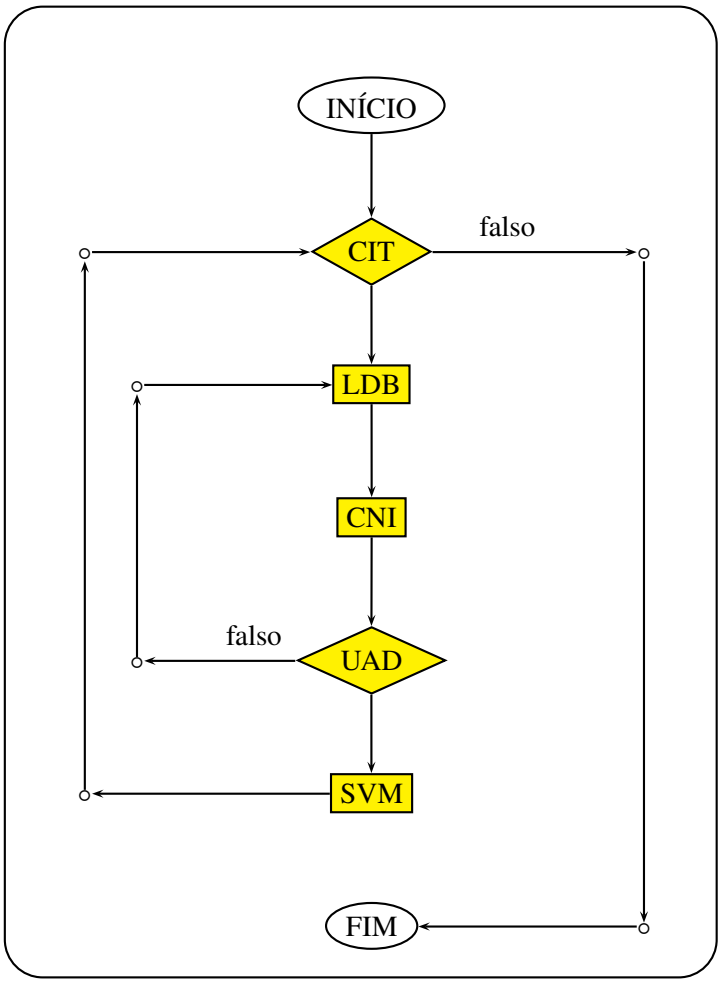

Figura 3.1 - Estrutura básica da estrutura proposta para treinamento do sistema de verificação de assinaturas manuscritas. As siglas estão explicadas ao longo do texto.

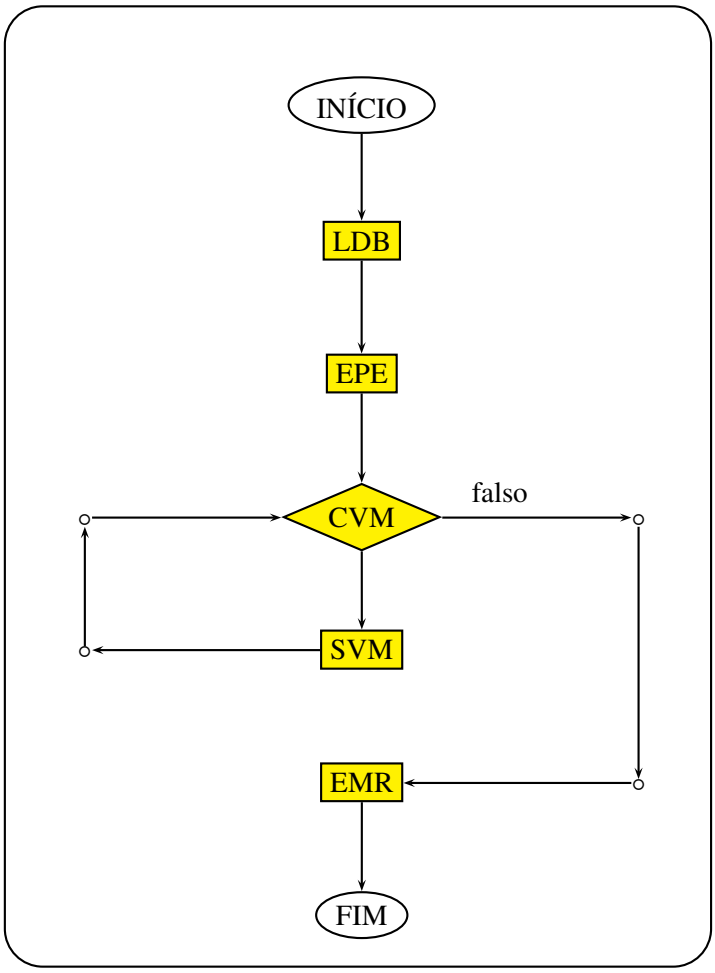

Figura 3.2 - Estrutura básica da estrutura proposta para aplicação pós-treinamento do sistema de verificação de assinaturas manuscritas. As siglas estão explicadas ao longo do texto.

Capítulo 4

Testes e Resultados

Neste capítulo, encontram-se os testes realizados, que foram divididos em diversos grupos, assim como os respectivos resultados obtidos. Cada um dos grupos de testes apresenta uma abordagem diferente em relação ao percentual de amostras alocadas para treinamento e teste do sistema.

4.1 Materiais e Métodos

Todos os sinais de assinaturas manuscritas utilizados foram capturados utilizando uma caneta com um microfone eletreto embutido. Este protótipo foi desenvolvido utilizando uma caneta comum, cabo de som stéreo e um plug P2 stéreo (Figura 4.1). Essas assinaturas, que foram utilizadas para treinamento e teste do sistema proposto, foram coletadas sob as mesmas circunstâncias, isto é, utilizando a mesma caneta e uma prancheta de acrílico comum (Figura 4.2).

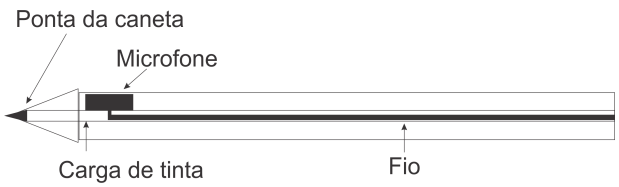

Figura 4.1 – Protótipo da caneta com microfone embutido

Figura 4.2 – Caneta com microfone embutido

O *software* utilizado para a coleta das assinaturas foi desenvolvido em linguagem Java e permitiu um melhor controle e agilidade no processo de coleta das assinaturas, além de garantir que todos os arquivos seriam gravados sob as mesmas circunstâncias de configuração. Em princípio, qualquer *software* de captura de áudio poderia ser empregado para a execução desta tarefa.

A base de dados foi separada por pastas nomeadas com o nome dos indivíduos participantes, e em cada uma das pastas foram armazenados os arquivos WAV, amostrados a uma taxa de 16000 Hz, 16-bits.

Os parâmetros utilizados nas experiências foram, conforme já descrito, as 25 energias obtidas a partir da Escala Bark. Ao serem submetidos ao processo de treinamento, os dados dos arquivos WAV são convertidos em um vetor de *double*, tratados e transformados em vetores de 25 valores. Eles são a base para treinamento da SVM, sendo que, a partir das SVMs de cada participante, o sistema faz a comparação entre suas respostas a partir da entrada de um arquivo WAV de assinatura desconhecido. A SVM considerada, partencente ao participante atual, é a que retorna o valor mais próximo do esperado, ou seja, o valor 1.

4.2 Bateria de Testes

Abaixo, apresenta-se a bateria completa dos testes realizados, incluindo tabelas e gráficos, assim como os resultados obtidos, cuja discussão e interpretação, que levam às conclusões fi-

nais, encontram-se no próximo capítulo.

O primeiro e o quinto grupo de testes utilizaram 2 arquivos para treinamento, e o restante dos arquivos da base para teste, utilizou vários filtros e suportes *wavelet* diferentes. O segundo e o sexto utilizaram 3 arquivos para treinamento. O terceiro e o sétimo utilizaram 4 arquivos para treinamento e, por fim, o quarto e o último grupo utilizaram 5 arquivos para treinamento, ou seja, 50% das amostras inicialmente coletadas de cada indivíduo participante do projeto. A ideia, aqui, foi comparar os resultados obtidos pela variação do número de entradas em cada SVM de treinamento.

Os testes dos grupos 1 a 4 (tabelas 4.1,4.2,4.3 e 4.4) foram realizados decompondo-se os sinais de entrada e saída até o nível médio $\frac{1}{2}\frac{log(n)}{log(2)}$ e os testes dos grupos 5 a 8 (tabelas 4.5, 4.6, 4.7 e 4.8) tiveram seus sinais decompostos até o nível máximo possível, a exemplo de (9, 15).

Os gráficos das figuras 4.3 e 4.4 apresentam um comparativo entre os percentuais de acertos obtidos com o emprego dos filtros *Wavelet* adotados no presente trabalho. O primeiro utiliza decomposição em nível médio na transformada *wavelet*, e o segundo, em nível máximo.

O próximo capítulo apresenta a discussão detalhada dos resultados apresentados nas tabelas e gráficos citados, assim como as conclusões e trabalhos futuros.

Tabela 4.1 - Testes do **grupo 1 (nível médio)** com 2 arquivos de treinamento, utilizando diversos filtros e suportes *wavelet*.

Filtro	*% de acertos*	*Nº de testes*	*Nº de acertos*
COIF30:	42,37%	118	50
DAUB56:	40,68%	118	48
DAUB58:	40,68%	118	48
DAUB46:	39,83%	118	47
DAUB48:	39,83%	118	47
DAUB54:	39,83%	118	47
DAUB66:	39,83%	118	47
DAUB40:	38,98%	118	46
DAUB44:	38,98%	118	46
DAUB50:	38,98%	118	46
DAUB52:	38,98%	118	46
DAUB60:	38,98%	118	46
DAUB64:	38,98%	118	46
DAUB34:	38,14%	118	45
DAUB36:	38,14%	118	45
DAUB42:	38,14%	118	45
DAUB62:	38,14%	118	45
DAUB68:	38,14%	118	45
DAUB70:	38,14%	118	45
COIF6:	38,14%	118	45
VAIDYANATHAN24:	38,14%	118	45
DAUB4:	37,29%	118	44
DAUB6:	37,29%	118	44
DAUB38:	37,29%	118	44
DAUB72:	37,29%	118	44
DAUB28:	36,44%	118	43
DAUB30:	36,44%	118	43
DAUB32:	36,44%	118	43
SYM16:	36,44%	118	43
DAUB74:	35,59%	118	42
HAAR:	34,75%	118	41
DAUB76:	34,75%	118	41
SYM8:	34,75%	118	41
COIF12:	34,75%	118	41
DAUB8:	33,90%	118	40
DAUB26:	33,90%	118	40
COIF24:	33,90%	118	40
DAUB10:	31,36%	118	37
DAUB14:	31,36%	118	37
DAUB24:	31,36%	118	37
COIF18:	31,36%	118	37
DAUB12:	30,51%	118	36
DAUB16:	30,51%	118	36
DAUB20:	30,51%	118	36
DAUB22:	29,66%	118	35
BEYLKIN18:	28,81%	118	34
DAUB18:	26,27%	118	31

Tabela 4.2 - Testes do **grupo 2 (nível médio)** com 3 arquivos de treinamento, utilizando diversos filtros e suportes *wavelet*.

Filtro	*% de acertos*	*Nº de testes*	*Nº de acertos*
DAUB76:	49,51%	103	51
DAUB70:	47,57%	103	49
DAUB74:	46,60%	103	48
DAUB72:	45,63%	103	47
COIF30:	45,63%	103	47
VAIDYANATHAN24:	45,63%	103	47
DAUB64:	44,66%	103	46
DAUB66:	44,66%	103	46
DAUB68:	44,66%	103	46
DAUB46:	43,69%	103	45
DAUB40:	42,72%	103	44
DAUB42:	42,72%	103	44
DAUB44:	42,72%	103	44
DAUB48:	42,72%	103	44
DAUB62:	42,72%	103	44
DAUB14:	41,75%	103	43
DAUB50:	41,75%	103	43
DAUB54:	41,75%	103	43
COIF24:	41,75%	103	43
DAUB10:	40,78%	103	42
DAUB28:	40,78%	103	42
DAUB30:	40,78%	103	42
DAUB38:	40,78%	103	42
DAUB52:	40,78%	103	42
DAUB60:	40,78%	103	42
COIF6:	40,78%	103	42
DAUB12:	39,81%	103	41
DAUB32:	39,81%	103	41
DAUB58:	39,81%	103	41
DAUB36:	38,83%	103	40
SYM16:	38,83%	103	40
BEYLKIN18:	38,83%	103	40
DAUB26:	37,86%	103	39
DAUB34:	37,86%	103	39
HAAR:	36,89%	103	38
DAUB56:	36,89%	103	38
DAUB4:	35,92%	103	37
DAUB18:	34,95%	103	36
DAUB20:	33,98%	103	35
DAUB24:	33,98%	103	35
DAUB22:	33,01%	103	34
DAUB16:	32,04%	103	33
DAUB6:	31,07%	103	32
COIF18:	31,07%	103	32
SYM8:	27,18%	103	28
DAUB8:	26,21%	103	27
COIF12:	26,21%	103	27

Tabela 4.3 - Testes do **grupo 3 (nível médio)** com 4 arquivos de treinamento, utilizando diversos filtros e suportes *wavelet*.

Filtro	*% de acertos*	*Nº de testes*	*Nº de acertos*
COIF30:	47,73%	88	42
DAUB50:	45,45%	88	40
DAUB56:	45,45%	88	40
BEYLKIN18:	45,45%	88	40
DAUB48:	44,32%	88	39
DAUB52:	44,32%	88	39
DAUB54:	44,32%	88	39
DAUB72:	44,32%	88	39
DAUB74:	44,32%	88	39
DAUB76:	44,32%	88	39
COIF6:	44,32%	88	39
DAUB58:	43,18%	88	38
DAUB68:	43,18%	88	38
VAIDYANATHAN24:	43,18%	88	38
DAUB4:	42,05%	88	37
DAUB46:	42,05%	88	37
DAUB62:	42,05%	88	37
DAUB70:	42,05%	88	37
SYM16:	42,05%	88	37
COIF24:	42,05%	88	37
DAUB24:	40,91%	88	36
DAUB44:	40,91%	88	36
DAUB60:	40,91%	88	36
DAUB6:	39,77%	88	35
DAUB8:	39,77%	88	35
DAUB10:	39,77%	88	35
DAUB26:	39,77%	88	35
DAUB28:	39,77%	88	35
DAUB42:	39,77%	88	35
DAUB64:	39,77%	88	35
DAUB66:	39,77%	88	35
DAUB16:	38,64%	88	34
DAUB30:	38,64%	88	34
DAUB32:	38,64%	88	34
DAUB40:	38,64%	88	34
SYM8:	38,64%	88	34
DAUB12:	37,50%	88	33
DAUB34:	37,50%	88	33
DAUB36:	37,50%	88	33
DAUB38:	37,50%	88	33
COIF12:	37,50%	88	33
HAAR:	36,36%	88	32
DAUB14:	36,36%	88	32
DAUB18:	35,23%	88	31
DAUB22:	35,23%	88	31
COIF18:	34,09%	88	30
DAUB20:	32,95%	88	29

Tabela 4.4 - Testes do **grupo 4 (nível médio)** com 5 arquivos de treinamento, utilizando diversos filtros e suportes *wavelet*.

Filtro	*% de acertos*	*Nº de testes*	*Nº de acertos*
DAUB4:	47,95%	73	35
SYM8:	47,95%	73	35
COIF6:	46,58%	73	34
COIF12:	46,58%	73	34
VAIDYANATHAN24:	46,58%	73	34
DAUB8:	43,84%	73	32
DAUB10:	43,84%	73	32
DAUB24:	43,84%	73	32
BEYLKIN18:	43,84%	73	32
DAUB18:	42,47%	73	31
DAUB22:	42,47%	73	31
DAUB20:	41,10%	73	30
DAUB52:	41,10%	73	30
DAUB70:	41,10%	73	30
HAAR:	39,73%	73	29
DAUB36:	39,73%	73	29
DAUB38:	39,73%	73	29
DAUB40:	39,73%	73	29
DAUB74:	39,73%	73	29
COIF30:	39,73%	73	29
DAUB6:	38,36%	73	28
DAUB12:	38,36%	73	28
DAUB14:	38,36%	73	28
DAUB28:	38,36%	73	28
DAUB30:	38,36%	73	28
DAUB32:	38,36%	73	28
DAUB48:	38,36%	73	28
DAUB50:	38,36%	73	28
DAUB54:	38,36%	73	28
DAUB58:	38,36%	73	28
DAUB16:	36,99%	73	27
DAUB34:	36,99%	73	27
DAUB66:	36,99%	73	27
DAUB68:	36,99%	73	27
DAUB72:	36,99%	73	27
DAUB76:	36,99%	73	27
COIF24:	36,99%	73	27
DAUB42:	35,62%	73	26
DAUB56:	35,62%	73	26
DAUB62:	35,62%	73	26
DAUB64:	35,62%	73	26
COIF18:	35,62%	73	26
DAUB44:	34,25%	73	25
DAUB46:	34,25%	73	25
DAUB26:	32,88%	73	24
DAUB60:	32,88%	73	24
SYM16:	32,88%	73	24

Tabela 4.5 - Testes do **grupo 5 (nível máximo)** com 2 arquivos de treinamento, utilizando diversos filtros e suportes *wavelet*.

Filtro	*% de acertos*	*Nº de testes*	*Nº de acertos*
DAUB22:	84,75%	118	100
DAUB28:	83,90%	118	99
DAUB32:	83,90%	118	99
DAUB40:	83,90%	118	99
DAUB46:	83,90%	118	99
BEYLKIN18:	83,90%	118	99
DAUB20:	83,05%	118	98
DAUB30:	83,05%	118	98
DAUB48:	83,05%	118	98
COIF30:	83,05%	118	98
DAUB24:	82,20%	118	97
DAUB38:	82,20%	118	97
SYM16:	82,20%	118	97
COIF24:	82,20%	118	97
DAUB12:	81,36%	118	96
DAUB14:	81,36%	118	96
DAUB54:	80,51%	118	95
DAUB58:	80,51%	118	95
DAUB66:	80,51%	118	95
COIF18:	80,51%	118	95
VAIDYANATHAN24:	80,51%	118	95
DAUB36:	79,66%	118	94
DAUB44:	79,66%	118	94
DAUB56:	79,66%	118	94
DAUB74:	79,66%	118	94
DAUB76:	79,66%	118	94
DAUB50:	78,81%	118	93
DAUB64:	78,81%	118	93
DAUB42:	77,97%	118	92
DAUB62:	77,97%	118	92
DAUB68:	77,97%	118	92
DAUB70:	77,97%	118	92
DAUB72:	77,97%	118	92
COIF12:	77,97%	118	92
DAUB16:	77,12%	118	91
DAUB52:	77,12%	118	91
DAUB60:	76,27%	118	90
DAUB34:	75,42%	118	89
DAUB18:	74,58%	118	88
DAUB6:	73,73%	118	87
DAUB26:	73,73%	118	87
SYM8:	72,88%	118	86
COIF6:	72,88%	118	86
DAUB10:	69,49%	118	82
DAUB8:	64,41%	118	76
DAUB4:	59,32%	118	70
HAAR:	42,37%	118	50

Tabela 4.6 - Testes do **grupo 6 (nível máximo)** com 3 arquivos de treinamento, utilizando diversos filtros e suportes *wavelet*.

Filtro	*% de acertos*	*Nº de testes*	*Nº de acertos*
DAUB38:	91,26%	103	94
COIF30:	89,32%	103	92
DAUB22:	87,38%	103	90
DAUB30:	87,38%	103	90
DAUB56:	87,38%	103	90
VAIDYANATHAN24:	87,38%	103	90
DAUB20:	86,41%	103	89
DAUB24:	86,41%	103	89
DAUB32:	86,41%	103	89
DAUB48:	86,41%	103	89
DAUB58:	86,41%	103	89
DAUB76:	86,41%	103	89
SYM16:	86,41%	103	89
BEYLKIN18:	86,41%	103	89
DAUB64:	85,44%	103	88
DAUB68:	85,44%	103	88
DAUB16:	84,47%	103	87
DAUB70:	84,47%	103	87
COIF24:	84,47%	103	87
DAUB14:	83,50%	103	86
DAUB26:	83,50%	103	86
DAUB28:	83,50%	103	86
DAUB34:	83,50%	103	86
DAUB50:	83,50%	103	86
DAUB52:	83,50%	103	86
DAUB60:	83,50%	103	86
DAUB66:	83,50%	103	86
COIF18:	83,50%	103	86
DAUB40:	82,52%	103	85
DAUB72:	82,52%	103	85
DAUB74:	82,52%	103	85
DAUB42:	81,55%	103	84
DAUB44:	81,55%	103	84
DAUB46:	81,55%	103	84
DAUB54:	80,58%	103	83
DAUB62:	80,58%	103	83
SYM8:	80,58%	103	83
COIF12:	79,61%	103	82
DAUB10:	78,64%	103	81
DAUB36:	78,64%	103	81
DAUB12:	77,67%	103	80
DAUB6:	76,70%	103	79
DAUB18:	75,73%	103	78
COIF6:	70,87%	103	73
DAUB8:	69,90%	103	72
DAUB4:	64,08%	103	66
HAAR:	60,19%	103	62

Tabela 4.7 - Testes do **grupo 7 (nível máximo)** com 4 arquivos de treinamento, utilizando diversos filtros e suportes *wavelet*.

