Universidade de São Paulo

Escola de Engenharia de São Carlos

Departamento de Engenharia Elétrica

Antonio José Homsi Goulart

Classificação automática de gênero musical baseada em entropia e fractais

São Carlos - SP

2012

Antonio José Homsi Goulart

Classificação automática de gênero musical baseada em entropia e fractais

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciências, Programa de Engenharia Elétrica.

Trata-se da versão corrigida da dissertação. A versão original se encontra disponível na EESC/USP que aloja o programa de Pós-Graduação em Engenharia Elétrica.

Área de concentração:
Sistemas Dinâmicos

Orientador:
Prof. Dr. Carlos Dias Maciel

São Carlos - SP
2012

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

G694c

Goulart, Antonio José Homsi
Classificação automática de gênero musical baseada em entropia e fractais. / Antonio José Homsi Goulart ; orientador Carlos Dias Maciel. São Carlos, 2012.

Dissertação (Mestrado - Programa de Pós-Graduação em Engenharia Elétrica e Área de Concentração em Sistemas Dinâmicos)-- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2012.

1. Processamento digital de sinais. 2. Classificação automática de gênero musical. 3. Entropia baseada em wavelet. 4. Lacunaridade. 5. SVM. 6. GMM. I. Título.

FOLHA DE JULGAMENTO

Candidato: Engenheiro Eletricista **ANTONIO JOSÉ HOMSI GOULART.**

Título da dissertação: "Classificação automática de gênero musical baseada em entropia e fractais".

Data da defesa: 16/02/2012

Comissão Julgadora: **Resultado:**

Prof. Associado **Carlos Dias Maciel (Orientador)** APROVADO
(Escola de Engenharia de São Carlos/EESC)

Prof. Associado **Ailton Akira Shinoda** APROVADO
(Universidade Estadual Paulista "Júlio de Mesquita Filho" /campus de Ilha Solteira)

Prof. Associado **Rodrigo Capobianco Guido** Aprovado
(Universidade Estadual Paulista "Júlio de Mesquita Filho" /campus de São José do Rio Preto)

Coordenador do Programa de Pós-Graduação em Engenharia Elétrica:
Prof. Titular **Denis Vinicius Coury**

Presidente da Comissão de Pós-Graduação:
Prof. Associado **Paulo Cesar Lima Segantine**

Para Cíntia

Agradecimentos

Aos professores e amigos Carlos e Rodrigo, pela ajuda, conhecimento e paciência.

Ao grupo de Computação Musical do IME, pelas idéias construtivas e pelas risadas.

As amigas queridas e aos amigos queridos, por dividir momentos legais e me incentivar.

A Momi e a Vóvi, por tudo.

"All this time the Guard was looking at her, first through a telescope, then through a microscope, and then through an opera-glass." [1]

"Through the looking-glass, and what Alice found there", by Lewis Carroll.

"(...) but I dare say you will supose that we could at least distinguish by sight the Triangles, Squares, and other figures, moving about as I have described them. On the contrary, we would see nothing of the kind, not at least so as to distinguish one figure from another. Nothing was visible, nor could be visible, to us, except straight Lines;"[2]

"Flatland", by Edwin A. Abbott.

[1] *Wavelets.*
[2] Classificadores.

Resumo

GOULART, A.J.H. *Classificação automática de gênero musical baseada em entropia e fractais.* 2012. Dissertação (Mestrado em Ciências) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2012.

A classificação automática de gênero musical tem como finalidade o conforto de ouvintes de músicas auxiliando no gerenciamento das coleções de músicas digitais. Existem sistemas que se baseiam em cabeçalhos de metadados (tais como nome de artista, gênero cadastrado, etc.) e também os que extraem parâmetros dos arquivos de música para a realização da tarefa. Enquanto a maioria dos trabalhos do segundo tipo utilizam-se do conteúdo rítmico e tímbrico, este utiliza-se apenas de conceitos da teoria da informação e da geometria de fractais. Entropia, lacunaridade e dimensão do fractal são os parâmetros que treinam os classificadores. Os testes foram realizados com duas coleções criadas para este trabalho e os resultados foram proeminentes.

Palavras-chave: classificação automática de gênero musical, entropia baseada em wavelet, lacunaridade, SVM, GMM.

Abstract

GOULART, A.J.H. *Automatic music genre classification based on entropy and fractals.* 2012. Dissertação (Mestrado em Ciências) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2012.

The goal of automatic music genre classification is giving music listeners ease and confort when managing digital music databases. Some systems are based on tags of metadata (such as artist name, genre labeled, etc.), while others explore characteristics from the music files to complete the task. While the majority of works of the second type analyse rhytmic, timbric and pitch content, this one explores only information theoretic and fractal geometry concepts. Entropy, fractal dimension and lacunarity are the parameters adopted to train the classifiers. Tests were carried out on two databases assembled by the author. Results were prominent.

Key-words: automatic music genre classification, wavelet based entropy, lacunarity, SVM, GMM.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

FD	*Fractal Dimension*
DWT	*Discrete Wavelet Transform*
STFT	*Short-time Fourier Transform*
MFCCs	*Mel-frequency cepstral coefficients*
MRA	*Multi-Resolution Analysis*
LDA	*Linear Discriminant Analysis*
k-NN	*k-Nearest Neighbours*
SVM	*Support vector machine*
QP	*Quadratic Programs*
RBF	*Radial Basis Functions*
GMM	*Gaussian Mixture Models*
EM	*Expectation Maximization*
ML	*Maximum Likelihood*
MP3	*MPEG-2 Audio Layer III*
WMA	*Windows Media Audio*
AIFF	*Audio Interchange File Format*
CELT	*Constrained Energy Lapped Transform*
AAC	*Advanced Audio Coding*
FLAC	*Free Lossless Audio Converter*
WAV	*Waveform Audio Format*
WT	*Wavelet Transform*
MIDI	*Musical Instrument Digital Interface*
WBE	*Wavelet Based Entropy*
FPE	*Fourier Power Spectrum*

Sumário

Capítulo 1

Introdução e Motivação

1.1 A importância da classificação automática de gêneros musicais

Atualmente é raro encontrar um computador conectado a Internet que não esteja fazendo um *download* de música. Dada a extrema facilidade de obtenção - lícita ou não - de arquivos de música digital em suas diversas extensões (mp3, aac, wma, CELT, aiff, flac, wav, entre outros), as coleções atingem tamanhos incríveis. Já não se "baixa" mais uma música ou um álbum, mas sim toda a discografia de um determinado artista. Tantos *gigabytes* de música armazenados tornam difícil a tarefa de localizar algumas, se estas não estiverem organizadas.

Devido à onipresença e à velocidade da internet, ao invés de fazer *downloads*, é também comum o consumo via *streaming*, tal como as rádios *on-line* (vide Apêndice III), artistas que disponibilizam suas músicas para serem ouvidas em suas páginas, sites de vídeos, televisão *on-line*, etc.

Finalizando os exemplos cotidianos, tem-se a extrema popularidade dos *players* portáteis. É praticamente impossível não encontrar pessoas na rua, no ônibus ou no metrô, que estejam com um par de fones de ouvido ligados em um *player* ou no próprio celular, ouvindo suas mp3 competindo com o ruído externo. Em academias de ginástica, mesmo com a batida ambiente tocando, muitas pessoas preferem suas próprias *playlists*.

Com relação aos estilos, pesquisas mostram que a procura de músicas é feita mais por gênero do que por artista [22] e que o gênero em questão influencia mais o "gostar" da música do que a própria música [25], [38], [19]. As rádios *on-line* selecionam para cada ouvinte músicas similares baseadas na escolha de um gênero. Nos *softwares* e *players* portáteis, listas de reprodução inteligentes são criadas com o mesmo princípio.

A intenção desses parágrafos introdutórios é ilustrar a necessidade do desenvolvimento e aprimoramento de sistemas de classificação inteligente de dados, busca, *data mining*, recuperação da informação (*information retrieval*), entre outros. No Capítulo 2 será mostrado que os trabalhos desenvolvidos utilizam-se de metadados vinculados aos arquivos de música ou de informações musicais extraídas diretamente dos arquivos de áudio, tal como o conteúdo rítmico e tímbrico.

1.2 Objetivos e Contribuições do trabalho

O objetivo específico deste trabalho é apresentar um sistema automático de classificação de gêneros musicais que se baseia, além do classificador inteligente, apenas nos conceitos de fractais e teoria da informação para realizar a tarefa, como entropia, dimensão de fractal e lacunaridade. A contribuição vem com o aprimoramento desses tipos de sistemas, dada a necessidade ilustrada. A alternativa apresentada pode, ainda, ser utilizada em conjunto com as tecnologias existentes numa tentativa de aprimorar ainda mais os resultados.

A palavra "apenas" foi utilizada no parágrafo anterior com o sentido de que este trabalho não utiliza informações musicais para a classificação. No Capítulo 2, esse tipo de informação será apresentada como conteúdo tímbrico, rítmico e conteúdo de pitch. A primeira diz respeito aos timbres explorados na música, já que diferentes instrumentos possuem diferentes distribuições de energia no espectro, e diferentes estilos musicais exploram diferentes orquestrações. A textura rítmica refere-se aos padrões de ritmos das composições e o conteúdo de *pitch* está associado às frequências (notas) utilizadas. Dannenberg *et.al.* apontam que a maioria dos trabalhos utiliza parâmetros perceptuais de baixo nível como detecção de pitch, acompanhamento de partitura (MIDI) e processamento de eventos. Como dito anteriormente o sistema aqui apresentado explora o conteúdo do áudio do ponto de vista informacional para realização da classificação.

Fica aqui a ressalva de que o autor não se baseia na filosofia de que gêneros musicais são perfeitamente classificáveis ou que existam gêneros bem definidos, muito menos que ele se encontra em posição de poder realizar tal classificação. A arte musical é dinâmica; diferentes estilos influenciam-se livremente, dando origens a complexas misturas e novos estilos. O objetivo secundário deste trabalho é investigar a música sob um ponto de vista informacional, e o

quanto e o que esse tipo de análise é capaz de dizer sobre os diferentes gêneros. Tais esclarecimentos abrem novas discussões e possibilidades para futuros trabalhos.

1.3 Organização do Trabalho

Este trabalho está organizado da seguinte forma: o Capítulo 2 apresenta uma revisão da literatura, mencionando o estado da arte em técnicas de classificação automática de gênero musical. Serão citados, mas sem explicação teórica, os vários tipos de parâmetros utilizados nos trabalhos da área, assim como os vários tipos de classificadores, e na sequência encontra-se um pouco de teoria acerca dos conceitos utilizados no sistema proposto, a saber:

- Entropia;
- A entropia baseada em *wavelet*;
- Dimensão de fractal;
- Lacunaridade;
- Transformada *Wavelet*;
- Máquinas de vetores suporte;
- Modelos de mistura de Gaussianas.

O Capítulo 3 descreve com detalhes o sistema proposto em suas diferentes arquiteturas experimentadas e os testes realizados. Os resultados obtidos são sumarizados no Capítulo 4 e finalmente as conclusões e contribuições do trabalho são discorridas no Capítulo 5. Após as referências bibliográficas encontram-se cinco apêndices: uma breve discussão sobre os elementos de teoria musical utilizados no texto, listas com as músicas utilizadas nos treinamentos e nos testes, o artigo publicado pelo autor no *IEEE International Symposium on Multimedia 2011*, os programas utilizados e finalmente, a reprodução de uma matéria sobre música na nuvem publicada em jornal.

Capítulo 2

Revisão Bibliográfica

Neste capítulo, é apresentada uma revisão da literatura sobre classificação automática de gênero musical e o estado da arte é exposto, assim como uma breve exposição dos conceitos teóricos utilizados neste trabalho. Mais precisamente, serão abordados os conceitos de entropia e entropia baseada em wavelet, geometria fractal (dimensão de fractal, lacunaridade) e transformada wavelet. Ainda, outros conceitos importantes como formato WAV de áudio, a medida estatística de desvio-padrão e o conceito de energia são expostos.

2.1 Trabalhos relacionados

Mckay e Fujinaga [22] discorreram sobre o por que pesquisadores devem continuar seu trabalho na área de *automatic music genre classification* (AMGC). Algumas das questões colocadas são:

- a ambiguidade e subjetividade na classificação e o dinamismo dos gêneros musicais. São necessários muito tempo e destreza para uma classificação manual, além do fato de que não há consenso entre humanos na divisão dos gêneros. Poucos possuem definições claras e normalmente sobrepõem-se;
- as classificações tendem a ser por artista ou por álbum, e não por músicas individuais;
- cabeçalhos de metadados encontrados nos arquivos de áudio não são confiáveis;
- finalmente, novos gêneros são introduzidos regularmente, e as fronteiras de cada gênero mudam com o tempo.

O trabalho de Dannenberg *et.al.* [8] foi baseado em informações MIDI. Utilizaram pitch, duração, contagem de notas, volume e variações de volume. Para o estágio de classificação utilizaram *naive-bayesian*, classificador linear e redes neurais. Neste trabalho, a capacidade de um músico improvisar em um determinado estilo era testada. Uma coleção foi criada para o treinamento dos classificadores. Foi utilizado um classificador para cada estilo treinados para retornar "sim" ou "nao" para o improviso em questão. Testando entre 4 estilos foi obtida uma classificação de 98%, e entre 8 estilos os resultados ficaram entre 77% e 90%. Na opinião dos autores é crucial que este sistema seja *real-time*, e este responde em 5 segundos.

Outro trabalho clássico na área foi realizado por Tzanetakis e Cook [39]. Foram propostos 3 conjuntos de parâmetros para representar a textura de timbres, o conteúdo rítmico e as informaçoes sobre pitch. Para a obtenção dos vetores de parâmetros foram utilizadas a Transformada de Fourier de tempo reduzido (STFT), coeficientes da banda MEL (MFCCs) e Transformada *Wavelet* (WT). Os classificadores utilizados, baseados em reconhecimento de padrões, foram *simple Gaussian*, *Gaussian Mixture Models* (GMM) e *k-nearest neigbour*. A classificação obtida foi de 61% utilizando 10 gêneros.

Li et. al. [20] fizeram um estudo comparativo entre parâmetros obtidos via *Daubechies Wavelet Coefficient Histograms* (DWCH) versus parâmetros baseados em conteúdo tímbrico, rítmico e de pitch. Como classificadores foram utilizadas Máquinas de Vetores Suporte (SVMs) e *Linear Discriminant Analysis* (LDA). Os testes compararam ainda o uso de trechos iniciais versus trechos do meio das músicas. A melhor classificação obtida, de 74,2%, foi obtida utilizando DWCH e SVM, utilizando trechos centrais das músicas (segundos 31 a 60).

Paradzinets *et.al.* [28] exploraram informação acústica, timbre e *beat* (ritmo), e também levaram em conta o sistema psicoacústico humano. Com os parâmetros acústicos obtidos via *Piecewise Gaussian Modeling* (PGM) [11], aplicaram filtros de banda crítica e de equalização de *loudness*. As características rítmicas foram obtidas por meio de histogramas 2D de *beats*, após a realização de uma Transformada *Wavelet*. Para as caracteristicas tímbricas detectaram pitch e suas respectivas amplitudes e computaram seus histogramas. Diferentes arquiteturas de redes neurais foram testadas. Concluiram que considerar a percepção humana foi melhor e que treinar diferentes redes neurais para os diferentes gêneros é melhor que treinar apenas uma com todas as classes. A taxa média de acerto que obtiveram nos testes foi de 49,3%.

Silla *et.al.* [36] adotaram múltiplos vetores de parâmetros extraídos de trechos do início, do meio e do final das músicas no domínio do tempo-frequência. Foram testadas classificações baseadas em *Naive-Bayes*, árvores de decisão, *k-Nearest neighbours*, SVMs e Redes Neurais multicamadas. A melhor classificação obtida foi de 65,06% utilizando a abordagem *Round-Robin*.

O método proposto por Ezzaidi e Rouat [12] divide as músicas em janelas e obtém MFCCs do espectro normalizado. Os classificadores, baseados em GMMs, alcançaram 99%

de acerto. O sistema de Panagakis e Kotropoulos [26] propõe parâmetros que consideram as características do sistema auditivo humano e classificação baseada em representação esparsa. Os resultados obtidos com as coleções GTZAN e ISMIR2004 foram as melhores já reportadas, respectivamente, 91% e 93,56%.

Como foi mostrado, os trabalhos exploram texturas tímbricas e rítmicas, as notas musicais presentes e outras informações musicais. Este trabalho explora uma releitura da entropia clássica de Shannon, representando a entropia do sinal do ponto de vista da sua dinâmica, e conceitos de fractais, que representam a autossimilaridade e a dispersão da informação.

2.2 A Transformada *Wavelet* Discreta (DWT)

De acordo com [17], na trasformada *wavelet* contínua, a função ψ, que na prática se parece com uma pequena onda (apelidada de ondaleta, vide Figura 2.1), é utilizada para criar uma família [1] de *wavelets* $\psi(at + b)$, onde a e b são números reais. O valor de a dilata (comprime ou estica) a função ψ e b a translada. A WT contínua leva um sinal $f(t)$ em uma funcão de duas variáveis, escala e tempo,

$$c(a, b) = \int_{-\infty}^{+\infty} f(t)\psi(at + b)dt. \tag{2.1}$$

Na WT discreta, as ondaletas são dilatadas e transladadas apenas por valores discretos, sendo que geralmente a dilatação ocorre em potências de 2, e as ondaletas ficam da forma

$$\psi(2^k + l),$$

sendo k e l inteiros. *Wavelets* ortogonais são um caso especial de wavelet discreta. Representam o sinal sem redundância e seus algoritmos são mais rápidos. São obtidas por meio de produtos escalares.

A WT decompõe um sinal em várias resoluções e o descreve no domínio da frequência, possibilitando a análise do sinal em diferentes escalas de tempo e de frequência. Daí vem o nome de domínio do tempo-frequência. Com a teoria da multirresolução, Stéphane Mallat uniu *wavelets* ortogonais à teoria de filtros utilizada em processamento de sinais [17]. Nessa abordagem, a ondaleta é atualizada por uma nova função, a função de escala (*scaling function*),

[1] Várias famílias podem ser utilizadas. Este trabalho é baseado na família Haar (vide Figura 2.1).

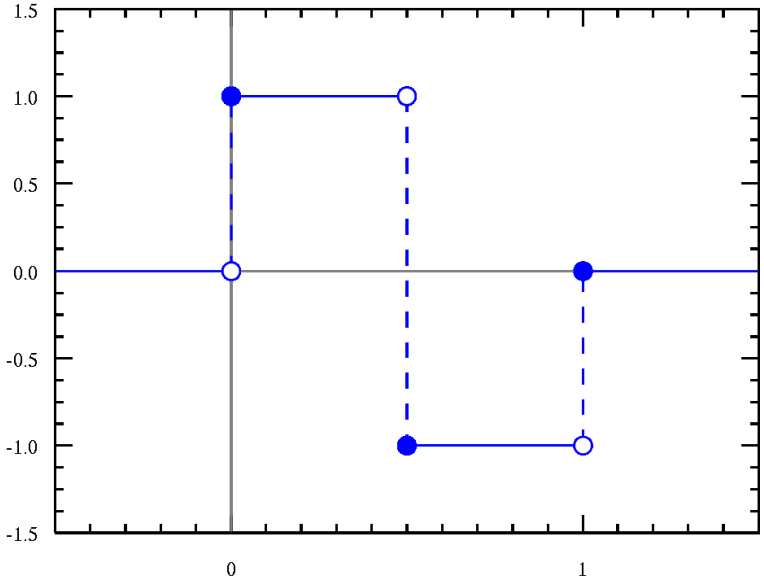

Figura 2.1 – A *wavelet* Haar. A mais simples das *wavelets*.

possibilitando várias visualizações do sinal em diferentes resoluções que diferem entre si por um fator de 2. Em um sentido, cada visualização aproxima o sinal com maior precisão, chegando no sinal original. No outro sentido as visualizações chegam em zero, não contendo informação alguma. Um detalhe é que em cada passo está incorporada a diferença de informação entre duas resoluções, por exemplo, os detalhes que precisam ser adicionados a uma imagem para que ela fique com uma resolução duas vezes melhor.

Na linguagem da WT, a função de escala é dilatada para gerar uma visualização do sinal em metade da resolução anterior. Na linguagem de processamento de sinais, um filtro passa-baixa e um passa-alta são aplicados ao sinal, e o sinal resultante é subamostrado, como mostra a Figura 2.2. Por exemplo, para se obter a transformada de um sinal de 32 amostras, primeiro deve-se agrupá-lo em pares e computar as diferenças e as médias entre cada par. Obtém-se assim 16 diferenças (coeficientes da *wavelet*) e 16 médias (coeficientes da função de escala). Em seguida, deve-se fazer o mesmo para os 16 coeficientes obtidos, depois para 8, e assim por diante. Como haviam 32 amostras, têm-se como resultado 31 coeficientes mais o último coeficiente da função de escala, que representa o valor médio do sinal.

Para a obtenção da transformada faz-se o produto interno das funções $f(t)$ e $\psi_{a,b}(t)$, ficando com a relação

$$W(a,b) = \int_{-\infty}^{\infty} f(t)\psi_{a,b}(t)dt = \langle f(t), \psi_{a,b}(t)\rangle. \tag{2.2}$$

Na prática, a DWT decompõe um sinal em diferentes faixas de frequência sem per-

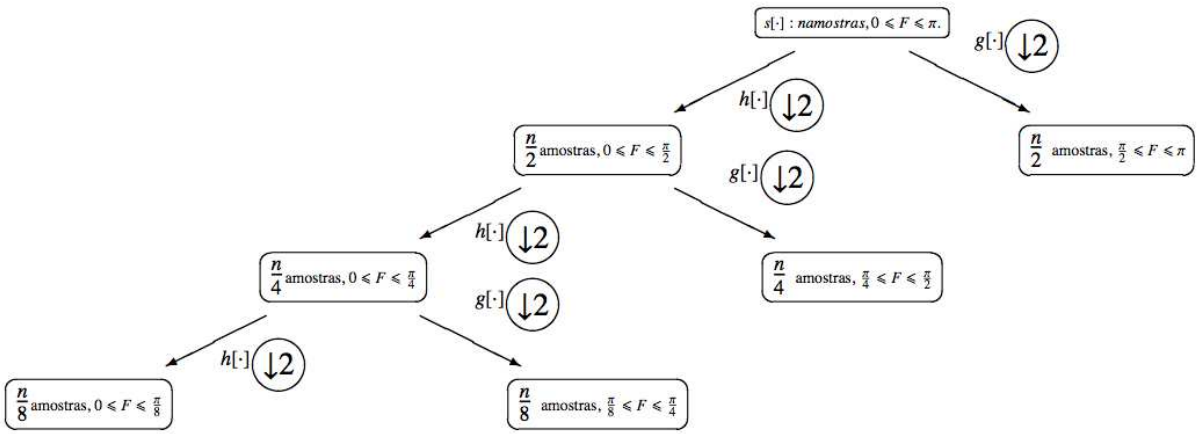

Figura 2.2 – Ilustração do funcionamento da DWT de um sinal de N amostras com máxima frequência π. A decomposição está representada até o terceiro nível. $h[.]$ é um filtro passa-baixa e $g[.]$ um passa-alta. Figura adaptada de [37].

der informação sobre a evolução deste sinal no tempo. Dessa maneira fica possível analisar o comportamento dinâmico deste sinal em diferentes bandas e em diferentes resoluções. No contexto deste trabalho, a análise da entropia nas diferentes faixas de frequência é de fundamental importância uma vez que diferentes estilos de música exploram diferentes faixas no espectro. Ainda, dentro de uma mesma banda de frequência, os timbres se diferenciam devido aos padrões de execução/interpretação para cada estilo e devido a variedade de instrumentos utilizados.

2.3 Entropia

2.3.1 A entropia de Shannon

Não existe apenas um conceito de entropia. Apesar de ter sido usado pela primeira vez na teoria da termodinâmica, Claude E. Shannon (1916-2001) propôs este nome para a incerteza em um processo de comunicação. Ele é considerado o precursor da Teoria da informação (ou Teoria

matemática da comunicação), que lida, entre outros assuntos, com sistemas de comunicação, transmissão de dados e códigos corretores de erros.

A teoria da informação considera a comunicação como um problema matemático baseado na estatística dos símbolos a serem utilizados no processo [29], ou seja, quanto maior for a incerteza do que poderá ser dito, maior é a entropia. É importante observar que para o cálculo da entropia não são utilizados os símbolos *per se*, mas apenas suas probabilidades. A grandeza entropia traduz a idéia de "informação", ou seja, mensagens mais improváveis geram mais informação que mensagens comuns.

Um exemplo clássico é o lançamento de uma moeda. Para o cálculo da entropia, Shannon propôs a equação

$$-\sum_{i=0}^{N} p_i \log_2(p_i), \tag{2.3}$$

onde p_i é a probabilidade de ocorrência do símbolo. O "N" no suporte representa todos os símbolos possíveis [29]. Para o caso da moeda honesta tem-se

$$H = -(\frac{1}{2}log_2\frac{1}{2} + \frac{1}{2}log_2\frac{1}{2}) = 1bit.$$

A unidade da entropia depende da base do logaritmo utilizada. Base 2 gera o resultado em bits. Outra maneira de interpretar o resultado da entropia é como o número médio de perguntas necessárias para adivinhar qual foi o evento ocorrido. É fácil observar que, neste exemplo, com uma simples pergunta sabemos exatamente qual face caiu virada para cima.

2.3.2 A entropia espectral

Como colocado por Rosso *et. al.* [34], uma abordagem natural para quantificar a complexidade de um sinal é considerar sua entropia espectral, definida a partir do espectro de potência [30] do mesmo. Esta medida quantifica quão concentrada ou espalhada a densidade de potência (FPE) do sinal se encontra. Um sinal senoidal, exemplo altamente ordenado, manifesta-se como um pulso bem definido no domínio da frequência. Logo, analisando o espectro, temos uma baixa entropia. Por outro lado, uma atividade desordenada, como um ruído, apresenta ampla resposta

em frequência, refletindo numa alta entropia.

No entanto, a aplicação da Transformada de Fourier em um sinal requer estacionaridade no sentido amplo deste [34] e ainda, a FT nada diz sobre a evolução dos padrões de frequência ao longo do tempo, logo, a entropia espectral não é definida em função do tempo [34]. Esse problema pode ser parcialmente resolvido, utilizando a Transformada de Fourier de tempo reduzido (STFT). Nessa abordagem, a FT é aplicada a janelas curtas do sinal que evoluem com o tempo, e sendo assim a questão da estacionaridade é parcialmente satisfeita e observa-se a evolução da frequência no tempo. A limitação dessa abordagem deve-se ao janelamento, atrelado ao princípio da incerteza [41]: se a janela é pequena, a resolução das frequências é baixa; se a janela é grande, a resolução temporal fica prejudicada. Estas limitações são importantes quando o sinal sob análise apresenta transientes no tempo, como é o caso de sinais musicais e de atividade cerebral (aplicação original da ferramenta utilizada neste trabalho).

Para superar essas limitações, uma entropia que evolui com o tempo pode ser definida a partir da representação em tempo-frequência do sinal, como é o caso da transformada *wavelet*, que nada assume quanto a estacionaridade e cujo único parâmetro de entrada é o sinal no domínio do tempo [5], [3], [4]. Nesse caso, a evolução dos padrões de frequência no tempo podem ser analisados com uma resolução ótima no tempo-frequência e, logo, além de refletir o grau de ordem ou desordem de um sinal, a entropia baseada em WT diz respeito ao processo dinâmico inerente a esse sinal [33]. Deve-se observar, entretanto, que o filtro *wavelet* utilizado na decomposição influencia diretamente esta conclusão e a questão da incerteza.

2.3.3 A entropia baseada em *wavelet*

Neste trabalho a entropia é calculada de uma maneira diferente daquela proposta por Shannon. Ao invés de abordar a probabilidade dos símbolos, como descrito anteriormente, esta abordagem chega no valor de entropia a partir da energia de trechos do sinal em questão, refletidos nos coeficientes de uma transformada *wavelet*, como comentado na seção anterior [1], [34], [18], [4].

Como já foi mencionado, a máxima entropia possível de uma distribuição (no caso, do sinal) ocorre quando esta é uma distribuição equiprovável, ou quando todos os p_i possuem mesmo valor [1]. Neste caso, a informação encontra-se igualmente distribuída ao longo do sinal. Qualquer outro tipo de distribuição resultará em um valor de entropia menor. Por outro

lado, quanto mais agrupada estiver a informação menor será a entropia do sinal. A entropia mínima ocorre quando toda a informação está contida em um único local, i.e., em apenas um valor de i, onde $p_i = 1$.

Nesta adaptação da fórmula clássica de entropia, o denominador equivale a energia total do sinal definido na janela sob análise [34], então

$$\Big(\sum_{j=0}^{1023} x_j^2 = 1\Big), \tag{2.4}$$

onde x_j é o valor de cada coeficiente da WT referente aquela janela de 1024 amostras. A energia normalizada de cada coeficiente x_i é a proporção relativa do total de energia contida naquele coeficiente. Na verdade, x_i e x_j estão diferenciados apenas para facilitar a compreensão do cálculo. Define-se então [1] a entropia em função das energias normalizadas dos coeficientes como

$$-\sum_{i=0}^{1023} p_i \log_2(p_i), \tag{2.5}$$

onde

$$p_i = \frac{{x_i}^2}{\Big(\sum\limits_{j=0}^{1023} x_j^2\Big)}. \tag{2.6}$$

É importante ressaltar que esta equação para o cálculo de p_i satisfaz os axiomas de Kolmogorov [27]:

- Primeiro axioma: para cada janela W_j, se $x_i \in W_j$, então $P(x_i) \geq 0$, $\forall x_i \in W_j$;
- Segundo axioma:

$$\sum_{i=0}^{1023} \frac{{x_i}^2}{(\sum\limits_{j=0}^{1023} x_j^2)} = 1; \tag{2.7}$$

- Terceiro axioma: tomando $X_1 = [x_i, x_{i+1}, ..., x_{i+m}]$ e $X_2 = [x_j, x_{j+1}, ..., x_{j+n}]$, sendo que $X_1 \cap X_2 = 0$, então $P(X_1 \cup X_2) = P(X_1) + P(X_2)$,

e sendo assim, p_i é adequada para ser interpretada como uma medida descritiva do espaço amostral.

No entanto, a fórmula como está colocada pressupõe a entidade "informação" diretamente relacionada com a amplitude do sinal. Não é verdade que baixas amplitudes (nula, inclusive) podem ser interpretadas como não-conteúdo. O silêncio, por exemplo, quando bem

explorado, pode ser tão belo quanto notas bem escolhidas. Pode-se ainda pensar que, ao introduzir uma pausa no lugar de determinada nota de uma escala ou ainda utilizar o silêncio em detrimento de resolver uma cadência, tem-se uma grande quantia de informação injetada, dada a imprevisibilidade do evento. Esta é apenas uma diferente interpretação de entropia, que calculada dessa maneira reflete a complexidade na dinâmica do sinal para cada faixa de frequência considerada.

Ainda nesse quesito, há que se tomar cuidado com o efeito da *loudness war* que se instaurou no início da década de 90 [24]. A "guerra" entre as gravadoras consiste em deixar seus lançamentos o mais "quente" possível (termo utilizado entre audiófilos e profissionais do áudio), por meio do processamento, analógico ou digital, das gravações. A consequência disso é que a variação de dinâmica nas músicas quase não mais existe, pois quase toda a música se encontra na região de volume intenso.

2.4 Geometria fractal

Como colocado em [23], a geometria é a linguagem matemática para descrever, relacionar e manipular formas. As formas euclidianas posuem apenas uma ou poucas características que podem ser alteradas em escala ou proporção. Já as formas fractais são independentes de escala e são auto-similares. A geometria euclidiana descreve precisamente objetos feitos pelo homem, mas esta descrição não é apropriada para formas encontradas na natureza, como a geometria fractal.

Enquanto as formas euclidianas possuem fórmulas simples, formas fractais são resultado de algoritmos recursivos. Uma forma fractal classica é o *Cantor set*, mostrado na Figura 2.3. Para sua realização executa-se o seguinte algoritmo:

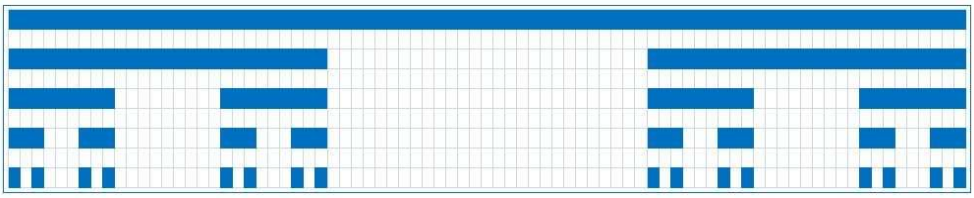

Figura 2.3 – O conjunto de Cantor. Cada pedaço que o compõe remete a estrutura inicial.

- comece com um segmento de reta de comprimento j;
- divida-o em 3 partes de mesmo comprimento $\frac{j}{3}$;
- remova a parte central e transfira os dois pedaços restantes para a linha abaixo (o segmento inicial continua);
- repita o processo.

Se olharmos qualquer pedaço do conjunto com *zoom*, veremos a mesma estrutura inicial. Na natureza, a auto-similaridade pode ser observada, por exemplo, em um relâmpago (Figura 2.4). Se olharmos bem de perto, enxergamos em pequenos pedaços do relâmpago a mesma estrutura do relâmpago inteiro. Depois de alguns zooms, as estruturas ficam muito pequenas para seguirmos adiante. No entanto, em uma idealização matemática, a propriedade de auto-similaridade de um fractal continua por estágios infinitos. Este fato leva ao surgimento de conceitos como dimensão de fractal, que são úteis para estruturas que não possuem tal "condição infinita"[23].

Figura 2.4 – Um relâmpago, exemplo de fractal que ocorre na natureza. Após *zoom*, a estrutura remete ao relâmpago inicial.

2.4.1 Dimensão de fractal

De acordo com [13], [37] um ponto possui dimensão 0, uma reta possui dimensão 1, uma superfície possui dimensão 2 e uma porção de espaço possui dimensão 3. Esses exemplos indicam que, para determinar a dimensão topológica de um objeto, faz-se uma correspondência unívoca desse objeto com um ente geométrico elementar. Diferentemente, a dimensão fractal (D) refere-se à dimensão espacial, o espaço ocupado por um sinal ou figura, podendo o valor correspondente ser um número fracionário. A dimensão do fractal, portanto, consiste em uma medida para o nível de irregularidade ou auto-similaridade desse sinal ou figura [13].

Existem vários algoritmos para o cálculo da dimensão de fractal. Para este trabalho foi utilizado o algoritmo de *box-counting* [2], que calcula o número de "caixas" necessárias para cobrir todo o sinal a ser analisado (como exemplo usaremos o sinal representado na Figura 2.5).

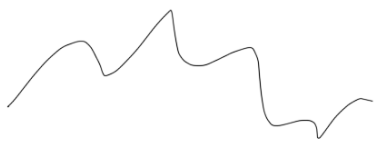

Figura 2.5 – Exemplo de um sinal qualquer, que contém N pedaços similares.

Para isso o sinal é colocado em um *grid* (quadriculado/grelha) com quadrados de tamanho δ. A Figura 2.6 ilustra tal procedimento.
Conta-se então o número de quadrados necessários para varrer todo o sinal. Logicamente, o número de caixas depende do tamanho escolhido para δ. A dimensão do fractal é dada [2] então pela equação

$$D = -\frac{log(N)}{log(\frac{1}{\delta})}. \tag{2.8}$$

Tomamos o limite com δ tendendo a zero para que caixas com tamanhos recursivamente distintos sejam usadas para cobrir o fractal. Ao se desenhar uma curva tomando valores diferentes para δ e contando o número de caixas necessárias para estes valores de δ, teremos o valor da

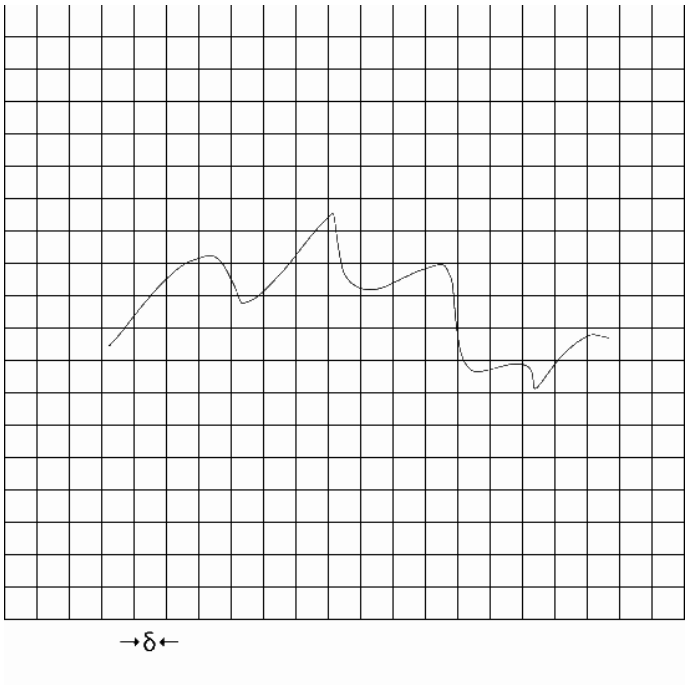

Figura 2.6 – Agora o sinal é colocado em um *grid* para que o número de caixas seja contado.

dimensão de fractal representado pela inclinação desta curva.

O algoritmo de *box-counting* é de fácil cálculo e pode ser rodado em formas auto-similares ou não. Estas formas podem, inclusive, estar em espaços de mais dimensões. Pensando em obejos 3D, as"*box*" contadas não serão quadrados no plano, mas cubos com as 3 dimensões [23]. Al-Akaidi mostra em [2] que este algortimo é muito preciso quando a dimensão de fractal de um sinal possui valor entre 1 e $1,5$ (que é o caso dos sinais musicais utilizados neste trabalho), (ou entre 2 e $2,5$ no caso de imagens), e, saindo dessa faixa, perde acurácia e é de difícil cálculo.

2.4.2 Lacunaridade

Lacunaridade é um termo que vem do latim *lacuna*, que significa intervalo [2]. Tem a ver com a distribuição dos tamanhos dos intervalos de informação e de vazio no sinal. É uma medida homóloga à dimensão de fractal, e contribui na descrição da textura de um fractal. Diferentes fractais podem apresentar mesma dimensão, mas parecerem bem diferentes por possuírem lacunaridades diferentes. A diferença entre eles reside na maneira em que os intervalos estão distribuídos [2].

Baixa lacunaridade, geralmente, representa homogeneidade, e altos valores de lacunaridade representam heterogeneidade. Quanto mais alto o valor da lacunaridade, maior será a variação da distribuição de pixels em uma imagem, por exemplo. Em outras palavras, lacunaridade alta significa que estes pixels estão agrupados em uma grande variedade de tamanhos de ilhas de pixels rodeadas por uma variedade de vazios, indicando heterogeneidade de um padrão ou textura.

Para o cálculo da lacunaridade é utilizado o algoritmo da *gliding-box* (caixa deslizante), que analisa a distribuição de massa do fractal [2], [23]. Como o princípio é o mesmo em qualquer dimensão será apresentado um exemplo de fractal 3D, por ser de mais fácil visualização. Este exemplo é adaptado de [23]. Considere as Figuras 2.7 e 2.8. São as duas primeiras iterações de um fractal. Os passos para o cálculo da lacunaridade deste fractal são descritos a seguir:

- ajuste o objeto dentro de um cubo de lado S (Bounding Box - BB), como mostra a Figura 2.9. A medida S é função do tamanho do objeto;

- tome um cubo de tamanho $s < S$ que vai deslizar (Gliding Box - GB) por toda a BB e computar a distribuição de massa, contando o número de GBs não-vazias. Ao se trabalhar com imagens, escolhe-se um limiar a partir do qual o conteúdo nas GBs é considerado não-vazio.

- Calcula-se então as probabilidades de massa para o objeto em questão e chega-se ao valor da lacunaridade.

Neste exemplo, tem-se uma GB com $S = 3$ e será feita uma análise com $s = 2$ (novamente, este exemplo e estes valores foram escolhidos dada a fácil visualização). Escolhido S, o número de GBs que devem ser analisadas é $N(s) = (S - s + 1)^3$. A frequência de GBs de lado s com distribuição de massa M, $n(M, s)$, dividida pelo total de GBs $N(s)$, define uma função de probabilidade

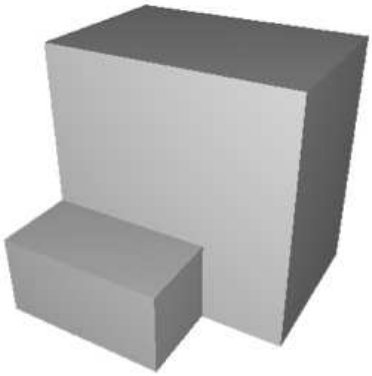

Figura 2.7 – Exemplo da primeira iteração de um fractal. Figura adaptada de [23].

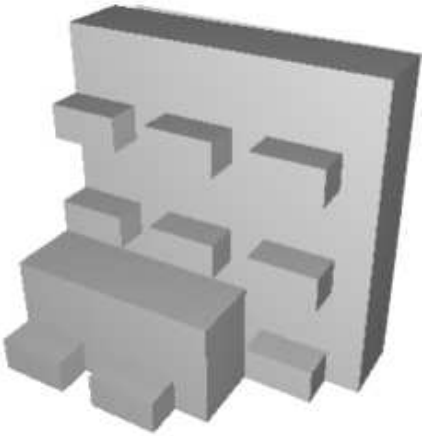

Figura 2.8 – Exemplo da segunda iteração de um fractal. Figura adaptada de [23].

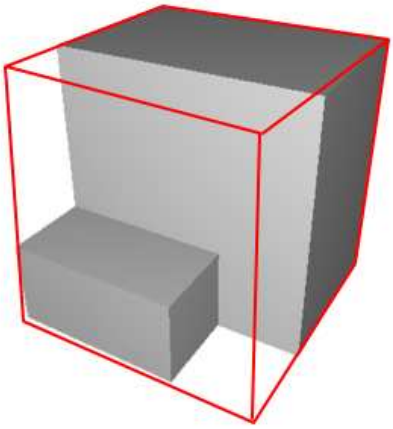

Figura 2.9 – Fractal colocado dentro da *bounding box*. Figura adaptada de [23].

$$Q(M, s) = \frac{n(M, s)}{N(s)}, \tag{2.9}$$

onde $Q(M, s)$ representa a probabilidade de uma GB de lado s possuir M voxels [2] não-vazios. A lacunaridade é então calculada (para GB de lado s) dividindo o segundo momento de Q(M,s) pelo quadrado do seu primeiro momento:

$$\Lambda(s) = \frac{\sum_{i=1}^{N} M^2 Q(M, s)}{(\sum_{i=1}^{N} MQ(M, s))^2}. \tag{2.10}$$

Neste caso, um cubo mágico pode ajudar na visualização. Note que 8 cubinhos de lado 2 (cada um sendo composto por 8 cubinhos de lado 1, a unidade mínima, neste caso, o voxel) varrem todo o cubo principal de lado 3 (o cálculo da lacunaridade admite a sobreposição de GBs). Analisando o cubo mágico, todas as 8 GBs possuirão massa 4.

Analisando a distribuição de massa do fractal em questão, começando pela parte de trás, as 4 GBs (de lado 2, portanto, de 8 voxels) possuem massa 8 (uma vez que todos seus voxels são não-vazios). Já das 4 GBs da parte da frente, 2 possuem massa 4, 1 possui massa 5 e a outra possui massa 6. Isso leva a tabela 2.1, mostrada a seguir.

Tabela 2.1: Resultados da análise da distribuição de massa para cálculo da lacunaridade.

Massa:i	**Frequência** $n(M_i, s)$	**Probabilidade** $Q(M_i, s)$	$M_i Q(M_i, s)$	$M_i^2 Q(M_i, s)$
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	2	0.25	1	4
5	1	0.125	0.625	3.125
6	1	0.125	0.75	0
7	0	0	0	4.5
8	4	0.5	4	32
Total	8	1	6.375	43.625

Para este fractal, temos que $\Lambda(s) = \frac{43,625}{(6,375)^2} = 1,073$. Analisando o cubo mágico, são 8 GBs com $M = 4$. Isso dá um valor unitário para a lacunaridade (este é o caso em que o objeto a ser analisado é a própria BB). É intuitivo pensar que o fractal apresentado é mais heterogêneo do que um cubo regular.