Filtro	*% de acertos*	*Nº de testes*	*Nº de acertos*
DAUB38:	92,05%	88	81
DAUB68:	92,05%	88	81
DAUB76:	92,05%	88	81
DAUB36:	90,91%	88	80
DAUB54:	90,91%	88	80
DAUB56:	90,91%	88	80
DAUB70:	90,91%	88	80
VAIDYANATHAN24:	90,91%	88	80
DAUB40:	89,77%	88	79
DAUB58:	89,77%	88	79
DAUB22:	88,64%	88	78
DAUB24:	88,64%	88	78
DAUB28:	88,64%	88	78
DAUB30:	88,64%	88	78
DAUB48:	88,64%	88	78
DAUB50:	88,64%	88	78
DAUB52:	88,64%	88	78
DAUB62:	88,64%	88	78
DAUB66:	88,64%	88	78
DAUB72:	88,64%	88	78
DAUB74:	88,64%	88	78
DAUB16:	87,50%	88	77
DAUB34:	87,50%	88	77
DAUB46:	87,50%	88	77
DAUB60:	87,50%	88	77
BEYLKIN18:	87,50%	88	77
DAUB26:	86,36%	88	76
DAUB42:	86,36%	88	76
DAUB44:	86,36%	88	76
DAUB32:	85,23%	88	75
DAUB64:	85,23%	88	75
COIF30:	85,23%	88	75
DAUB14:	84,09%	88	74
DAUB20:	84,09%	88	74
DAUB18:	82,95%	88	73
SYM16:	81,82%	88	72
COIF24:	81,82%	88	72
SYM8:	79,55%	88	70
DAUB6:	78,41%	88	69
DAUB12:	78,41%	88	69
COIF18:	78,41%	88	69
DAUB8:	76,14%	88	67
COIF6:	73,86%	88	65
DAUB10:	72,73%	88	64
COIF12:	72,73%	88	64
DAUB4:	62,50%	88	55
HAAR:	57,95%	88	51

Tabela 4.8 - Testes do **grupo 8 (nível máximo)** com 5 arquivos de treinamento, utilizando diversos filtros e suportes *wavelet*.

Filtro	*% de acertos*	*Nº de testes*	*Nº de acertos*
DAUB68:	91,78%	73	67
DAUB76:	91,78%	73	67
VAIDYANATHAN24:	91,78%	73	67
DAUB54:	90,41%	73	66
DAUB58:	90,41%	73	66
DAUB70:	90,41%	73	66
BEYLKIN18:	90,41%	73	66
DAUB26:	89,04%	73	65
DAUB30:	89,04%	73	65
DAUB32:	89,04%	73	65
DAUB38:	89,04%	73	65
DAUB40:	89,04%	73	65
DAUB44:	89,04%	73	65
DAUB48:	89,04%	73	65
DAUB56:	89,04%	73	65
DAUB66:	89,04%	73	65
DAUB22:	87,67%	73	64
DAUB34:	87,67%	73	64
DAUB36:	87,67%	73	64
DAUB50:	87,67%	73	64
DAUB52:	87,67%	73	64
DAUB60:	87,67%	73	64
DAUB62:	87,67%	73	64
DAUB74:	87,67%	73	64
DAUB24:	86,30%	73	63
DAUB42:	86,30%	73	63
DAUB72:	84,93%	73	62
DAUB16:	83,56%	73	61
DAUB64:	83,56%	73	61
DAUB14:	82,19%	73	60
DAUB18:	80,82%	73	59
DAUB28:	80,82%	73	59
DAUB46:	80,82%	73	59
DAUB8:	79,45%	73	58
COIF24:	79,45%	73	58
COIF30:	79,45%	73	58
SYM16:	78,08%	73	57
SYM8:	76,71%	73	56
DAUB10:	75,34%	73	55
COIF6:	75,34%	73	55
COIF18:	75,34%	73	55
DAUB6:	73,97%	73	54
DAUB20:	72,60%	73	53
DAUB12:	71,23%	73	52
COIF12:	71,23%	73	52
HAAR:	61,64%	73	45
DAUB4:	58,90%	73	43

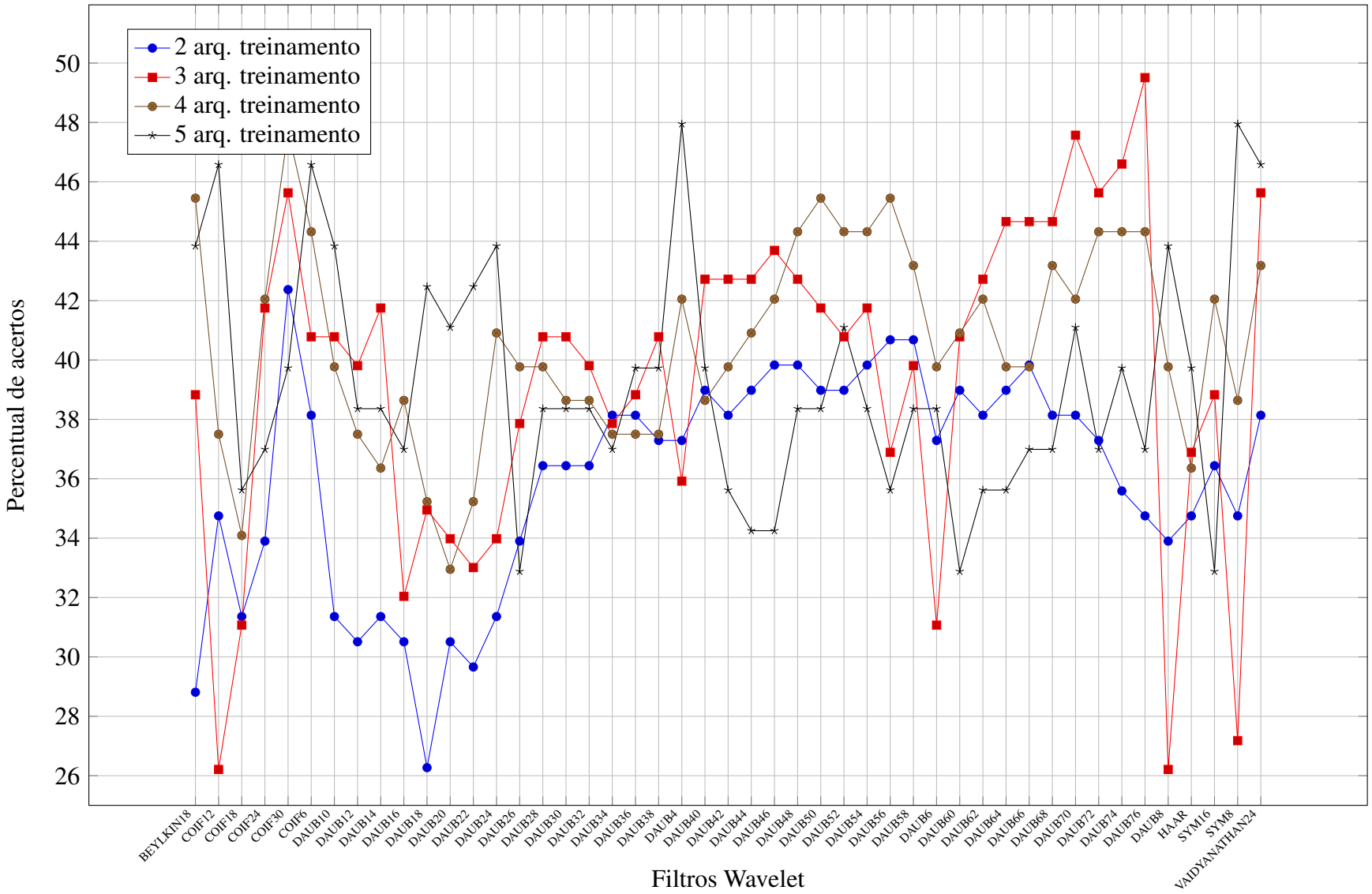

Figura 4.3 - Gráfico de aproveitamento dos filtros utilizando nível médio de decomposição da transformada Wavelet.

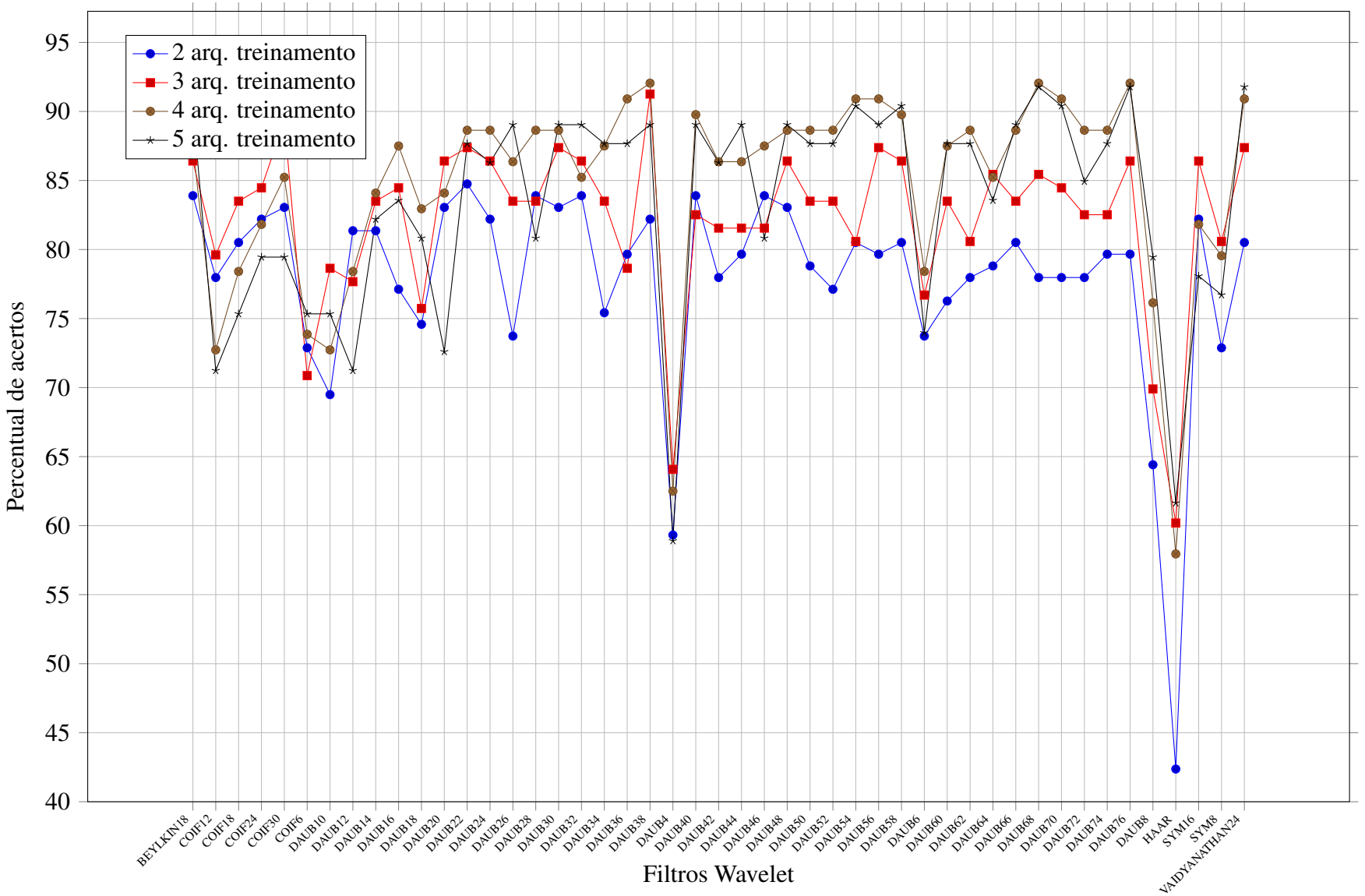

Figura 4.4 - Gráfico de aproveitamento dos filtros utilizando nível máximo de decomposição da transformada Wavelet.

Capítulo 5

Conclusões e Trabalhos Futuros

Após uma revisão bibliográfica de importantes conceitos empregados no presente trabalho, apresentou-se um sistema não-invasivo para biometria, baseado no ruído emitido pela caneta ao assinar, utilizando-se, na análise, os parâmetros extraídos por uma caneta especial com microfone eletreto embutido. O sistema foi projetado com base em uma estrutura composta por uma SVM para cada usuário integrante do banco de assinaturas do sistema com *kernels* Gaussianos, associados com uma rotina de decisão.

Nesta abordagem proposta, cada uma das classes consideradas fica a cargo de uma SVM exclusiva, isto é, cada SVM é treinada levando em conta apenas uma das classes. Essa concepção faz com que a ideia original por trás da SVM seja modificada, pois, em vez de determinar um hiperplano ótimo para separar as classes, o treinamento da forma como foi realizado faz com que um processo de otimização seja implementado, buscando reduzir ao máximo a distância dos pontos no espaço até o ponto objetivo, que foi definido como 1 no caso do indivíduo representado pela referida SVM.

Por fim, no momento da aplicação do sistema para realizar uma classificação, a rotina de decisão retrocitada cuida de verificar qual das SVMs tem sua função objetivo mais próxima dos parâmetros de entrada analisados.

Os testes realizados com os parâmetros extraídos dos arquivos de dados brutos, transformados nas 25 energias predominantes do sinal, mostraram-se eficientes no reconhecimento de usuários por meio dos ruídos produzidos por suas assinaturas. Os resultados mais promissores utilizam 4 arquivos para treinamento.

Embora não tenham sido reportados com mais detalhes, foram realizados diversos testes com outros tipos de parâmetros de entrada no sistema classificador, tais como as entropias do sinal. Essas abordagens não ofereceram, entretanto, resultados nem sequer próximos dos melhores resultados obtidos com a abordagem proposta. Ainda na abordagem proposta, a variação dos *kernels* das SVMs utilizadas não forneceu resultados melhores do que o *kernel* Gaussiano.

Observou-se também que, conforme reportado anteriormente, a variação dos filtros *wavelet* no cálculo das energias das sub-bandas críticas de frequência causou alterações pouco significantes nos resultados, o que mostrou que o classificador projetado se adaptou bem a tais mudanças. De qualquer forma, foi observado que os melhores resultados nas classificações foram obtidos com o uso dos filtros da família Daubechies de suporte a partir de 38 (figura 4.4). Isso significa que as respostas em frequência próximas das ideais e as respostas em fase quase linear, que são as características dessas famílias de filtros, são importantes, isto é, os conteúdos de frequências teoricamente especificados para as bandas críticas devem ser respeitados e, além disso, o atraso quase constante inerente ao processo de filtragem com filtros de fase quase linear também é fator importante. A faixa adotada para o tamanho do suporte dos filtros, que equilibra resolução no tempo e na frequência, deve ser levada em conta.

De maneira geral, foi possível observar uma considerável capacidade de generalização do sistema de classificação proposto. Com base na tabela 4.7 e no gráfico da figura 4.4 do capítulo anterior, é possível notar que o uso de 4 arquivos de assinaturas para treinar o sistema já foi suficiente para a obtenção de um alto índice de acertos nas classificações. É importante também salientar que, quanto menor for o número de assinaturas necessárias para treinar o sistema, menos penoso e mais rápido será o processo de inclusão dos indivíduos no sistema. Além disso, é perfeitamente possível a implementação do sistema aqui proposto em associação com outro sistema de biometria, como o de verificação da imagem da assinatura, também utilizando processamento de sinais. Esta abordagem poderia facilmente resultar em um sistema com maiores taxas de reconhecimento, visto que, nossos resultados utilizando a abordagem proposta já mostram reconhecimento por volta de 90%. Fato este que permitiria uma redução do número de assinaturas para a formação da base de dados de treinamento, para 3 ou 2 arquivos, no caso da adoção de mais de um processo de biometria.

Em termos de trabalhos futuros, pretende-se aprimorar o sistema proposto, aumentando o número de elementos integrantes da base de dados (usuários e suas assinaturas), além de desenvolver em *hardware* (DSP ou FPGA) o projeto de caneta, base para assinatura e aplicativo de gerenciamento. Outra proposta para trabalhos futuros está centrada na inclusão de parâmetros não acústicos, como em (9), que devem permitir melhora no desempenho do sistema proposto.

Referências

1 BRAVO, J.C.L. *Autenticação pessoal baseada no som da assinatura.* 2006. 113p. Dissertação (Mestrado). Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas, 2006.

2 RATHA, N.; BOLLE, R. *Automatic fingerprint recognition systems.* New York: Springer-Verlag, 2004.

3 VIGLIAZZI, D. *Biometria:* medidas de segurança. Florianópolis: Visual Books, 2006.

4 WILSON, C.; BOLLE, R. *Vein pattern recognition:* a privacy-enhancing biometric. Boca Raton: CRC Press,2010.

5 NIXON, M.S.; TAN, T.N.; CHELLAPPA, R. *Human identification based on gait.* New York: Springer, 2006.

6 BORGES, M.E. *Algoritmo genético para otimização de arquiteturas de redes MLP.* Disponível em: <http://www.borges-solutions.com/algoritmo-genetico-para-otimizacao-de-arquiteturas-de-redes-mlp/ >. Acesso em: 20 fev. 2013.

7 DENG, L.; O'SHAUGHNESSY, O. *Speech processing:* a dynamic and optimization-oriented approach. New York: Marcel Dekker, 2003.

8 SILVA, I.N.; SPATTI, D.H.; FLAUZINO R.A. *Redes neurais artificiais:* para engenharia e ciências aplicadas. São Paulo: Artliber, 2010.

9 SOUZA, L.M. *Detecção inteligente de patologias na laringe baseada em máquinas de vetores de suporte e na transformada wavelet.* 2010. 102p. Dissertação (Mestrado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2010.

10 BARANOSKI, F. L. *Verificação da autoria de documentos manuscritos usando SVM.* 2005. 88p. Dissertação (Mestrado) - Programa de Pós-Graduação em Informática Aplicada, Pontifícia Universidade Católica do Paraná, Curitiba, 2005.

11 LORENA, A. C.; CARVALHO, A. C. P. L. F. Uma introdução às support vector machines. *Revista de Informática Teórica e Aplicada*, v.14, n.2, p. 43-67, 2007.

12 BOSI, M; GOLDBERG, R. E. *Introduction to digital audio coding and standards*. Boston-USA: Kruwer Academic Press, 2003.

13 MEYER, P.L. *Probabilidade*: aplicações à estatística. Rio de Janeiro: LTC, 1983.

14 LATHI, B.P. *Sinais e sistemas lineares.* 2. ed. Porto Alegre: Bookman, 2008.

15 FONSECA, E.S. *Wavelets, prediçcão linear e LS-SVM aplicados na análise e classificação de sinais de vozes patológicas*. 2008. 107p. Tese (Doutorado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2008.

16 BARBON JUNIOR, S. *Dynamic time warping baseado na transformada wavelet*. 2007. 113p. Dissertação (Mestrado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2007.

17 VIEIRA, L. S. *Conversão de voz baseada na transformada wavelet*. 2008. 91p. Dissertação (Mestrado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2008.

18 FANTINATO, P. C. *Segmentação de voz baseada na análise fractal e na transformada wavelet*. 2009. 123p. Dissertação (Mestrado) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2009.

19 ADDISON, P. S. *The illustrated Wavelet transform handbook*: introductory theory and applications in science, engineering, medicine and finance. Edinburg-UK: Institute of Physics Publishing, 2002.

20 WALKER, J. S. *A primer on wavelets and their scientific applications*. Whashington-USA: Chapman and Hall/CRC, 1999.

21 JENSEN, A. et al. *Ripples in mathematics*: the discrete wavelet transform. New York-USA: Springer-Verlag, 2000.

22 HAYKIN, S.; VEEN, B. V. *Sinais e sistemas*. Porto Alegre: Bookman, 2002.

23 AKAY, M. *Time-frequency and wavelets in biomedical signal processing*. New York: IEEE Press, 1998.

24 WILLIAMS, J. R.; AMARATUNGA, K. Introduction to wavelets in engineering. *International Journal for Numerical Methods in Engineering*, v. 37, n.1, p. 2365-2388, 1994.

25 LIPSCHUTZ, S. *Álgebra linear*. 3. ed. São Paulo: Makron Books, 1994.

26 STRANG, G.; NGUYEN, T. *Wavelets and filter-banks*. Wellesley: Wellesley-Cambridge Academic Press, 1997.

27 DEITEL H.M. at. al. *Advanced Java 2 platform:* how to program. Upper Saddle River: Prentice Hall, 2001.

Apêndice I - Coeficientes dos filtros *wavelet* utilizados nas experiências.

Coeficientes referentes aos filtros $h[\cdot]$ (passa-baixas de análise).