[2] Equivalente de pixel em 3D.

2.5 Classificadores

2.5.1 *Support Vector Machines* - Máquinas de vetores suporte

O SVM é um algoritmo de aprendizado supervisionado que infere de um conjunto de exemplos cuja classe é conhecida uma função capaz de predizer a qual classe pertencem novos exemplos desconhecidos [21]. Sendo assim a saída do classificador é uma função matemática definida no espaço que contém os exemplos de aprendizado. O algoritmo SVM decorre das funções lineares, as mais simples das funções de decisão.

Funções de decisão lineares consistem de um limite de decisão que é um hiperplano que separa duas regiões diferentes no espaço. Este hiperplano é visualizado como uma linha em 2D, um plano em 3D, etc. Tais funções de decisão são expressas por uma função do vetor $\overrightarrow{x}$ e pode assumir os valores +1 ou -1, que representam as classes distintas.

A função pode ser parametrizada por um escalar b e por um vetor de pesos w, e fica expressa como

$$f(\overrightarrow{x}) = \langle \overrightarrow{w}, \overrightarrow{x} \rangle + b, \tag{2.11}$$

onde $\langle \overrightarrow{w}, \overrightarrow{x} \rangle$ é o produto interno entre $\overrightarrow{w}$ e $\overrightarrow{x}$ e é obtido com $\langle \overrightarrow{w}, \overrightarrow{x} \rangle = \sum_{i=1}^{d} w_i x_i$, sendo que d é a dimensão. O que a SVM faz é escolher os parâmetros w e b da função de decisão linear que melhor generaliza para exemplos desconhecidos, ou seja, que escolhe a melhor margem de separação das classes, aquela que além de separar corretamente duas classes está tão distante quanto possível dos exemplos de treinamento [21], como mostrado na Figura 2.10. Isso elimina, ou pelo menos reduz a chance de que pequenas perturbações alterem o resultado da classificação, como aconteceria no primeiro caso da Figura 2.10.

Existem duas classes, uma representada por $y = +1$ e outra por $y = -1$. Para achar um hiperplano que separe-as corretamente deve-se satisfazer $\langle \overrightarrow{w}, \overrightarrow{x_i} \rangle + b > 0$, para todo $y_i = 1$ e $\langle \overrightarrow{w}, \overrightarrow{x_i} \rangle + b < 0$, para todo $y_i = -1$, o que equivale satisfazer $(\langle \overrightarrow{w}, \overrightarrow{x_i} \rangle + b)y_i > 1, \forall i$. Mas a intenção é encontrar o hiperplano com maior margem, sendo esta a distância entre o hiperplano e o exemplo de treinamento mais próximo. A Figura 2.11 ilustra este processo. A margem é dada por $\frac{2}{\sqrt{\langle \overrightarrow{w}, \overrightarrow{w} \rangle}}$. Chega-se então no problema de otimização

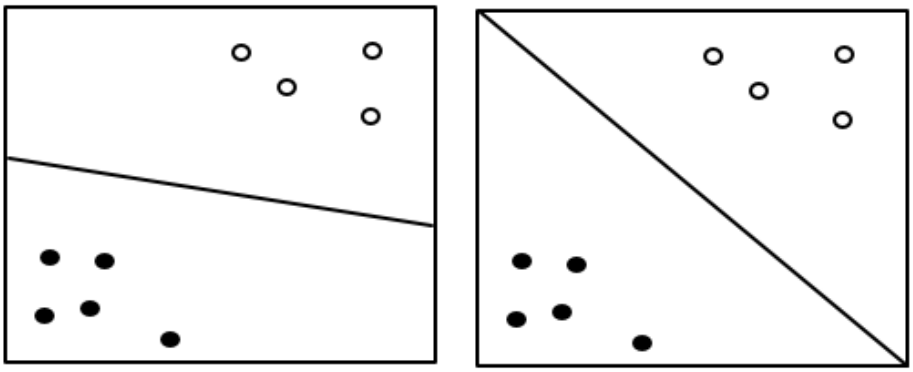

Figura 2.10 – Separação das classes. Note que a função à direita separa melhor. Figura adaptada de [21].

$$min_{w,b}\frac{1}{2}\langle\overrightarrow{w},\overrightarrow{w}\rangle, \tag{2.12}$$

sendo que

$$(\langle\overrightarrow{w},\overrightarrow{x_i}\rangle + b)y_i \geq 1, \forall i.$$

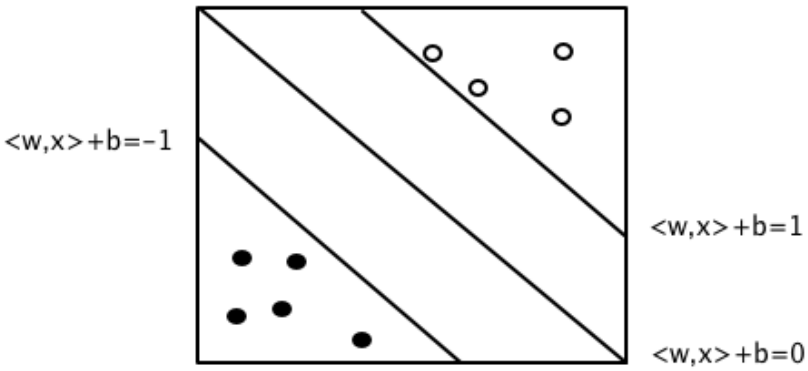

Figura 2.11 – Hiperplano ótimo - os exemplos por onde passam $\langle w, x\rangle + b = 1$ e $\langle w, x\rangle + b = -1$ são os vetores suporte. Figura adaptada de [21].

No entanto, a classificação baseada em funções lineares é de pouca aplicação, uma vez que dificilmente os dados apresentam-se como nos exemplos citados. Na maioria dos problemas reais, a separação nao é possível linearmente, então não haverá $\overrightarrow{w}$ e b que satisfaçam as condições. Precisamos então permitir que alguns dados violem as condições, ou seja, passamos a permitir que alguns exemplos de treinamento situem-se no lado ao qual não pertencem [21], como exemplificado na Figura 2.12. Usando deste artifício, é possível classificar conjuntos de dados que não são linearmente separáveis e além disso a capacidade de generalização do

classificador aumenta [21].

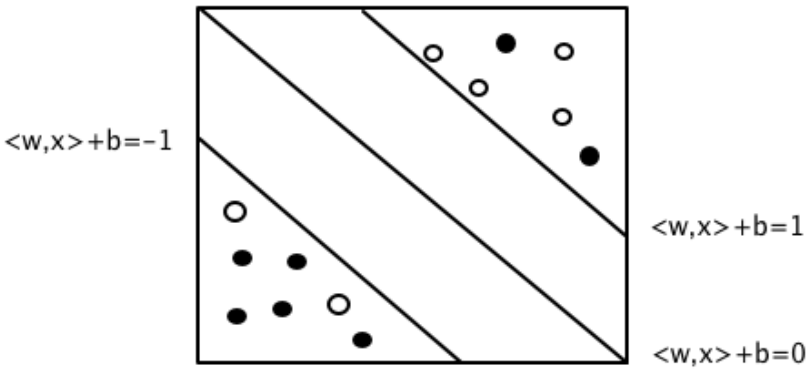

Figura 2.12 – Aqui não há possibilidade de separação linear. Necessidade de *soft margin*. Figura adaptada de [21].

Faz-se necessário então introduzir novos parâmetros no sistema, C e ξ, os quais variamos para balancear as condições, que no caso são o tamanho da margem e a correta classificação dos dados de treinamento [21]. Chega-se então no problema de otimização

$$min_{w,b,\xi}(\frac{1}{2}\langle\overrightarrow{w},\overrightarrow{w}\rangle + C\sum_{i=1}^{m}\xi_i), \tag{2.13}$$

sendo que

$$(\langle\overrightarrow{w},\overrightarrow{x_i}\rangle + b)y_i + \xi \geq 1, \forall i.$$

Note que ξ será subtraído de 1 do lado direito da equação, então ξ_i representa o quanto permitimos que a margem seja violada pelo exemplo x_i. Note ainda que C multiplica a soma de todos os ξ_i. Então se o valor de C aumenta, aumentando a margem, cresce também o número de exemplos de treinamento que violarão a mesma.

Para conjuntos de dados com distribuição simples o que foi apresentado até o momento é suficiente para a tarefa de classificação. No entanto, conforme aumenta a complexidade dos dados, surge a necessidade de funções cada vez mais complexas, como é o caso da Figura 2.13. Neste exemplo não é possível classificar com uma função de decisão linear. Pode-se pensar em uma circunferência para a classificação, que apresentaria uma boa capacidade de generalização, mas pode-se imaginar um mapeamento ϕ, que leva os dados para outra dimensão, aonde será possível uma separação linear com uma boa margem e com capacidade de generalização [21]. Segundo [11], com uma função apropriada de mapeamento $\phi(t)$ não-linear para uma dimensão alta o suficiente, sempre será possível separar duas classes diferentes por um hiperplano.

Como comentado em [21], em 1998 Vapnik e Chervonenkis provaram que isso seria possível [40], convertendo o problema da SVM com *soft margin* em um problema equivalente com multplicadores de Lagrange α. O novo problema, resolvido variando os α_i se resume em

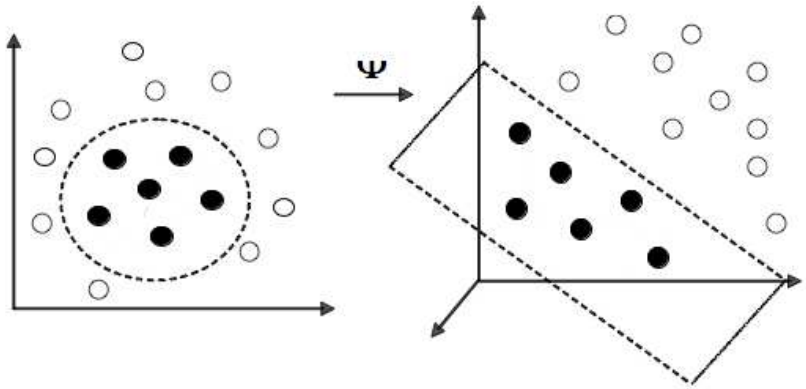

Figura 2.13 – Necessidade de mapeamento não-linear antes da aplicação da SVM. Figura adaptada de [21].

$$min_{\alpha}\frac{1}{2}\sum_{i,j=1}^{m}(y_i y_j \alpha_i \alpha_j (\overrightarrow{x_i}.\overrightarrow{x_j}) - \alpha_i), \tag{2.14}$$

em que

$$\sum_{i=1}^{m} y_i \alpha_i = 0, \qquad 0 \leq \alpha_i \leq C, \qquad i = 1, 2, ...m.$$

A correspondência de α_i com as variáveis primárias w e b se dá por

$$w = \sum_{i=1}^{m} \alpha_i y_i \overrightarrow{x_i}, \tag{2.15}$$

com

$$\alpha_i(y_i(\langle \overrightarrow{w}, \overrightarrow{x_i} \rangle + b) - 1) = 0.$$

Pela linearidade do produto interno fica-se com a função de decisão

$$f(x) = \langle \overrightarrow{w}, \overrightarrow{x} \rangle + b = \sum_{i=1}^{m} \alpha_i y_i \langle \overrightarrow{x_i}, \overrightarrow{x} \rangle + b. \tag{2.16}$$

Uma vez que os vetores de treinamento apenas aparecem sob a forma de seus produtos internos, os vetores de treinamento em si são dispensáveis para a obtenção dos valores ótimos dos α_i e de b, e também para o cálculo de $f(x)$ [21]. A consequência disso é que ao invés de mapear explicitamente todos os dados no novo espaço (a partir de ϕ) e então realizar a classificação por SVM, fica possível operar no espaço original, dado que temos uma função kernel k(.,.) que equivale ao produto interno dos dados mapeados no novo espaço [21], ou seja, $k(x, y) = \langle \phi(x), \phi(y) \rangle$.

Na prática não é preciso se preocupar muito com a exata natureza do mapeamento ϕ. É

importante preocupar-se em encontrar um função $k(x, y)$ que represente uma boa medida de similaridade entre os vetores x e y [21]. Além disso, $k(x, y)$ deve satisfazer as condições de Mercer [11]. Em virtude do sucesso das SVMs, este procedimento de mapear implicitamente os dados por meio da função k, chamado de *kernel trick* é encontrado num vasto leque de aplicações [35].

Encontrar os α_i é um problema da classe dos programas quadráticos (*Quadratic Programs - QP*). Sendo assim, uma propriedade que o SVM QP pode aproveitar é o da esparsidade - o fato de que em muitos casos a solução ótima terá a maioria dos α_i iguais a 0 [21]. Os vetores que possuírem $\alpha_i = 0$ não contribuem para a função de decisão, e aqueles com $\alpha_i \neq 0$ serão os vetores suporte. Da mesma forma que na classificação linear, os vetores suporte são os exemplos em que a classificação é mais difícil, pois são os que se encontram mais próximos do limite de decisão.

Neste trabalho, e na maioria das aplicações, o kernel utilizado é o gaussiano, que é da forma

$$k(x, y) = exp(-\frac{\|\overrightarrow{x} - \overrightarrow{y}\|^2}{\sigma^2}), \qquad \sigma \neq 0. \tag{2.17}$$

O kernel gaussiano, que realiza o mapeamento num espaço dimensional infinito, é similar a distribuição de probabilidade gaussiana e é um grupo de funções kernel conhecido como *Radial Basis Functions*. RBFs são funções kernel que dependem apenas da distância geométrica entre $\overrightarrow{x}$ e $\overrightarrow{y}$.

Ao se trabalhar com SVMs, é extremamente importante a preocupação com a possibilidade de ocorrência de *overfitting*. Neste caso, a função de decisão consegue mapear bem o conjunto de treinamento, mas perde a capacidade de generalização para novos exemplos, quando na verdade o que se deseja é o oposto - um sistema com boa capacidade de generalização, mesmo que incapaz de mapear perfeitamente os vetores de treinamento.

2.5.2 *Gaussian Mixture Models* - Modelos de mistura de Gaussianas

Um modelo de mistura de gaussianas (GMM) é uma função densidade de probabilidade que representa uma soma ponderada de densidades com componentes gaussianas. Os parâmetros do GMM são estimados a partir de dados de treinamento utilizando o algoritmo iterativo *Expectation Maximization* (EM), ou então por estimativa a partir de um modelo prévio bem treinado, método conhecido como *Maximum a Posteriori* (MAP).

A soma ponderada de M densidades com componentes gaussianos é dada por

$$p(x|y) = \sum_{i=1}^{M} w_i g(\overrightarrow{x}|\overrightarrow{\mu_i}, \Sigma i), \tag{2.18}$$

onde $\overrightarrow{x}$ é um vetor N-dimensional de medição de parâmetros, w_i são os pesos da mistura e $g(\overrightarrow{x}|\overrightarrow{\mu_i}, \Sigma i)$ são os componentes de densidade gaussiana. Cada uma dessas componentes é uma função gaussiana N-variada da forma

$$g(\overrightarrow{x}|\overrightarrow{\mu_i}, \Sigma i) = \frac{1}{(2\pi)^{\frac{N}{2}}|\Sigma_i|^{\frac{1}{2}}} exp(-\frac{1}{2}(\overrightarrow{x} - \overrightarrow{\mu_i})'\Sigma_i^{-1}(\overrightarrow{x} - \overrightarrow{\mu_i})), \tag{2.19}$$

sendo $\overrightarrow{\mu_i}$ o vetor das médias e Σ_i a matriz de covariância. Os pesos satisfazem a condição $\sum_{i=1}^{M} w_i = 1$. Tais parâmetros são representados coletivamente pela notação

$$\lambda = \{w_i, \overrightarrow{\mu_i}, \Sigma_i\}, \qquad i = 1, ..., M.$$

É importante notar que por se tratar de uma mistura de gaussianas, os parâmetros podem ser compartilhados, ou amarrados entre os diversos componentes [31], por exemplo, a partir de uma matriz de covariância comum para todos os componentes. A escolha do modelo normalmente é determinada pela quantidade de dados disponíveis para estimar os parâmetros do GMM e qual será a aplicação. Os modelos variam quanto ao número de componentes, tipo de matriz de covariância e tipo de amarramento dos parâmetros.

Devido a sua alta capacidade de representar um grande número de classes de amostras, GMMs são largamente utilizados em sistemas biométricos, particularmente em sistemas de reconhecimento de locutor [31]. Como mostrado no Capítulo 2, os trabalhos com os melhores resultados fizeram uso de clasificadores baseados em GMM. Um de seus atributos poderosos é a capacidade de aproximar suavemente densidades arbitrariamente distribuídas. Gray [16] coloca que o modelo clássico uni-modal gaussiano representa distribuições de parâmetros por meio de uma posição (vetor de médias) e e de uma forma elíptica (matriz de covariância), enquanto que um modelo de *nearest neighbour* (vizinhos mais próximos) representa a distribuição por um conjunto discreto de *templates* característicos. Já o GMM funciona como um híbrido entre estes dois modelos utilizando um conjunto discreto de funções gaussianas, cada uma com sua média e matriz de covariância, permitindo uma melhor capacidade de modelamento.

Uma colocação interessante de Reynolds [31] [32] é que o uso de GMM para representar distribuições de parâmetros em um sistema biométrico também pode ser motivado pela noção intuitiva de que componentes individuais podem modelar classes ocultas inerentes ao fenômeno. Por exemplo, em reconhecimento de locutores, é razoável assumir que o espaço acústico de parâmetros relacionados ao espectro corresponda aos enventos fonéticos de um lo-

cutor, como vogais, sons nasais ou sons fricativos. Essas classes acústicas refletem algumas dependências entre as configurações do trato vocal de um locutor que são úteis para a tarefa de reconhecimento do mesmo. $\overrightarrow{\mu_i}$ decorre do formato espectral da i-ésima classe acústica e Σ_i reflete variações nesse espectro. Da mesma forma, podemos pensar que diferentes configurações (de instrumentos, ritmos, harmonia, melodia, etc) exploradas em diferentes estilos musicais geram classes ocultas específicas para cada um desses estilos.

Estimativa dos parâmetros do GMM

Sabendo a configuração do GMM e tendo em mão um conjunto de exemplos de treinamento, queremos determinar os parâmetros λ do modelo que melhor representam as distribuições dos vetores de treinamento. Neste trabalho foi utilizado o método de *Maximum Likelihood* (ML), que se resume num caso especial do algoritmo de *Expectation Maximization* (EM), como será mostrado a seguir.

O objetivo da estimativa por ML é encontrar os parâmetros do modelo que maximizam a verossimilhança do GMM, dados os vetores de treino. Para uma sequência de T vetores de treino $X = \{\overrightarrow{x_1}, ..., \overrightarrow{x_T}\}$, a verossimilhança do GMM, assumindo independência entre os vetores é dada por

$$p(X|\lambda) = \prod_{t=1}^{T} p(\overrightarrow{x_t}|\lambda). \tag{2.20}$$

Esta expressão é uma função não-linear dos parâmetros λ, então maximização direta não é possível [31]. No entanto, a estimativa dos parâmetros ML podem ser obtidas iterativamente utilizando um caso especial do algoritmo EM [9].

A idéia básica do algoritmo EM é, começando com um modelo inicial λ, estimar um novo modelo $\overline{\lambda}$, sendo que $p(X|\overline{\lambda}) \geq p(X|\lambda)$. O novo modelo passa a ser o modelo inicial para a próxima iteração, e o processo se repete até que um limite de convergência seja atingido. O modelo inicial é obtido pelo método de vizinhos mais próximos [31]. As fórmulas utilizadas nas iterações são

$$\overline{w_i} = \frac{1}{T} \sum_{t=1}^{T} Pr(i|x_t, \lambda) \tag{2.21}$$

para os pesos da mistura,

$$\overline{\mu_i} = \frac{\sum_{t=1}^{T} Pr(i|x_t, \lambda) x_t}{\sum_{t=1}^{T} Pr(i|x_t, \lambda)} \tag{2.22}$$

para as médias e

$$\overline{\sigma_i}^2 = \frac{\sum_{t=1}^{T} Pr(i|x_t, \lambda)x_t^2}{\sum_{t=1}^{T} Pr(i|x_t, \lambda)} - \overline{\mu_i}^2 \tag{2.23}$$

para as variâncias, onde x_t e μ_i referem-se a elementos arbitrários dos vetores $\overrightarrow{x_t}$ e $\overrightarrow{\mu_i}$, enquanto $\overline{\sigma_i}^2$ refere-se a elementos da matriz Σ_i, de covariância diagonal. [3]

A probabilidade *a posteriori* para o componente i é dada por

$$Pr(i|\overrightarrow{x_t}, \lambda) = \frac{w_i g(\overrightarrow{x_t}|\mu_i, \Sigma_i)}{\sum_{k=1}^{M} w_k g(\overrightarrow{x_t}|\overrightarrow{\mu_k}, \Sigma_k)}. \tag{2.24}$$

2.6 Outros conceitos relevantes

2.6.1 O formato de áudio WAV

Um arquivo no formato WAV é composto por um cabeçalho sub-dividido em duas partes chamadas *chunk 1* e *chunk 2*, seguidas pelos valores de amplitudes das amostras, que fazem parte de uma porção identificada como *chunk* de dados [6]. De acordo com a especificação do formato, no caso de sinais amostrados com uma quantização de 16 bits, a faixa de valores para cada amostra vai de −32768 até 32767. A especificação de cada um dos *chunks* mencionados consta nas Tabelas 2.2, 2.3 e 2.4.

Tabela 2.2 – *Chunk* 1 do formato WAV: componentes e descrições dos 12 bytes integrantes

Bytes	**Descrição**
0 a 3	*string* ascii “RIFF”
4 a 7	comprimento do *chunk*
8 a 11	*string* ascii “WAV”

[3]Esta matriz é uma matriz de covariância diagonal, uma vez que foi assumida independência entre os vetores.

Tabela 2.3 – *Chunk* 2 do formato WAV: componentes e descrições dos 24 bytes integrantes

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

Tabela 2.4 – *Chunk* de dados do formato WAV: componentes e descrições dos bytes integrantes

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

2.6.2 A medida estatística de desvio-padrão

Além dos valores de interesse propriamente ditos, neste trabalho faz-se uso do valor do desvio-padrão destes valores, que representa uma medida de variação nos valores dos parâmetros ao longo da distribuição [37]. Para um vetor genérico $\vec{x}$, de comprimento N, a sua variância σ^2 pode ser definida como

$$\sigma^2 = \frac{1}{N}\sum_{i=0}^{N-1} {x_i}^2 - \left(\frac{1}{N}\sum_{i=0}^{N-1} x_i\right)^2 . \tag{2.25}$$

O desvio-padrão σ é a raiz quadrada do valor da variância.

2.6.3 O conceito de energia

A energia de um sinal de N amostras discretas é um escalar definido como

$$E = \sum_{i=0}^{N-1} {x_i}^2, \tag{2.26}$$

sendo x_i o valor de cada uma das amostras. No presente trabalho, as energias das amostras e das janelas de áudio de várias sub-bandas de frequências de um sinal são utilizadas para o cálculo de entropias de acordo com o método apresentado em [1] e [34], descrito neste capítulo.

Capítulo 3

Descrição da técnica proposta e dos testes

Este capítulo descreve as diferentes combinações entre parâmetros e tipos de classificadores propostas para a tarefa, assim como os testes realizados com cada uma das configurações. Encontra-se também uma discussão sobre a escolha das músicas utilizadas e no final do capítulo encontram-se os algoritmos detalhando cada cenário testado.

3.1 A Arquitetura do Sistema Proposto

O sistema proposto consiste de duas etapas. Na primeira são extraídas características dos arquivos de áudio e são formados os vetores de parâmetros a serem utilizados na segunda etapa, que consiste no treinamento do classificador. Concluída a segunda etapa cada música a ser testada pode ser processada e o sistema retorna qual é o estilo mais provável da música em questão. A seguir serão mostrados os parâmetros e arquiteturas dos classificadores utilizados.

3.1.1 Primeira etapa: vetor de parâmetros.

Três tipos de vetores de parâmetros foram testados. Os três possuem os dois primeiros passos em comum, descritos a seguir. Primeiramente, o cabeçalho do arquivo WAV é removido, res-

tando apenas os valores das amostras do áudio. Em seguida, é feito o janelamento do arquivo. O arquivo de áudio é dividido em janelas de N amostras, com sobreposição de 50%. Por exemplo, para o caso de $N = 1024$ amostras, a primeira janela vai de x_0 até x_{1023}, a segunda vai de x_{512} até x_{1535} e assim por diante.

O primeiro sistema projetado é baseado em um vetor de 5 parâmetros. Após os dois primeiros passos, calcula-se a entropia de cada janela de acordo com a equação apresentada no capítulo 2. Com o valor da entropia de cada janela calculado, obtém-se os cinco elementos do vetor de parâmetros, calculados no domínio do tempo:

- média das entropias;
- desvio-padrão das entropias;
- maior de todas as entropias;
- menor de todas as entropias;
- maior diferença da entropia entre janelas vizinhas.

A única diferença para o segundo tipo de vetor de parâmetros reside no fato de que antes de se calcular o valor da entropia de cada janela, o sinal de música é submetido a uma Transformada *Wavelet* e passa para o domínio do tempo-frequência, ficando da seguinte forma:

- média das entropias (extraídas após DWT);
- desvio-padrão das entropias (DWT);
- maior de todas as entropias (DWT);
- menor de todas as entropias (DWT);
- maior diferença da entropia entre janelas vizinhas (DWT).

Também foi testada uma pequena variação deste tipo de vetor, adicionando um sexto elemento ao vetor, a dimensão de fractal, obtida no domínio do tempo.

Já o terceiro tipo de vetor de parâmetros consiste de 10 elementos. Após os dois passos iniciais, a lacunaridade de cada janela é calculada (no domínio do tempo). Em seguida é aplicada a Transformada *Wavelet*, e então calcula-se o valor da entropia de cada janela no domínio do tempo-frequência. Com o valor da lacunaridade e da entropia de cada janela, forma-se o vetor de parâmetros:

- média das lacunaridades (domínio do tempo);
- desvio-padrão das lacunaridades (tempo);

- maior de todas as lacunaridades (tempo);
- menor de todas as lacunaridades (tempo);
- maior diferença da lacunaridade entre janelas vizinhas (tempo);
- média das entropias (DWT);
- desvio-padrão das entropias (DWT);
- maior de todas as entropias (DWT);
- menor de todas as entropias (DWT);
- maior diferença da entropia entre janelas vizinhas (DWT).

3.1.2 Segunda etapa: classificador.

Dois tipos de classificadores foram experimentados, em três arquiteturas diferentes. A primeira arquitetura é baseada em SVMs. Para cada gênero musical em questão, uma SVM é treinada para retornar valores positivos caso detecte seu estilo e valores negativos caso detecte algum dos outros estilos.

O segundo tipo de arquitetura também explora uma SVM para cada estilo possível. No entanto, cada SVM é treinada apenas com seu gênero, e nada é "ensinado" sobre os outros estilos. Em ambos os casos, a música a ser classificada passa pelas SVMs de todos os estilos, e adota-se como resultado o estilo cuja SVM retornou o maior valor. A terceira arquitetura consiste em modelos de mistura de gaussianas (GMMs), que retornam uma probabilidade para cada gênero em questão, adotando-se como resultado o estilo com maior probabilidade.

Uma observação interessante é a de que o sistema de classificação do segundo tipo, por treinar cada uma das SVMs levando em conta apenas os elementos de uma classe particular a ser identificada, é relativamente distinto dos esquemas tradicionais de classificação. Em particular, cada uma das SVMs, isoladamente, ao invés de criar uma margem de separação entre elementos de classes distintas, minimiza as distâncias de cada ponto do espaço até o ponto 1 ou -1, conforme o caso, ou seja, seleciona a máxima função de transferência. Estes resultados serão observados na seção seguinte, assim como os resultados obtidos das diversas combinações entre vetores de parâmetros e arquitetura de classificadores.

3.2 Músicas utilizadas

Duas coleções de músicas foram montadas para a realização dos testes. Uma delas é baseada em 3 estilos musicais (intuitivamente bem diferentes), sendo eles o *blues*, a música clássica e o *lounge* (música de coquetel). A outra é baseada em 4 estilos brasileiros, sendo eles a bossa-nova, o samba, o axé e o forró, numa tentativa de dificultar a classificação adicionando mais um gênero e aumentando a similaridade entre eles. As músicas foram extraídas diretamente de CDs, a 44100 amostras por segundo e 16 bits de resolução, formato WAV. No Apêndice II encontram-se os nomes de todas as músicas utilizadas, assim como dos intérpretes.

É importante ressaltar que a classificação destes estilos é subjetiva, como já foi mencionado. A escolha dos gêneros e das músicas, assim como essa prévia classificação, foram realizadas pelo autor e serviram de referência para os testes que seguiram.

3.3 Métodos

Os testes foram realizados utilizando os diferentes tipos de vetores de parâmetros descritos em 3.1.1. As diferentes configurações serão discutidas nesta seção. Os algoritmos utilizados em cada cenário encontram-se detalhados na seção seguinte (3.4) e ilustram com clareza cada configuração experimentada.

Primeiramente foram utilizados apenas os vetores com valores de entropia e classificação baseada em SVM. Foram testadas as 4 possíveis combinações (entropias no tempo e entropias no tempo-frequência / SVM tipo 1 e SVM tipo 2). Na sequência foram realizados testes considerando também a dimensão de fractal, utilizando o melhor cenário obtido nos testes até então. Estes testes foram realizados em cima da coleção de 3 estilos, utilizando a música inteira para extração dos parâmetros.

Para os testes com a coleção de 4 estilos de músicas brasileiras foi utilizado o terceiro tipo de vetor de parâmetros. Sendo assim, entropia e lacunaridade são a base para a classificação. O classificador utilizado nesta etapa é baseado em GMMs. Neste grupo de testes, apenas o trecho central das músicas foi utilizado para a extração dos parâmetros. O quarto inicial e o quarto final não foram considerados. Tal medida, além de possibilitar economia compu-

tacional, é ao mesmo tempo uma tentativa de não confundir os classificadores com informações distorcidas sobre determinado gênero, como passagens no início e no final das músicas. Muitas canções apresentam uma longa introdução com conteúdo diferente daquele que será apresentado ao longo da mesma. Da mesma maneira, o final das músicas pode apresentar elementos dissonantes em relação ao resto da composição.

Algumas arquiteturas foram abandonadas logo no início de seus testes. Com a coleção de 4 gêneros foi testada a mesma configuração utilizada com a coleção de 3 músicas: entropia e SVM. Logo nos primeiros testes percebeu-se que não seria possível alcançar bons resultados com essa configuração. A partir desse ponto a lacunaridade foi adotada como uma maneira de enriquecer os parâmetros utilizados na classificação. No entanto, tal esforço ainda não foi suficiente. Então, o classificador foi alterado para o GMM, medida que gerou resultados proeminentes, e esta configuração foi referência para os testes com os estilos brasileiros. Dado o sucesso desta arquitetura uma nova rodada de testes foi realizada aplicando-a na coleção de 3 estilos, para efeito de comparação direta com o sistema baseado em SVMs.

Foram também testadas janelas com $N = 2048$ amostras, mas os resultados logo mostraram-se muito piores que o uso de $N = 1024$, e sendo assim tal tamanho de janela logo foi descartado. Esta abordagem foi experimentada tanto em SVMs quanto em GMMs.

Os programas (vide apêndice III) foram escritos em C e C++ para plataforma UNIX com colaboração de R.C. Guido [15], [14], [13], [37]. Todos os algoritmos utilizados para extração de parâmetros e classificação (etapa de testes) possuem ordem de complexidade $O(n)$, ou seja, linear em relação ao tamanho do arquivo de entrada.

3.4 Algoritmos

Os algoritmos das técnicas utilizadas para obtenção das classificações são mostrados a seguir. Como diferentes cenários foram testados os algoritmos servem como guias para as tabelas de resultados obtidas que são mostradas no capítulo seguinte (Tabelas 4.1 a 4.7).

3.4.1 Algoritmo referente a Tabela 4.1.

- passo 1: separar a base de dados entre “músicas para treinamento” e “músicas para teste”;
- passo 2: extrair os dados brutos dos arquivos (músicas);
- passo 3: normalizar as amostras dos arquivos;
- passo 4: separar as amostras em janelas de 1024 amostras cada uma;
- passo 5: calcular a entropia de cada janela;
- passo 6: calcular a média e o desvio-padrão das entropias;
- passo 7: encontrar o maior valor de entropia;
- passo 8: encontrar o menor valor de entropia;
- passo 9: calcular as diferenças entre as entropias das janelas vizinhas, armazenando o maior desses valores;
- passo 10: para cada música, formar um vetor de parâmetros com cinco características, descrito no capítulo 4 como sendo o vetor de parâmetros do primeiro tipo;
- passo 11: treinar as SVMs (tipo 1, descrito no capítulo 4) com os vetores de parâmetros das músicas reservadas para teste;
- passo 12: testar a classificação de cada um dos vetores de parâmetros das músicas reservadas para teste.

3.4.2 Algoritmo referente a Tabela 4.2.

- passo 1: separar a base de dados entre “músicas para treinamento” e “músicas para teste”;
- passo 2: extrair os dados brutos dos arquivos (músicas);
- passo 3: normalizar as amostras dos arquivos;
- passo 4: separar as amostras em janelas de 1024 amostras cada uma;
- passo 5: aplicar a DWT em cada janela;
- passo 6: calcular a entropia, baseada em *wavelets*, de cada janela;
- passo 7: calcular a média das entropias;
- passo 8: calcular o desvio-padrão das entropias;
- passo 9: encontrar o maior valor de entropia;
- passo 10: encontrar o menor valor de entropia;
- passo 11: calcular as diferenças entre as entropias das janelas vizinhas, armazenando o maior desses valores;
- passo 12: para cada música, formar o vetor de parâmetros com cinco características, descrito no capítulo 4 como sendo o vetor de parâmetros do segundo tipo;
- passo 13: treinar as SVMs (tipo 1, descrito no capítulo 4) com os vetores de parâmetros das músicas reservadas para treinamento;
- passo 14: testar a classificação de cada um dos vetores de parâmetros das músicas reservadas para teste.

3.4.3 Algoritmo referente a Tabela 4.3.

- passo 1: separar a base de dados entre “músicas para treinamento” e “músicas para teste”;
- passo 2: extrair os dados brutos dos arquivos (músicas);
- passo 3: normalizar as amostras dos arquivos;
- passo 4: separar as amostras em janelas de 1024 amostras cada uma;
- passo 5: calcular a entropia de cada janela;
- passo 6: calcular a média das entropias;
- passo 7: calcular o desvio-padrão das entropias;
- passo 8: encontrar o maior valor de entropia;
- passo 9: encontrar o menor valor de entropia;
- passo 10: calcular as diferenças entre entropias de janelas vizinhas, armazenando o maior desses valores;
- passo 11: para cada música, formar um vetor de parâmetros com cinco características, descrito no capítulo 4 como sendo o vetor de parâmetros do primeiro tipo;
- passo 12: treinar SVMs (tipo 2, descrito no capítulo 4) com os vetores de parâmetros das músicas reservadas para treinamento;
- passo 13: testar a classificação de cada um dos vetores de parâmetros das músicas reservadas para teste.

3.4.4 Algoritmo referente a Tabela 4.4.

- passo 1: separar a base de dados entre “músicas para treinamento” e “músicas para teste”;
- passo 2: extrair os dados brutos dos arquivos (músicas);
- passo 3: normalizar as amostras dos arquivos;
- passo 4: separar as amostras em janelas de 1024 amostras cada uma;
- passo 5: aplicar a DWT em cada janela;
- passo 6: calcular a entropia baseada em *wavelet* de cada janela;
- passo 7: calcular a média das entropias;
- passo 8: calcular o desvio-padrão das entropias;
- passo 9: encontrar o maior valor de entropia;
- passo 10: encontrar o menor valor de entropia;
- passo 11: calcular as diferenças entre as entropias de janelas vizinhas, armazenando o maior valor;
- passo 12: para cada música, formar um vetor de parâmetros com cinco características, descrito no capítulo 4 como vetor de parâmetros do segundo tipo;
- passo 13: treinar SVMs (tipo 2, descrito no capítulo 4) com os vetores de parâmetros das músicas reservadas para treinamento;
- passo 14: testar a classificação de cada um dos vetores de parâmetros das músicas reservadas para teste.

3.4.5 Algoritmo referente a Tabela 4.5.

- passo 1: separar a base de dados entre “músicas para treinamento” e “músicas para teste”;
- passo 2: extrair os dados brutos dos arquivos (músicas);
- passo 3: normalizar as amostras dos arquivos;
- passo 4: separar as amostras em janelas de 1024 amostras cada uma;
- passo 5: calcular a dimensão de fractal de cada janela;
- passo 6: aplicar a DWT em cada janela;
- passo 7: calcular a entropia baseada em *wavelet* de cada janela;
- passo 8: calcular as médias das entropias;
- passo 9: calcular o desvio-padrão das entropias;
- passo 10: encontrar o maior valor de entropia;
- passo 11: encontrar o menor valor de entropia;
- passo 12: calcular as diferenças entre entropias de janelas vizinhas, armazenando o maior valor;
- passo 13: para cada música, formar um vetor de parâmetros com seis características, descrito no capítulo 4 como vetor de parâmetros do segundo tipo com alteração;
- passo 14: treinar as SVMs (tipo 2, descrito no capítulo 4) com os vetores de parâmetros das músicas reservadas para treinamento;
- passo 15: testar a classificação de cada um dos vetores de parâmetros das músicas reservadas para teste.

3.4.6 Algoritmo referente as Tabelas 4.6 e 4.7.

- passo 1: separar a base de dados entre “músicas para treinamento” e “músicas para teste”;
- passo 2: extrair os dados brutos dos arquivos (músicas);
- passo 3: normalizar as amostras dos arquivos;
- passo 4: separar as amostras em janelas de 1024 amostras cada uma;
- passo 5: calcular a lacunaridade de cada janela;
- passo 6: calcular a média das lacunaridades;
- passo 7: calcular o desvio-padrão das lacunaridades;
- passo 8: encontrar o maior valor de lacunaridade;
- passo 9: encontrar o menor valor de lacunaridade;
- passo 10: calcular as diferenças entre lacunaridades de janelas vizinhas, armazenando o maior valor;
- passo 12: aplicar a DWT em cada janela;
- passo 13: calcular a entropia baseada em *wavelet* de cada janela;
- passo 14: calcular a média das entropias;
- passo 15: calcular o desvio-padrão das entropias;
- passo 16: encontrar o maior valor de entropia;
- passo 17: encontrar o menor valor de entropia;
- passo 18: calcular as diferenças entre entropias de janelas vizinhas, armazenando o maior valor;
- passo 20: para cada música, formar um vetor de parâmetros com dez características, descrito no capítulo 4 como vetor de parâmetros do terceiro tipo;
- passo 21: treinar os GMMs com os vetores das músicas reservadas para treinamento;
- passo 22: testar a classificação de cada um dos vetores de parâmetros das músicas reservadas para teste.

Capítulo 4

Resultados

Este capítulo descreve os resultados obtidos com as diferentes arquiteturas propostas, comprovando o sucesso do sistema.

A primeira tabela (Tabela 4.1) mostra a classificação obtida utilizando entropias extraídas no domínio do tempo e classificador baseado em SVM sendo ensinado sobre todas as classes. Pouco treinamento possibilitou bons resultados, mas de uma maneira geral, utilizando também entropia no tempo, a Tabela 4.2 mostra que o classificador baseado em SVMs treinadas apenas com suas classes apresentam maior acurácia. Com 30% da base sendo utilizada para treinamento, 13 dos 21 *blues* foram corretamente classificados, assim como 15 das 21 clássicas e 20 das 21 *lounge*.

Este fato pôde ser confirmado também com as entropias extraídas no domínio do tempo-frequência. A Tabela 4.3 mostra resultados piores em comparação com a Tabela 4.4, cujos resultados foram obtidos justamente com o segundo tipo de arquitetura de SVM mencionado no capítulo anterior. Neste cenário, com apenas 10% da base servindo como treinamento, 23 dos 27 *blues* foram corretamente classificados, assim como 24 das 27 músicas clássicas e também 24 das 27 *lounge*.

O melhor cenário utilizando entropias e SVMs foi aquele em que os parâmetros foram extraídos no tempo-frequência (entropia baseada em *wavelet*) e uma SVM treinada para cada estilo em questão (Tabela 4.4). Numa tentativa de melhorar ainda mais a acurácia, foi adicionada ao vetor de parâmetros a dimensão de fractal como um novo elemento, mas a Tabela 4.5 mostra que tal medida não apresentou melhorias no sistema.

A última arquitetura testada, que utiliza classificador baseado em GMMs, entropia baseada em *wavelet* e lacunaridade, apresentou também alto grau de classificação. Este esquema foi testado nas duas bases. A tabela 4.6 mostra os resultados referentes a coleção com 4 estilos brasileiros, e a tabela 4.7 a classificação obtida utilizando a base com 3 estilos.

Tabela 4.1: Classificação obtida utilizando entropia extraída no tempo e arquitetura de SVM do tipo 1, como descrito no texto. Na primeira e na segunda coluna encontram-se, respectivamente, a porcentagem da base e o número de músicas de cada estilo utilizadas para fins de treino e de teste. Os valores em negrito correspondem as situações em que ocorreram os melhores resultados.

Treino	Teste	Blues	Clássica	Lounge
10% (3)	90% (27)	77, 8%	92.6%	51, 8%
20% (6)	80% (24)	**91.8%**	**83.3%**	33.3%
30% (9)	70% (21)	**80,9%**	80, 9%	47.6%
40% (12)	60% (18)	72.2%	72.2%	50%
50% (15)	50% (15)	**80%**	73.3%	**86.8%**
60% (18)	40% (12)	58.3%	66.8%	50%
70% (21)	30% (9)	**88,9%**	**88,9%**	66.8%
80% (24)	20% (6)	50%	33.3%	16.8%
90% (27)	10% (3)	66.8%	66.8%	0%

Tabela 4.2: Classificação obtida utilizando entropia no tempo e classificador com segundo tipo de arquitetura de SVM.

Treino	Teste	Blues	Clássica	Lounge
10% (3)	90% (27)	37%	**92.6%**	**85.2%**
20% (6)	80% (24)	37.5%	79.2%	66.7%
30% (9)	70% (21)	61, 9%	71.4%	**95.2%**
40% (12)	60% (18)	61.1%	**83.3%**	**94.5%**
50% (15)	50% (15)	46.8%	**93.3%**	60%
60% (18)	40% (12)	33.3%	**83.3%**	41.8%
70% (21)	30% (9)	33.3%	**100%**	44.5%
80% (24)	20% (6)	50%	**100%**	50%
90% (27)	10% (3)	33.3%	66.8%	**100%**

Tabela 4.3: Classificação obtida utilizando valores de entropia extraídos no tempo-frequência e classificação baseada no primeiro tipo de arquitetura de SVM.

Treino	Teste	Blues	Clássica	Lounge
10% (3)	90% (27)	51, 8%	**85.2%**	**92.6%**
20% (6)	80% (24)	66.8%	66.8%	79.2%
30% (9)	70% (21)	**85.8%**	47.6%	61, 9%
40% (12)	60% (18)	66.8%	**83.3%**	61.1%
50% (15)	50% (15)	66.8%	73.3%	53.3%
60% (18)	40% (12)	58.3%	66.8%	66.8%
70% (21)	30% (9)	**100%**	66.8%	77.8%
80% (24)	20% (6)	**100%**	66.8%	50%
90% (27)	10% (3)	**100%**	66.8%	**100%**

Tabela 4.4: Classificação obtida utilizando valores de entropia extraídos no tempo-frequência e classificador com segundo tipo de arquitetura de SVM. Esta tabela apresenta o melhor resultado para a coleção de 3 gêneros. Notar, na primeira linha, que pouco treinamento possibilitou uma ótima classificação.