- Haar :
  {0.7071067, 0.7071067};

- Daub4 :
  { 4.82962913144e-01, 8.36516303737e-01, 2.24143868042e-01, -1.29409522551e-01};

- Daub6 :
  { 3.32670552950e-01, 8.06891509311e-01, 4.59877502118e-01, -1.35011020010e-01, -8.54412738820e-02, 3.52262918857e-02};

- Daub8 :
  { 2.30377813308e-01, 7.14846570552e-01, 6.30880767929e-01, -2.79837694168e-02, -1.87034811719e-01, 3.08413818355e-02, 3.28830116668e-02, -1.05974017850e-02};

- Daub10 :
  { 1.60102397974e-01, 6.03829269797e-01, 7.24308528437e-01, 1.38428145901e-01, -2.42294887066e-01, -3.22448695846e-02, 7.75714938400e-02, -6.24149021279e-03, -1.25807519990e-02, 3.33572528547e-03};

- Daub12 :
  { 1.11540743350e-01, 4.94623890398e-01, 7.51133908021e-01, 3.15250351709e-01, -2.26264693965e-01, -1.29766867567e-01, 9.75016055873e-02, 2.75228655303e-02, -3.15820393174e-02, 5.53842201161e-04, 4.77725751094e-03, -1.07730108530e-03};

- Daub14 :
  { 7.78520540850e-02, 3.96539319481e-01, 7.29132090846e-01, 4.69782287405e-01, -1.43906003928e-01, -2.24036184993e-01, 7.13092192668e-02, 8.06126091510e-02, -3.80299369350e-02, -1.65745416306e-02, 1.25509985560e-02, 4.29577972921e-04, -1.80164070404e-03, 3.53713799974e-04};

- Daub16 :
  { 5.44158422431e-02, 3.12871590914e-01, 6.75630736297e-01, 5.85354683654e-01, -1.58291052563e-02, -2.84015542961e-01, 4.72484573913e-04, 1.28747426620e-01, -1.73693010018e-02, -4.40882539307e-02, 1.39810279173e-02, 8.74609404740e-03, -4.87035299345e-03, -3.91740373376e-04, 6.75449406450e-04, -1.17476784124e-04};

- Daub18 :
  { 3.80779473638e-02, 2.43834674612e-01, 6.04823123690e-01, 6.57288078051e-01, 1.33197385825e-01, -2.93273783279e-01, -9.68407832229e-02, 1.48540749338e-01, 3.07256814793e-02, -6.76328290613e-02, 2.50947114831e-04, 2.23616621236e-02, -4.72320475775e-03, -4.28150368246e-03, 1.84764688305e-03, 2.30385763523e-04, -2.51963188942e-04, 3.93473203162e-05};

- Daub20 :
  { 2.66700579005e-02, 1.88176800077e-01, 5.27201188931e-01, 6.88459039453e-01, 2.81172343660e-01, -2.49846424327e-01, -1.95946274377e-01, 1.27369340335e-01, 9.30573646035e-02, -7.13941471663e-02, -2.94575368218e-02, 3.32126740593e-02, 3.60655356695e-03, -1.07331754833e-02, 1.39535174705e-03, 1.99240529518e-03, -6.85856694959e-04, -1.16466855129e-04, 9.35886703200e-05, -1.32642028945e-05};

- Daub22 :
  { 1.86942977614e-02, 1.44067021150e-01, 4.49899764356e-01, 6.85686774916e-01, 4.11964368947e-01, -1.62275245027e-01, -2.74230846817e-01, 6.60435881966e-02, 1.49812012466e-01, -4.64799551166e-02, -6.64387856950e-02, 3.13350902190e-02, 2.08409043601e-02, -1.53648209062e-02, -3.34085887301e-03, 4.92841765605e-03, -3.08592858815e-04, -8.93023250666e-04, 2.49152523552e-04, 5.44390746993e-05, -3.46349841869e-05, 4.49427427723e-06};

- Daub24 :
  { 1.31122579572e-02, 1.09566272821e-01, 3.77355135214e-01, 6.57198722579e-01, 5.15886478427e-01, -4.47638856537e-02, -3.16178453752e-01, -2.37792572560e-02, 1.82478605927e-01, 5.35956967435e-03, -9.64321200965e-02, 1.08491302558e-02, 4.15462774950e-02, -1.22186490697e-02, -1.28408251983e-02, 6.71149900879e-03, 2.24860724099e-03, -2.17950361862e-03, 6.54512821250e-06, 3.88653062820e-04, -8.85041092082e-05, -2.42415457570e-05, 1.27769522193e-05, -1.52907175806e-06};

- Daub26 :
  { 9.20213353896e-03, 8.28612438729e-02, 3.11996322160e-01, 6.11055851158e-01, 5.88889570431e-01, 8.69857261796e-02, -3.14972907711e-01, -1.24576730750e-01, 1.79476079429e-01, 7.29489336567e-02, -1.05807618187e-01, -2.64884064753e-02, 5.61394771002e-02, 2.37997225405e-03, -2.38314207103e-02, 3.92394144879e-03, 7.25558940161e-03, -2.76191123465e-03, -1.31567391189e-03, 9.32326130867e-04, 4.92515251262e-05, -1.65128988556e-04, 3.06785375793e-05, 1.04419305714e-05, -4.70041647936e-06, 5.22003509845e-07};

- Daub28 :
  { 6.46115346008e-03, 6.23647588493e-02, 2.54850267792e-01, 5.54305617940e-01, 6.31187849104e-01, 2.18670687758e-01, -2.71688552278e-01, -2.18033529993e-01, 1.38395213864e-01, 1.39989016584e-01, -8.67484115681e-02, -7.15489555040e-02, 5.52371262592e-02, 2.69814083079e-02, -3.01853515403e-02, -5.61504953035e-03, 1.27894932663e-02, -7.46218989268e-04, -3.84963886802e-03, 1.06169108560e-03, 7.08021154235e-04, -3.86831947312e-04, -4.17772457703e-05, 6.87550425269e-05, -1.03372091845e-05, -4.38970490178e-06, 1.72499467536e-06, -1.78713996831e-07};

- Daub30 :
  { 4.53853736157e-03, 4.67433948927e-02, 2.06023863986e-01, 4.92631771708e-01, 6.45813140357e-01, 3.39002535454e-01, -1.93204139609e-01, -2.88882596566e-01, 6.52829528487e-02, 1.90146714007e-01, -3.96661765557e-02, -1.11120936037e-01, 3.38771439235e-02, 5.47805505845e-02, -2.57670073284e-02, -2.08100501696e-02, 1.50839180278e-02, 5.10100036040e-03, -6.48773456031e-03, -2.41756490761e-04, 1.94332398038e-03, -3.73482354137e-04, -3.59565244362e-04, 1.55896489920e-04, 2.57926991553e-05, -2.81332962660e-05, 3.36298718173e-06, 1.81127040794e-06, -6.31688232588e-07, 6.13335991330e-08};

- Daub32 :
  { 3.18922092534e-03, 3.49077143236e-02, 1.65064283488e-01, 4.30312722846e-01, 6.37356332083e-01, 4.40290256886e-01, -8.97510894024e-02, -3.27063310527e-01, -2.79182081330e-02, 2.11190693947e-

01, 2.73402637527e-02, -1.32388305563e-01, -6.23972275247e-03, 7.59242360442e-02, -7.58897436885e-03, -3.68883976917e-02, 1.02976596409e-02, 1.39937688598e-02, -6.99001456341e-03, -3.64427962149e-03, 3.12802338120e-03, 4.07896980849e-04, -9.41021749359e-04, 1.14241520038e-04, 1.74787245225e-04, -6.10359662141e-05, -1.39456689882e-05, 1.13366086612e-05, -1.04357134231e-06, -7.36365678545e-07, 2.30878408685e-07, -2.10933963010e-08};

- Daub34 :
  { 2.24180700103e-03, 2.59853937036e-02, 1.31214903307e-01, 3.70350724152e-01, 6.10996615684e-01, 5.18315764056e-01, 2.73149704032e-02, -3.28320748363e-01, -1.26599752215e-01, 1.97310589565e-01, 1.01135489177e-01, -1.26815691778e-01, -5.70914196316e-02, 8.11059866541e-02, 2.23123361781e-02, -4.69224383892e-02, -3.27095553581e-03, 2.27336765839e-02, -3.04298998135e-03, -8.60292152032e-03, 2.96799669152e-03, 2.30120524215e-03, -1.43684530480e-03, -3.28132519409e-04, 4.39465427768e-04, -2.56101095665e-05, -8.20480320245e-05, 2.31868137987e-05, 6.99060098507e-06, -4.50594247722e-06, 3.01654960999e-07, 2.95770093331e-07, -8.42394844600e-08, 7.26749296856e-09};

- Daub36 :
  { 1.57631021844e-03, 1.92885317241e-02, 1.03588465822e-01, 3.14678941337e-01, 5.71826807766e-01, 5.71801654888e-01, 1.47223111969e-01, -2.93654040736e-01, -2.16480934005e-01, 1.49533975565e-01, 1.67081312763e-01, -9.23318841508e-02, -1.06752246659e-01, 6.48872162119e-02, 5.70512477385e-02, -4.45261419029e-02, -2.37332103958e-02, 2.66707059264e-02, 6.26216795430e-03, -1.30514809466e-02, 1.18630033858e-04, 4.94334360546e-03, -1.11873266699e-03, -1.34059629833e-03, 6.28465682965e-04, 2.13581561910e-04, -1.98648552311e-04, -1.53591712353e-07, 3.74123788074e-05, -8.52060253744e-06, -3.33263447888e-06, 1.76871298362e-06, -7.69163268988e-08, -1.17609876702e-07, 3.06883586304e-08, -2.50793445494e-09};

- Daub38 :
  { 1.10866976318e-03, 1.42810984507e-02, 8.12781132654e-02, 2.64388431740e-01, 5.24436377464e-01, 6.01704549127e-01, 2.60894952651e-01, -2.28091394215e-01, -2.85838631755e-01, 7.46522697081e-02, 2.12349743306e-01, -3.35185419023e-02, -1.42785695038e-01, 2.75843506256e-02, 8.69067555558e-02, -2.65012362501e-02, -4.56742262772e-02, 2.16237674095e-02, 1.93755498891e-02, -1.39883886785e-02, -5.86692228101e-03, 7.04074736710e-03, 7.68954359257e-04, -2.68755180070e-03, 3.41808653458e-04, 7.35802520505e-04, -2.60676135678e-04, -1.24600791734e-04, 8.71127046721e-05, 5.10595048707e-06, -1.66401762971e-05, 3.01096431629e-06, 1.53193147669e-06, -6.86275565776e-07, 1.44708829879e-08, 4.63693777578e-08, -1.11640206703e-08, 8.66684883899e-10};

- Daub40 :
  { 7.79953613666e-04, 1.05493946249e-02, 6.34237804590e-02, 2.19942113551e-01, 4.72696185310e-01, 6.10493238938e-01, 3.61502298739e-01, -1.39212088011e-01, -3.26786800434e-01, -1.67270883090e-02, 2.28291050819e-01, 3.98502464577e-02, -1.55458750707e-01, -2.47168273386e-02, 1.02291719174e-01, 5.63224685730e-03, -6.17228996246e-02, 5.87468181181e-03, 3.22942995307e-02, -8.78932492390e-03, -1.38105261371e-02, 6.72162730225e-03, 4.42054238704e-03, -3.58149425960e-03, -8.31562172822e-04, 1.39255961932e-03, -5.34975984399e-05, -3.85104748699e-04, 1.01532889736e-04, 6.77428082837e-05, -3.71058618339e-05, -4.37614386218e-06, 7.24124828767e-06, -1.01199401001e-06, -6.84707959700e-07, 2.63392422627e-07, 2.01432202355e-10, -1.81484324829e-08, 4.05612705555e-09, -2.99883648961e-10};

- Daub42 :
  { 5.48822509852e-04, 7.77663905235e-03, 4.92477715381e-02, 1.81359625440e-01, 4.19687944939e-

01, 6.01506094935e-01, 4.44590451927e-01, -3.57229196172e-02, -3.35664089530e-01, -1.12397071568e-01, 2.11564527680e-01, 1.15233298439e-01, -1.39940424932e-01, -8.17759429808e-02, 9.66003903237e-02, 4.57234057492e-02, -6.49775048937e-02, -1.86538592021e-02, 3.97268354278e-02, 3.35775639033e-03, -2.08920536779e-02, 2.40347092080e-03, 8.98882438197e-03, -2.89133434858e-03, -2.95837403893e-03, 1.71660704063e-03, 6.39418500512e-04, -6.90671117082e-04, -3.19640627768e-05, 1.93664650416e-04, -3.63552025008e-05, -3.49966598498e-05, 1.53548250927e-05, 2.79033053981e-06, -3.09001716454e-06, 3.16609544236e-07, 2.99213663046e-07, -1.00040087903e-07, -2.25401497467e-09, 7.05803354123e-09, -1.47195419765e-09, 1.03880557102e-10};

- Daub44 :
  { 3.86263231491e-04, 5.72185463133e-03, 3.80699372364e-02, 1.48367540890e-01, 3.67728683446e-01, 5.78432731009e-01, 5.07901090622e-01, 7.37245011836e-02, -3.12726580428e-01, -2.00568406104e-01, 1.64093188106e-01, 1.79973187992e-01, -9.71107984091e-02, -1.31768137686e-01, 6.80763143927e-02, 8.45573763668e-02, -5.13642542974e-02, -4.65308118275e-02, 3.69708466206e-02, 2.05867076275e-02, -2.34800013444e-02, -6.21378284936e-03, 1.25647252183e-02, 3.00137398507e-04, -5.45569198615e-03, 1.04426073918e-03, 1.82701049565e-03, -7.70690988123e-04, -4.23787399839e-04, 3.28609414213e-04, 4.34589990453e-05, -9.40522363481e-05, 1.13743496621e-05, 1.73737569575e-05, -6.16672931646e-06, -1.56517913199e-06, 1.29518205731e-06, -8.77987987336e-08, -1.28333622875e-07, 3.76122874933e-08, 1.68017140492e-09, -2.72962314663e-09, 5.33593882166e-10, -3.60211348433e-11};

- Daub46 :
  { 2.71904194128e-04, 4.20274889318e-03, 2.93100036578e-02, 1.20515531783e-01, 3.18450813852e-01, 5.44931147873e-01, 5.51018517241e-01, 1.81392625363e-01, -2.61392148030e-01, -2.71402098607e-01, 9.21254070824e-02, 2.23573658242e-01, -3.30374470942e-02, -1.64011321531e-01, 2.02830745756e-02, 1.12297043618e-01, -2.11262123562e-02, -7.02073915749e-02, 2.17658568344e-02, 3.84953325225e-02, -1.85235136501e-02, -1.75371010030e-02, 1.27519439315e-02, 6.03184065002e-03, -7.07531927370e-03, -1.13486547335e-03, 3.12287644981e-03, -2.46501400516e-04, -1.06123122888e-03, 3.19420492709e-04, 2.56762452007e-04, -1.50021850349e-04, -3.37889483412e-05, 4.42607120310e-05, -2.63520788924e-06, -8.34787556785e-06, 2.39756954684e-06, 8.14757483477e-07, -5.33900540520e-07, 1.85309178563e-08, 5.41754917953e-08, -1.39993549543e-08, -9.47288590181e-10, 1.05044645369e-09, -1.93240511131e-10, 1.25020330235e-11};

- Daub48 :
  { 1.91435800947e-04, 3.08208171490e-03, 2.24823399497e-02, 9.72622358336e-02, 2.72908916067e-01, 5.04371040839e-01, 5.74939221095e-01, 2.80985553233e-01, -1.87271406885e-01, -3.17943078999e-01, 4.77661368434e-03, 2.39237388780e-01, 4.25287296414e-02, -1.71175351370e-01, -3.87771735779e-02, 1.21016303469e-01, 2.09801137091e-02, -8.21616542080e-02, -4.57843624181e-03, 5.13016200399e-02, -4.94470942812e-03, -2.82131070949e-02, 7.66172188164e-03, 1.30499708710e-02, -6.29143537001e-03, -4.74656878632e-03, 3.73604617828e-03, 1.15376493683e-03, -1.69645681897e-03, -4.41618485614e-05, 5.86127059318e-04, -1.18123323796e-04, -1.46007981776e-04, 6.55938863930e-05, 2.18324146046e-05, -2.02288829261e-05, 1.34115775080e-08, 3.90110033859e-06, -8.98025314393e-07, -4.03250775687e-07, 2.16633965327e-07, -5.05764541979e-10, -2.25574038817e-08, 5.15777678967e-09, 4.74837582425e-10, -4.02465864458e-10, 6.99180115763e-11, -4.34278250380e-12};

- Daub50 :
  { 1.34802979347e-04, 2.25695959185e-03, 1.71867412540e-02, 7.80358628721e-02, 2.31693507886e-01, 4.59683415146e-01, 5.81636896746e-01, 3.67885074802e-01, -9.71746409646e-02, -3.36473079641e-01, -8.75876145876e-02, 2.24537819745e-01, 1.18155286719e-01, -1.50560213750e-01, -9.85086152899e-

02, 1.06633805018e-01, 6.67521644940e-02, -7.70841110565e-02, -3.71739628611e-02, 5.36179093987e-02, 1.55426059291e-02, -3.40423204606e-02, -3.07983679484e-03, 1.89228044766e-02, -1.98942578220e-03, -8.86070261804e-03, 2.72693625873e-03, 3.32270777397e-03, -1.84248429020e-03, -8.99977423746e-04, 8.77258193674e-04, 1.15321244046e-04, -3.09880099098e-04, 3.54371452327e-05, 7.90464000396e-05, -2.73304811996e-05, -1.27719529319e-05, 8.99066139306e-06, 5.23282770815e-07, -1.77920133265e-06, 3.21203751886e-07, 1.92280679014e-07, -8.65694173227e-08, -2.61159855611e-09, 9.27922448008e-09, -1.88041575506e-09, -2.22847491022e-10, 1.53590157016e-10, -2.52762516346e-11, 1.50969208282e-12};

- Daub52 :
  { 9.49379575071e-05, 1.65052023353e-03, 1.30975542925e-02, 6.22747440251e-02, 1.95039438716e-01, 4.13292962278e-01, 5.73669043034e-01, 4.39158311789e-01, 1.77407678098e-03, -3.26384593691e-01, -1.74839961289e-01, 1.81291832311e-01, 1.82755409589e-01, -1.04323900285e-01, -1.47977193275e-01, 6.98231861132e-02, 1.06482405249e-01, -5.34485616814e-02, -6.86547596040e-02, 4.22321857963e-02, 3.85357159711e-02, -3.13781103630e-02, -1.77609035683e-02, 2.07349201799e-02, 5.82958055531e-03, -1.17854979061e-02, -5.28738399262e-04, 5.60194723942e-03, -9.39058250473e-04, -2.14553028156e-03, 8.38348805654e-04, 6.16138220457e-04, -4.31955707426e-04, -1.06057474828e-04, 1.57479523860e-04, -5.27779549303e-06, -4.10967399639e-05, 1.07422154087e-05, 7.00007868296e-06, -3.88740016185e-06, -4.65046322064e-07, 7.93921063370e-07, -1.07900423757e-07, -8.90446637016e-08, 3.40779562129e-08, 2.16932825985e-09, -3.77601047853e-09, 6.78004724582e-10, 1.00230319104e-10, -5. 84040818534e-11, 9.13051001637e-12, -5.25187122424e-13};

- Daub54 :
  { 6.68713138543e-05, 1.20553123167e-03, 9.95258878087e-03, 4.94525999829e-02, 1.62922027502e-01, 3.67110214125e-01, 5.53849860990e-01, 4.93406122677e-01, 1.02840855061e-01, -2.89716803314e-01, -2.48264581903e-01, 1.14823019517e-01, 2.27273288414e-01, -3.87864186318e-02, -1.78031740959e-01, 1.57993974602e-02, 1.31197971717e-01, -1.40627515558e-02, -9.10229065295e-02, 1.73110182654e-02, 5.79694057347e-02, -1.85124935619e-02, -3.27390666310e-02, 1.61469669223e-02, 1.56655956489e-02, -1.15771864589e-02, -5.86209634546e-03, 6.85663560968e-03, 1.34262687730e-03, -3.33285446952e-03, 1.45752962593e-04, 1.30117745024e-03, -3.41835122691e-04, -3.87901857410e-04, 2.01971987969e-04, 7.66005838706e-05, -7.71114551779e-05, -3.51748361490e-06, 2.06344264773e-05, -3.90116407063e-06, -3.65750090818e-06, 1.63436962472e-06, 3.05088068625e-07, -3.47246814739e-07, 3.28655896805e-08, 4.02625505286e-08, -1.32133227399e-08, -1.30946560685e-09, 1.52161498477e-09, -2.41552692801e-10, -4.37498622429e-11, 2.21366208806e-11, -3.29579012247e-12, 1.82818835288e-13};

- Daub56 :
  { 4.71080777501e-05, 8.79498515984e-04, 7.54265037764e-03, 3.90926081154e-02, 1.35137914253e-01, 3.22563361285e-01, 5.24998231630e-01, 5.30516293441e-01, 2.00176144045e-01, -2.30498954047e-01, -3.01327809532e-01, 3.28578791633e-02, 2.45808151373e-01, 3.69068853157e-02, -1.82877330732e-01, -4.68382337445e-02, 1.34627567910e-01, 3.44786312750e-02, -9.76853558056e-02, -1.73419228313e-02, 6.77478955019e-02, 3.44801895554e-03, -4.33333686160e-02, 4.43173291006e-03, 2.46880600101e-02, -6.81554976455e-03, -1.20635919682e-02, 5.83881662774e-03, 4.78486311245e-03, -3.72546124707e-03, -1.36037384563e-03, 1.87599866820e-03, 1.41567239314e-04, -7.48674955911e-04, 1.15465606365e-04, 2.29579098223e-04, -8.90390149004e-05, -4.90771341619e-05, 3.64140121105e-05, 4.63866498139e-06, -1.00432604133e-05, 1.24790031757e-06, 1.84036373451e-06, -6.67021547995e-07, -1.75746117320e-07, 1.49066001353e-07, -8.26238731562e-09, -1.78413869087e-08, 5.04404705638e-09, 6.94454032894e-10, -6.07704124722e-10, 8.49222001105e-11, 1.86736726378e-11, -8.36549047125e-12, 1.18885053340e-12, -6.36777235471e-14};

- Daub58 :
  { 3.31896627984e-05, 6.40951680304e-04, 5.70212651777e-03, 3.07735802214e-02, 1.11370116951e-01, 2.80653455970e-01, 4.89758804762e-01, 5.51374432758e-01, 2.89105238335e-01, -1.54028734459e-01, -3.30040948917e-01, -5.57068000729e-02, 2.36105236153e-01, 1.12419174873e-01, -1.60877988594e-01, -1.07845949938e-01, 1.14472295893e-01, 8.32207471624e-02, -8.51254926156e-02, -5.50274895253e-02, 6.34791645842e-02, 3.05315432727e-02, -4.51879812777e-02, -1.29171425542e-02, 2.94704318717e-02, 2.64832730767e-03, -1.70412245736e-02, 1.73788033272e-03, 8.46972549356e-03, -2.55080712778e-03, -3.47379898968e-03, 1.87712092572e-03, 1.08705394222e-03, -1.00077832708e-03, -2.00071136307e-04, 4.11128345474e-04, -2.29201804121e-05, -1.29304484008e-04, 3.64502606856e-05, 2.91334475016e-05, -1.65732839530e-05, -3.59364480402e-06, 4.75060924645e-06, -3.02905459205e-07, -8.97570175063e-07, 2.63389838699e-07, 9.38719741109e-08, -6.28615692201e-08, 1.07659190661e-09, 7.76897885477e-09, -1.89399538617e-09, -3.42680086326e-10, 2.40709945350e-10, -2.94058925076e-11, -7.83250973362e-12, 3.15276241337e-12, -4.28565487006e-13, 2.21919131158e-14};

- Daub60 :
  { 2.33861617273e-05, 4.66637950428e-04, 4.30079716504e-03, 2.41308326715e-02, 9.12383040670e-02, 2.42020670940e-01, 4.50487821853e-01, 5.57572232912e-01, 3.66242683371e-01, -6.61836707759e-02, -3.32966975020e-01, -1.41968513330e-01, 1.99462121580e-01, 1.77829873244e-01, -1.14558219432e-01, -1.57236817959e-01, 7.27786589703e-02, 1.22747746045e-01, -5.38064654582e-02, -8.76586900363e-02, 4.38016646714e-02, 5.67123657447e-02, -3.56733974967e-02, -3.22637589193e-02, 2.70786195952e-02, 1.52879607698e-02, -1.83997438681e-02, -5.29685966613e-03, 1.09156316583e-02, 6.19671756497e-04, -5.53073014819e-03, 8.43384586662e-04, 2.32452009406e-03, -8.60927696811e-04, -7.67878250438e-04, 5.05094823903e-04, 1.72482584235e-04, -2.16171830116e-04, -8.54830546758e-06, 6.98200837080e-05, -1.33971686329e-05, -1.63615247872e-05, 7.25214553589e-06, 2.32754909849e-06, -2.18726767699e-06, 1.09947433852e-08, 4.26166232601e-07, -1.00041468235e-07, -4.76437996513e-08, 2.60544275497e-08, 5.55339786139e-10, -3.33110568046e-09, 6.98486269183e-10, 1.61362297827e-10, -9.46138799727e-11, 1.00010513139e-11, 3.23942863853e-12, -1.18523759210e-12, 1.54399757084e-13, -7.73794263095e-15};

- Daub62 :
  { 1.64801338645e-05, 3.39412203776e-04, 3.23688406862e-03, 1.88536916129e-02, 7.43360930116e-02, 2.07012874485e-01, 4.09192200037e-01, 5.51139840914e-01, 4.29468808206e-01, 2.71692124973e-02, -3.10955118319e-01, -2.17978485523e-01, 1.40178288765e-01, 2.24966711473e-01, -4.99263491604e-02, -1.86962360895e-01, 1.54369884294e-02, 1.45089500931e-01, -8.13983227346e-03, -1.07612773323e-01, 1.09412974523e-02, 7.53536117432e-02, -1.48800266181e-02, -4.86190754648e-02, 1.61541715659e-02, 2.80476193667e-02, -1.42762752777e-02, -1.39005529392e-02, 1.05176394873e-02, 5.51616357331e-03, -6.52085237587e-03, -1.42826422321e-03, 3.39306677671e-03, -6.39790110601e-05, -1.45904174198e-03, 3.43139829690e-04, 4.99881617563e-04, -2.39658346940e-04, -1.24341161725e-04, 1.08958435041e-04, 1.50133572744e-05, -3.63125515786e-05, 4.03452023518e-06, 8.79530134269e-06, -3.03514236589e-06, -1.36906023094e-06, 9.81001542204e-07, 5.32725065697e-08, -1.97592512917e-07, 3.61682651733e-08, 2.32830971382e-08, -1.06152960215e-08, -6.47431168795e-10, 1.40856815102e-09, -2.52404395415e-10, -7.34893003248e-11, 3.69210880887e-11, -3.32700896712e-12, -1.32433491724e-12, 4.44546709629e-13, -5.55944205057e-14, 2.69938287976e-15};

- Daub64 :
  { 1.16146330213e-05, 2.46656690638e-04, 2.43126191957e-03, 1.46810463814e-02, 6.02574991203e-02, 1.75750783639e-01, 3.67509628597e-01, 5.34317919340e-01, 4.77809163733e-01, 1.20630538265e-01, -2.66698181476e-01, -2.77421581558e-01, 6.47133548055e-02, 2.48310642356e-01, 2.46624448396e-

02, -1.92102344708e-01, -4.89951171846e-02, 1.45232079475e-01, 4.44049081999e-02, -1.09456113116e-01, -2.96278725084e-02, 8.08741406384e-02, 1.41061515161e-02, -5.69263140624e-02, -2.38026446493e-03, 3.70514579235e-02, -4.14590766082e-03, -2.16628228363e-02, 6.16752731068e-03, 1.10174007154e-02, -5.41156825727e-03, -4.64921675118e-03, 3.62722464068e-03, 1.46895510046e-03, -1.96474055582e-03, -2.21167872957e-04, 8.67305851845e-04, -1.02453731060e-04, -3.05965442382e-04, 1.05391546173e-04, 8.10367832913e-05, -5.25980928268e-05, -1.29404577940e-05, 1.82426840198e-05, -6.36178153226e-07, -4.55830957626e-06, 1.20288903632e-06, 7.56004762559e-07, -4.28597069315e-07, -5.00336186874e-08, 8.96596631195e-08, -1.21992435948e-08, -1.10438302172e-08, 4.25042231198e-09, 4.38438779994e-10, -5.88109146263e-10, 8.90472379622e-11, 3.26327074133e-11, -1.43091876516e-11, 1.07561065350e-12, 5.36148222961e-13, -1.66380048943e-13, 2.00071530381e-14, -9.42101913953e-16};

- Daub66 :
  { 8.18635831417e-06, 1.79101615370e-04, 1.82270943516e-03, 1.13959433745e-02, 4.86146665317e-02, 1.48186313180e-01, 3.26718130117e-01, 5.09376172514e-01, 5.11254770583e-01, 2.09582350713e-01, -2.04202622398e-01, -3.15997410766e-01, -1.92783394369e-02, 2.45420612119e-01, 9.98515586803e-02, -1.71428099051e-01, -1.10844133116e-01, 1.21967856403e-01, 9.47880880506e-02, -9.11469683513e-02, -7.03024850540e-02, 7.01911439409e-02, 4.57345618938e-02, -5.34712513358e-02, -2.52485829774e-02, 3.86870607602e-02, 1.07032658200e-02, -2.57287617547e-02, -2.16775861735e-03, 1.53169541158e-02, -1.59428878241e-03, -7.95354038705e-03, 2.38906240816e-03, 3.48080095340e-03, -1.86071821445e-03, -1.20430925760e-03, 1.07438069635e-03, 2.72730584733e-04, -4.90832900759e-04, 4.39316625176e-06, 1.78043189825e-04, -4.16043851627e-05, -4.92956442341e-05, 2.42333539881e-05, 9.07080575782e-06, -8.86612136675e-06, -3.60751610287e-07, 2.28837127614e-06, -4.42692340795e-07, -3.98579129198e-07, 1.82244333257e-07, 3.37797270373e-08, -3.98783819851e-08, 3.67286357683e-09, 5.11121185734e-09, -1.67139267725e-09, -2.49640210524e-10, 2.42683310230e-10, -3.04957445394e-11, -1.42023685988e-11, 5.50941472076e-12, -3.34348121895e-13, -2.15248838683e-13, 6.21474024717e-14, -7.19651054536e-15, 3.28937367841e-16};

- Daub68 :
  { 5.77051063273e-06, 1.29947620067e-04, 1.36406139005e-03, 8.81988940388e-03, 3.90488413517e-02, 1.24152482111e-01, 2.87765059233e-01, 4.78478746279e-01, 5.30555099656e-01, 2.90366329507e-01, -1.28246842174e-01, -3.31525301508e-01, -1.03891915515e-01, 2.16907220187e-01, 1.66601750412e-01, -1.27337358223e-01, -1.60924927177e-01, 7.79918469379e-02, 1.34125960271e-01, -5.44829680641e-02, -1.02947596992e-01, 4.35760946496e-02, 7.31852354367e-02, -3.70128384178e-02, -4.74385596452e-02, 3.07397465739e-02, 2.72283507563e-02, -2.36717379228e-02, -1.31439800166e-02, 1.64093741998e-02, 4.71364926099e-03, -1.00455067083e-02, -6.19474884515e-04, 5.33495076875e-03, -7.69212797506e-04, -2.39945394353e-03, 8.58995987436e-04, 8.75199906407e-04, -5.52735576214e-04, -2.32673214023e-04, 2.65077239755e-04, 2.66005001845e-05, -9.91469777078e-05, 1.35311722724e-05, 2.84495141969e-05, -1.05765749425e-05, -5.71082651099e-06, 4.16987175854e-06, 4.97971810142e-07, -1.11630653481e-06, 1.44819570833e-07, 2.02599066666e-07, -7.52670174041e-08, -1.99034650153e-08, 1.74042333293e-08, -8.66574426136e-10, -2.31650194699e-09, 6.44637821032e-10, 1.30041031860e-10, -9.90477453763e-11, 1.00420873546e-11, 6.08012535400e-12, -2.10787910891e-12, 9.79945115821e-14, 8.57919405179e-14, -2.31708370390e-14, 2.58733838193e-15, -1.14894475448e-16};

- Daub70 :
  { 4.06793406114e-06, 9.42146947557e-05, 1.01912268037e-03, 6.80729288431e-03, 3.12362885114e-02, 1.03404455861e-01, 2.51307378994e-01, 4.43592739224e-01, 5.37008427509e-01, 3.60345640518e-01, -4.38838818739e-02, -3.23822864912e-01, -1.81786976766e-01, 1.66041357490e-01, 2.17299289321e-01, -6.52628713106e-02, -1.91919589298e-01, 1.93095446660e-02, 1.55292480396e-01, -4.75268083411e-

03, -1.20585522643e-01, 4.73422917264e-03, 8.99135475707e-02, -9.31855894990e-03, -6.33560374404e-02, 1.32285495850e-02, 4.12546930647e-02, -1.43668397842e-02, -2.41694978016e-02, 1.27664567156e-02, 1.22894360081e-02, -9.57779789923e-03, -5.08599164923e-03, 6.13775458674e-03, 1.42808879407e-03, -3.35764438092e-03, 7.61596943517e-06, 1.54963746970e-03, -3.34669216425e-04, -5.86481031899e-04, 2.64832881996e-04, 1.70001228366e-04, -1.36588307226e-04, -2.97699596284e-05, 5.30414312291e-05, -2.43700152682e-06, -1.57244207727e-05, 4.30804786171e-06, 3.35334586287e-06, -1.89592961769e-06, -3.90393173328e-07, 5.30236861690e-07, -3.70030837820e-08, -9.99039694453e-08, 3.00818865071e-08, 1.08490273378e-08, -7.45811655289e-09, 5.89795131038e-11, 1.03082334548e-09, -2.43354557375e-10, -6.40793825650e-11, 4.00053662725e-11, -3.12563935710e-12, -2.56706547615e-12, 8.01508853368e-13, -2.59795432889e-14, -3.39772085679e-14, 8.62403743472e-15, -9.29801252932e-16, 4.01462871233e-17};

- Daub72 :
  { 2.86792518275e-06, 6.82602867854e-05, 7.60215109966e-04, 5.24029737740e-03, 2.48905656448e-02, 8.56520925952e-02, 2.17756953097e-01, 4.06433697708e-01, 5.32266895260e-01, 4.17875335600e-01, 4.39751975293e-02, -2.94421039589e-01, -2.46807036978e-01, 9.81142041631e-02, 2.46537277608e-01, 7.27851509579e-03, -1.99337205608e-01, -4.58614007463e-02, 1.54106236627e-01, 5.02761800735e-02, -1.18803754310e-01, -3.98808535755e-02, 9.11567822580e-02, 2.50387214495e-02, -6.82090166368e-02, -1.13191003168e-02, 4.85130835478e-02, 1.42497266176e-03, -3.19807206776e-02, 3.98404019871e-03, 1.90635947806e-02, -5.65781324505e-03, -9.99026347328e-03, 5.02298910666e-03, 4.41348483535e-03, -3.48454144540e-03, -1.50307406629e-03, 1.99079377185e-03, 2.77681279571e-04, -9.46340382326e-04, 8.61456575899e-05, 3.69350728496e-04, -1.15511889584e-04, -1.13189946808e-04, 6.69474119693e-05, 2.37510668366e-05, -2.73139082465e-05, -1.18347105998e-06, 8.37221819816e-06, -1.58614578243e-06, -1.87081160285e-06, 8.31142127970e-07, 2.54842352255e-07, -2.45537765843e-07, 2.75324907333e-09, 4.79904346545e-08, -1.15609368881e-08, -5.61278434332e-09, 3.13884169578e-09, 1.09081555371e-10, -4.51254577856e-10, 8.96241820385e-11, 3.03742909811e-11, -1.59971668926e-11, 8.87684628721e-13, 1.07096935711e-12, -3.02928502697e-13, 5.54226318263e-15, 1.33807138629e-14, -3.20462854340e-15, 3.33997198481e-16, -1.40327417537e-17};

- Daub74 :
  { 2.02206086249e-06, 4.94234375062e-05, 5.66241837706e-04, 4.02414036825e-03, 1.97622861538e-02, 7.05848259771e-02, 1.87326331862e-01, 3.68440972400e-01, 5.18167040855e-01, 4.62207553661e-01, 1.30878963233e-01, -2.46180429761e-01, -2.94375915262e-01, 1.96715004523e-02, 2.51523254360e-01, 8.18060283872e-02, -1.81962291778e-01, -1.08451713823e-01, 1.29929646959e-01, 1.01780296838e-01, -9.66075406166e-02, -8.23302119065e-02, 7.50476199483e-02, 5.95674108715e-02, -5.92568156326e-02, -3.82538294793e-02, 4.58079441512e-02, 2.09728005925e-02, -3.35235840641e-02, -8.83349389041e-03, 2.26186515445e-02, 1.69047238348e-03, -1.37639819628e-02, 1.51930577883e-03, 7.38775745285e-03, -2.24805318700e-03, -3.39452327640e-03, 1.81687134380e-03, 1.26393425811e-03, -1.11148486531e-03, -3.28078847088e-04, 5.49053277337e-04, 1.53443902319e-05, -2.20894403245e-04, 4.33672612594e-05, 7.05513878206e-05, -3.09866292761e-05, -1.63916249616e-05, 1.35432771841e-05, 1.84994500311e-06, -4.30994155659e-06, 4.85473139699e-07, 1.00212139929e-06, -3.49494860344e-07, -1.50988538867e-07, 1.10903123221e-07, 5.35065751546e-09, -2.25219383672e-08, 4.22448570636e-09, 2.79397446595e-09, -1.29720500146e-09, -1.03141112909e-10, 1.94616489408e-10, -3.20339824412e-11, -1.39841571553e-11, 6.33495544097e-12, -2.09636319423e-13, -4.42161240987e-13, 1.13805283092e-13, -4.51888960746e-16, -5.24302569188e-15, 1.18901238750e-15, -1.19928033585e-16, 4.90661506493e-18};

- Daub76 :
  { 1.42577664167e-06, 3.57625199426e-05, 4.21170266472e-04, 3.08308811925e-03, 1.56372493475e-

02, 5.78899436128e-02, 1.60071993564e-01, 3.30775781411e-01, 4.96591175311e-01, 4.93356078517e-01, 2.13050571355e-01, -1.82867667708e-01, -3.21675637808e-01, -6.22665060478e-02, 2.32125963835e-01, 1.49985119618e-01, -1.41795685973e-01, -1.59912565158e-01, 8.56381215561e-02, 1.41414734073e-01, -5.65864586307e-02, -1.14731170710e-01, 4.30958954330e-02, 8.72043982620e-02, -3.66051034028e-02, -6.17662087084e-02, 3.19898775315e-02, 4.00549811051e-02, -2.68914938808e-02, -2.31141340205e-02, 2.09046452556e-02, 1.12904972786e-02, -1.47018820653e-02, -4.13130665603e-03, 9.21478503219e-03, 5.62571574840e-04, -5.07131450921e-03, 7.16982182106e-04, 2.40069778189e-03, -8.44862666553e-04, -9.42461407722e-04, 5.81075975053e-04, 2.81763925038e-04, -3.03102046072e-04, -4.55568269666e-05, 1.26204335016e-04, -1.15540910383e-05, -4.17514164854e-05, 1.33417614992e-05, 1.03735918404e-05, -6.45673042846e-06, -1.55084435011e-06, 2.14996026993e-06, -8.48708758607e-08, -5.18773373887e-07, 1.39637754550e-07, 8.40035104689e-08, -4.88475793745e-08, -5.42427480028e-09, 1.03470453927e-08, -1.43632948779e-09, -1.34919775398e-09, 5.26113255735e-10, 6.73233649018e-11, -8.27825652253e-11, 1.10169293459e-11, 6.29153731703e-12, -2.48478923756e-12, 2.62649650406e-14, 1.80866123627e-13, -4.24981781957e-14, -4.56339716212e-16, 2.04509967678e-15, -4.40530704248e-16, 4.30459683955e-17, -1.71615245108e-18};

- sym8:
  {0.032223100604, -0.012603967262, -0.099219543577, 0.297857795606, 0.803738751807, 0.497618667633, -0.029635527646, -0.075765714789};

- sym16:
  {0.001889950333, -0.000302920515, -0.014952258337, 0.003808752014, 0.049137179674, -0.027219029917, -0.051945838108, 0.364441894835, 0.777185751701, 0.481359651258, -0.061273359068, -0.143294238351, 0.007607487325, 0.031695087811, -0.000542132332, -0.003382415951};

- coif6:
  {-0.072732619513, 0.337897662458, 0.852572020212, 0.384864846864, -0.072732619513, -0.015655728135};

- coif12:
  {0.016387336464, -0.041464936782, -0.067372554722, 0.386110066823, 0.812723635450, 0.417005184424, -0.076488599079, -0.059434418647, 0.023680171946, 0.005611434819, -0.001823208871, -0.000720549445};

- coif18:
  {-0.003793512864, 0.007782596427, 0.023452696142, -0.065771911282, -0.061123390003, 0.405176902410, 0.793777222626, 0.428483476378, -0.071799821619, -0.082301927107, 0.034555027573, 0.015880544864, -0.009007976137, -0.002574517689, 0.001117518771, 0.000466216960, -0.000070983303, -0.000034599773};

- coif24:
  {0.000892313669, -0.001629492013, -0.007346166328, 0.016068943965, 0.026682300156, -0.081266699681, -0.056077313317, 0.415308407030, 0.782238930921, 0.434386056491, -0.066627474263, -0.096220442034, 0.039334427123, 0.025082261845, -0.015211731528, -0.005658286687, 0.003751436157, 0.001266561929, -0.000589020756, -0.000259974552, 0.000062339034, 0.000031229876, -0.000003259680, -0.000001784985};

- coif30:
  {-0.000212080840, 0.000358589688, 0.002178236358, -0.004159358782, -0.010131117521, 0.023408156788, 0.028168028974, -0.091920010569, -0.052043163181, 0.421566206733, 0.774289603730, 0.437991626216, -0.062035963969, -0.105574208714, 0.041289208754, 0.032683574270, -0.019761778945, -0.009164231163, 0.006764185449, 0.002433373213, -0.001662863702, -0.000638131343, 0.000302259582, 0.000140541150, -0.000041340432, -0.000021315027, 0.000003734655, 0.000002063762, -0.000000167443, -0.000000095177};

- Beylkin18 :
  {0.099305765374353, 0.424215360812961, 0.699825214056600,
  0.449718251149468, -0.110927598348234, -0.264497231446384, 0.026900308803690, 0.155538731877093,
  -0.017520746266529, -0.088543630622924, 0.019679866044322, 0.042916387274192, -0.017460408696028,
  -0.014365807968852, 0.010040411844631, 0.0014842347824723, -0.002736031626258, 0.0006404853285212};

- Vaidyanathan24:
  {-0.000062906118, 0.000343631905, -0.000453956620, -0.000944897136, 0.002843834547, 0.000708137504,
  -0.008839103409, 0.003153847056, 0.019687215010, -0.014853448005, -0.035470398607, 0.038742619293,
  0.055892523691, -0.077709750902, -0.083928884366, 0.131971661417, 0.135084227129, -0.194450471766,
  -0.263494802488, 0.201612161775, 0.635601059872, 0.572797793211, 0.250184129505, 0.045799334111}.