Treino	Teste	Blues	Clássica	Lounge	Acurácia geral
10% (3)	90% (27)	**85.1%**	**88,8%**	**88,8%**	87.57%
20% (6)	80% (24)	79.2%	66.8%	79.2%	75.07%
30% (9)	70% (21)	**80,9%**	**85.8%**	**85.8%**	84.17%
40% (12)	60% (18)	72.2%	**88,8%**	**83.3%**	81.44%
50% (15)	50% (15)	**80%**	**86.6%**	**80%**	82.2%
60% (18)	40% (12)	**83.3%**	**91.8%**	**83.3%**	86.14%
70% (21)	30% (9)	77.8%	**88,8%**	77.8%	81.47%
80% (24)	20% (6)	**100%**	66.8%	50%	72.27%
90% (27)	10% (3)	**100%**	66.8%	**100%**	88.94%

Tabela 4.5: Novo parâmetro adicionado, a dimensão de fractal calculada no domínio do tempo. Entropias extraídas no tempo-frequência e classificador baseado na segunda arquitetura de SVM.

Treino	Teste	Blues	Clássica	Lounge
10% (3)	90% (27)	66.8%	66.8%	**96.3%**
20% (6)	80% (24)	75%	66.8%	**83.3%**
30% (9)	70% (21)	76.2%	76.2%	**90.5%**
40% (12)	60% (18)	72.2%	61.1%	**83.3%**
50% (15)	50% (15)	**86.6%**	**86.6%**	**100%**
60% (18)	40% (12)	66.8%	**91.8%**	**83.3%**
70% (21)	30% (9)	66.8%	**100%**	77.8%
80% (24)	20% (6)	**83.3%**	**100%**	**83.3%**
90% (27)	10% (3)	**100%**	**100%**	**100%**

Tabela 4.6: Classificação obtida utilizando o terceiro tipo de vetor de parâmetros, com entropia e lacunaridade. Nestes testes o classificador utilizado foi o GMM. Teste com a coleção de 4 estilos.

Treino	Teste	Axé	Bossa	Forró	Samba	Acurácia geral
6	14	71.43%	100%	71.43%	35.72%	69.65%
10	10	90%	90%	90%	30%	75%
12	8	87.5%	75%	100%	50%	78.13%

Tabela 4.7: Último teste. Entropia baseada em wavelet em conjunto com lacunaridade como parâmetros para classificação baseada em GMM. Teste com a coleção de 3 estilos.

Treino	Teste	Blues	Clássica	Lounge	Acurácia geral
4	21	71.43%	33.3%	90.48%	65.07%
8	17	64.7%	88.24%	88.24%	80.39%
10	15	80%	80%	93.3%	84.44%
15	10	80%	80%	100%	86.67%
20	5	80%	100%	100%	93.33%

Finalmente, a Tabela 4.8 mostra um comparativo entre os dois conjuntos [1] testados, com aproximadamente 30% da base de dados sendo utilizada para fins de treinamento. Já na Tabela 4.9 encontra-se o melhor resultado geral obtido com cada conjunto. As Tabelas 4.10, 4.11 e 4.12 representam as matrizes de confusão dos resultados obtidos nestes casos.

É importante ressaltar que os resultados obtidos são uma média de 5 rodadas de testes, alternando quais músicas das coleções foram utilizadas para fins de treinamento e de testes, procedimento denominado *cross-validation*.

Tabela 4.8: Comparativo entre os conjuntos de testes propostos. Notar que para um treinamento utilizando aproximadamente 30% da base o conjunto 1 apresenta melhores resultados que o conjunto 1*. No entanto, aumentando para 40% o treinamento em 1* o resultado é superado.

Conjunto	Número de estilos	% da base nos testes	Acurácia geral
1	3	30%	84.17%
2	4	30%	69.65%
1*	3	32%	80.39%
1*	3	40%	84.44%

Tabela 4.9: Comparativo entre os conjuntos propostos no trabalho. Melhor resultado geral obtido em cada caso. Para o primeiro conjunto (SVMs), 10% da base foi utilizada para treino. No caso do segundo conjunto (GMMs), o treinamento foi realizado com 60% da base de dados. No conjunto 1*, treinamento com 80% da base. Notar que treinando o conjunto 1 com 80% da base o resultado piora em relação ao treinamento deste mesmo conjunto com 10% da base.

Conjunto	Número de estilos	% da base nos testes	Acurácia geral
1	3	10%	87.56%
2	4	60%	78.13%
1	3	80%	72.27%
1*	3	80%	93.33%

[1] O conjunto 1 representa a base de dados com 3 estilos (*blues*, clássica, *lounge*), entropia, classificador SVM. O conjunto 2 representa a base com 4 estilos brasileiros, entropia, lacunaridade, e GMMs. O conjunto 1* representa a base de dados do conjunto 1, mas utilizando os parâmetros e tipo de classificador do conjunto 2.

Tabela 4.10: Matriz de confusão referente ao melhor cenário obtido com o conjunto 1. Notar o baixo valor dos elementos fora da diagonal principal.

Confusão	Blues	Clássica	Lounge
Blues	17	1	3
Clássica	0	18	3
Lounge	3	0	18

Tabela 4.11: Matriz de confusão referente ao melhor cenário obtido com o conjunto 1*. Notar que neste caso apenas um elemento fora da diagonal principal não é nulo.

Confusão	Blues	Clássica	Lounge
Blues	4	0	1
Clássica	0	5	0
Lounge	0	0	5

Tabela 4.12: Matriz de confusão referente ao melhor cenário obtido com o conjunto 2. Neste cenário, nenhum elemento da coluna referente ao gênero “samba” é nulo.

Confusão	Axé	Bossa	Forró	Samba
Axé	7	0	0	1
Bossa	0	6	0	2
Forró	1	0	6	1
Samba	0	2	2	4

A matriz de confusão ideal possui os elementos da diagonal principal com valores os maiores possíveis [2] e todos os outros elementos nulos. De uma maneira geral, os resultados obtidos representam uma boa classificação, uma vez que os elementos fora da diagonal principal são nulos ou de baixo valor.

[2]Neste caso, o valor máximo a ser representado é o número de músicas de cada estilo testadas.

Capítulo 5

Discussão e conclusões

O trabalho apresentou configurações a serem utilizadas para classificação de sinais, mais precisamente, classificação de gêneros musicais. É importante ressaltar que a eficácia destes sistemas é observada analisando o conjunto de ferramentas, então os parâmetros utilizados nunca são analisados separadamente da arquitetura de classificação. Em outras palavras, fixando um esquema de classificação, podemos dizer qual tipo de parâmetro é mais interessante em conjunto com o mesmo.

A divisão do áudio em janelas permite uma análise pormenorizada dos arquivos, ao invés de examinar o sinal como um todo. Janelas menores, com 1024 amostras de áudio (ou de coeficientes *wavelet*), mostraram ser mais eficientes para fins de classificação musical do que as janelas com 2048 amostras. Pelo princípio da incerteza, janelas menores propiciam melhor resolução no tempo, possibilitando uma melhor análise dos transientes. Em contrapartida, janelas maiores apresentam melhor resolução nas frequências presentes, em detrimento da noção temporal. Pode ser que, para a tarefa de classificação de gêneros, o primeiro caso melhor diferencie as classes, uma vez que janelas de 2048 amostras não apresentaram bons resultados.

SVMs e GMMs foram testadas. Com apenas 3 estilos possíveis, as SVMs mostraram-se eficientes na classificação. Na literatura, é confirmado o fato de que SVMs são extremamente eficientes para separar uma classe de outra, são ótimos classificadores binários. No entanto, com 4 estilos em prova, mais similares (ao menos intuitivamente) entre si, a classificação apresentada não retornou bons resultados. O treinamento das SVMs é realizado ensinando o que representa uma classe, sendo que todas as outras classes ficam como a outra possibilidade. Conforme aumenta o numero de gêneros possíveis, os dados embaralham-se e a margem dos vetores suporte vai diminuindo, gerando erros na classificação. Cabe a ressalva de que o segundo tipo de arquitetura testada para a SVM não funciona baseada em vetores de suporte, mas sim como uma função que escolhe a máxima função de transferência entre aquela de cada estilo.

Esta abordagem torna-se interessante pois no caso de ensinar o classificador sobre o que não é o seu estilo, novos exemplos que não são estilos previamente mencionados geram con-

fusão na classificação, uma vez que dados semelhantes aos seus não foram mapeados. Desta maneira, considerar a maximização da função de transferência, ou função objetivo, vai de encontro com escolher qual gênero mais se assemelha com cada novo exemplo testado.

Os GMMs entraram então como uma tentativa de melhorar a classificação, e apresentaram ótimos resultados, possibilitando uma boa classificação para a coleção de 4 estilos e melhorando ainda mais a acurácia na classificação das músicas da base com 3 estilos. GMMs geralmente apresentam bons resultados pois utilizar um conjunto de gaussianas, cada uma com suas características (média, desvio-padrão, matriz de covariância), possibilita uma aproximação mais suave e precisa da distribuição do fenômeno que se deseja analisar. Além disso, os fenômenos naturais geralmente obedecem a um padrão de gaussianas, aumentando a chance de que uma modelagem por curvas de Gauss apresente alto grau de semelhanca com o fenômeno sob análise. Para sinais aleatórios, de fato, o teorema do limite central mostra que a distribuição obedece a padrões de gaussianas.

Nos testes com a coleção de 3 estilos de músicas o sinal inteiro era utilizado para a extração dos parâmetros, mas a literatura sugere que utilizar apenas alguns trechos específicos da música a ser classificada não só não atrapalha a classificação, como também a melhora. A justificativa pode vir do fato de que há um intervalo de tempo para que as músicas alcancem sua mensagem principal, sua parte mais importante. Em outras palavras, a introdução pode apresentar longos trechos de preparação para a música que está se construindo, e quando isso ocorre, o gênero daquele trecho pode desviar do gênero principal. O mesmo vale para o final das músicas, que pode ser interpretado como um trecho de fechamento da “história que foi contada”. Desse ponto de vista (ou de escuta) é inclusive melhor evitar basear a classificação nesses trechos iniciais e finais.

Entretanto, muitos afirmam que as músicas, quando atingem sua porção central, mudam de estilo. Esse trecho em que ocorre a mudança é conhecido como “outro” na estrutura das músicas. Este fato realmente acontece, mas da forma como os testes foram realizados, a saber, utilizando os dois quartos centrais, a presença do “outro”, quando ocorre, dilui-se em meio a dois quartos da música.

Relembrando, o cálculo da dimensão do fractal pode ser aplicado a estruturas que na verdade não são fractais. No entanto, o fato de essa medida não ter ajudado na classificação pode estar sugerindo que a FD dos sinais musicais, a princípio, fala pouco sobre o gênero em questão, ou ainda, que independentemente do quanto fala sobre os gêneros, fala aproximadamente o mesmo sobre todos. Uma futura rodada de testes utilizando entropia, lacunaridade e dimensão de fractal pode julgar se a FD não é um bom parâmetro a ser utilizado na classificação de áudio, uma vez que as duas medidas de geometria fractal apresentadas neste trabalho, juntamente, descrevem muito bem sinais de imagem. É importante ressaltar que existem vários métodos para o cálculo da FD, cada um apropriado para uma faixa de valores, e o *box-counting*,

utilizado neste trabalho, é adequado.

Já a lacunaridade mostrou ser um bom parâmetro para auxiliar na classificação. Sua inclusão no vetor de parâmetros elevou o nível de acerto (entre os 4 estilos brasileiros) aproximadamente de 60% para 78%. A lacunaridade, juntamente com a dimensão de fractal, descreve a textura de um sinal analisado. É possível que os diferentes estilos musicais apresentem distribuições de informação bem definidas para cada estilo, pois é justamente esse o ponto da análise de lacunaridade.

A entropia baseada em *wavelet*, logo nos primeiros testes, apenas com 3 estilos, mostrou ótimos resultados de classificação. Isso leva a crer que diferentes gêneros musicais possuem diferentes padrões de informação, do ponto de vista do seu agrupamento e dinâmica, uma vez que assim foi definido o cálculo das entropias nesse trabalho, diferentemente da forma clássica. A WBE já é amplamente utilizada em análise de sinais de eletroencefalograma e sinais de voz para reconhecimento de locutor e de detecção de patologias na laringe. Os resultados obtidos neste trabalho sugerem que mais trabalhos e estudos na área de sinais musicais devem ser realizados com essa ferramenta, e que variações na forma de obter a entropia também devem ser testadas, pois a quantidade de informação que um evento transmite é uma rica fonte de informação sobre o mesmo.

A família Haar, utilizada neste trabalho, é a mais simples das famílias de ondaletas. As *wavelets* dessa família constituem filtros de apenas 2 coeficientes. Sendo assim, o resultado de filtragens a partir destas funções é pobre e apresenta ruído e interferências das janelas adjacentes. No entanto, é justamente esta interferência que torna essa família interessante para esta aplicação, uma vez que a principal informação está na dinâmica temporal do sinal, e não na perfeita resposta em frequência.

É importante ressaltar que a entropia utilizada como parâmetro neste trabalho nao é aquela proposta por Shannon, que é baseada na probabilidade de ocorrência dos símbolos possíveis, sendo que símbolos mais improváveis geram mais informação. A entropia no sistema aqui proposto para classificação não considera as probabilidades, mas sim a energia normalizada de trechos do sinal em questão, analisando assim sua distribuição em termos do agrupamento da sua dinâmica.

Finalizando, uma análise geral dos números obtidos mostra que utilizando a SVM do primeiro tipo (Tabela 4.1), 20% de treinamento já proporcionou resultado excelente para *blues* e clássica, e que com 50% de treinamento, o que ainda é aceitável, proporciona boa classificação também para *lounge* em conjunto com os demais gêneros. Já com o segundo tipo de arquitetura de SVM (Tabela 4.2), um treino com cerca de 30% da base também mostrou resultados bastante promissores.

As matrizes de confusão refentes a coleção com 3 estilos (Tabelas 4.10 e 4.11) mostram que para estes estilos as poucas classificações incorretas aconteceram entre *blues* e *lounge*.

Isso pode decorrer do fato que a música clássica explora mais a dinâmica em relação aos outros dois gêneros, e a análise da dinâmica é justamente o que a WBE realiza. Sendo assim, *blues* e *lounge* estariam se confundindo por suas dinâmicas situarem-se na mesma faixa. Em relação a coleção de músicas brasileiras, a Tabela 4.12 mostra que todos os gêneros foram confundidos com o samba em alguns casos, e que os sambas testados também foram confundidos com outros estilos. A matriz parece mostrar que, da perspectiva da análise da dinâmica nas diferentes faixas do espectro estes gêneros se influenciam e se referenciam.

De forma geral, parâmetros extraídos no domínio tempo-frequência (Tabela 4.3) mostram que o aumento no treinamento não proporciona ganho considerável em acurácia com a primeira arquitetura de SVM. Já com a outra arquiterura (Tabela 4.4), percebe-se um padrão bastante satisfatório em termos de acurácia para os três gêneros envolvidos. Neste último caso, a inclusão da dimensão de fractal como parâmetro (Tabela 4.5) não proporciona ganho significativo, pelo contrário, mostra inadequação em certos casos.

Por fim, na arquitetura com o classificador GMM (Tabelas 4.6 e 4.7), pode-se observar que o principal fator de destaque é o fato de que além de proporcionarem uma considerável classificação, não ha indícios de *overfitting*, uma vez que o aumento da base de treinamento não causa perda de acurácia.

Devido a dificuldade de obtenção de bases extensas, as coleções utilizadas neste trabalho são modestas, e fazem-se necessárias bases maiores para realizar mais testes de classificação e afirmar com maior margem de segurança a eficiência do sistema proposto. Fica como idéia de trabalho futuro aplicar o sistema proposto nas bases de dados divulgadas para disputas de classificação, como a base do ISMIR (*International Symposium on Music Information Retrieval*), para comparar diretamente o sistema com os melhores trabalhos na área. Tal resultado complementará os subsídios para responder se entropia baseada em *wavelet* e geometria fractal (conceitos de auto-similaridade e distribuição dos padrões de informação) garantem uma boa descrição de sinais musicais, ou se a exploração de informação musical é também necessária.

Referências

[1] 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.

[2] ALAKAIDI, M. *Fractal Speech Processing*. Cambridge University Press, 2004.

[3] BLANCO, S.; DATTELLIS, C.; ISAACSON, S.; ROSSO, O.A.; SIRNE, R. *Timefrequency analysis of electroencephalogram series (II): gabor and wavelet transform*. Physical Review E (54), pp. 6661-72, 1996.

[4] BLANCO, S.; FIGLIOLA, A.; QUIAN QUIROGA, R.; ROSSO, O.A.; SERRANO, E. *Time-frequency analysis of electroencephalogram series (III): wavelet packets and information cost function*. Physical Review E (57), pp. 932-40, 1998.

[5] BLANCO, S.; QUIAN QUIROGA, R; ROSSO, O.A.; KOCHEN, S. *Time-frequency analysis of electroencephalogram series*. Physical Review E (51), pp. 2624-31, 1995.

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

[7] COVER, T.; THOMAS, J. *Elements of Information Theory, 2nd edition*. John Wiley & Sons, 2006.

[8] DANNENBERG, R. B.; THOM, B.; WATSON, D. *A Machine Learning Approach to Musical Style Recognition*. Proceedings of the International Computer Music Conference, pp. 344-7, 1997.

[9] DEMPSTER, A.; LAIRD, N.; RUBIN, D. *Maximum Likelihood from Incomplete Data via the EM Algorithm*. Journal of the Royal Statistical Society 39(1), pp. 138, 1977.

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

[11] DUDA, O.; HART, P.E.; STORK, D.G. *Pattern Classification, second edition*. John Wiley & Sons, 2001.

[12] EZZAIDI, H.; ROUAT, J. *Automatic Musical Genre Classification Using Divergence and Average Information Measures* M. World Academy of Science, Engineering and Technology, 15, 2006.

[13] 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.

[14] GOULART, A.J.H.; GUIDO, R.C.; MACIEL, C.D. *Exploring different approaches for Music Genre Classification.* Expert systems with applications, esperando revisão.

[15] GOULART, A.J.H.; GUIDO, R.C.; MACIEL, C.D.; PAULO, K.C.S.; SILVA, I.N. *Music Genre Classification based on entropy and fractal lacunarity.* Proceedings of the IEEE International Symposium on Multimedia, 2011.

[16] GRAY, R. *Vector Quantization*. IEEE ASSP Magazine 429, 1984.

[17] HUBBARD, B.B. *The World According to Wavelets: The Story of a Mathematical Technique in the Making, Second Edition.* A K Peters, 1998.

[18] JAYASREE, T.; DEVARAJ, D.; SUKANESH, R. *Classification of Transients using Wavelet Based Entropy and Radial Basis Neural Networks*. International Journal of Computer and Electrical Engineering, Vol. 1, No. 5, pp. 1793-8163, 2009.

[19] LEE, J. H.; DOWNIE, J. S. *Survey of Music Information Needs, Uses, and Seeking Behaviours: Preliminary Findings*. Proceedings of the International Conference on Music Information Retrieval, 2004.

[20] LI, T.; OGIHARA, M.; LI, Q. *A Comparative study on content-based Music Genre Classification*. Proceedings of the 26th Annual International ACM SI-GIR Conference on Research and Development in Information Retrieval, Toronto, ACM Press, pages 282-289, 2003.

[21] LOVELL, B. C.; WALDER, C. J. *Business Applications and Computational Intelligence*. Idea Group, 2006.

[22] MCKAY, C.; FUJINAGA, I; *Musical genre classification: Is it worth pursuing and how can it be improved?* Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR-06), 2006.

[23] MELO, R.H.C. *Using fractal characteristics such as fractal dimension, lacunarity and succolarity to characterize texture patterns on images*. Tese de mestrado. Universidade Federal Fluminense, 2007.

[24] MILNER, G. *Perfecting Sound Forever: An Aural History of Recorded Music.* Faber and Faber, 2009.

[25] NORTH, A. C.; HARGREAVES, D. J. *Liking for Musical Styles*. Music Scientae, vol. 1, no. 1, pp. 109-28, 1997.

[26] PANAGAKIS, Y.; KOTROPOULOS, C.; ARCE, G. R. *Music genre classification via sparse representations of auditory temporal modulations*. 17th European Signal Processing Conference (EUSIPCO), 2009.

[27] PAPOULIS, A.; Pillai, S.U. *Probability, Random Variables and Stochastic Processes, fourth edition.* Mc-Graw Hill, 2002.

[28] PARADZINETS, A; HARB, H.; CHEN, L. *Multiexpert system for automatic music genre classification*. liris.cnrs.fr/Documents/Liris-4224.pdf, Research report, 2009.

[29] PIERCE, J. *An Introduction to Information Theory: Symbols, Signals and Noise, second editon.* Courier Dover Publications, 1980.

[30] POWELL, C.E.; PERCIVAL, I.C. *A spectral entropy method for distinguishing regular and irregular motion of Hamiltonian systems*. Journal of Physics A: Mathematical and Theoretical (12), pp. 2053-71, 1979.

[31] REYNOLDS, D. *Gaussian Mixture Models*. Tutorial.

[32] REYNOLDS, D. *A Gaussian Mixture Modeling Approach to Text-Independent Speaker Identification*. PhD thesis, Georgia Institute of Technology 1992.

[33] ROSSO, O.A.; BLANCO, S. *Characterization of dynamical evolution of electroencephalogram time series*. 1999, unpublished.

[34] ROSSO, O.;BLANCO, S.; YORDANOVA, J.; KOLEV, V.; FIGLIOLA, A.; SCHURMANN, M.; BASAR, E. *Wavelet entropy: a new tool for analysis of short duration brain electrical signals*. Journal of Neuroscience Methods 105, pp. 6575, 2001.

[35] SCHOLKOPF, B.; SMOLA, A. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. Cambridge, MA: MIT Press, 2002.

[36] SILLA JR., C.N.; KOERICH, A.L.; KAESTNER, C.A.A.*A machine learning approach to automatic music genre classification*. Journal of the Brazilian Computer Society, vol.14, no.3, Campinas, September 2008.

[37] SOUZA, L.M. *Detecção inteligente de patologias na laringe baseada em máquinas de vetores de suporte e na transformada wavelet.* Tese de mestrado. Universidade de São Paulo, 2010.