Apêndice II - Código fonte do sistema proposto.

```
1  // Metodo da classe principal
2  // organiza os processos de treinamento e teste
3  private static void energiasSVM(int num, String[] nomes, List<Filtro> filtros, boolean
       isNivelMaximo) {
4          for (Filtro filtro : filtros) {
5
6              File[] arquivosTreinamento = null;
7              File[] arquivosTeste = null;
8              File[] arquivos = null;
9
10             int encontrados = 0;
11             int arquivosTestados = 0;
12
13             // Criando as SVMs
14             NeuralNetwork[] nn = new NeuralNetwork[nomes.length];
15             Double[] V1 = null;
16             Double[] V2 = null;
17             Double[] V3 = null;
18             Double[] V4 = null;
19
20             // Captura os parametros para a neural
21             int numero_de_amostras_de_cada_exemplo_de_entrada = 25; // 25 energias
22             int numero_de_exemplos_de_treinamento = num;              // valor inicial
23             int numero_de_amostras_de_cada_exemplo_de_saida = 1;
24
25
26             // treinamento
27             for (int n = 0; n < nomes.length; n++) {
28                 arquivosTreinamento = new File("/dados/coleta/Treinamento/" +
                       nomes[n]).listFiles();
29
30                 arquivosTeste = new File("/dados/coleta/Teste/"+nomes[n]).listFiles();
31                 arquivos = new File[arquivosTreinamento.length+arquivosTeste.length];
32
33                 // juntando todos os arquivos em um unico vetor 'arquivos'
34                 System.arraycopy(arquivosTreinamento, 0, arquivos, 0,
                       arquivosTreinamento.length);
35                 System.arraycopy(arquivosTeste, 0, arquivos, arquivosTreinamento.length,
                       arquivosTeste.length);
36
37                 arquivosTreinamento = new File[numero_de_exemplos_de_treinamento];
38                 arquivosTeste = new File[arquivos.length-numero_de_exemplos_de_treinamento];
39
40                 // vetor de treinamento
41                 System.arraycopy(arquivos, 0, arquivosTreinamento, 0,
                       arquivosTreinamento.length);
42
43                 // vetor de testes
44                 System.arraycopy(arquivos, arquivosTreinamento.length, arquivosTeste, 0,
```

```
                    arquivosTeste.length);

                int arq = 0;
                for (File f : arquivosTreinamento) {
                    System.out.println("... " + f.getPath());
                    Wave wave = new Wave(f.getPath());

                    // captura os dados em 'e'
                    Double[] e;
                    if ((e = Serialization.recover("wave_" + f.getName(), filtro.getNome()))
                        == null) {
                        e = Wavelet.energias(wave, filtro, isNivelMaximo);
                        Serialization.persist(e, "wave_" + f.getName(), filtro.getNome());
                    } else {
                        //System.out.println("  treinamento "+nomes[n]+" recuperou
                            wave_"+f.getName());
                    }

                    V2 = new Double[numero_de_exemplos_de_treinamento];
                    Arrays.fill(V2, 1.0);

                    if (arq == 0) {
                        V1 = e;
                    } else {
                        Double[] b = new Double[V1.length + e.length];
                        System.arraycopy(V1, 0, b, 0, V1.length);
                        System.arraycopy(e, 0, b, V1.length, e.length);
                        V1 = b;

                    }
                    arq++;

                }

                nn[n] = new NeuralNetwork(numero_de_amostras_de_cada_exemplo_de_entrada,
                    numero_de_exemplos_de_treinamento,
                    numero_de_amostras_de_cada_exemplo_de_saida);
                nn[n].train(V1, V2); // treina a neural
            }


            // teste
            for (int t = 0; t < nomes.length; t++) {

                arquivosTreinamento = new
                    File("/dados/coleta/Treinamento/"+nomes[t]).listFiles();
                arquivosTeste = new File("/dados/coleta/Teste/"+nomes[t]).listFiles();
                arquivos = new File[arquivosTreinamento.length+arquivosTeste.length];

                // juntando todos os arquivos em um unico vetor 'arquivos'
                System.arraycopy(arquivosTreinamento, 0, arquivos, 0,
                    arquivosTreinamento.length);
                System.arraycopy(arquivosTeste, 0, arquivos, arquivosTreinamento.length,
                    arquivosTeste.length);

                arquivosTreinamento = new File[numero_de_exemplos_de_treinamento];
                arquivosTeste = new File[arquivos.length-numero_de_exemplos_de_treinamento];

                // vetor de treinamento
```

```
96                  System.arraycopy(arquivos, 0, arquivosTreinamento, 0,
                        arquivosTreinamento.length);
97
98                  // vetor de testes
99                  System.arraycopy(arquivos, arquivosTreinamento.length, arquivosTeste, 0,
                        arquivosTeste.length);
100
101                 for (File f : arquivosTeste) {
102                     double menor = 2.0;
103                     int resultado = -1;
104
105                     System.out.println("␣testando␣" + f.getPath() + "...");
106                     Wave wave = new Wave(f.getPath());
107
108                     Double[] e;
109                     if ((e = Serialization.recover("wave_" + f.getName(), filtro.getNome()))
                            == null) {
110                         e = Wavelet.energias(wave, filtro, isNivelMaximo);
111                         Serialization.persist(e, "wave_" + f.getName(), filtro.getNome());
112                     }
113
114                     V3 = e;
115                     V4 = new Double[]{0.0};// uma saida so
116
117                     for (int j = 0; j < nn.length; j++) {
118                         nn[j].pass(V3, V4);
119                         double r = Math.abs(V4[0] - 1);
120
121                         System.out.println("␣==>␣teste␣com␣" + nomes[j] + "=" + r);
122
123                         if (r < menor) {
124                             menor = r;
125                             resultado = j;
126                         }
127                     }
128
129                     // resultado
130                     arquivosTestados++;
131                     System.out.println("␣␣Resultado[" + nomes[resultado] + "]:␣" + menor +
                            "\n\n");
132                     if (nomes[t].equals(nomes[resultado])) {
133                         encontrados++;
134                     }
135                 }
136
137             }
138
139             System.out.println("\n\n␣␣Total␣de␣arquivos␣testados:␣" + arquivosTestados);
140             System.out.println("␣␣Encontrados:␣" + encontrados);
141             double perc = ((double) (encontrados / (double) arquivosTestados) * 100);
142             System.out.println("␣␣Aproveitamento␣com␣filtro␣" + filtro.getNome() + ":␣" +
                    perc + "%");
143             System.out.println("␣␣Numero_de_exemplos_de_treinamento:" +
                    numero_de_exemplos_de_treinamento);
144         }
145     }
```

```
1  // classe abstrata Wavelet
2  // executa os principais processos do presente trabalho, como definicao dos filtros,
       normalizacao dos dados e definicao dos vetores de caracteristicas
3  public abstract class Wavelet {
4
5      private static final List<Filtro> filtros = new ArrayList<Filtro>();
6
7      static {
8          filtros.add(new Filtro("HAAR", new double[]{1 / Math.sqrt(2), 1 / Math.sqrt(2)}));
9          filtros.add(new Filtro("DAUB4", new double[]{4.82962913144e-01, 8.36516303737e-01,
               2.24143868042e-01, -1.29409522551e-01}));
10         filtros.add(new Filtro("DAUB6", new double[]{3.32670552950e-01, 8.06891509311e-01,
               4.59877502118e-01, -1.35011020010e-01, -8.54412738820e-02, 3.52262918857e-02}));
11         filtros.add(new Filtro("DAUB8", new double[]{2.30377813308e-01, 7.14846570552e-01,
               6.30880767929e-01, -2.79837694168e-02, -1.87034811719e-01, 3.08413818355e-02,
               3.28830116668e-02, -1.05974017850e-02}));
12         filtros.add(new Filtro("DAUB10", new double[]{1.60102397974e-01, 6.03829269797e-01,
               7.24308528437e-01, 1.38428145901e-01, -2.42294887066e-01, -3.22448695846e-02,
               7.75714938400e-02, -6.24149021279e-03, -1.25807519990e-02, 3.33572528547e-03}));
13         filtros.add(new Filtro("DAUB12", new double[]{1.11540743350e-01, 4.94623890398e-01,
               7.51133908021e-01, 3.15250351709e-01, -2.26264693965e-01, -1.29766867567e-01,
               9.75016055873e-02, 2.75228655303e-02, -3.15820393174e-02, 5.53842201161e-04,
               4.77725751094e-03, -1.07730108530e-03}));
14         filtros.add(new Filtro("DAUB14", new double[]{7.78520540850e-02, 3.96539319481e-01,
               7.29132090846e-01, 4.69782287405e-01, -1.43906003928e-01, -2.24036184993e-01,
               7.13092192668e-02, 8.06126091510e-02, -3.80299369350e-02, -1.65745416306e-02,
               1.25509985560e-02, 4.29577972921e-04, -1.80164070404e-03, 3.53713799974e-04}));
15         // 7
16         filtros.add(new Filtro("DAUB16", new double[]{5.44158422431e-02, 3.12871590914e-01,
               6.75630736297e-01, 5.85354683654e-01, -1.58291052563e-02, -2.84015542961e-01,
               4.72484573913e-04, 1.28747426620e-01, -1.73693010018e-02, -4.40882539307e-02,
               1.39810279173e-02, 8.74609404740e-03, -4.87035299345e-03, -3.91740373376e-04,
               6.75449406450e-04, -1.17476784124e-04}));
17         filtros.add(new Filtro("DAUB18", new double[]{3.80779473638e-02, 2.43834674612e-01,
               6.04823123690e-01, 6.57288078051e-01, 1.33197385825e-01, -2.93273783279e-01,
               -9.68407832229e-02, 1.48540749338e-01, 3.07256814793e-02, -6.76328290613e-02,
               2.50947114831e-04, 2.23616621236e-02, -4.72320475775e-03, -4.28150368246e-03,
               1.84764688305e-03, 2.30385763523e-04, -2.51963188942e-04, 3.93473203162e-05}));
18         filtros.add(new Filtro("DAUB20", new double[]{2.66700579005e-02, 1.88176800077e-01,
               5.27201188931e-01, 6.88459039453e-01, 2.81172343660e-01, -2.49846424327e-01,
               -1.95946274377e-01, 1.27369340335e-01, 9.30573646035e-02, -7.13941471663e-02,
               -2.94575368218e-02, 3.32126740593e-02, 3.60655356695e-03, -1.07331754833e-02,
               1.39535174705e-03, 1.99240529518e-03, -6.85856694959e-04, -1.16466855129e-04,
               9.35886703200e-05, -1.32642028945e-05}));
19         filtros.add(new Filtro("DAUB22", new double[]{1.86942977614e-02, 1.44067021150e-01,
               4.49899764356e-01, 6.85686774916e-01, 4.11964368947e-01, -1.62275245027e-01,
               -2.74230846817e-01, 6.60435881966e-02, 1.49812012466e-01, -4.64799551166e-02,
               -6.64387856950e-02, 3.13350902190e-02, 2.08409043601e-02, -1.53648209062e-02,
               -3.34085887301e-03, 4.92841765605e-03, -3.08592858815e-04, -8.93023250666e-04,
               2.49152523552e-04, 5.44390746993e-05, -3.46349841869e-05, 4.49427427723e-06}));
20         filtros.add(new Filtro("DAUB24", new double[]{1.31122579572e-02, 1.09566272821e-01,
               3.77355135214e-01, 6.57198722579e-01, 5.15886478427e-01, -4.47638856537e-02,
               -3.16178453752e-01, -2.37792572560e-02, 1.82478605927e-01, 5.35956967435e-03,
               -9.64321200965e-02, 1.08491302558e-02, 4.15462774950e-02, -1.22186490697e-02,
               -1.28408251983e-02, 6.71149900879e-03, 2.24860724099e-03, -2.17950361862e-03,
               6.54512821250e-06, 3.88653062820e-04, -8.85041092082e-05, -2.42415457570e-05,
               1.27769522193e-05, -1.52907175806e-06}));
21         // 12
```

```
22          filtros.add(new Filtro("DAUB26", new double[]{9.20213353896e-03, 8.28612438729e-02,
                3.11996322160e-01, 6.11055851158e-01, 5.88889570431e-01, 8.69857261796e-02,
                -3.14972907711e-01, -1.24576730750e-01, 1.79476079429e-01, 7.29489336567e-02,
                -1.05807618187e-01, -2.64884064753e-02, 5.61394771002e-02, 2.37997225405e-03,
                -2.38314207103e-02, 3.92394144879e-03, 7.25558940161e-03, -2.76191123465e-03,
                -1.31567391189e-03, 9.32326130867e-04, 4.92515251262e-05, -1.65128988556e-04,
                3.06785375793e-05, 1.04419305714e-05, -4.70041647936e-06, 5.22003509845e-07}));
23          filtros.add(new Filtro("DAUB28", new double[]{6.46115346008e-03, 6.23647588493e-02,
                2.54850267792e-01, 5.54305617940e-01, 6.31187849104e-01, 2.18670687758e-01,
                -2.71688552278e-01, -2.18033529993e-01, 1.38395213864e-01, 1.39989016584e-01,
                -8.67484115681e-02, -7.15489555040e-02, 5.52371262592e-02, 2.69814083079e-02,
                -3.01853515403e-02, -5.61504953035e-03, 1.27894932663e-02, -7.46218989268e-04,
                -3.84963886802e-03, 1.06169108560e-03, 7.08021154235e-04, -3.86831947312e-04,
                -4.17772457703e-05, 6.87550425269e-05, -1.03372091845e-05, -4.38970490178e-06,
                1.72499467536e-06, -1.78713996831e-07}));
24          filtros.add(new Filtro("DAUB30", new double[]{4.53853736157e-03, 4.67433948927e-02,
                2.06023863986e-01, 4.92631771708e-01, 6.45813140357e-01, 3.39002535454e-01,
                -1.93204139609e-01, -2.88882596566e-01, 6.52829528487e-02, 1.90146714007e-01,
                -3.96661765557e-02, -1.11120936037e-01, 3.38771439235e-02, 5.47805505845e-02,
                -2.57670073284e-02, -2.08100501696e-02, 1.50839180278e-02, 5.10100036040e-03,
                -6.48773456031e-03, -2.41756490761e-04, 1.94332398038e-03, -3.73482354137e-04,
                -3.59565244362e-04, 1.55896489920e-04, 2.57926991553e-05, -2.81332962660e-05,
                3.36298718173e-06, 1.81127040794e-06, -6.31688232588e-07, 6.13335991330e-08}));
25          filtros.add(new Filtro("DAUB32", new double[]{3.18922092534e-03, 3.49077143236e-02,
                1.65064283488e-01, 4.30312722846e-01, 6.37356332083e-01, 4.40290256886e-01,
                -8.97510894024e-02, -3.27063310527e-01, -2.79182081330e-02, 2.11190693947e-01,
                2.73402637527e-02, -1.32388305563e-01, -6.23972275247e-03, 7.59242360442e-02,
                -7.58897436885e-03, -3.68883976917e-02, 1.02976596409e-02, 1.39937688598e-02,
                -6.99001456341e-03, -3.64427962149e-03, 3.12802338120e-03, 4.07896980849e-04,
                -9.41021749359e-04, 1.14241520038e-04, 1.74787245225e-04, -6.10359662141e-05,
                -1.39456689882e-05, 1.13366086612e-05, -1.04357134231e-06, -7.36365678545e-07,
                2.30878408685e-07, -2.10933963010e-08}));
26          // 16
27          filtros.add(new Filtro("DAUB34", new double[]{2.24180700103e-03, 2.59853937036e-02,
                1.31214903307e-01, 3.70350724152e-01, 6.10996615684e-01, 5.18315764056e-01,
                2.73149704032e-02, -3.28320748363e-01, -1.26599752215e-01, 1.97310589565e-01,
                1.01135489177e-01, -1.26815691778e-01, -5.70914196316e-02, 8.11059866541e-02,
                2.23123361781e-02, -4.69224383892e-02, -3.27095553581e-03, 2.27336765839e-02,
                -3.04298998135e-03, -8.60292152032e-03, 2.96799669152e-03, 2.30120524215e-03,
                -1.43684530480e-03, -3.28132519409e-04, 4.39465427768e-04, -2.56101095665e-05,
                -8.20480320245e-05, 2.31868137987e-05, 6.99060098507e-06, -4.50594247722e-06,
                3.01654960999e-07, 2.95770093331e-07, -8.42394844600e-08, 7.26749296856e-09}));
28          filtros.add(new Filtro("DAUB36", new double[]{1.57631021844e-03, 1.92885317241e-02,
                1.03588465822e-01, 3.14678941337e-01, 5.71826807766e-01, 5.71801654888e-01,
                1.47223111969e-01, -2.93654040736e-01, -2.16480934005e-01, 1.49533975565e-01,
                1.67081312763e-01, -9.23318841508e-02, -1.06752246659e-01, 6.48872162119e-02,
                5.70512477385e-02, -4.45261419029e-02, -2.37332103958e-02, 2.66707059264e-02,
                6.26216795430e-03, -1.30514809466e-02, 1.18630033858e-04, 4.94334360546e-03,
                -1.11873266699e-03, -1.34059629833e-03, 6.28465682965e-04, 2.13581561910e-04,
                -1.98648552311e-04, -1.53591712353e-07, 3.74123788074e-05, -8.52060253744e-06,
                -3.33263447888e-06, 1.76871298362e-06, -7.69163268988e-08, -1.17609876702e-07,
                3.06883586304e-08, -2.50793445494e-09}));
29          filtros.add(new Filtro("DAUB38", new double[]{1.10866976318e-03, 1.42810984507e-02,
                8.12781132654e-02, 2.64388431740e-01, 5.24436377464e-01, 6.01704549127e-01,
                2.60894952651e-01, -2.28091394215e-01, -2.85838631755e-01, 7.46522697081e-02,
                2.12349743306e-01, -3.35185419023e-02, -1.42785695038e-01, 2.75843506256e-02,
                8.69067555558e-02, -2.65012362501e-02, -4.56742262772e-02, 2.16237674095e-02,
                1.93755498891e-02, -1.39883886785e-02, -5.86692228101e-03, 7.04074736710e-03,
                7.68954359257e-04, -2.68755180070e-03, 3.41808653458e-04, 7.35802520505e-04,
```

```
            -2.60676135678e-04, -1.24600791734e-04, 8.71127046721e-05, 5.10595048707e-06,
            -1.66401762971e-05, 3.01096431629e-06, 1.53193147669e-06, -6.86275565776e-07,
            1.44708829879e-08, 4.63693777578e-08, -1.11640206703e-08, 8.66684883899e-10}));
30          filtros.add(new Filtro("DAUB40", new double[]{7.79953613666e-04, 1.05493946249e-02,
            6.34237804590e-02, 2.19942113551e-01, 4.72696185310e-01, 6.10493238938e-01,
            3.61502298739e-01, -1.39212088011e-01, -3.26786800434e-01, -1.67270883090e-02,
            2.28291050819e-01, 3.98502464577e-02, -1.55458750707e-01, -2.47168273386e-02,
            1.02291719174e-01, 5.63224685730e-03, -6.17228996246e-02, 5.87468181181e-03,
            3.22942995307e-02, -8.78932492390e-03, -1.38105261371e-02, 6.72162730225e-03,
            4.42054238704e-03, -3.58149425960e-03, -8.31562172822e-04, 1.39255961932e-03,
            -5.34975984399e-05, -3.85104748699e-04, 1.01532889736e-04, 6.77428082837e-05,
            -3.71058618339e-05, -4.37614386218e-06, 7.24124828767e-06, -1.01199401001e-06,
            -6.84707959700e-07, 2.63392422627e-07, 2.01432202355e-10, -1.81484324829e-08,
            4.05612705555e-09, -2.99883648961e-10}));
31          filtros.add(new Filtro("DAUB42", new double[]{5.48822509852e-04, 7.77663905235e-03,
            4.92477715381e-02, 1.81359625440e-01, 4.19687944939e-01, 6.01506094935e-01,
            4.44590451927e-01, -3.57229196172e-02, -3.35664089530e-01, -1.12397071568e-01,
            2.11564527680e-01, 1.15233298439e-01, -1.39940424932e-01, -8.17759429808e-02,
            9.66003903237e-02, 4.57234057492e-02, -6.49775048937e-02, -1.86538592021e-02,
            3.97268354278e-02, 3.35775639033e-03, -2.08920536779e-02, 2.40347092080e-03,
            8.98882438197e-03, -2.89133434858e-03, -2.95837403893e-03, 1.71660704063e-03,
            6.39418500512e-04, -6.90671117082e-04, -3.19640627768e-05, 1.93664650416e-04,
            -3.63552025008e-05, -3.49966598498e-05, 1.53548250927e-05, 2.79033053981e-06,
            -3.09001716454e-06, 3.16609544236e-07, 2.99213663046e-07, -1.00040087903e-07,
            -2.25401497467e-09, 7.05803354123e-09, -1.47195419765e-09, 1.03880557102e-10}));
32          filtros.add(new Filtro("DAUB44", new double[]{3.86263231491e-04, 5.72185463133e-03,
            3.80699372364e-02, 1.48367540890e-01, 3.67728683446e-01, 5.78432731009e-01,
            5.07901090622e-01, 7.37245011836e-02, -3.12726580428e-01, -2.00568406104e-01,
            1.64093188106e-01, 1.79973187992e-01, -9.71107984091e-02, -1.31768137686e-01,
            6.80763143927e-02, 8.45573763668e-02, -5.13642542974e-02, -4.65308118275e-02,
            3.69708466206e-02, 2.05867076275e-02, -2.34800013444e-02, -6.21378284936e-03,
            1.25647252183e-02, 3.00137398507e-04, -5.45569198615e-03, 1.04426073918e-03,
            1.82701049565e-03, -7.70690988123e-04, -4.23787399839e-04, 3.28609414213e-04,
            4.34589990453e-05, -9.40522363481e-05, 1.13743496621e-05, 1.73737569575e-05,
            -6.16672931646e-06, -1.56517913199e-06, 1.29518205731e-06, -8.77987987336e-08,
            -1.28333622875e-07, 3.76122874933e-08, 1.68017140492e-09, -2.72962314663e-09,
            5.33593882166e-10, -3.60211348433e-11}));
33          filtros.add(new Filtro("DAUB46", new double[]{2.71904194128e-04, 4.20274889318e-03,
            2.93100036578e-02, 1.20515531783e-01, 3.18450813852e-01, 5.44931147873e-01,
            5.51018517241e-01, 1.81392625363e-01, -2.61392148030e-01, -2.71402098607e-01,
            9.21254070824e-02, 2.23573658242e-01, -3.30374470942e-02, -1.64011321531e-01,
            2.02830745756e-02, 1.12297043618e-01, -2.11262123562e-02, -7.02073915749e-02,
            2.17658568344e-02, 3.84953325225e-02, -1.85235136501e-02, -1.75371010030e-02,
            1.27519439315e-02, 6.03184065002e-03, -7.07531927370e-03, -1.13486547335e-03,
            3.12287644981e-03, -2.46501400516e-04, -1.06123122888e-03, 3.19420492709e-04,
            2.56762452007e-04, -1.50021850349e-04, -3.37889483412e-05, 4.42607120310e-05,
            -2.63520788924e-06, -8.34787556785e-06, 2.39756954684e-06, 8.14757483477e-07,
            -5.33900540520e-07, 1.85309178563e-08, 5.41754917953e-08, -1.39993549543e-08,
            -9.47288590181e-10, 1.05044645369e-09, -1.93240511131e-10, 1.25020330235e-11}));
34          filtros.add(new Filtro("DAUB48", new double[]{1.91435800947e-04, 3.08208171490e-03,
            2.24823399497e-02, 9.72622358336e-02, 2.72908916067e-01, 5.04371040839e-01,
            5.74939221095e-01, 2.80985553233e-01, -1.87271406885e-01, -3.17943078999e-01,
            4.77661368434e-03, 2.39237388780e-01, 4.25287296414e-02, -1.71175351370e-01,
            -3.87771735779e-02, 1.21016303469e-01, 2.09801137091e-02, -8.21616542080e-02,
            -4.57843624181e-03, 5.13016200399e-02, -4.94470942812e-03, -2.82131070949e-02,
            7.66172188164e-03, 1.30499708710e-02, -6.29143537001e-03, -4.74656878632e-03,
            3.73604617828e-03, 1.15376493683e-03, -1.69645681897e-03, -4.41618485614e-05,
            5.86127059318e-04, -1.18123323796e-04, -1.46007981776e-04, 6.55938863930e-05,
            2.18324146046e-05, -2.02288829261e-05, 1.34115775080e-08, 3.90110033859e-06,
```

```
            -8.98025314393e-07, -4.03250775687e-07, 2.16633965327e-07, -5.05764541979e-10,
            -2.25574038817e-08, 5.15777678967e-09, 4.74837582425e-10, -4.02465864458e-10,
            6.99180115763e-11, -4.34278250380e-12}));
        filtros.add(new Filtro("DAUB50", new double[]{1.34802979347e-04, 2.25695959185e-03,
            1.71867412540e-02, 7.80358628721e-02, 2.31693507886e-01, 4.59683415146e-01,
            5.81636896746e-01, 3.67885074802e-01, -9.71746409646e-02, -3.36473079641e-01,
            -8.75876145876e-02, 2.24537819745e-01, 1.18155286719e-01, -1.50560213750e-01,
            -9.85086152899e-02, 1.06633805018e-01, 6.67521644940e-02, -7.70841110565e-02,
            -3.71739628611e-02, 5.36179093987e-02, 1.55426059291e-02, -3.40423204606e-02,
            -3.07983679484e-03, 1.89228044766e-02, -1.98942578220e-03, -8.86070261804e-03,
            2.72693625873e-03, 3.32270777397e-03, -1.84248429020e-03, -8.99977423746e-04,
            8.77258193674e-04, 1.15321244046e-04, -3.09880099098e-04, 3.54371452327e-05,
            7.90464000396e-05, -2.73304811996e-05, -1.27719529319e-05, 8.99066139306e-06,
            5.23282770815e-07, -1.77920133265e-06, 3.21203751886e-07, 1.92280679014e-07,
            -8.65694173227e-08, -2.61159855611e-09, 9.27922448008e-09, -1.88041575506e-09,
            -2.22847491022e-10, 1.53590157016e-10, -2.52762516346e-11, 1.50969208282e-
12}));
        filtros.add(new Filtro("DAUB52", new double[]{9.49379575071e-05, 1.65052023353e-03,
            1.30975542925e-02, 6.22747440251e-02, 1.95039438716e-01, 4.13292962278e-01,
            5.73669043034e-01, 4.39158311789e-01, 1.77407678098e-03, -3.26384593691e-01,
            -1.74839961289e-01, 1.81291832311e-01, 1.82755409589e-01, -1.04323900285e-01,
            -1.47977193275e-01, 6.98231861132e-02, 1.06482405249e-01, -5.34485616814e-02,
            -6.86547596040e-02, 4.22321857963e-02, 3.85357159711e-02, -3.13781103630e-02,
            -1.77609035683e-02, 2.07349201799e-02, 5.82958055531e-03, -1.17854979061e-02,
            -5.28738399262e-04, 5.60194723942e-03, -9.39058250473e-04, -2.14553028156e-03,
            8.38348805654e-04, 6.16138220457e-04, -4.31955707426e-04, -1.06057474828e-04,
            1.57479523860e-04, -5.27779549303e-06, -4.10967399639e-05, 1.07422154087e-05,
            7.00007868296e-06, -3.88740016185e-06, -4.65046322064e-07, 7.93921063370e-07,
            -1.07900423757e-07, -8.90446637016e-08, 3.40779562129e-08, 2.16932825985e-09,
            -3.77601047853e-09, 6.78004724582e-10, 1.00230319104e-10, -5.84040818534e-
11, 9.13051001637e-12, -5.25187122424e-13}));
        filtros.add(new Filtro("DAUB54", new double[]{6.68713138543e-05, 1.20553123167e-03,
            9.95258878087e-03, 4.94525999829e-02, 1.62922027502e-01, 3.67110214125e-01,
            5.53849860990e-01, 4.93406122677e-01, 1.02840855061e-01, -2.89716803314e-01,
            -2.48264581903e-01, 1.14823019517e-01, 2.27273288414e-01, -3.87864186318e-02,
            -1.78031740959e-01, 1.57993974602e-02, 1.31197971717e-01, -1.40627515558e-02,
            -9.10229065295e-02, 1.73110182654e-02, 5.79694057347e-02, -1.85124935619e-02,
            -3.27390666310e-02, 1.61469669223e-02, 1.56655956489e-02, -1.15771864589e-02,
            -5.86209634546e-03, 6.85663560968e-03, 1.34262687730e-03, -3.33285446952e-03,
            1.45752962593e-04, 1.30117745024e-03, -3.41835122691e-04, -3.87901857410e-04,
            2.01971987969e-04, 7.66005838706e-05, -7.71114551779e-05, -3.51748361490e-06,
            2.06344264773e-05, -3.90116407063e-06, -3.65750090818e-06, 1.63436962472e-06,
            3.05088068625e-07, -3.47246814739e-07, 3.28655896805e-08, 4.02625505286e-08,
            -1.32133227399e-08, -1.30946560685e-09, 1.52161498477e-09, -2.41552692801e-10,
 -4.37498622429e-11, 2.21366208806e-11, -3.29579012247e-12, 1.82818835288e-13}));
        //27
        filtros.add(new Filtro("DAUB56", new double[]{4.71080777501e-05, 8.79498515984e-04,
            7.54265037764e-03, 3.90926081154e-02, 1.35137914253e-01, 3.22563361285e-01,
            5.24998231630e-01, 5.30516293441e-01, 2.00176144045e-01, -2.30498954047e-01,
            -3.01327809532e-01, 3.28578791633e-02, 2.45808151373e-01, 3.69068853157e-02,
            -1.82877330732e-01, -4.68382337445e-02, 1.34627567910e-01, 3.44786312750e-02,
            -9.76853558056e-02, -1.73419228313e-02, 6.77478955019e-02, 3.44801895554e-03,
            -4.33333686160e-02, 4.43173291006e-03, 2.46880600101e-02, -6.81554976455e-03,
            -1.20635919682e-02, 5.83881662774e-03, 4.78486311245e-03, -3.72546124707e-03,
            -1.36037384563e-03, 1.87599866820e-03, 1.41567239314e-04, -7.48674955911e-04,
            1.15465606365e-04, 2.29579098223e-04, -8.90390149004e-05, -4.90771341619e-05,
            3.64140121105e-05, 4.63866498139e-06, -1.00432604133e-05, 1.24790031757e-06,
            1.84036373451e-06, -6.67021547995e-07, -1.75746117320e-07, 1.49066001353e-07,
            -8.26238731562e-09, -1.78413869087e-08, 5.04404705638e-09, 6.94454032894e-10, -
```

```
43  6.07704124722e-10, 8.49222001105e-11, 1.86736726378e-11, -8.36549047125e-12,
        1.18885053340e-12, -6.36777235471e-14}));
44          filtros.add(new Filtro("DAUB58", new double[]{3.31896627984e-05, 6.40951680304e-04,
                5.70212651777e-03, 3.07735802214e-02, 1.11370116951e-01, 2.80653455970e-01,
                4.89758804762e-01, 5.51374432758e-01, 2.89105238335e-01, -1.54028734459e-01,
                -3.30040948917e-01, -5.57068000729e-02, 2.36105236153e-01, 1.12419174873e-01,
                -1.60877988594e-01, -1.07845949938e-01, 1.14472295893e-01, 8.32207471624e-02,
                -8.51254926156e-02, -5.50274895253e-02, 6.34791645842e-02, 3.05315432727e-02,
                -4.51879812777e-02, -1.29171425542e-02, 2.94704318717e-02, 2.64832730767e-03,
                -1.70412245736e-02, 1.73788033272e-03, 8.46972549356e-03, -2.55080712778e-03,
                -3.47379898968e-03, 1.87712092572e-03, 1.08705394222e-03, -1.00077832708e-03,
                -2.00071136307e-04, 4.11128345474e-04, -2.29201804121e-05, -1.29304484008e-04,
                3.64502606856e-05, 2.91334475016e-05, -1.65732839530e-05, -3.59364480402e-06,
                4.75060924645e-06, -3.02905459205e-07, -8.97570175063e-07, 2.63389838699e-07,
                9.38719741109e-08, -6.28615692201e-08, 1.07659190661e-09, 7.76897885477e-09,
45   -1.89399538617e-09, -3.42680086326e-10, 2.40709945350e-10, -2.94058925076e-11,
        -7.83250973362e-12, 3.15276241337e-12, -4.28565487006e-13, 2.21919131158e-14}));
46          filtros.add(new Filtro("DAUB60", new double[]{2.33861617273e-05, 4.66637950428e-04,
                4.30079716504e-03, 2.41308326715e-02, 9.12383040670e-02, 2.42020670940e-01,
                4.50487821853e-01, 5.57572232912e-01, 3.66242683371e-01, -6.61836707759e-02,
                -3.32966975020e-01, -1.41968513330e-01, 1.99462121580e-01, 1.77829873244e-01,
                -1.14558219432e-01, -1.57236817959e-01, 7.27786589703e-02, 1.22747746045e-01,
                -5.38064654582e-02, -8.76586900363e-02, 4.38016646714e-02, 5.67123657447e-02,
                -3.56733974967e-02, -3.22637589193e-02, 2.70786195952e-02, 1.52879607698e-02,
                -1.83997438681e-02, -5.29685966613e-03, 1.09156316583e-02, 6.19671756497e-04,
                -5.53073014819e-03, 8.43384586662e-04, 2.32452009406e-03, -8.60927696811e-04,
                -7.67878250438e-04, 5.05094823903e-04, 1.72482584235e-04, -2.16171830116e-04,
                -8.54830546758e-06, 6.98200837080e-05, -1.33971686329e-05, -1.63615247872e-05,
                7.25214553589e-06, 2.32754909849e-06, -2.18726767699e-06, 1.09947433852e-08,
                4.26166232601e-07, -1.00041468235e-07, -4.76437996513e-08, 2.60544275497e-08,
47   5.55339786139e-10, -3.33110568046e-09, 6.98486269183e-10, 1.61362297827e-10,
        -9.46138799727e-11, 1.00010513139e-11, 3.23942863853e-12, -1.18523759210e-12,
        1.54399757084e-13, -7.73794263095e-15}));
48          filtros.add(new Filtro("DAUB62", new double[]{1.64801338645e-05, 3.39412203776e-04,
                3.23688406862e-03, 1.88536916129e-02, 7.43360930116e-02, 2.07012874485e-01,
                4.09192200037e-01, 5.51139840914e-01, 4.29468808206e-01, 2.71692124973e-02,
                -3.10955118319e-01, -2.17978485523e-01, 1.40178288765e-01, 2.24966711473e-01,
                -4.99263491604e-02, -1.86962360895e-01, 1.54369884294e-02, 1.45089500931e-01,
                -8.13983227346e-03, -1.07612773323e-01, 1.09412974523e-02, 7.53536117432e-02,
                -1.48800266181e-02, -4.86190754648e-02, 1.61541715659e-02, 2.80476193667e-02,
                -1.42762752777e-02, -1.39005529392e-02, 1.05176394873e-02, 5.51616357331e-03,
                -6.52085237587e-03, -1.42826422321e-03, 3.39306677671e-03, -6.39790110601e-05,
                -1.45904174198e-03, 3.43139829690e-04, 4.99881617563e-04, -2.39658346940e-04,
                -1.24341161725e-04, 1.08958435041e-04, 1.50133572744e-05, -3.63125515786e-05,
                4.03452023518e-06, 8.79530134269e-06, -3.03514236589e-06, -1.36906023094e-06,
                9.81001542204e-07, 5.32725065697e-08, -1.97592512917e-07, 3.61682651733e-08,
49  2.32830971382e-08, -1.06152960215e-08, -6.47431168795e-10, 1.40856815102e-09,
        -2.52404395415e-10, -7.34893003248e-11, 3.69210880887e-11, -3.32700896712e-12,
        -1.32433491724e-12, 4.44546709629e-13, -5.55944205057e-14, 2.69938287976e-15}));
50          //31
51          filtros.add(new Filtro("DAUB64", new double[]{1.16146330213e-05, 2.46656690638e-04,
                2.43126191957e-03, 1.46810463814e-02, 6.02574991203e-02, 1.75750783639e-01,
                3.67509628597e-01, 5.34317919340e-01, 4.77809163733e-01, 1.20630538265e-01,
                -2.66698181476e-01, -2.77421581558e-01, 6.47133548055e-02, 2.48310642356e-01,
                2.46624448396e-02, -1.92102344708e-01, -4.89951171846e-02, 1.45232079475e-01,
                4.44049081999e-02, -1.09456113116e-01, -2.96278725084e-02, 8.08741406384e-02,
                1.41061515161e-02, -5.69263140624e-02, -2.38026446493e-03, 3.70514579235e-02,
                -4.14590766082e-03, -2.16628228363e-02, 6.16752731068e-03, 1.10174007154e-02,
                -5.41156825727e-03, -4.64921675118e-03, 3.62722464068e-03, 1.46895510046e-03,
```

```
            -1.96474055582e-03, -2.21167872957e-04, 8.67305851845e-04, -1.02453731060e-04,
            -3.05965442382e-04, 1.05391546173e-04, 8.10367832913e-05, -5.25980928268e-05,
            -1.29404577940e-05, 1.82426840198e-05, -6.36178153226e-07, -4.55830957626e-06,
            1.20288903632e-06, 7.56004762559e-07, -4.28597069315e-07, -5.00336186874e-
52  08, 8.96596631195e-08, -1.21992435948e-08, -1.10438302172e-08, 4.25042231198e-09,
        4.38438779994e-10, -5.88109146263e-10, 8.90472379622e-11, 3.26327074133e-11,
        -1.43091876516e-11, 1.07561065350e-12, 5.36148222961e-13, -1.66380048943e-13,
        2.00071530381e-14, -9.42101913953e-16}));
53          filtros.add(new Filtro("DAUB66", new double[]{8.18635831417e-06, 1.79101615370e-04,
                1.82270943516e-03, 1.13959433745e-02, 4.86146665317e-02, 1.48186313180e-01,
                3.26718130117e-01, 5.09376172514e-01, 5.11254770583e-01, 2.09582350713e-01,
                -2.04202622398e-01, -3.15997410766e-01, -1.92783394369e-02, 2.45420612119e-01,
                9.98515586803e-02, -1.71428099051e-01, -1.10844133116e-01, 1.21967856403e-01,
                9.47880880506e-02, -9.11469683513e-02, -7.03024850540e-02, 7.01911439409e-02,
                4.57345618938e-02, -5.34712513358e-02, -2.52485829774e-02, 3.86870607602e-02,
                1.07032658200e-02, -2.57287617547e-02, -2.16775861735e-03, 1.53169541158e-02,
                -1.59428878241e-03, -7.95354038705e-03, 2.38906240816e-03, 3.48080095340e-03,
                -1.86071821445e-03, -1.20430925760e-03, 1.07438069635e-03, 2.72730584733e-04,
                -4.90832900759e-04, 4.39316625176e-06, 1.78043189825e-04, -4.16043851627e-05,
                -4.92956442341e-05, 2.42333539881e-05, 9.07080575782e-06, -8.86612136675e-06,
                -3.60751610287e-07, 2.28837127614e-06, -4.42692340795e-07, -3.98579129198e-
54  07, 1.82244333257e-07, 3.37797270373e-08, -3.98783819851e-08, 3.67286357683e-09,
        5.11121185734e-09, -1.67139267725e-09, -2.49640210524e-10, 2.42683310230e-10,
        -3.04957445394e-11, -1.42023685988e-11, 5.50941472076e-12, -3.34348121895e-13,
        -2.15248838683e-13, 6.21474024717e-14, -7.19651054536e-15, 3.28937367841e-16}));
55          filtros.add(new Filtro("DAUB68", new double[]{5.77051063273e-06, 1.29947620067e-04,
                1.36406139005e-03, 8.81988940388e-03, 3.90488413517e-02, 1.24152482111e-01,
                2.87765059233e-01, 4.78478746279e-01, 5.30555099656e-01, 2.90366329507e-01,
                -1.28246842174e-01, -3.31525301508e-01, -1.03891915515e-01, 2.16907220187e-01,
                1.66601750412e-01, -1.27337358223e-01, -1.60924927177e-01, 7.79918469379e-02,
                1.34125960271e-01, -5.44829680641e-02, -1.02947596992e-01, 4.35760946496e-02,
                7.31852354367e-02, -3.70128384178e-02, -4.74385596452e-02, 3.07397465739e-02,
                2.72283507563e-02, -2.36717379228e-02, -1.31439800166e-02, 1.64093741998e-02,
                4.71364926099e-03, -1.00455067083e-02, -6.19474884515e-04, 5.33495076875e-03,
                -7.69212797506e-04, -2.39945394353e-03, 8.58995987436e-04, 8.75199906407e-04,
                -5.52735576214e-04, -2.32673214023e-04, 2.65077239755e-04, 2.66005001845e-05,
                -9.91469777078e-05, 1.35311722724e-05, 2.84495141969e-05, -1.05765749425e-05,
                -5.71082651099e-06, 4.16987175854e-06, 4.97971810142e-07, -1.11630653481e-06,
56   1.44819570833e-07, 2.02599066666e-07, -7.52670174041e-08, -1.99034650153e-08,
        1.74042333293e-08, -8.66574426136e-10, -2.31650194699e-09, 6.44637821032e-10,
        1.30041031860e-10, -9.90477453763e-11, 1.00420873546e-11, 6.08012535400e-12,
        -2.10787910891e-12, 9.79945115821e-14, 8.57919405179e-14, -2.31708370390e-14,
        2.58733838193e-15, -1.14894475448e-16}));
57          filtros.add(new Filtro("DAUB70", new double[]{4.06793406114e-06, 9.42146947557e-05,
                1.01912268037e-03, 6.80729288431e-03, 3.12362885114e-02, 1.03404455861e-01,
                2.51307378994e-01, 4.43592739224e-01, 5.37008427509e-01, 3.60345640518e-01,
                -4.38838818739e-02, -3.23822864912e-01, -1.81786976766e-01, 1.66041357490e-01,
                2.17299289321e-01, -6.52628713106e-02, -1.91919589298e-01, 1.93095446660e-02,
                1.55292480396e-01, -4.75268083411e-03, -1.20585522643e-01, 4.73422917264e-03,
                8.99135475707e-02, -9.31855894990e-03, -6.33560374404e-02, 1.32285495850e-02,
                4.12546930647e-02, -1.43668397842e-02, -2.41694978016e-02, 1.27664567156e-02,
                1.22894360081e-02, -9.57779789923e-03, -5.08599164923e-03, 6.13775458674e-03,
                1.42808879407e-03, -3.35764438092e-03, 7.61596943517e-06, 1.54963746970e-03,
                -3.34669216425e-04, -5.86481031899e-04, 2.64832881996e-04, 1.70001228366e-04,
                -1.36588307226e-04, -2.97699596284e-05, 5.30414312291e-05, -2.43700152682e-06,
                -1.57244207727e-05, 4.30804786171e-06, 3.35334586287e-06, -1.89592961769e-06,
58   -3.90393173328e-07, 5.30236861690e-07, -3.70030837820e-08, -9.99039694453e-08,
        3.00818865071e-08, 1.08490273378e-08, -7.45811655289e-09, 5.89795131038e-11,
        1.03082334548e-09, -2.43354557375e-10, -6.40793825650e-11, 4.00053662725e-11,
```

```
      -3.12563935710e-12, -2.56706547615e-12, 8.01508853368e-13, -2.59795432889e-14,
      -3.39772085679e-14, 8.62403743472e-15, -9.29801252932e-16, 4.01462871233e-17}));
59        filtros.add(new Filtro("DAUB72", new double[]{2.86792518275e-06, 6.82602867854e-05,
              7.60215109966e-04, 5.24029737740e-03, 2.48905656448e-02, 8.56520925952e-02,
              2.17756953097e-01, 4.06433697708e-01, 5.32266895260e-01, 4.17875335600e-01,
              4.39751975293e-02, -2.94421039589e-01, -2.46807036978e-01, 9.81142041631e-02,
              2.46537277608e-01, 7.27851509579e-03, -1.99337205608e-01, -4.58614007463e-02,
              1.54106236627e-01, 5.02761800735e-02, -1.18803754310e-01, -3.98808535755e-02,
              9.11567822580e-02, 2.50387214495e-02, -6.82090166368e-02, -1.13191003168e-02,
              4.85130835478e-02, 1.42497266176e-03, -3.19807206776e-02, 3.98404019871e-03,
              1.90635947806e-02, -5.65781324505e-03, -9.99026347328e-03, 5.02298910666e-03,
              4.41348483535e-03, -3.48454144540e-03, -1.50307406629e-03, 1.99079377185e-03,
              2.77681279571e-04, -9.46340382326e-04, 8.61456575899e-05, 3.69350728496e-04,
              -1.15511889584e-04, -1.13189946808e-04, 6.69474119693e-05, 2.37510668366e-05,
              -2.73139082465e-05, -1.18347105998e-06, 8.37221819816e-06, -1.58614578243e-06, -
60  1.87081160285e-06, 8.31142127970e-07, 2.54842352255e-07, -2.45537765843e-07,
      2.75324907333e-09, 4.79904346545e-08, -1.15609368881e-08, -5.61278434332e-09,
      3.13884169578e-09, 1.09081555371e-10, -4.51254577856e-10, 8.96241820385e-11,
      3.03742909811e-11, -1.59971668926e-11, 8.87684628721e-13, 1.07096935711e-12,
      -3.02928502697e-13, 5.54226318263e-15, 1.33807138629e-14, -3.20462854340e-15,
      3.33997198481e-16, -1.40327417537e-17}));
61        filtros.add(new Filtro("DAUB74", new double[]{2.02206086249e-06, 4.94234375062e-05,
              5.66241837706e-04, 4.02414036825e-03, 1.97622861538e-02, 7.05848259771e-02,
              1.87326331862e-01, 3.68440972400e-01, 5.18167040855e-01, 4.62207553661e-01,
              1.30878963233e-01, -2.46180429761e-01, -2.94375915262e-01, 1.96715004523e-02,
              2.51523254360e-01, 8.18060283872e-02, -1.81962291778e-01, -1.08451713823e-01,
              1.29929646959e-01, 1.01780296838e-01, -9.66075406166e-02, -8.23302119065e-02,
              7.50476199483e-02, 5.95674108715e-02, -5.92568156326e-02, -3.82538294793e-02,
              4.58079441512e-02, 2.09728005925e-02, -3.35235840641e-02, -8.83349389041e-03,
              2.26186515445e-02, 1.69047238348e-03, -1.37639819628e-02, 1.51930577883e-03,
              7.38775745285e-03, -2.24805318700e-03, -3.39452327640e-03, 1.81687134380e-03,
              1.26393425811e-03, -1.11148486531e-03, -3.28078847088e-04, 5.49053277337e-04,
              1.53443902319e-05, -2.20894403245e-04, 4.33672612594e-05, 7.05513878206e-05,
              -3.09866292761e-05, -1.63916249616e-05, 1.35432771841e-05, 1.84994500311e-06, -
62  4.30994155659e-06, 4.85473139699e-07, 1.00212139929e-06, -3.49494860344e-07,
      -1.50988538867e-07, 1.10903123221e-07, 5.35065751546e-09, -2.25219383672e-08,
      4.22448570636e-09, 2.79397446595e-09, -1.29720500146e-09, -1.03141112909e-10,
      1.94616489408e-10, -3.20339824412e-11, -1.39841571553e-11, 6.33495544097e-12,
      -2.09636319423e-13, -4.42161240987e-13, 1.13805283092e-13, -4.51888960746e-16,
      -5.24302569188e-15, 1.18901238750e-15, -1.19928033585e-16, 4.90661506493e-18}));
63        filtros.add(new Filtro("DAUB76", new double[]{1.42577664167e-06, 3.57625199426e-05,
              4.21170266472e-04, 3.08308811925e-03, 1.56372493475e-02, 5.78899436128e-02,
              1.60071993564e-01, 3.30775781411e-01, 4.96591175311e-01, 4.93356078517e-01,
              2.13050571355e-01, -1.82867667708e-01, -3.21675637808e-01, -6.22665060478e-02,
              2.32125963835e-01, 1.49985119618e-01, -1.41795685973e-01, -1.59912565158e-01,
              8.56381215561e-02, 1.41414734073e-01, -5.65864586307e-02, -1.14731170710e-01,
              4.30958954330e-02, 8.72043982620e-02, -3.66051034028e-02, -6.17662087084e-02,
              3.19898775315e-02, 4.00549811051e-02, -2.68914938808e-02, -2.31141340205e-02,
              2.09046452556e-02, 1.12904972786e-02, -1.47018820653e-02, -4.13130665603e-03,
              9.21478503219e-03, 5.62571574840e-04, -5.07131450921e-03, 7.16982182106e-04,
              2.40069778189e-03, -8.44862666553e-04, -9.42461407722e-04, 5.81075975053e-04,
              2.81763925038e-04, -3.03102046072e-04, -4.55568269666e-05, 1.26204335016e-04,
              -1.15540910383e-05, -4.17514164854e-05, 1.33417614992e-05, 1.03735918404e-05,
64  -6.45673042846e-06, -1.55084435011e-06, 2.14996026993e-06, -8.48708758607e-08,
      -5.18773373887e-07, 1.39637754550e-07, 8.40035104689e-08, -4.88475793745e-08,
      -5.42427480028e-09, 1.03470453927e-08, -1.43632948779e-09, -1.34919775398e-09,
      5.26113255735e-10, 6.73233649018e-11, -8.27825652253e-11, 1.10169293459e-11,
      6.29153731703e-12, -2.48478923756e-12, 2.62649650406e-14, 1.80866123627e-13,
      -4.24981781957e-14, -4.56339716212e-16, 2.04509967678e-15, -4.40530704248e-16,
```

```
    4.30459683955e-17, -1.71615245108e-18}));
65      filtros.add(new Filtro("SYM8", new double[]{0.032223100604, -0.012603967262,
            -0.099219543577, 0.297857795606, 0.803738751807, 0.497618667633, -0.029635527646,
            -0.075765714789}));
66      filtros.add(new Filtro("SYM16", new double[]{0.001889950333, -0.000302920515,
            -0.014952258337, 0.003808752014, 0.049137179674, -0.027219029917,
            -0.051945838108, 0.364441894835, 0.777185751701, 0.481359651258, -0.061273359068,
            -0.143294238351, 0.007607487325, 0.031695087811, -0.000542132332,
            -0.003382415951}));
67      filtros.add(new Filtro("COIF6", new double[]{-0.072732619513, 0.337897662458,
            0.852572020212, 0.384864846864, -0.072732619513, -0.015655728135}));
68      filtros.add(new Filtro("COIF12", new double[]{0.016387336464, -0.041464936782,
            -0.067372554722, 0.386110066823, 0.812723635450, 0.417005184424, -0.076488599079,
            -0.059434418647, 0.023680171946, 0.005611434819, -0.001823208871,
            -0.000720549445}));
69      filtros.add(new Filtro("COIF18", new double[]{-0.003793512864, 0.007782596427,
            0.023452696142, -0.065771911282, -0.061123390003, 0.405176902410, 0.793777222626,
            0.428483476378, -0.071799821619, -0.082301927107, 0.034555027573, 0.015880544864,
            -0.009007976137, -0.002574517689, 0.001117518771, 0.000466216960,
            -0.000070983303, -0.000034599773}));
70      filtros.add(new Filtro("COIF24", new double[]{0.000892313669, -0.001629492013,
            -0.007346166328, 0.016068943965, 0.026682300156, -0.081266699681,
            -0.056077313317, 0.415308407030, 0.782238930921, 0.434386056491, -0.066627474263,
            -0.096220442034, 0.039334427123, 0.025082261845, -0.015211731528,
            -0.005658286687, 0.003751436157, 0.001266561929, -0.000589020756,
            -0.000259974552, 0.000062339034, 0.000031229876, -0.000003259680,
            -0.000001784985}));
71      filtros.add(new Filtro("COIF30", new double[]{-0.000212080840, 0.000358589688,
            0.002178236358, -0.004159358782, -0.010131117521, 0.023408156788, 0.028168028974,
            -0.091920010569, -0.052043163181, 0.421566206733, 0.774289603730, 0.437991626216,
            -0.062035963969, -0.105574208714, 0.041289208754, 0.032683574270,
            -0.019761778945, -0.009164231163, 0.006764185449, 0.002433373213,
            -0.001662863702, -0.000638131343, 0.000302259582, 0.000140541150,
            -0.000041340432, -0.000021315027, 0.000003734655, 0.000002063762,
            -0.000000167443, -0.000000095177}));
72      filtros.add(new Filtro("BEYLKIN18", new double[]{0.099305765374353,
            0.424215360812961, 0.699825214056600, 0.449718251149468, -0.110927598348234,
            -0.264497231446384, 0.026900308803690, 0.155538731877093, -0.017520746266529,
            -0.088543630622924, 0.019679866044322, 0.042916387274192, -0.017460408696028,
            -0.014365807968852, 0.010040411844631, 0.0014842347824723, -0.002736031626258,
            0.0006404853285212}));
73      filtros.add(new Filtro("VAIDYANATHAN24", new double[]{-0.000062906118,
            0.000343631905, -0.000453956620, -0.000944897136, 0.002843834547, 0.000708137504,
            -0.008839103409, 0.003153847056, 0.019687215010, -0.014853448005,
            -0.035470398607, 0.038742619293, 0.055892523691, -0.077709750902,
            -0.083928884366, 0.131971661417, 0.135084227129, -0.194450471766,
            -0.263494802488, 0.201612161775, 0.635601059872, 0.572797793211, 0.250184129505,
            0.045799334111}));
74  }
75
76  public static List<Filtro> getFiltros() {
77      return filtros;
78  }
79
80  public static Double[] energias(Wave wave, Filtro filtro, boolean isNivelMaximo){
81
82      // normalizando
83      Double dados[] = normalizaWave(wave);
84
```

```
85          // Calculando o nivel maximo que se pode decompor o vetor
86          int nivelMaximo = (int) (Math.log(dados.length) / Math.log(2));
87
88          int nivel = nivelMaximo;
89
90          if (!isNivelMaximo)
91              nivel /=2; // nivel medio
92
93          // Transformada Wavelet Packet nivel MaXIMO
94          TransformadaWavelet.transformada_wavelet_packet_nivel_k(dados, nivel, filtro);
95
96          // Definindo o numero de sub-bandas
97          int s = 25;
98
99          // Definindo a resolucao
100         double r = (Integer.parseInt(wave.getTaxaAmostragem())) / Math.pow(2, nivel + 1);
101         //System.out.println("Resolucao: "+r);
102
103         // Definindo o vetor de caracteristicas a partir da escala bark
104         int inicio = 0;
105         int fim = 0;
106         Double[] energia = new Double[s];
107         Arrays.fill(energia, 0.0);// inicializando o vetor de objetos
108
109         for (int i = 0; i < s; i++) {
110             switch (i) {
111                 case 0:
112                     inicio = 0;
113                     fim = (int) (100 / r);
114                     break;
115                 case 1:
116                     inicio = (int) (100 / r) + 1;
117                     fim = (int) (200 / r);
118                     break;
119                 case 2:
120                     inicio = (int) (200 / r) + 1;
121                     fim = (int) (300 / r);
122                     break;
123                 case 3:
124                     inicio = (int) (300 / r) + 1;
125                     fim = (int) (400 / r);
126                     break;
127                 case 4:
128                     inicio = (int) (400 / r) + 1;
129                     fim = (int) (510 / r);
130                     break;
131                 case 5:
132                     inicio = (int) (510 / r) + 1;
133                     fim = (int) (630 / r);
134                     break;
135                 case 6:
136                     inicio = (int) (630 / r) + 1;
137                     fim = (int) (770 / r);
138                     break;
139                 case 7:
140                     inicio = (int) (770 / r) + 1;
141                     fim = (int) (920 / r);
142                     break;
143                 case 8:
```

```
144                     inicio = (int) (920 / r) + 1;
145                     fim = (int) (1080 / r);
146                     break;
147                 case 9:
148                     inicio = (int) (1080 / r) + 1;
149                     fim = (int) (1270 / r);
150                     break;
151                 case 10:
152                     inicio = (int) (1270 / r) + 1;
153                     fim = (int) (1480 / r);
154                     break;
155                 case 11:
156                     inicio = (int) (1480 / r) + 1;
157                     fim = (int) (1720 / r);
158                     break;
159                 case 12:
160                     inicio = (int) (1720 / r) + 1;
161                     fim = (int) (2000 / r);
162                     break;
163                 case 13:
164                     inicio = (int) (2000 / r) + 1;
165                     fim = (int) (2320 / r);
166                     break;
167                 case 14:
168                     inicio = (int) (2320 / r) + 1;
169                     fim = (int) (2700 / r);
170                     break;
171                 case 15:
172                     inicio = (int) (2700 / r) + 1;
173                     fim = (int) (3150 / r);
174                     break;
175                 case 16:
176                     inicio = (int) (3150 / r) + 1;
177                     fim = (int) (3700 / r);
178                     break;
179                 case 17:
180                     inicio = (int) (3700 / r) + 1;
181                     fim = (int) (4400 / r);
182                     break;
183                 case 18:
184                     inicio = (int) (4400 / r) + 1;
185                     fim = (int) (5300 / r);
186                     break;
187                 case 19:
188                     inicio = (int) (5300 / r) + 1;
189                     fim = (int) (6400 / r);
190                     break;
191                 case 20:
192                     inicio = (int) (6400 / r) + 1;
193                     fim = (int) (7700 / r);
194                     break;
195                 case 21:
196                     inicio = (int) (7700 / r) + 1;
197                     fim = (int) (9500 / r);
198                     break;
199                 case 22:
200                     inicio = (int) (9500 / r) + 1;
201                     fim = (int) (12000 / r);
202                     break;
```

```
203                 case 23:
204                     inicio = (int) (12000 / r) + 1;
205                     fim = (int) (15500 / r);
206                     break;
207                 case 24:
208                     inicio = (int) (15500 / r) + 1;
209                     fim = (int) (22050 / r);
210                     break;
211             }
212             for (int j = inicio; j < fim; j++) {
213                 energia[i] += dados[j] * dados[j];
214             }
215         }
216 
217         // Achando a maior amplitude em modulo
218         double maior_amplitude = 0.0;
219         for (int i = 0; i < s; i++) {
220             if (energia[i] > maior_amplitude) {
221                 maior_amplitude = energia[i];
222             }
223         }
224 
225         // Dividindo as amostras pelo modulo da maior delas
226         for (int i = 0; i < s; i++) {
227             energia[i] /= maior_amplitude;
228             //System.out.println("energia["+i+"]="+energia[i]);
229         }
230 
231         return energia;
232     }
233 
234     private static Double[] normalizaWave(Wave wave) {
235 
236         Double[] dados = null;
237 
238         try{
239             AbrirWave aw = new AbrirWave(wave);
240             dados = new Double[wave.getTamanhoChunk()];
241 
242             for (int i = 0; i < dados.length; i++)
243                 dados[i] = wave.getDoubleData()[i];
244 
245             // Aumentando o comprimento do vetor e preenchendo com 0 ate que o comprimento
                    fique sendo uma potencia de 2
246             int tamanhoOriginal = dados.length;
247             int y = 1;
248             while (y < tamanhoOriginal) {
249                 y *= 2;
250             }
251             int tamanhoNovo = y;
252             Double[] c = new Double[tamanhoNovo];
253             Arrays.fill(c, 0.0);
254             System.arraycopy(dados, 0, c, 0, tamanhoOriginal);
255             dados=c;
256 
257             // Achando a maior amplitude em modulo
258             double maiorAmplitude = Math.abs(Collections.max(Arrays.asList(dados)));
259 
260             // Dividindo as amostras pela maior amplitude
```

```
261             for (int i=0; i<dados.length; i++)
262                 dados[i]/=maiorAmplitude;
263
264
265              // Achando o valor medio do vetor e subtraindo de cada amostra
266             double valorMedio=0.0;
267             for (Double d: dados)
268                 valorMedio+=d;
269             valorMedio = valorMedio/dados.length;
270             for (int i=0; i<dados.length; i++)
271                 dados[i]=dados[i]-valorMedio;
272
273         }
274         catch(Exception e){
275             e.printStackTrace();
276         }
277
278         return dados;
279     }
280 }
```

```
1  // classe Wave
2  // encapsula os dados relacionados aos arquivos .wav
3  public class Wave {
4
5      private short[] rowData ;
6      private double[]doubleData;
7      private FileReader leitorArquivo;
8      private File arquivo;
9      private int tamanho;
10     private String tipoArquivo;
11     private String formato;
12     private String indentificador;
13     private int comprimentoChunk;
14     private String categoriaFormato;
15     private String numCanais;
16     private String taxaAmostragem;