[38] TEKMAN, H. G.; HORTACSU, N. *Aspects of Stylistic Knowledge: What are Different Styles Like and Why Do We Listen to Them?* Psychology of Music, vol. 30, no. 1, pp. 28-47, 2002.

[39] TZANETAKIS, G.; COOK, P. *Musical Genre Classification of Audio Signals*. IEEE Transactions on Speech and Audio Processing, vol. 10, no. 5, pp. 293-302, 2002.

[40] VAPNIK, V. N. *Statistical learning theory*. New York: Wiley, 1998.

[41] WORNELL, G. *Signal processing with fractals: a wavelet-based approach*. Prentice Hall, 1996.

Apêndice I - Melodia, harmonia, ritmo e timbre

Alguns termos utilizados em teoria musical, como melodia, harmonia, ritmo, e timbre foram mencionados ao longo do texto. Este apêndice contém uma breve explicação sobre cada um desses termos.

Melodia
É uma sucessão de notas musicais e de silêncios. É aquele elemento musical que cantamos sozinhos ou assoviamos. A melodia possui um sentido na obra, e se apóia na harmonia e no ritmo.

Harmonia
Acontece quando duas ou mais notas musicais soam simultaneamente. A harmonia se organiza em acordes, que são combinações de notas, e obedece a uma série de estruturas e relações específicas. No entanto, tais regras podem ser quebradas.

Ritmo
Refere-se a colocação das notas no tempo, e como essas notas se relacionam com a pulsação da música. A melodia e a harmonia organizam-se em função do ritmo. A pulsação é a forma como contamos o andamento da música, por exemplo, batendo o pé no chão ou mexendo a cabeça.

Timbre
É a característica que permite a distinção entre diferentes fontes sonoras. Se ouvirmos uma guitarra tocando a mesma nota que um trumpete, o que nos possibilita diferencia-las é o timbre instrumental. Os dois instrumentos irão produzir as mesmas notas na frequência, mas com características sonoras distintas.

Apêndice II - Músicas utilizadas nos testes

Blues

Crossroads Soundtrack, by Ry Cooder

- Crossroads
- Down In Mississippi
- Cotton Needs Pickin’
- Viola Lee Blues
- See You In Hell Blind Boy
- Nitty Gritty Mississippi
- He Made A Woman Out Of Me
- Feelin’ Bad Blues
- Willie Brown Blues
- Walkin’ Away Blues

Atlantic Blues: Guitar

- Blind Willie Mctell - Broke Down Engine
- Mississipi Fred McDowell - Shake ’Em on Down
- John Lee Hooker - My Baby Don’t Love Me
- Stick McGhee - Tall Pretty Woman
- Texas Johnny Brown - Blues Rock
- Texas Johnny Brown - There Goes The Blues
- Texas Johnny Brown - Bongo boogie
- Texas Johnny Brown - Two Bones and a Pick
- Chuck Norris - Let Me Know
- Guitar Slim - Down Through the Years
- Cornell Dupree - Okie Dokie Stomp
- Big Joe Turner - T. V. Mama
- Al King - Reconsider Baby
- Mickey Baker - Midnight Midnight
- Big Joe Turner - I Smell Trouble
- Al King - Why I Sing the Blues
- Al King - Crosscut Saw
- Al King - Born Under a Bad Sign
- Jr. John Hammond - Shake For Me
- Steve Ray Vaughan - Flood Down In Texas

Música clássica

Tchaikovsky

Regente: Yuri Simonov

O Quebra-Nozes, Suíte Op. 71A

- Abertura
- Marcha
- O Chocolate
- O Café
- O chá
- Trepak
- Dana dos Mirlitões
- A Mãe Cegonha e os Polichinelos
- Dança da Fada do Açúcar
- Valsa das Flores
- Pas-de-Deux
- Valsa Final e Apoteose

Beethoven

Regente: Mark Emler

- Abertura ”Egmont”op. 84

Sinfonia n. 6 em fá maior op. 68 “Pastoral”

- Erwachen heiterer Empfindungen bei der Ankunft auf dem Lande
- Szene am Bach (Andante molto mosso)
- Lustiges Zusammensein der Landleute (Allegro)
- Gewitter, Sturm (Allegro)
- Hirtengesang: Frohe und dankbare Gefhle nach dem Sturm (Allegretto)

Bach

Violino: Jonathan Carney

- Tocata e Fuga em Ré Menor, BWV 565
- Cantata n 51,“Jauchzet Gott In Allen Landen”, BWV 51
- Cantata n 140, “Wachet Auf”, BWV 140
- Concerto de Brandenburgo n 4, BWV 1049 (I. Allegro)
- Ária na Corda Sol da Suíte n 3, BWV 1068
- Concerto para Cravo n 1 em Ré Menor, BWV 1052 (I. Allegro)
- Cantata n 208, “Sheep May Safely Graze”, BWV 208
- Sute n 2 em Si Menor para Flauta, BWV 1067 - Polonaise
- Sute n 2 em Si Menor para Flauta, BWV 1067 - Minueto
- Sute n 2 em Si Menor para Flauta, BWV 1067 - Badinerie
- Cantata n 147, ”Jesus Alegria dos Homens”, BWV 147
- Concerto de Brandenburgo n 2, BWV 1047(III. Allegro Assai)

Lounge

Compilação de vários álbuns do Buddha Lounge

- Roedelius - Poetry
- Gary Stadler with Stephanie - Dona Cre Tun
- Althea W. - Abo Daylight
- Opera To Relax - From Life 2 Life
- Drum N Space - The Audience
- David and Steve Gordon - Reverence
- Svensson - In the Move
- Marcator - What is Right
- Delago - Second Day
- Zingaia - Divine Flame
- Alquimia - Night of the Alebrijes
- Roedelius - Glass from Jasper
- Peter Mergener - Rain in Australia
- Achillea - Cape Porcupine
- Tya - Why
- Ginkgo Garden - One and Twain
- Nasser Kilada - Samah
- Tau - Touche’ (Sutra Edit)
- Hands Upon Black Earth - Bhajya Sahita
- David and Steve Gordon - Empowered (Club Remix)
- Stella Maris - Northern Lights

- David Gordon - Mangoville
- The Moontrane Conductors - In For The Night (Buddha Edit)
- David and Steve Gordon - Descent to the Lowerworld
- One At Last - Hamana Nale (Lotus Mix)
- Althea W. - Sky Walk
- Alcyone - Velvet Sutra
- Jaya Lakshmi - Radha Pranam
- Marcator - Necromantra
- Opera To Relax - Upon the Temple Bell

Bossa-Nova

Fogueira Três - Bossa Nova

- Garota de Ipanema
- Amazonas
- Corcovado
- Nostalgia da bossa
- The gentle rain
- Água de beber
- Batida diferente
- Days of wine and roses
- O barquinho
- Triste

Tom Maior

- Desafinado
- Estrada do sol
- O nosso amor
- Este seu olhar
- Eu sei que vou te amar
- A felicidade
- Chega de saudade
- Modinha
- As praias desertas
- Janelas abertas

Samba

Adoniran Barbosa

- Tiro ao Álvaro
- Saudosa maloca
- Prova de carinho
- Iracema
- Trem das onze
- Despejo na favela
- Samba do Arnesto
- As mariposas
- Vila Esperança
- Samba italiano

Cartola

- Disfarça e chora
- Amor proibido
- Silêncio de um cipestre
- Peito vazio
- As rosas não falam
- A canção que chegou
- Autonomia
- Quem me vê sorrindo
- Alvorada
- Sim

Axé

Asa de Águia - Ao vivo

- Dia Dos Namorados
- Oba Vou Passear
- Padang
- Leva Eu
- Cocobambu
- Porto Seguro
- Noronha
- Ficar com Voçê
- Sonho De Amor
- Na Bahia Ai Ai

É o Tchan - Do Brasil

- Ralando O Tchan (A Danca Do Ventre)
- Bambolê
- Mão Boba
- Nega Vá
- Disque Tchan (Alô É Tchan)
- Pot-Pourri (Sambas Da Bahia)
- I Love You
- De Bem Com A Vida
- Simbora, Neném (Olha A Lua, Alah)
- Tarde De Domingo

Forró

Cavalo de Pau - Noda de Cajú

- Deixa
- De Braço dado com a solidão
- Saudade de casa
- Passos na areia
- Meu cavalo de pau
- Ainda te quero
- Brincar de amar
- Dor da ausência
- Seis cordas
- Juras de amor

Mastruz com Leite - Flor do Mamulengo

- Princípio, meio e fim
- Se lembra coração
- Meu bê-a-bá
- Jogo aberto
- Meeiro
- Somos Mastruz com Leite
- Rala coxa
- Flor do mamulengo
- Passeando pelo sertão
- Homem pequeno

Apêndice III - Códigos-fonte

Neste apêndice encontram-se os códigos-fonte utilizados neste trabalho. Os programas referentes a extração de parâmetros foram desenvolvidos pelo autor com colaboração de Rodrigo Capobianco Guido. Os programas referentes aos classificadores e as bibliotecas para as funções de transformada *wavelet* foram desenvolvidos por R.C. Guido.

O trabalho encontra-se dividido em várias etapas. Os programas para extração de parâmetros geram arquivos .txt com os dados que servirão de treinamento para o classificador. Da mesma maneira, primeiro extraem-se os parâmetros de cada música a ser classificada para então usá-los como dados de entrada no programa de classificação.

```
//bibliotecas de funcoes de transformada wavelet

//////////////////////////////////////////////////////////////
//   PERMISSAO PARA USAR LIVREMENTE DESDE QUE CITADA A FONTE
//   AUTOR: Prof. Dr. Rodrigo C. Guido
//   guido@ifsc.usp.br
//   IFSC/USP    2006
//////////////////////////////////////////////////////////////
//wavelet.h
//bibliotecas de funcoes de transformada wavelet
#include<math.h>
const long qtd_max_col_matriz=4; //quantidade de colunas da matriz bidimensioanl, usada
apenas
//nas funcoes bidimensionais. Modificar isso, conforme a quantidade de colunas da matriz a ser
//usada no arquivo cpp. O valor 4 acima e' apenas um exemplo.

//-------------------------------------------------------------
void transformada_wavelet(double* f, long n, int nivel, char ordem, double h[], int ch)
{
double *g=new double[ch];
for(int i=0;i<ch;i++)
      {
      g[i]=h[ch-i-1];
      if(i%2!=0)
         g[i]*=-1;
      }
int cg=ch;
long j=0;
double* t = new double[n];
if(ordem=='n') // n de normal para wavelet
            {
            for(long i=0;i<n;i+=2)  //trend
                        {
                        t[j]=0;
                        for(long k=0;k<ch;k++)
                                    t[j]+=f[(i+k)%n]*h[k];
                        j++;
                        }
            for(long i=0;i<n;i+=2) //fluctuation
                        {
                        t[j]=0;
                        for(long k=0;k<cg;k++)
                                    t[j]+=f[(i+k)%n]*g[k];
                        j++;
                        }
            }
else // i de invertido para wavelet packet
            {
            for(long i=0;i<n;i+=2) //fluctuation
                        {
                        t[j]=0;
                        for(long k=0;k<cg;k++)
                                    t[j]+=f[(i+k)%n]*g[k];
                        j++;
                        }
            for(long i=0;i<n;i+=2)  //trend
                        {
                        t[j]=0;
                        for(long k=0;k<ch;k++)
```

//continuacao bibliotecas de funcoes de transformada wavelet

```
                                t[j]+=f[(i+k)%n]*h[k];
                        j++;
                        }
            }
for(long i=0;i<n;i++)
            f[i]=t[i];
nivel--;
n/=2;
delete(t);
if(nivel>0)
      transformada_wavelet(&f[0],n,nivel,ordem,h,ch);
}
//-------------------------------------------------------------
void transformada_wavelet_packet(double* f, long n, int nivel, double h[], int ch)
{
long inicio=0;
long comprimento=n;
for(int i=1;i<=nivel;i++) // por exemplo, para nivel 5, vou chamar 5 vezes a fun?o de
tranasformada, cada vez em n?el 1.
            {
            inicio=0;
            comprimento=(int)(n/pow(2,i-1));
            for(int j=0;j<pow(2,i-1);j++)
                        {
                        if(j%2==0)
                                    transformada_wavelet(&f[inicio],comprimento,1,'n',h,ch); // n de
ordem normal: primeiro passa-baixa e depois passa-alta
                        else
                                    transformada_wavelet(&f[inicio],comprimento,1,'i',h,ch); // i de
invertido: primeiro passa-alta e depois passa-baixa
                        inicio+=comprimento;
                        }
            }
}
//-------------------------------------------------------------
void transformada_wavelet_inversa(double* f, long n, int nivel, char ordem, double h[], int ch)
{
double* g=new double[ch];
for(int i=0;i<ch;i++)
      {
      g[i]=h[ch-i-1];
      if(i%2!=0)
         g[i]*=-1;
      }

double* sfi=new double[ch];
for(int i=0;i<ch;i+=2)
            sfi[i]=h[ch-2-i];
for(int i=0;i<ch;i+=2)
            sfi[i+1]=g[ch-2-i];
int csfi=ch;

double *wfi=new double[csfi];
for(int i=0;i<ch;i+=2)
            wfi[i]=h[ch-1-i];
for(int i=0;i<ch;i+=2)
```

```
//continuacao bibliotecas de funcoes de transformada wavelet


		wfi[i+1]=g[ch-1-i];

long comprimento_do_subsinal=2*(long)((n)/(pow(2,nivel)));
double* subsinal=new double[comprimento_do_subsinal];
if(ordem=='n') //normal
		{
		for(long i=0;i<comprimento_do_subsinal;i+=2)
			{
		subsinal[i]=f[(int)(i/2)];
			subsinal[i+1]=f[(int)(i/2)+(int)(comprimento_do_subsinal/2)];
			}
		}
else // i  // invertido  -> alguns casos da packet
		{
		for(long i=0;i<comprimento_do_subsinal;i+=2)
			{
			subsinal[i]=f[(int)(i/2)+(int)(comprimento_do_subsinal/2)];
			subsinal[i+1]=f[(int)(i/2)];
			}
		}

long start;
if(comprimento_do_subsinal>=csfi)
		{
		if(comprimento_do_subsinal-csfi > 0)
		   start=(comprimento_do_subsinal-csfi)+2;
		else
		   start=-(comprimento_do_subsinal-csfi)+2;
		}
else
		{
		long comprimento_matricial_do_sinal=2;
		while(comprimento_matricial_do_sinal<csfi)
			comprimento_matricial_do_sinal+=comprimento_matricial_do_sinal;
		start=comprimento_matricial_do_sinal-csfi+2;
		}

for(long j=0;j<comprimento_do_subsinal;j+=2)
	{
		f[j]=0;
		f[j+1]=0;
		for(int k=0;k<csfi;k++)
			{
			f[j]+=sfi[k]*subsinal[(start+k)%(comprimento_do_subsinal)];
			f[j+1]+=wfi[k]*subsinal[(start+k)%(comprimento_do_subsinal)];
			}
		start+=2;
		}
nivel--;
delete(subsinal);
if(nivel>0)
	transformada_wavelet_inversa(&f[0],n,nivel,ordem,h,ch);
}
//-------------------------------------------------------
void transformada_wavelet_packet_inversa(double* f, long n, int nivel, double h[], int ch)
{
```

```
//continuacao bibliotecas de funcoes de transformada wavelet


long inicio=0;
long comprimento=n;
for(int i=nivel;i>=1;i--) // por exemplo, para nivel 5, vou chamar 5 vezes a fun?o de
tranasformada, cada vez em n?el 1.
	{
	inicio=0;
	comprimento=(int)(n/pow(2,i-1));
	for(int j=0;j<pow(2,i-1);j++)
		{
		if(j%2==0)
			transformada_wavelet_inversa(&f[inicio],comprimento,1,'n',h,ch); // n
de ordem normal: primeiro passa-baixa e depois passa-alta
		else
			transformada_wavelet_inversa(&f[inicio],comprimento,1,'i',h,ch); // i
de invertido: primeiro passa-alta e depois passa-baixa
		inicio+=comprimento;
		}
	}
}
//-------------------------------------------------------------
void transformada_wavelet_bidimensional(double f[][qtd_max_col_matriz], long li, long ci, long n,
long m, int nivel, char ordem_l, char ordem_c, double h[], int ch)
{
double* vetor_linha=new double[m];
for(int i=li;i<li+m;i++)
	{
	for(int j=ci;j<ci+m;j++)
		vetor_linha[j-ci]=f[i][j];
	transformada_wavelet(&vetor_linha[0],m,1,ordem_l,h,ch);
	for(int j=ci;j<ci+m;j++)
		f[i][j]=vetor_linha[j-ci];
	}


double* vetor_coluna=new double[n];
for(int j=ci;j<ci+m;j++)
	{
	for(int i=li;i<li+n;i++)
		vetor_coluna[i-li]=f[i][j];
	transformada_wavelet(&vetor_coluna[0],n,1,ordem_c,h,ch);
	for(int i=li;i<li+n;i++)
		f[i][j]=vetor_coluna[i-li];
	}

nivel--;
n/=2;
m/=2;
delete(vetor_linha);
delete(vetor_coluna);
if(nivel>0)
	transformada_wavelet_bidimensional(f, li, ci, n, m, nivel, ordem_l, ordem_c,h,ch);
}
//-------------------------------------------------------------
void transformada_wavelet_bidimensional_inversa(double f[][qtd_max_col_matriz], long li, long
ci, long n, long m, int nivel, char ordem_l, char ordem_c, double h[], int ch)
{
```

```
//continuacao bibliotecas de funcoes de transformada wavelet



long quantidade_de_linhas_da_submatriz=2*(long)((n)/(pow(2,nivel)));
long quantidade_de_colunas_da_submatriz=2*(long)((m)/(pow(2,nivel)));


double* vetor_linha=new double[quantidade_de_colunas_da_submatriz];
for(int i=li;i<li+quantidade_de_linhas_da_submatriz;i++)
	{
	for(int j=ci;j<ci+quantidade_de_colunas_da_submatriz;j++)
		vetor_linha[j-ci]=f[i][j];
	transformada_wavelet_inversa(&vetor_linha[0],quantidade_de_colunas_da_submatriz,1,
ordem_l,h,ch);
	for(int j=ci;j<ci+quantidade_de_colunas_da_submatriz;j++)
		f[i][j]=vetor_linha[j-ci];
	}


double* vetor_coluna=new double[quantidade_de_linhas_da_submatriz];
for(int j=ci;j<ci+quantidade_de_colunas_da_submatriz;j++)
	{
	for(int i=li;i<li+quantidade_de_linhas_da_submatriz;i++)
		vetor_coluna[i-li]=f[i][j];
	transformada_wavelet_inversa(&vetor_coluna[0],quantidade_de_linhas_da_submatriz,1,
ordem_c,h,ch);
	for(int i=li;i<li+quantidade_de_linhas_da_submatriz;i++)
		f[i][j]=vetor_coluna[i-li];
	}

nivel--;
delete(vetor_linha);
delete(vetor_coluna);
if(nivel>0)
	transformada_wavelet_bidimensional_inversa(f, li, ci, n, m, nivel, ordem_l, ordem_c,h,ch);
}
//-----------------------------------------------------------------
void transformada_wavelet_packet_bidimensional(double f[][qtd_max_col_matriz], long n, long
m, int nivel, double h[], int ch)
{
long inicio_da_linha;
long inicio_da_coluna;
long comprimento_da_linha;
long comprimento_da_coluna;

for(int k=1;k<=nivel;k++) // por exemplo, para nivel 5, vou chamar 5 vezes a fun?o de
transformada, cada vez em n?el 1.
	{
	inicio_da_linha=0;
	inicio_da_coluna=0;

	comprimento_da_linha=(int)(m/pow(2,k-1));
	comprimento_da_coluna=(int)(n/pow(2,k-1));

	for(int i=0;i<pow(2,k-1);i++)
		{
		inicio_da_coluna=0;
		for(int j=0;j<pow(2,k-1);j++)
```

//continuacao bibliotecas de funcoes de transformada wavelet

```
                                {
                                if((i%2==0)&&(j%2==0))
                                        transformada_wavelet_bidimensional(f,inicio_da_linha,
inicio_da_coluna, comprimento_da_linha,comprimento_da_coluna,1,'n','n',h,ch);
                                else if ((i%2==0)&&(j%2!=0))
                                        transformada_wavelet_bidimensional(f,inicio_da_linha,
inicio_da_coluna, comprimento_da_linha,comprimento_da_coluna,1,'i','n',h,ch);
                                else if ((i%2!=0)&&(j%2==0))
                                        transformada_wavelet_bidimensional(f,inicio_da_linha,
inicio_da_coluna, comprimento_da_linha,comprimento_da_coluna,1,'n','i',h,ch);
                                else
                                        transformada_wavelet_bidimensional(f,inicio_da_linha,
inicio_da_coluna, comprimento_da_linha,comprimento_da_coluna,1,'i','i',h,ch);
                                inicio_da_coluna+=comprimento_da_coluna;
                                }
                        inicio_da_linha+=comprimento_da_linha;
                        }
                }
}
//-----------------------------------------------------------
void transformada_wavelet_packet_bidimensional_inversa(double f[][qtd_max_col_matriz], long
n, long m, int nivel, double h[], int ch)
{
long inicio_da_linha;
long inicio_da_coluna;
long comprimento_da_linha;
long comprimento_da_coluna;

for(int k=nivel;k>=1;k--)
        {
        inicio_da_linha=0;
        inicio_da_coluna=0;

        comprimento_da_linha=(int)(m/pow(2,k-1));
        comprimento_da_coluna=(int)(n/pow(2,k-1));

        for(int i=0;i<pow(2,k-1);i++)
                {
                inicio_da_coluna=0;
                for(int j=0;j<pow(2,k-1);j++)
                        {
                        if((i%2==0)&&(j%2==0))
                                transformada_wavelet_bidimensional_inversa(f,inicio_da_linha,
inicio_da_coluna, comprimento_da_linha,comprimento_da_coluna,1,'n','n',h,ch);
                        else if ((i%2==0)&&(j%2!=0))
                                transformada_wavelet_bidimensional_inversa(f,inicio_da_linha,
inicio_da_coluna, comprimento_da_linha,comprimento_da_coluna,1,'i','n',h,ch);
                        else if ((i%2!=0)&&(j%2==0))
                                transformada_wavelet_bidimensional_inversa(f,inicio_da_linha,
inicio_da_coluna, comprimento_da_linha,comprimento_da_coluna,1,'n','i',h,ch);
                        else
                                transformada_wavelet_bidimensional_inversa(f,inicio_da_linha,
inicio_da_coluna, comprimento_da_linha,comprimento_da_coluna,1,'i','i',h,ch);
                        inicio_da_coluna+=comprimento_da_coluna;
                        }
```