17     private int mediaBits;
18     private int alinhamento;
19     private int resolucao;
20     private String indentificacao;
21     private int tamanhoChunk;
22     private String path;
23     private FileInputStream fis;
24
25     public Wave() {
26     }
27
28     public Wave(String path) {
29         this.path = path;
30     }
31
32     public String getPath(){
33         return this.path;
34     }
35
```

```
36      public void setPath(String path){
37          this.path=path;
38      }
39
40      public FileReader getLeitorArquivo() {
41          return leitorArquivo;
42      }
43
44      public void setLeitorArquivo(FileReader leitorArquivo) {
45          this.leitorArquivo = leitorArquivo;
46      }
47
48      public File getArquivo() {
49          return arquivo;
50      }
51
52      public void setArquivo(File arquivo) {
53          this.arquivo = arquivo;
54      }
55
56      public int getTamanho() {
57          return tamanho;
58      }
59
60      public void setTamanho(int tamanho) {
61          this.tamanho = tamanho;
62      }
63
64      public String getTipoArquivo() {
65          return tipoArquivo;
66      }
67
68      public void setTipoArquivo(String tipoArquivo) {
69          this.tipoArquivo = tipoArquivo;
70      }
71
72      public String getFormato() {
73          return formato;
74      }
75
76      public void setFormato(String formato) {
77          this.formato = formato;
78      }
79
80      public String getIndentificador() {
81          return indentificador;
82      }
83
84      public void setIndentificador(String indentificador) {
85          this.indentificador = indentificador;
86      }
87
88      public int getComprimentoChunk() {
89          return comprimentoChunk;
90      }
91
92      public void setComprimentoChunk(int comprimentoChunk) {
93          this.comprimentoChunk = comprimentoChunk;
94      }
```

```
95
96      public String getCategoriaFormato() {
97          return categoriaFormato;
98      }
99
100     public void setCategoriaFormato(String categoriaFormato) {
101         this.categoriaFormato = categoriaFormato;
102     }
103
104     public String getNumCanais() {
105         return numCanais;
106     }
107
108     public void setNumCanais(String numCanais) {
109         this.numCanais = numCanais;
110     }
111
112     public String getTaxaAmostragem() {
113         return taxaAmostragem;
114     }
115
116     public void setTaxaAmostragem(String taxaAmostragem) {
117         this.taxaAmostragem = taxaAmostragem;
118     }
119
120     public int getMediaBits() {
121         return mediaBits;
122     }
123
124     public void setMediaBits(int mediaBits) {
125         this.mediaBits = mediaBits;
126     }
127
128     public int getAlinhamento() {
129         return alinhamento;
130     }
131
132     public void setAlinhamento(int alinhamento) {
133         this.alinhamento = alinhamento;
134     }
135
136     public int getResolucao() {
137         return resolucao;
138     }
139
140     public void setResolucao(int resolucao) {
141         this.resolucao = resolucao;
142     }
143
144     public String getIndentificacao() {
145         return indentificacao;
146     }
147
148     public void setIndentificacao(String indentificacao) {
149         this.indentificacao = indentificacao;
150     }
151
152     public int getTamanhoChunk() {
153         return tamanhoChunk;
```

```
154     }
155
156     public void setTamanhoChunk(int comprimento) {
157         this.tamanhoChunk = comprimento;
158     }
159
160     public short[] getRowData() {
161         return rowData;
162     }
163
164     public void setRowData(short[] rowData) {
165         this.rowData = rowData;
166     }
167
168     public double[] getDoubleData() {
169         return doubleData;
170     }
171
172     public void setDoubleData(double[] doubleData) {
173         this.doubleData = doubleData;
174     }
175
176     /**
177      * @return the fis
178      */
179     public FileInputStream getFis() {
180         return fis;
181     }
182
183     /**
184      * @param fis the fis to set
185      */
186     public void setFis(FileInputStream fis) {
187         this.fis = fis;
188     }
189 }
```

```
1  // classe Filtro
2  // encapsula os dados relacionados aos filtros wavelet
3  public class Filtro {
4
5      private String nome;
6      private double[] coeficientes;
7
8      public Filtro(String nome, double[] coeficientes) {
9          this.nome = nome;
10         this.coeficientes = coeficientes;
11     }
12
13     public double[] getCoeficientes() {
14         return coeficientes;
15     }
16
17     public void setCoeficientes(double[] coeficientes) {
18         this.coeficientes = coeficientes;
19     }
```

```
20
21      public String getNome() {
22          return nome;
23      }
24
25      public void setNome(String nome) {
26          this.nome = nome;
27      }
28  }
```

```
1   // classe TransformadaWavelet
2   // executa a transformada Wavelet no vetor de dados submetido, utilizando o filtro
        especificado
3   public abstract class TransformadaWavelet {
4
5       public static void transformada_wavelet_nivel_k(Double[] f, int inicio, int n, int nivel,
            char ordem, Filtro filtro) {
6
7           double[] sf = filtro.getCoeficientes();
8           int csf = sf.length;
9           Double[] wf = new Double[csf];
10          for (int i = 0; i < csf; i++) {
11              wf[i] = sf[csf - i - 1];
12              if (i % 2 != 0) {
13                  wf[i] *= -1;
14              }
15          }
16          int cwf = csf;
17          int j = 0;
18          Double[] t = new Double[n];
19          if (ordem == 'n') // n de normal para wavelet
20          {
21              for (int i = 0; i < n; i += 2) //trend
22              {
23                  t[j] = 0.0;
24                  for (int k = 0; k < csf; k++) {
25                      t[j] += f[inicio + (i + k) % n] * sf[k];
26                  }
27                  j++;
28              }
29              for (int i = 0; i < n; i += 2) //fluctuation
30              {
31                  t[j] = 0.0;
32                  for (int k = 0; k < cwf; k++) {
33                      t[j] += f[inicio + (i + k) % n] * wf[k];
34                      //System.out.println(inicio+(i+k)%n);
35                      //try{System.in.read();}catch(Exception e){}
36
37                  }
38                  j++;
39              }
40          } else // i de invertido para wavelet packet
41          {
42              for (int i = 0; i < n; i += 2) //fluctuation
43              {
44                  t[j] = 0.0;
```

```
45                  for (int k = 0; k < cwf; k++) {
46                      t[j] += f[inicio + (i + k) % n] * wf[k];
47                  }
48                  j++;
49              }
50              for (int i = 0; i < n; i += 2) //trend
51              {
52                  t[j] = 0.0;
53                  for (int k = 0; k < csf; k++) {
54                      t[j] += f[inicio + (i + k) % n] * sf[k];
55                  }
56                  j++;
57              }
58          }
59          for (int i = 0; i < n; i++) {
60              f[inicio + i] = t[i];
61          }
62          nivel--;
63          n /= 2;
64
65          if (nivel > 0) {
66              transformada_wavelet_nivel_k(f, inicio, n, nivel, ordem, filtro);
67          }
68
69
70          //return f;
71      }
72
73      public static void transformada_wavelet_packet_nivel_k(Double[] wave, int nivel, Filtro
            filtro) {
74          int inicio = 0;
75          int comprimento = wave.length;
76          for (int i = 1; i <= nivel; i++) // por exemplo, para nivel 5, vou chamar 5 vezes a
                funcao de transformada, cada vez em n?el 1.
77          {
78              inicio = 0;
79              comprimento = (int) (wave.length / Math.pow(2, i - 1));
80              for (int j = 0; j < Math.pow(2, i - 1); j++) {
81                  if (j % 2 == 0) {
82                      transformada_wavelet_nivel_k(wave, inicio, comprimento, 1, 'n', filtro);
                            // n de ordem normal: primeiro passa-baixa e depois passa-alta
83                  } else {
84                      transformada_wavelet_nivel_k(wave, inicio, comprimento, 1, 'i', filtro);
                            // i de invertido: primeiro passa-alta e depois passa-baixa
85                  }
86                  inicio += comprimento;
87                  //System.out.println("inicio: "+inicio);
88              }
89          }
90          //return wave.getDoubleData();
91      }
92
93  }
```

```
1   // classe Serialization
2   // serializa os dados processados em disco economizando tempo em futuras inclusoes de
```

```
    individuos e arquivos no processo
3 public class Serialization implements Serializable{
4
5     public static void persist(NeuralNetwork c, String name, String nome_filtro) {
6         try {
7             File dir = new File("serializacao/" + nome_filtro);
8             dir.mkdirs();
9             FileOutputStream fo = new FileOutputStream("serializacao/" + nome_filtro + "/" +
                  name + ".ser");
10            ObjectOutputStream oo = new ObjectOutputStream(fo);
11            oo.writeObject(c);
12            oo.close();
13        } catch (FileNotFoundException e) {
14        } catch (Exception e) {
15            System.out.println("Erro ao persistir: " + e.getMessage());
16            //e.printStackTrace();
17        }
18    }
19
20    public static NeuralNetwork recoverNN(String name, String nome_filtro) {
21        NeuralNetwork c=null;
22        try {
23            FileInputStream fi = new FileInputStream("serializacao/" + nome_filtro + "/" +
                  name + ".ser");
24            ObjectInputStream oi = new ObjectInputStream(fi);
25            c = (NeuralNetwork) oi.readObject();
26            oi.close();
27        } catch (FileNotFoundException e) {
28            //e.printStackTrace();
29        } catch (Exception e) {
30            System.out.println("Erro ao restaurar: " + e.getMessage());
31            //e.printStackTrace();
32        }
33        return c;
34    }
35
36    public static void persist(Double[] c, String name, String nome_filtro) {
37        try {
38
39        File dir = new File("serializacao/" + nome_filtro);
40        dir.mkdirs();
41        FileOutputStream fo = new FileOutputStream("serializacao/" + nome_filtro + "/" + name
              + ".ser");
42            ObjectOutputStream oo = new ObjectOutputStream(fo);
43            oo.writeObject(c);
44            oo.close();
45        } catch (FileNotFoundException e) {
46        } catch (Exception e) {
47            System.out.println("Erro ao persistir: " + e.getMessage());
48            //e.printStackTrace();
49        }
50    }
51
52    public static Double[] recover(String name, String nome_filtro) {
53        Double[] c=null;
54        try {
55            FileInputStream fi = new FileInputStream("serializacao/" + nome_filtro + "/" +
                  name + ".ser");
56            ObjectInputStream oi = new ObjectInputStream(fi);
```

```
57              c = (Double[]) oi.readObject();
58              oi.close();
59          } catch (FileNotFoundException e) {
60          } catch (Exception e) {
61              System.out.println("Erro ao restaurar: " + e.getMessage());
62              //e.printStackTrace();
63          }
64          return c;
65      }
66  }
```

```
1   // classe NeuralNetwork
2   // executa as classificacoes utilizando Support Vector Machines
3   public class NeuralNetwork implements Serializable {
4
5       private Double[] a;
6       private Double[][] b;
7       private Double[][] w;
8       private Double[][] fis;
9       private int number_of_input_neurons;
10      private int number_of_hidden_neurons;
11      private int number_of_output_neurons;
12
13      public NeuralNetwork(int n_i_n, int n_h_n, int n_o_n) {
14          number_of_input_neurons = n_i_n;
15          number_of_hidden_neurons = n_h_n;
16          number_of_output_neurons = n_o_n;
17
18          a = new Double[number_of_hidden_neurons];
19          b = new Double[number_of_hidden_neurons][];
20          for (int i = 0; i < number_of_hidden_neurons; i++) {
21              b[i] = new Double[number_of_input_neurons];
22          }
23          w = new Double[number_of_output_neurons][];
24          for (int i = 0; i < number_of_output_neurons; i++) {
25              w[i] = new Double[number_of_hidden_neurons];
26          }
27          for (int i = 0; i < number_of_output_neurons; i++) {
28              for (int j = 0; j < number_of_hidden_neurons; j++) {
29                  w[i][j] = 0.0;
30              }
31          }
32          fis = new Double[number_of_hidden_neurons][];
33          for (int i = 0; i < number_of_hidden_neurons; i++) {
34              fis[i] = new Double[number_of_hidden_neurons];
35          }
36      }
37
38      public void pass(Double[] input_signal, Double[] output_signal) {
39          for (int i = 0; i < number_of_output_neurons; i++) {
40              output_signal[i] = 0.0;
41              for (int j = 0; j < number_of_hidden_neurons; j++) {
42                  output_signal[i] += w[i][j] * svm(input_signal, b[j],
                        number_of_input_neurons, a[j]);
43              }
44          }
```

```
45      }
46
47      public void train(Double[] input_signal, Double[] desired_output_signal) {
48  //non-supervised training
49  //centers
50          int k = 0;
51          for (int i = 0; i < number_of_hidden_neurons; i++) {
52              for (int j = 0; j < number_of_input_neurons; j++) {
53                  b[i][j] = input_signal[j + k];
54              }
55              k += number_of_input_neurons;
56          }
57  //variances
58          double maior = 0;
59          for (int i = 0; i < number_of_hidden_neurons; i++) {
60              for (int j = i; j < number_of_hidden_neurons; j++) {
61                  if (dist(b[i], b[j], number_of_input_neurons) > maior) {
62                      maior = dist(b[i], b[j], number_of_input_neurons);
63                  }
64              }
65          }
66
67          for (int i = 0; i < number_of_hidden_neurons; i++) {
68              a[i] = maior;
69          }
70  //supervised training
71  //weights
72
73          Double[] outs = new Double[number_of_hidden_neurons];
74          double m;
75          int lcpivo;
76          int p;
77
78          for (int c = 0; c < number_of_output_neurons; c++) {
79              //System.out.println("\nNEURON " + c);
80              p = 0;
81              for (int i = 0; i < number_of_hidden_neurons; i++) {
82                  for (int j = 0; j < number_of_hidden_neurons; j++) {
83                      fis[i][j] = svm(Arrays.copyOfRange(input_signal, p, input_signal.length),
                            b[j], number_of_input_neurons, a[j]);
84                  }
85                  p += number_of_input_neurons;
86              }
87
88              p = 0;
89              for (int u = 0; u < number_of_hidden_neurons; u++) {
90                  outs[u] = desired_output_signal[p + c];
91                  p += number_of_output_neurons;
92              }
93
94              maior = fis[0][0];
95              for (int i = 0; i < number_of_hidden_neurons; i++) {
96                  for (int j = 0; j < number_of_hidden_neurons; j++) {
97                      if (fis[i][j] > maior) {
98                          maior = fis[i][j];
99                      }
100                 }
101                 for (int j = 0; j < number_of_hidden_neurons; j++) {
102                     fis[i][j] /= maior;
```

```
103                 }
104                 outs[i] /= maior;
105             }
106
107             //escalonando o sistema
108             lcpivo = 0;
109             do {
110                 for (int kk = lcpivo; kk < number_of_hidden_neurons - 1; kk++) {
111                     m = (-fis[kk + 1][lcpivo] / fis[lcpivo][lcpivo]);
112                     for (int j = lcpivo; j < number_of_hidden_neurons; j++) {
113                         fis[kk + 1][j] += fis[lcpivo][j] * m;
114                     }
115                     outs[kk + 1] += outs[lcpivo] * m;
116                 }
117                 lcpivo++;
118             } while (lcpivo < number_of_hidden_neurons - 1);
119
120             //solucao do sistema escalonado
121             System.arraycopy(outs, 0, w[c], 0, number_of_hidden_neurons);
122
123             //System.out.println("number_of_hidden_neurons: " + number_of_hidden_neurons);
124             w[c][number_of_hidden_neurons - 1] = outs[number_of_hidden_neurons - 1] /
                    fis[number_of_hidden_neurons - 1][number_of_hidden_neurons - 1];
125             for (int i = (number_of_hidden_neurons - 2); i >= 0; i--) {
126                 for (int j = i; j < number_of_hidden_neurons-1; j++) {
127                     //System.out.println("w[" + c + "][" + i + "] -= fis[" + i + "][" + (j +
                            1) + "] * w[" + c + "][" + (j + 1) + "]");
128                     w[c][i] -= fis[i][j + 1] * w[c][j + 1];
129                 }
130                 w[c][i] /= fis[i][i];
131             }
132         }
133     }
134
135     public double svm(Double[] input_signal, Double[] mean, int number_of_input_neurons,
            double variance) //kernel gaussiano
136     {
137         if (variance == 0) {
138             return (Math.exp(-(Math.pow(dist(input_signal, mean, number_of_input_neurons), 2)
                    / (2 * variance + 0.0000000000001))));
139         }
140         return (Math.exp(-(Math.pow(dist(input_signal, mean, number_of_input_neurons), 2) /
                (2 * variance))));
141     }
142
143     public double dist(Double[] x, Double[] y, int length) {
144         double d = 0;
145         for (int i = 0; i < length; i++) {
146             d += Math.pow((x[i] - y[i]), 2);
147         }
148         return (Math.sqrt(d));
149     }
150
151
152 }
```

```
1  // classe AbrirWave
2  // extrai os dados brutos de um arquivo .wav
3  public class AbrirWave {
4
5      private Wave wave;
6      private FileInputStream fis;
7
8      public AbrirWave(Wave wave) throws FileNotFoundException, IOException{
9          //wave.setPath(url);
10         File arquivo = new File(wave.getPath());
11         wave.setLeitorArquivo(new FileReader(arquivo));
12         fis = new FileInputStream(arquivo);
13         this.wave=wave;
14         montaCabecalho();
15     }
16
17     public void montaCabecalho() {
18         //wave.setRowData(new short[5000000]);
19         int data = 36;
20         boolean achaData = true;
21         int j = 0;
22         short s[] = new short[5000000];
23         try {
24             while ((s[j] = (short) fis.read()) != -1) {
25                 j++;
26             }
27             //System.out.println("j: "+j);
28             wave.setRowData(new short[j]);
29             wave.setRowData(Arrays.copyOfRange(s, 0, j));
30
31         } catch (IOException ex) {
32             Logger.getLogger(AbrirWave.class.getName()).log(Level.SEVERE, null, ex);
33         }
34
35         wave.setTipoArquivo((char) wave.getRowData()[0] + "" + (char) wave.getRowData()[1] +
               "" + (char) wave.getRowData()[2] + "" + (char) wave.getRowData()[3]);
36         wave.setTamanho((int) (char4ParaLong(vChar4((char) wave.getRowData()[4], (char)
               wave.getRowData()[5], (char) wave.getRowData()[6], (char) wave.getRowData()[7]))
               + 8));
37 /*        System.out.println(""+wave.getRowData()[24]);
38         System.out.println(""+wave.getRowData()[25]);
39         System.out.println(""+wave.getRowData()[26]);
40         System.out.println(""+wave.getRowData()[27]);
41         System.out.println("tamanho: ");
42         System.out.println(""+wave.getRowData()[4]);
43         System.out.println(""+wave.getRowData()[5]);
44         System.out.println(""+wave.getRowData()[6]);
45         System.out.println(""+wave.getRowData()[7]);*/
46         wave.setFormato((char) wave.getRowData()[8] + "" + (char) wave.getRowData()[9] + "" +
               (char) wave.getRowData()[10] + "" + (char) wave.getRowData()[11]);
47         wave.setIndentificador((char) wave.getRowData()[12] + "" + (char)
               wave.getRowData()[13] + "" + (char) wave.getRowData()[14] + "" + (char)
               wave.getRowData()[15]);
48         wave.setComprimentoChunk((int) wave.getRowData()[16]);
49
50
51         if (((int) wave.getRowData()[20]) == 1) {
52             wave.setCategoriaFormato("PCM");
53
```

```
54          } else if (((int) wave.getRowData()[20]) == 0) {
55              wave.setCategoriaFormato("outro");
56          }
57
58          if (((int) wave.getRowData()[22]) == 1) {
59              wave.setNumCanais("Mono");
60
61          } else if (((int) wave.getRowData()[22]) == 2) {
62              wave.setNumCanais("Estereo");
63
64          }
65          wave.setTaxaAmostragem(Long.toString(char4ParaLong(vChar4((char)
                wave.getRowData()[24], (char) wave.getRowData()[25], (char)
                wave.getRowData()[26], (char) wave.getRowData()[27]))));
66          wave.setMediaBits((int) wave.getRowData()[28]);
67          wave.setAlinhamento((int) wave.getRowData()[32]);
68          wave.setResolucao((int) wave.getRowData()[34]);
69
70          while (achaData) {
71              //System.out.println("wave.getRowData()["+data+"]="+(char)wave.getRowData()[data]);
72              if (wave.getRowData()[data] == 'd') {
73                  achaData = false;
74              } else {
75                  data++;
76              }
77          }
78          wave.setIndentificacao((char) wave.getRowData()[data] + "" + (char)
                wave.getRowData()[++data] + "" + (char) wave.getRowData()[++data] + "" + (char)
                wave.getRowData()[++data]);
79          /*System.out.println("data:"+(data+1)+"="+(wave.getRowData()[data+1]));
80          System.out.println("data:"+(data+2)+"="+(wave.getRowData()[data+2]));
81          System.out.println("data:"+(data+3)+"="+(wave.getRowData()[data+3]));
82          System.out.println("data:"+(data+4)+"="+(wave.getRowData()[data+4]));
83          System.out.println("40 a 43===");
84          System.out.println("data:"+(40)+"="+(wave.getRowData()[40]));
85          System.out.println("data:"+(41)+"="+(wave.getRowData()[41]));
86          System.out.println("data:"+(42)+"="+(wave.getRowData()[42]));
87          System.out.println("data:"+(43)+"="+(wave.getRowData()[43]));*/
88          wave.setTamanhoChunk((int) char4ParaLong(vChar4((char) wave.getRowData()[++data],
                (char) wave.getRowData()[++data], (char) wave.getRowData()[++data], (char)
                wave.getRowData()[++data]))/2);
89
90          /*System.out.println("CABEcALHO WAVE");
91          System.out.println("");
92          System.out.println("Aquivo tipo : " + wave.getTipoArquivo());
93          System.out.println("Tamanho : " + wave.getTamanho());
94          System.out.println("Formato : " + wave.getFormato());
95          System.out.println("Identificador :  " + wave.getIndentificador());
96          System.out.println("Comprimento Chunk : " + wave.getComprimentoChunk());
97          System.out.println("Formato : " + wave.getCategoriaFormato());
98          System.out.println("Numero de Canais : " + wave.getNumCanais());
99          System.out.println("Taxa de Amostragem : " + wave.getTaxaAmostragem());
100         System.out.println("Media de bits/s : " + wave.getMediaBits());
101         System.out.println("Alinhamento de Bloco : " + wave.getAlinhamento());
102         System.out.println("Resolucao : " + wave.getResolucao()+" bits");
103         System.out.println("Identificacao : " + wave.getIndentificacao());
104         System.out.println("Comprimento Chunk : " + wave.getTamanhoChunk());
105         System.out.println*/
106
```

```
107         wave.setDoubleData(new double[wave.getTamanhoChunk()]);
108
109         for (j = 0; j < wave.getTamanhoChunk(); j++) {
110             wave.getDoubleData()[j] = char2ParaLong(vChar2((char) wave.getRowData()[++data],
                    (char) wave.getRowData()[++data]));
111             // if (j>wave.getTamanhoChunk()*0.49 && j<wave.getTamanhoChunk()*0.50001)
112                 //System.out.println("(" + j + ") " + (int) wave.getRowData()[data - 1] + " :
                        " + (int) wave.getRowData()[data] + " = " +
                        char2ParaLong(vChar2((char)wave.getRowData()[data - 1],
                        (char)wave.getRowData()[data])));
113
114         }
115     /*
116     int i = 0;
117     for (short c : wave.getRowData()) {
118     if (i < 100) {
119     System.out.println("  (" + i + ")" + (char) c + "|" + c);
120     i++;
121     }
122     }*/
123
124     }
125
126     //retorna um vetor com 4 char
127     public char[] vChar4(char a, char b, char c, char d) {
128         char retorno[] = {a, b, c, d};
129         return retorno;
130     }
131
132     //retorna um vetor de com 2 char
133     public char[] vChar2(char a, char b) {
134         char retorno[] = {a, b};
135         return retorno;
136     }
137
138     public long char4ParaLong(char[] bytes) {
139         char[] apoio = {(char) 0, (char) 0, (char) 0, (char) 0};
140         String binario = "";
141         String etapaBinario = "";
142         int comprimento = 0;
143         for (int i = 0; i < bytes.length; i++) {
144             apoio[i] = bytes[bytes.length - i - 1];
145             etapaBinario = Integer.toBinaryString(bytes[bytes.length - i - 1]);
146             comprimento = etapaBinario.length();
147             for (int j = 0; j < 8 - comprimento; j++) {
148                 etapaBinario = 0 + etapaBinario;
149             }
150             binario += etapaBinario;
151         }
152         Long retorno = Long.parseLong(binario, 2);
153         return (retorno);
154     }
155
156     public Long char2ParaLong(char[] bytes) {
157
158         char[] apoio = {(char) 0, (char) 0};
159         String binario = "";
160         String binarioInversa = "";
161         String etapaBinario = "";
```

```
162
163         int comprimento = 0;
164         Long retorno;
165
166         for (int i = 0; i < bytes.length; i++) {
167             apoio[i] = bytes[bytes.length - i - 1];
168             etapaBinario = Integer.toBinaryString(bytes[bytes.length - i - 1]);
169             comprimento = etapaBinario.length();
170
171             for (int j = 0; j < 8 - comprimento; j++) {
172                 etapaBinario = 0 + etapaBinario;
173             }
174             binario += etapaBinario;
175         }
176         //se o primeiro numero for 1 troca todos os numeros e adiciona o sinal negativo
177         if (binario.charAt(0) == '1') {
178             for (int j = 0; j < binario.length() - 1; j++) {
179                 if (binario.charAt(j) == '0') {
180                     binarioInversa += '1';
181
182                 } else {
183                     binarioInversa += '0';
184                 }
185             }
186             binarioInversa += binario.charAt(binario.length() - 1);
187             binarioInversa = '-' + binarioInversa;
188             //se o ultimo numero for 0 subtrai 2
189
190
191             if (binarioInversa.charAt(binarioInversa.length() - 1) == '0') {
192                 retorno = (Long.parseLong(binarioInversa, 2) - 2);
193
194             } else {
195                 retorno = Long.parseLong(binarioInversa, 2);
196             }
197         } else {
198             binarioInversa = binario;
199             retorno = Long.parseLong(binarioInversa, 2);
200         }
201         return (retorno);
202     }
203 }
```

Apêndice III - Publicações durante o mestrado.

Durante o curso de mestrado do autor, os seguintes artigos foram publicados:

- PUIA, V. M.; VIEIRA, L. S.; GUIDO, R. C.; BARBON, S.; ESCOLA, J. P. L.; FERREIRA, A. A.; VINHOTE, R. D. Reconhecimento de locutores usando Wavelets. In: CONGRESSO DE INICIAÇÃO CIENTÍFICA DA UFSCAR, 15., 2007, São Carlos. *Anais...* São Carlos: UFSCar, 2007.

- PUIA, V.M.; VIEIRA, L. S.; GUIDO, R. C.; BARBON, S.; ESCOLA, J. P. L. Construção de um sistema biométrico baseado em processamento de voz humana. In: SIMPÓSIO DE INICIAÇÃO CIENTÍFICA E TECNOLÓGICA, 9., 2007, São Paulo *Anais...* São Paulo: FAPESP, 2007.

- FANTINATO, P. C.; GUIDO, R. C.; CHEN, S. H.; SANTOS, B. L. S.; VIEIRA, L. S.; BARBON JUNIOR, S.; RODRIGUES, L. C.; SANCHEZ, F. L.; ESCOLA, J. P. L.; SOUZA, L. M.; MACIEL, C. D.; SCALASSARA, P. R.; PEREIRA, J. C. A fractal-based approach for speech segmentation. In: IEEE INTERNATIONAL SYMPOSIUM ON MULTIMEDIA (ISM2008), 10., 2008, Berkeley. *Proceedings...* Berkeley: IEEE, 2008. p. 551-555. v. 1.

- GUIDO, R. C.; BARBON JÚNIOR, S.; VIEIRA, L. S.; RODRIGUES, L. C.; FANTINATO, P. C.; ESCOLA, J. P. L.; SOUZA, L. M.; ALMEIDA, L. O. B.; SLAETS, J. F. W. Autenticação biométrica por comando de voz com implementação em hardware dedicado. In: WORKSHOP NO AMBITO DO PROJECTO -AMBIENTES PARA CO-PROJETO DE HARDWARE/SOFTWARE EM PLATAFORMAS FPGAS COM APLICAÇÃO EM ROBÓTICA MÓVEL, 2., 2008, São Carlos, *Anais...* São Carlos, 2008. CD-ROM.

- GUIDO, R.C.; BARBON JÚNIOR, S.; VIEIRA, L.S.; FANTINATO, P.; SANCHEZ, F.L.; ESCOLA, J.P.; BOLOGNINI, E.; SOUZA, L.; ZULATO, P.; RIBEIRO, J., LACERDA, M. Speech compression based on the discrete Shapelet transform. In: CNMAC – CONGRESSO NACIONAL DE MATEMÁTICA APLICADA E COMPUTACIONAL. 31., 2008. Recife. *Anais...* Recife-PE: CNMAC, 2008. CD ROM.

- GUIDO, R.C.; CHEN, S.H.; BARBON, S.; SOUZA, L.M.; VIEIRA, L.S.; RODRIGUES, L.C.; ESCOLA, J.P.L.; ZULATO, P.; LACERDA, M.A.; RIBEIRO, J. On the determination of Epsilon during discriminative GMM training. In: IEEE INTERNATIONAL SYMPOSIUM ON MULTIMEDIA. 2010. Taichung. *Anais...* New Jersey: IEEE Press, 2010. p. 362-364.

- ESCOLA, J.P.L.; BARBON JÚNIOR, S.; GUIDO, R.C. Extração de características do ruído emitido pela caneta ao assinar. In: CONGRESSO DE MATEMÁTICA APLICADA E COMPUTACIONAL. 2011. Uberlândia. *Anais...* Uberlândia, 2011. CD ROM.

- COTES, L.V.; TAVARES, M.R.; SOLGON, R.; BARBON JÚNIOR, S.; ESCOLA, J.P.L. Transformada discreta wavelet aplicada a sinais de áudio. In: SIMPÓSIO DE INICIAÇÃO CIENTÍFICA E TECNOLÓGICA, 14., 2012, São Paulo. *Anais...* São Paulo: FATEC, 2012. CD ROM.

- ESCOLA, J.P.L.; BARBON JÚNIOR, S. Biometria baseada no ruído caligráfico, wavelets e support vector machine. In: SIMPÓSIO DE TECNOLOGIA DA INFORMAÇÃO. 2012. *Anais...* São José do Rio Preto, 2012. CD ROM.

- GUIDO, R.C.; BARBON JUNIOR, S.; SOLGON, R.; SILVA PAULO, K.C.; RODRIGUES, L.C.; DA SILVA, I.N.; ESCOLA, J.P. Introducing the discriminative paraconsistent machine (DPM). Information Sciences, v. 221, p. 389-402, 2013.