//continuacao bibliotecas de funcoes de transformada wavelet

```
                    inicio_da_linha+=comprimento_da_linha;
                    }
          }
}
//-----------------------------------------------------------------------
```

```
//Calculo da Entropia baseada em Wavelet (Wavelet based Entropy)

//Permissao para usar livremente desde que citada a fonte

//Autores:
//Rodrigo Capobianco Guido        (guido@ifsc.usp.br)
//Antonio Jose Homsi Goulart      (antonio.goulart@usp.br)

#include<iostream.h>
#include<stdio.h>
#include<math.h>
#include<string.h>
#include "wavelet.h"
//---------------------------------------
main(int argc,char* n[])
{
void modifica_dados_brutos(double*,long);
short converte2de8para1de16(unsigned char, unsigned char);

FILE* fw=fopen("/Users/AntonioJHG/Desktop/treinamento.txt","a"); //escolher destino
fclose(fw);

for(int k=1;k<argc;k++)
        {
        FILE* fr;

        if(((fr=fopen(n[k],"rb"))!=NULL) )
        {
    struct
                {
                        unsigned char riff[4];
                        unsigned int len;
                } riff_header;
    fread(&riff_header,sizeof(riff_header),1,fr);

    cout<<"\nArquivo do tipo:
"<<riff_header.riff[0]<<riff_header.riff[1]<<riff_header.riff[2]<<riff_header.riff[3];
    cout<<"\nTamanho excluindo header: "<<riff_header.len;

    ////////////////////////////////////////////////////////////////////////////

    unsigned char wave[4];
    fread(&wave,sizeof(wave),1,fr); //////

    cout<<"\nSub-Tipo: "<<wave[0]<<wave[1]<<wave[2]<<wave[3];

    ////////////////////////////////////////////////////////////////////////////
    struct
                {
                        unsigned char id[4];
                        unsigned int len;
                } riff_chunk;
    fread(&riff_chunk,sizeof(riff_chunk),1,fr);

    cout<<"\nIdentificador:
"<<riff_chunk.id[0]<<riff_chunk.id[1]<<riff_chunk.id[2]<<riff_chunk.id[3];
    cout<<"\nComprimento do chunk apos header: "<<riff_chunk.len;
```

//continuacao Calculo da Entropia baseada em Wavelet (Wavelet based Entropy)

```
	////////////////////////////////////////////////////////////////////////////////
	struct
		{
			unsigned short formattag;
			unsigned short numberofchannels;
			unsigned int samplingrate;
			unsigned int avgbytespersecond;
			unsigned short blockalign;
		}  wave_chunk;
	fread(&wave_chunk,sizeof(wave_chunk),1,fr);

		//tratamento de uma excessao que costuma aparecer em alguns arquivos wav...
O correto seriam 16 bytes, as vezes aparecem 18 ou mais...
		if(riff_chunk.len>16)
		{
			unsigned char excesso;
			for(int i=0;i<riff_chunk.len-16;i++)
			{
				fread(&excesso,sizeof(excesso),1,fr);
			}
		}
		//fim do tratamento da excess?
	cout<<"\nCategoria do formato: "<<wave_chunk.formattag;
	cout<<"\nNumero de canais: "<<wave_chunk.numberofchannels;
	cout<<"\nTaxa de amostragem: "<<wave_chunk.samplingrate;
	cout<<"\nMedia do num. de bps: "<<wave_chunk.avgbytespersecond;
	cout<<"\nAlinhamento do bloco em bytes: "<<wave_chunk.blockalign;

	////////////////////////////////////////////////////////////////////////////////

	if(wave_chunk.formattag==1) //PCM
		{
			int resolucao=(wave_chunk.avgbytespersecond *
8)/(wave_chunk.numberofchannels * wave_chunk.samplingrate);
			cout<<"\nResolucao: "<<resolucao;

			struct
			{
				unsigned char data[4];
				unsigned int chunk_size;
			} header_data_chunk;

			fread(&header_data_chunk,sizeof(header_data_chunk),1,fr);

			cout<<"\nIdentificacao:
"<<header_data_chunk.data[0]<<header_data_chunk.data[1]<<header_data_chunk.data[2]<<hea
der_data_chunk.data[3];
			cout<<"\nTamanho do chunk de dados:
"<<header_data_chunk.chunk_size;
			cout<<"\nNumero de frames para amostrar:
"<<header_data_chunk.chunk_size/wave_chunk.blockalign;

			long
tamanho_da_janela=header_data_chunk.chunk_size/wave_chunk.blockalign;

			cout<<"\nTamanho da janela: "<<tamanho_da_janela;
```

//continuacao Calculo da Entropia baseada em Wavelet (Wavelet based Entropy)

```
                    if((resolucao==8) && (wave_chunk.numberofchannels==1))
                    {
                              unsigned char waveformdata;
                              double* amostras_no_tempo = new
double[tamanho_da_janela];
                              for(long i=0;i<tamanho_da_janela;i++)
                              {
                                        fread(&waveformdata,sizeof(waveformdata),1,fr);
                                        amostras_no_tempo[i]=(double)waveformdata;
                              }


          modifica_dados_brutos(&amostras_no_tempo[0],tamanho_da_janela);
                    }
                    else if((resolucao==8) && (wave_chunk.numberofchannels==2))
                    {
                              unsigned char waveformdata_right;
                              unsigned char waveformdata_left;
                              double* amostras_no_tempo_left = new
double[tamanho_da_janela];
                              double* amostras_no_tempo_right = new
double[tamanho_da_janela];
                              for(long i=0;i<tamanho_da_janela;i++)
                              {

          fread(&waveformdata_left,sizeof(waveformdata_left),1,fr);

          fread(&waveformdata_right,sizeof(waveformdata_right),1,fr);


          amostras_no_tempo_right[i]=(double)waveformdata_right;

          amostras_no_tempo_left[i]=(double)waveformdata_left;
                              }


          modifica_dados_brutos(&amostras_no_tempo_left[0],tamanho_da_janela);

          modifica_dados_brutos(&amostras_no_tempo_right[0],tamanho_da_janela);
                    }
                    else if((resolucao==16) && (wave_chunk.numberofchannels==1))
                    {
                              unsigned char waveformdata_lsb, waveformdata_msb;
                              double* amostras_no_tempo = new
double[tamanho_da_janela];
                              for(long i=0;i<tamanho_da_janela;i++)
                              {

          fread(&waveformdata_lsb,sizeof(waveformdata_lsb),1,fr);

          fread(&waveformdata_msb,sizeof(waveformdata_msb),1,fr);

          amostras_no_tempo[i]=(double)converte2de8para1de16(waveformdata_lsb,waveformd
ata_msb);
                              }
```

//continuacao Calculo da Entropia baseada em Wavelet (Wavelet based Entropy)

```
		modifica_dados_brutos(&amostras_no_tempo[0],tamanho_da_janela);
				}
				else if ((resolucao==16) && (wave_chunk.numberofchannels==2))
				{
						unsigned char waveformdata_lsb_left, waveformdata_lsb_right,
waveformdata_msb_left, waveformdata_msb_right;
						double* amostras_no_tempo_left = new
double[tamanho_da_janela];
						double* amostras_no_tempo_right = new
double[tamanho_da_janela];
						for(long i=0;i<tamanho_da_janela;i++)
						{

		fread(&waveformdata_lsb_left,sizeof(waveformdata_lsb_left),1,fr);

		fread(&waveformdata_msb_left,sizeof(waveformdata_msb_left),1,fr);

		fread(&waveformdata_lsb_right,sizeof(waveformdata_lsb_right),1,fr);

		fread(&waveformdata_msb_right,sizeof(waveformdata_msb_right),1,fr);

		amostras_no_tempo_left[i]=(double)converte2de8para1de16(waveformdata_lsb_left,wa
veformdata_msb_left);

		amostras_no_tempo_right[i]=(double)converte2de8para1de16(waveformdata_lsb_right,
waveformdata_msb_right);
						}


		modifica_dados_brutos(&amostras_no_tempo_left[0],tamanho_da_janela);

		modifica_dados_brutos(&amostras_no_tempo_right[0],tamanho_da_janela);
				}
				else
				{
						cout<<"Resolucao ou nmero de canais invalido(s)";
						exit(0);
				}
			}
	else
				cout<<"FORA DO FORMATO PCM...";
	fclose(fr);
		}
		else
	cout<<"Arquivo nao existe ou nao pode ser aberto";
		cout<<"\n\n\n";
 }
}
//-------------------------------------------------------------
short converte2de8para1de16(unsigned char lsb, unsigned char msb)
{
		return(((msb&0x80)>>7)*(32768) +
			((msb&0x40)>>6)*(16384) +
			((msb&0x20)>>5)*(8192) +
```

//continuacao Calculo da Entropia baseada em Wavelet (Wavelet based Entropy)

```
                        ((msb&0x10)>>4)*(4096) +
                        ((msb&0x08)>>3)*(2048) +
                        ((msb&0x04)>>2)*(1024) +
                        ((msb&0x02)>>1)*(512) +
                        ((msb&0x01))*(256) +
                        ((lsb&0x80)>>7)*(128) +
                        ((lsb&0x40)>>6)*(64) +
                        ((lsb&0x20)>>5)*(32) +
                        ((lsb&0x10)>>4)*(16) +
                        ((lsb&0x08)>>3)*(8) +
                        ((lsb&0x04)>>2)*(4) +
                        ((lsb&0x02)>>1)*(2) +
                        (lsb&0x01));
}
//-------------------------------------------------------

void modifica_dados_brutos(double* dados, long tamanho)
{
        void realiza_processamento(double*,long);

        double maior=fabs(dados[0]);
        for(long i=1;i<tamanho;i++)
                if(fabs(dados[i])>maior)
                        maior=fabs(dados[i]);

        for(long i=0;i<tamanho;i++)
                dados[i]/=maior;

        double* saida=new double[tamanho];
        long j=0;
        for(long i=0;i<tamanho;i++)
                if(dados[i]!=0)
                        {
                                saida[j]=dados[i];
                                j++;
                        }

        for(long i=0;i<tamanho;i++)
                dados[i]=saida[i];
        tamanho=j-1;


        realiza_processamento(dados,tamanho);
}


//-------------------------------------------------------
void realiza_processamento(double* dados, long tamanho)
{

        FILE* f=fopen("/Users/AntonioJHG/Desktop/treinamento.txt","a"); //escolher destino

        double ent, den;

        double* vetor_ent =new double[(int)(tamanho/512) + 1];
```

//continuacao Calculo da Entropia baseada em Wavelet (Wavelet based Entropy)

```
	double h[]={1/sqrt(2),1/sqrt(2)};
	int ch=(int)(sizeof(h)/sizeof(double));
	for (int inicio = 0; inicio < tamanho  - 1024; inicio += 512)	//incremento de 512 para
avancar janela por janela com sobreposicao de 50%
	{

		transformada_wavelet_packet(&dados[inicio],1024,10,h,ch);


		den = 0;
		for (long i = inicio; i < inicio + 1024; i++)	//incremento de 1 em 1 para fazer
cada elemento dentro de cada janela
		{
			den = den + (dados[i]*dados[i]);
		}
		ent = 0;
		for (long i = inicio; i < inicio + 1024; i++)
			{
			if(dados[i]!=0)
				ent = ent -
((pow(dados[i],2)/den)*log2(pow(dados[i],2)/den));		//calculo entropia
			}
		vetor_ent[inicio/512] = ent;


		transformada_wavelet_packet_inversa(&dados[inicio],1024,10,h,ch);

	}

	//-----------------------------------------------------------------

	double maior = fabs(vetor_ent[0]);
	double menor = fabs(vetor_ent[0]);

	double soma = 0;
	double media;

	for (long i=0; i<(int)(((tamanho-1024)/512)+1); i++)
		{
		if (fabs(vetor_ent[i])>maior)
			maior = fabs(vetor_ent[i]);

		if (fabs(vetor_ent[i])<menor)
			menor = fabs(vetor_ent[i]);


		soma = soma + vetor_ent[i];
		}

	media = soma/(int)(((tamanho-1024)/512)+1);


	double soma_std = 0;
	for (long i=0; i<(int)(((tamanho-1024)/512)+1); i++)
		soma_std = soma_std + pow((media - vetor_ent[i]),2);
```

```
//continuacao Calculo da Entropia baseada em Wavelet (Wavelet based Entropy)


        double desv_pad = sqrt(soma_std/(int)(((tamanho-1024)/512)+1));

        double* delta_janelas = new double[(int)(((tamanho-1024)/512))];

        for (long i=1; i<(int)(((tamanho-1024)/512)+1); i++)
                {
                delta_janelas[i] = fabs(vetor_ent[i] - vetor_ent[i-1]);
                std::cout<<"\n"<<i<<"\t"<<delta_janelas[i];


                }

        double maior_deltas = fabs(delta_janelas[0]);
        double menor_deltas = fabs(delta_janelas[0]);

        for (long i=0; i<(int)(((tamanho-1024)/512)); i++)
                {
                if (fabs(delta_janelas[i])>maior_deltas)
                        maior_deltas = fabs(delta_janelas[i]);
                if (fabs(delta_janelas[i])<menor_deltas)
                        menor_deltas = fabs(delta_janelas[i]);
                }

         fprintf(f,"%.5f,%.5f,%.5f,%.5f,%.5f,",media,desv_pad,maior,menor,maior_deltas);

        fclose(f);


        //-------------------------------------------------------------------------------

}

double log2 (double x)
{
        return(log10(x)/log10(2));
}

//////////////////////////////////////////////
//////////////////////////////////////////////
```

```
//calculo da lacunaridade e dimensao de fractal

#include<iostream.h>
#include<stdio.h>
#include<math.h>
#include<string.h>
#include "wavelet.h"
//-------------------------------------
double lac(double*,int);
double box_counting(double*,int);

main(int argc,char* n[])
{
void modifica_dados_brutos(double*,int);
short converte2de8para1de16(unsigned char, unsigned char);

FILE* fw=fopen("/Users/AntonioJHG/Desktop/treinamento.txt","a");
fclose(fw);

for(int k=1;k<argc;k++)
        {
        FILE* fr;

        if(((fr=fopen(n[k],"rb"))!=NULL) )
        {
    struct
                {
                        unsigned char riff[4];
                        unsigned int len;
                } riff_header;
    fread(&riff_header,sizeof(riff_header),1,fr);

    cout<<"\nArquivo do tipo:
"<<riff_header.riff[0]<<riff_header.riff[1]<<riff_header.riff[2]<<riff_header.riff[3];
    cout<<"\nTamanho excluindo header: "<<riff_header.len;

    ////////////////////////////////////////////////////////////////////////////

    unsigned char wave[4];
    fread(&wave,sizeof(wave),1,fr); //////

    cout<<"\nSub-Tipo: "<<wave[0]<<wave[1]<<wave[2]<<wave[3];

    ////////////////////////////////////////////////////////////////////////////
    struct
                {
                        unsigned char id[4];
                        unsigned int len;
                } riff_chunk;
    fread(&riff_chunk,sizeof(riff_chunk),1,fr);

    cout<<"\nIdentificador:
"<<riff_chunk.id[0]<<riff_chunk.id[1]<<riff_chunk.id[2]<<riff_chunk.id[3];
    cout<<"\nComprimento do chunk apos header: "<<riff_chunk.len;

    ////////////////////////////////////////////////////////////////////////////
    struct
                {
                        unsigned short formattag;
```

```
//continuacao calculo da lacunaridade e dimensao de fractal



                        unsigned short numberofchannels;
                        unsigned int samplingrate;
                        unsigned int avgbytespersecond;
                        unsigned short blockalign;
                }  wave_chunk;
    fread(&wave_chunk,sizeof(wave_chunk),1,fr);

                //tratamento de uma excessao que costuma aparecer em alguns arquivos wav...
O correto seriam 16 bytes, as vezes aparecem 18 ou mais...
                if(riff_chunk.len>16)
                {
                        unsigned char excesso;
                        for(int i=0;i<riff_chunk.len-16;i++)
                        {
                                fread(&excesso,sizeof(excesso),1,fr);
                        }
                }
                //fim do tratamento da excess?
    cout<<"\nCategoria do formato: "<<wave_chunk.formattag;
    cout<<"\nNumero de canais: "<<wave_chunk.numberofchannels;
    cout<<"\nTaxa de amostragem: "<<wave_chunk.samplingrate;
    cout<<"\nMedia do num. de bps: "<<wave_chunk.avgbytespersecond;
    cout<<"\nAlinhamento do bloco em bytes: "<<wave_chunk.blockalign;

    ////////////////////////////////////////////////////////////////////////////////

    if(wave_chunk.formattag==1) //PCM
                {
                        int resolucao=(wave_chunk.avgbytespersecond *
8)/(wave_chunk.numberofchannels * wave_chunk.samplingrate);// pq nao bitssample
                        cout<<"\nResolucao: "<<resolucao;

                        struct
                        {
                                unsigned char data[4];
                                unsigned int chunk_size;
                        } header_data_chunk;

                        fread(&header_data_chunk,sizeof(header_data_chunk),1,fr);

                        cout<<"\nIdentificacao:
"<<header_data_chunk.data[0]<<header_data_chunk.data[1]<<header_data_chunk.data[2]<<hea
der_data_chunk.data[3];
                        cout<<"\nTamanho do chunk de dados:
"<<header_data_chunk.chunk_size;
                        cout<<"\nNumero de frames para amostrar:
"<<header_data_chunk.chunk_size/wave_chunk.blockalign;

                        int
tamanho_da_janela=header_data_chunk.chunk_size/wave_chunk.blockalign;

                        cout<<"\nTamanho da janela: "<<tamanho_da_janela;
                        if((resolucao==8) && (wave_chunk.numberofchannels==1))
                        {
                                unsigned char waveformdata;
```

//continuacao calculo da lacunaridade e dimensao de fractal

```
                                        double* amostras_no_tempo = new
double[tamanho_da_janela];
                                        for(int i=0;i<tamanho_da_janela;i++)
                                        {
                                                fread(&waveformdata,sizeof(waveformdata),1,fr);
                                                amostras_no_tempo[i]=(double)waveformdata;
                                        }


        modifica_dados_brutos(&amostras_no_tempo[0],tamanho_da_janela);
                                }
                                else if((resolucao==8) && (wave_chunk.numberofchannels==2))
                                {
                                        unsigned char waveformdata_right;
                                        unsigned char waveformdata_left;
                                        double* amostras_no_tempo_left = new
double[tamanho_da_janela];
                                        double* amostras_no_tempo_right = new
double[tamanho_da_janela];
                                        for(int i=0;i<tamanho_da_janela;i++)
                                        {

        fread(&waveformdata_left,sizeof(waveformdata_left),1,fr);

        fread(&waveformdata_right,sizeof(waveformdata_right),1,fr);


        amostras_no_tempo_right[i]=(double)waveformdata_right;

        amostras_no_tempo_left[i]=(double)waveformdata_left;
                                        }


        modifica_dados_brutos(&amostras_no_tempo_left[0],tamanho_da_janela);

        modifica_dados_brutos(&amostras_no_tempo_right[0],tamanho_da_janela);
                                }
                                else if((resolucao==16) && (wave_chunk.numberofchannels==1))
                                {
                                        unsigned char waveformdata_lsb, waveformdata_msb;
                                        double* amostras_no_tempo = new
double[tamanho_da_janela];
                                        for(int i=0;i<tamanho_da_janela;i++)
                                        {

        fread(&waveformdata_lsb,sizeof(waveformdata_lsb),1,fr);

        fread(&waveformdata_msb,sizeof(waveformdata_msb),1,fr);

        amostras_no_tempo[i]=(double)converte2de8para1de16(waveformdata_lsb,waveformd
ata_msb);
                                        }
```

```
//continuacao calculo da lacunaridade e dimensao de fractal



		modifica_dados_brutos(&amostras_no_tempo[0],tamanho_da_janela);
				}
				else if ((resolucao==16) && (wave_chunk.numberofchannels==2))
				{
					unsigned char waveformdata_lsb_left, waveformdata_lsb_right,
waveformdata_msb_left, waveformdata_msb_right;
					double* amostras_no_tempo_left = new
double[tamanho_da_janela];
					double* amostras_no_tempo_right = new
double[tamanho_da_janela];
					for(int i=0;i<tamanho_da_janela;i++)
					{

		fread(&waveformdata_lsb_left,sizeof(waveformdata_lsb_left),1,fr);

		fread(&waveformdata_msb_left,sizeof(waveformdata_msb_left),1,fr);

		fread(&waveformdata_lsb_right,sizeof(waveformdata_lsb_right),1,fr);

		fread(&waveformdata_msb_right,sizeof(waveformdata_msb_right),1,fr);

		amostras_no_tempo_left[i]=(double)converte2de8para1de16(waveformdata_lsb_left,wa
veformdata_msb_left);

		amostras_no_tempo_right[i]=(double)converte2de8para1de16(waveformdata_lsb_right,
waveformdata_msb_right);
					}


		modifica_dados_brutos(&amostras_no_tempo_left[0],tamanho_da_janela);

		modifica_dados_brutos(&amostras_no_tempo_right[0],tamanho_da_janela);
				}
				else
				{
					cout<<"Resolucao ou nmero de canais invalido(s)";
					exit(0);
				}
			}
	else
				cout<<"FORA DO FORMATO PCM...";
	fclose(fr);
		}
		else
	cout<<"Arquivo nao existe ou nao pode ser aberto";
		cout<<"\n\n\n";
 }
}
//-------------------------------------------------------------
short converte2de8para1de16(unsigned char lsb, unsigned char msb)
{
		return(((msb&0x80)>>7)*(32768) +
			((msb&0x40)>>6)*(16384) +
			((msb&0x20)>>5)*(8192) +
			((msb&0x10)>>4)*(4096) +
```

```
//continuacao calculo da lacunaridade e dimensao de fractal



                        ((msb&0x08)>>3)*(2048) +
                        ((msb&0x04)>>2)*(1024) +
                        ((msb&0x02)>>1)*(512) +
                        ((msb&0x01))*(256) +
                        ((lsb&0x80)>>7)*(128) +
                        ((lsb&0x40)>>6)*(64) +
                        ((lsb&0x20)>>5)*(32) +
                        ((lsb&0x10)>>4)*(16) +
                        ((lsb&0x08)>>3)*(8) +
                        ((lsb&0x04)>>2)*(4) +
                        ((lsb&0x02)>>1)*(2) +
                        (lsb&0x01));
}
//-------------------------------------------------------

void modifica_dados_brutos(double* dados, int tamanho)
{
        void realiza_processamento(double*,int);

        double maior=fabs(dados[0]);
        for(int i=1;i<tamanho;i++)
                if(fabs(dados[i])>maior)
                        maior=fabs(dados[i]);

        for(int i=0;i<tamanho;i++)
                dados[i]/=maior;

        double* saida=new double[tamanho];
        int j=0;
        for(int i=0;i<tamanho;i++)
                if(dados[i]!=0)
                        {
                                saida[j]=dados[i];
                                j++;
                        }

        for(int i=0;i<tamanho;i++)
                dados[i]=saida[i];
        tamanho=j-1;


        realiza_processamento(dados,tamanho);
}


//-------------------------------------------------------
void realiza_processamento(double* dados, int tamanho)
{

        FILE* f=fopen("/Users/AntonioJHG/Desktop/treinamento.txt","a");

        double par, den;

        double* vetor_par =new double[(int)(tamanho/512) + 1];
```

```
//continuacao calculo da lacunaridade e dimensao de fractal


	for (int inicio = 0; inicio < tamanho  - 10*1024; inicio += 512)		//incremento de 512
para avancar janela por janela com sobreposicao de 50%
	{

	vetor_par[(int)(inicio/512)] = lac(&dados[inicio],1024);	//quando quiser utilizar
box_counting comentar esa linha

	//vetor_par[(int)(inicio/512)] = box_counting(&dados[inicio],1024);
	//comentar linha de cima e abrir esta

	}

	//-----------------------------------------------------------------

	double maior = fabs(vetor_par[0]);
	double menor = fabs(vetor_par[0]);

	double soma = 0;
	double media;

	for (int i=0; i<(int)(((tamanho-10*1024)/512)+1); i++)
		{
		if (fabs(vetor_par[i])>maior)
			maior = fabs(vetor_par[i]);

		if (fabs(vetor_par[i])<menor)
			menor = fabs(vetor_par[i]);


		soma = soma + vetor_par[i];
		}

	media = soma/(int)(((tamanho-10*1024)/512)+1);


	double soma_std = 0;
	for (int i=0; i<(int)(((tamanho-10*1024)/512)+1); i++)
		soma_std = soma_std + pow((media - vetor_par[i]),2);

	double desv_pad = sqrt(soma_std/(int)(((tamanho-10*1024)/512)+1));

	double* delta_janelas = new double[(int)(((tamanho-10*1024)/512))];

	for (int i=1; i<(int)(((tamanho-10*1024)/512)+1); i++)
		{
		delta_janelas[i] = fabs(vetor_par[i] - vetor_par[i-1]);
		std::cout<<"\n"<<i<<"\t"<<delta_janelas[i];


		}

	double maior_deltas = fabs(delta_janelas[0]);
	double menor_deltas = fabs(delta_janelas[0]);

	for (int i=0; i<(int)(((tamanho-10*1024)/512)); i++)
```

//continuacao calculo da lacunaridade e dimensao de fractal

```
                    {
                    if (fabs(delta_janelas[i])>maior_deltas)
                              maior_deltas = fabs(delta_janelas[i]);
                    if (fabs(delta_janelas[i])<menor_deltas)
                              menor_deltas = fabs(delta_janelas[i]);
                    }

          fprintf(f,"%.5f,%.5f,%.5f,%.5f,%.5f,",media,desv_pad,maior,menor,maior_deltas);

          fclose(f);



}
/////////////////////////////////////////////////////
/////////////////////////////////////////////////////

double lac(double* x, int n)
{
double* s=new double[n];
for(int i=0;i<n;i++)
{  s[i]=x[i]; }

double menor=s[0];
for(int i=1;i<n;i++)
  if(s[i]<menor)
    menor=s[i];
for(int i=1;i<n;i++)
  s[i]-=menor;

double maior=s[0];
for(int i=1;i<n;i++)
  if(s[i]>maior)
    maior=s[i];
for(int i=1;i<n;i++)
  s[i]/=maior;

double media=0;
for(int i=0;i<n;i++)
  media+=s[i];
media/=(double)(n);

for(int i=0;i<n;i++)
  s[i]/=(double)(media);

for(int i=0;i<n;i++)
  s[i]-=1;

for(int i=0;i<n;i++)
  s[i]=fabs(s[i]);

media=0;
for(int i=0;i<n;i++)
  media+=s[i];
media/=(double)(n);
```

//continuacao calculo da lacunaridade e dimensao de fractal

```
return(media);
}           //aqui fecha a lac

//////////////////////////////////////////////////////
//////////////////////////////////////////////////////
double box_counting(double* p, int n)
{
double log2(double);
int bc(double*,int,int);

double* s = new double[n];
for(int i=0;i<n;i++)
  s[i]=p[i];

double menor=s[0];
for(int i=1;i<n;i++)
  if(s[i]<menor)
    menor=s[i];

for(int i=0;i<n;i++)
  s[i]-=menor;

double maior=s[0];
for(int i=1;i<n;i++)
  if(s[i]>maior)
    maior=s[i];

for(int i=0;i<n;i++)
  s[i]=(s[i]/maior)*n;
//signal fits now inside a square

double*x = new double[(int)(log2(n))];
double*y = new double[(int)(log2(n))];

x[0]=n;
y[0]=1;
for(int i=1;i<(int)(log2(n));i++)
  {
  x[i]=(int)(x[i-1]/2.0);
  y[i]=bc(s,n,(int)(x[i]));
  }

for(int i=0;i<((int)(log2(n)));i++)
  {
  x[i]=log2(x[i]);
  y[i]=log2(y[i]);
  }

double sx=0;
double sy=0;
double sxy=0;
double sx2=0;
for(int i=0;i<((int)(log2(n)));i++)
  {
```

//continuacao calculo da lacunaridade e dimensao de fractal

```
  sx+=x[i];
  sy+=y[i];
  sxy+=x[i]*y[i];
  sx2+=x[i]*x[i];
  }
return((-((sx*sy-((int)(log2(n)))*sxy)/(sx*sx-((int)(log2(n)))*sx2))));
}
//////////////////////////////////////////////////////
//////////////////////////////////////////////////////
int bc(double* v,int t,int q)
{
double maior;
double menor;
int c=0;
for(int i=0;i<(int)((double)t/(double)q);i++)
  {
  maior=v[0];
  menor=v[0];
  for(int j=i*q;j<(i+1)*q;j++)
    {
    if(v[j]>maior)
      maior=v[j];
    if(v[j]<menor)
      menor=v[j];
    }
  c+=(maior-menor+1);
  }
return(c);
}
//////////////////////////////////////////////////////
//////////////////////////////////////////////////////
double log2(double x)
{
return(log10(x)/log10(2));
}

//////////////////////////////////////////////
//////////////////////////////////////////////
```

```
//classificador SVM

#include "svm.h"
#include<math.h>
#include<iostream>
double svm(double*,double*,int,double);
double dist(double*,double*,int);
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
neural_network::neural_network()
{

}
//////////////////////////////////////////////////////////////////
neural_network::~neural_network()
{

}
//////////////////////////////////////////////////////////////////
void neural_network::init(int n_i_n,int n_h_n,int n_o_n)
{
number_of_input_neurons=n_i_n;
number_of_hidden_neurons=n_h_n;
number_of_output_neurons=n_o_n;

a=new double[number_of_hidden_neurons];
b=new double*[number_of_hidden_neurons];
for(int i=0;i<number_of_hidden_neurons;i++)
	b[i]=new double[number_of_input_neurons];
w=new double*[number_of_output_neurons];
for(int i=0;i<number_of_output_neurons;i++)
	w[i]=new double[number_of_hidden_neurons];
for(int i=0;i<number_of_output_neurons;i++)
	for(int j=0;j<number_of_hidden_neurons;j++)
		w[i][j]=0;
fis=new double*[number_of_hidden_neurons];
for(int i=0;i<number_of_hidden_neurons;i++)
	fis[i]=new double[number_of_hidden_neurons];
}
//////////////////////////////////////////////////////////////////
void neural_network::pass(double* input_signal, double* output_signal)
{
for(int i=0;i<number_of_output_neurons;i++)
	{
	output_signal[i]=0;
	for(int j=0;j<number_of_hidden_neurons;j++)

	output_signal[i]+=w[i][j]*svm(&input_signal[0],&b[j][0],number_of_input_neurons,a[j]);
	}
}
//////////////////////////////////////////////////////////////////
void neural_network::train(double* input_signal, double* desired_output_signal)
{
//non-supervised training
//centers
int k=0;
for(int i=0;i<number_of_hidden_neurons;i++)
	{
	for(int j=0;j<number_of_input_neurons;j++)
```

```
//continuacao classificador SVM


                b[i][j]=input_signal[j+k];
        k+=number_of_input_neurons;
        }
//variances
double maior=0;
for(int i=0;i<number_of_hidden_neurons;i++)
        for(int j=i;j<number_of_hidden_neurons;j++)
                if(dist(&b[i][0],&b[j][0],number_of_input_neurons)>maior)
                        maior=dist(&b[i][0],&b[j][0],number_of_input_neurons);

for(int i=0;i<number_of_hidden_neurons;i++)
        a[i]=maior;
//supervised training
//weights
std::cout<<"\n\n";
double* outs=new double[number_of_hidden_neurons];
double m;
int lcpivo;
int p;
for(int c=0;c<number_of_output_neurons;c++)
        {
        std::cout<<"\nNEURON "<<c;
        p=0;
        for(int i=0;i<number_of_hidden_neurons;i++)
                {
                for(int j=0;j<number_of_hidden_neurons;j++)
                        fis[i][j]=svm(&input_signal[p],&b[j][0],number_of_input_neurons,a[j]);
                p+=number_of_input_neurons;
                }

        p=0;
        for(int u=0;u<number_of_hidden_neurons;u++)
                {
                outs[u]=desired_output_signal[p+c];
                p+=number_of_output_neurons;
                }

        maior=fis[0][0];
        for(int i=0;i<number_of_hidden_neurons;i++)
            {
            for(int j=0;j<number_of_hidden_neurons;j++)
                if(fis[i][j]>maior)
                 maior=fis[i][j];
        for(int j=0;j<number_of_hidden_neurons;j++)
        fis[i][j]/=maior;
        outs[i]/=maior;
        }

        //escalonando o sistema

        lcpivo=0;
        do
                {
                for(int k=lcpivo;k<number_of_hidden_neurons-1;k++)
                    {
                                if (fis[lcpivo][lcpivo]==0)
```

//continuacao classificador SVM

```
                                        m=0;
                                else
                                        m=(-fis[k+1][lcpivo]/fis[lcpivo][lcpivo]);
                for(int j=lcpivo + 1;j<number_of_hidden_neurons;j++)
                        fis[k+1][j]+=fis[lcpivo][j]*m;
                                fis[k+1][lcpivo]=0;
                                outs[k+1]+=outs[lcpivo]*m;
                }
                lcpivo++;
                }while(lcpivo<number_of_hidden_neurons-1);

        //solucao do sistema escalonado
        for(int i=0;i<number_of_hidden_neurons;i++)
        w[c][i]=outs[i];
        if(fis[number_of_hidden_neurons-1][number_of_hidden_neurons-1]==0)
                w[c][number_of_hidden_neurons-1]=0;
        else
                w[c][number_of_hidden_neurons-1]=outs[number_of_hidden_neurons-
1]/fis[number_of_hidden_neurons-1][number_of_hidden_neurons-1];
        for(int i=(number_of_hidden_neurons-2);i>=0;i--)
        {
        for(int j=i;j<number_of_hidden_neurons-1;j++)
        w[c][i]-=fis[i][j+1]*w[c][j+1];
        if(fis[i][i]==0)
                                w[c][i]=0;
                        else
                                w[c][i]/=fis[i][i];
                        }
        }
}
//////////////////////////////////////////////////////////////////
double svm(double* input_signal,double* mean,int number_of_input_neurons,double variance)
//kernel gaussiano
{
if(variance==0)
  return(exp(-
(pow(dist(input_signal,mean,number_of_input_neurons),2)/(2*variance+0.0000000000001))));
return(exp(-(pow(dist(input_signal,mean,number_of_input_neurons),2)/(2*variance))));
}
//////////////////////////////////////////////////////////////////
double dist(double* x, double* y, int length)
{
double d=0;
for(int i=0;i<length;i++)
        d+=pow((x[i]-y[i]),2);
return(sqrt(d));
}
```

```
//biblioteca classificador SVM (svm.h)

//////////////////////////////////////////////////
//////////////////////////////////////////////////
class neural_network
{
private:

	double* a;
	double** b;
	double** w;
	double** fis;

	int number_of_input_neurons;
	int number_of_hidden_neurons;
	int number_of_output_neurons;

public:
	neural_network();
	~neural_network();
	void init(int,int,int);
	void train(double*,double*);//input vector, desired output vector
	void pass(double*,double*);//input vector, output vector.
};
```

```
//classificador GMM

#include "gau.h"
#include<math.h>
#include<iostream>
double media(double*,int);
double variancia(double*,int);
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
neural_network::neural_network()
{

}
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
neural_network::~neural_network()
{

}
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
void neural_network::init(int n_i_n,int n_t_e,int n_o_n)
{
number_of_input_neurons=n_i_n;
number_of_training_examples=n_t_e;
number_of_examples_in_each_class=n_o_n;

number_of_classes=(int)(number_of_training_examples/number_of_examples_in_each_class);

mean=new double*[number_of_classes];
for(int i=0;i<number_of_classes;i++)
	mean[i]=new double[number_of_input_neurons];
variance=new double*[number_of_classes];
for(int i=0;i<number_of_classes;i++)
	variance[i]=new double[number_of_input_neurons];
}
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
int neural_network::pass(double* input_signal)
{
double* p=new double[number_of_classes];
for(int i=0;i<number_of_classes;i++)
	p[i]=1;
for(int i=0;i<number_of_classes;i++)
	for(int j=0;j<number_of_input_neurons;j++)
		p[i]*=/*(1/(sqrt(variance[i][j]))*sqrt(2*3.1415927))**/exp(-
pow(input_signal[j]-mean[i][j],2)/(2*variance[i][j]));

double maior=p[0];
int i_maior=0;
for(int i=0;i<number_of_classes;i++)
	{
	printf("\np[%d] = %.20f",i+1,p[i]);
	if(p[i]>maior)
		{
		maior=p[i];
		i_maior=i;
		}
	}
```

```
//continuacao classificador GMM


return(i_maior+1);
}
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
void neural_network::train(double* input_signal)
{

double** s=new double*[number_of_input_neurons];
for(int i=0;i<number_of_input_neurons;i++)
	s[i]=new double[number_of_training_examples];

for(int i=0;i<number_of_input_neurons;i++)
	for(int j=0;j<number_of_training_examples;j++)
		s[i][j]=input_signal[i+(j*number_of_input_neurons)];

for(int i=0;i<number_of_classes;i++)
	for(int j=0;j<number_of_input_neurons;j++)
		{

	mean[i][j]=media(&s[j][i*number_of_examples_in_each_class],number_of_examples_in_e
ach_class);

	variance[i][j]=variancia(&s[j][i*number_of_examples_in_each_class],number_of_example
s_in_each_class);
		}
}
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
double media(double* signal ,int t)
{
double m=0;
for(int i=0;i<t;i++)
	m+=signal[i];
m/=t;
}
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
double variancia(double* signal ,int t)
{
double ex2=0;
for(int i=0;i<t;i++)
	ex2+=signal[i]*signal[i];
ex2/=t;

double ex=0;
for(int i=0;i<t;i++)
	ex+=signal[i];
ex/=t;

return(ex2 - (ex*ex));
}
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
```

```
//biblioteca classificador GMM (gau.h)

//////////////////////////////////////////////////////
//////////////////////////////////////////////////////
class neural_network
{
private:

	double** mean;
	double** variance;

	int number_of_input_neurons;
	int number_of_training_examples;
	int number_of_examples_in_each_class;
	int number_of_classes;
public:
	neural_network();
	~neural_network();
	void init(int,int,int);
	void train(double*);//input vector
	int pass(double*);//input vector
};
```

Apêndice IV - Artigo IEEE ISM 2011

Neste apêndice encontra-se o artigo publicado no evento "*7th Institute of Electrical and Electronic Engineers International Symposium on Multimedia*", que ocorreu nos dias 5, 6 e 7 de dezembro de 2011 em Dana Point, na Califórnia.

Music Genre Classification based on entropy and fractal lacunarity

Antonio Jose Homsi Goulart, Carlos Dias Maciel, Rodrigo Capobianco Guido,
Katia Cristina Silva Paulo, Ivan Nunes da Silva
School of Engineering at São Carlos, University of São Paulo.
Av. Trabalhador São Carlense 400, 13566-590, São Carlos, SP, Brazil.

Abstract

In this letter, we present an automatic music genre classification scheme based on a Gaussian Mixture Model (GMM) classifier. The proposed technique adopts entropies and lacunarities as features for the classifications. Tests were carried out with four styles of Brazilian music, namely Axé, Bossa Nova, Forró, and Samba.

1 Introduction

Given the ease of music downloading, streaming, on-line radio stations access and media storage on personal computers and portable players, music database management is a must nowadays. The opportunity to rapidly choose a tune or genre among thousands, or even a smart playlist generation calls for precise automatic music genre classification systems.

Is was also shown that users browse more by genre than by artist or recommendation [7] and that genre influence the liking more than the music itself, playing an important role in appreciation and cognition [8],[9]. Nevertheless, music genre classification is ambiguous and subjective. Some claim that not even humans are able to classify genres that does not even have clear definitions [6]. So, systems based on metadata will not get high accuracy. It will be shown in section 2 some works that does not count on metadata for the classification, but in parameters extracted directly from the songs. Most of them explore timbral and rhytmic content. Our work, on the other hand, explores information theoretic concepts, like entropies, and lacunarities, which come from fractal theory.

The remainder of this work is organized as follows. Section 2 presents a review on literature about music genre classification techniques, covering the state-of-the-art in the field. The proposed approach is described with details in section 3. Section 4 lists the tests carried out and, lastly, useful comments and conclusions are included in section 5.

2 Literature review

Mckay and Fujinaga [6] elaborated a paper on why should researchers continue efforts to enhance the area of automatic music genre classification (AMGC). The issues they point out are related to the ambiguity and subjectivity in the classifications and the dynamism of music styles. It takes a lot of expertise and time to manually classify recordings, and also there is limited agreement among human annotators when classifying music by genre. Very few genres have clear definitions and there is often significant overlap among them. Also, classifications tend to be by artist or album rather than by individual recordings, and metadata found in mp3 tags tend to have unreliable annotations. Finally, new genres are introduced regularly, and the understanding of existing genres change with time.

The ground-breaking work of Dannemberg *et. al.* [10], based on naive Bayesian and neural network approaches, identifies one out of four styles of a musician improvisation. They were testing a performer's ability to consistently produce intentional and different styles. A database was elaborated to train the classifiers, and an accuracy of 98% was achieved when classifying among four styles. When using eight classifiers, trained to return "yes" or "no" for eight different styles, they got an overall accuracy of 77% to 90%.

Another classic work in the area is the one of Tzanetakis and Cook [11]. They proposed three different feature sets to represent timbral texture, rhythmic and pitch content. Short-time Fourier Transform (STFT), Mel-frequency Cepstral Coeffi-

cients (MFCCs), Wavelet Transform (WT) [4], and some additional parameters were used to obtain feature vectors. With these vectors, they could train statistical pattern recognition classifiers such as simple Gaussian, Gaussian Mixture Model, and k-Nearest Neighbour [5], by using real world audio collections. They achieved correct classifications of 61% for ten musical genres.

Li *et. al.* [12] worked on a comparative study between timbral textural, rhythmic content features and pitch content features versus features based on Daubechies Wavelet Coefficient Histograms (DWCH) [4]. For the classifications, they used Support Vector Machines (SVM), Linear Discriminant Analysis (LDA) and some other learning methods [5]. They also tested the use of One-Against-All (OAA) and Round-Robin (RR) approaches [5]. They used both first seconds of and middle parts of songs to carry out tests. The best overall accuracy (74.2%) was achieved when using DWCH features and an SVM classifier based on the OAA approach, being this test carried out with middle parts of songs (seconds 31 to 60).

Ezzaidi and Rouat [13] proposed two methods. They divided the musical pieces into frames and then got MFCCs from averaged spectral energies. Finally, for comparison purposes, they used Gaussian Mixture Models (GMMs), obtaining a maximum of 99% recognition.

Silla *et. al.* [14] adopted multiple feature vectors that were selected from different time segments from the beginning, middle and final parts of the music, and pattern recognition ensemble approach, according to a space-time decomposition dimension. Naive-Bayes, decision trees, k Nearest-neighbors, SVMs and Multilayer Perceptron Neural Networks were employed. The best accuracy obtained was 65.06% when using Round-Robin on Space-time ensemble.

Panagakis and Kotropoulos [15] proposed a music genre classification framework that considers the properties of the auditory human perception system, i.e., 2D auditory temporal modulations representing music and genre classification based on sparse representation. The accuracies they obtained outperformed any rate ever reported for the GTZAN [11] and ISMIR2004 datasets, i.e., 91% and 93.56%, respectively.

Paradzinets *et. al.* [16] explored acoustic information, beat-related and timbre characteristics. To obtain acoustic information they used Piecewise Gaussian Modeling (PGM) [5] features enhanced by modeling of human auditory filter. To do so, they obtained the PGM features, then applied critical bands filter, equal loudness and specific loudness sensation. To extract the beat-related characteristics, they used wavelet transforms, getting the 2D-beat histograms. For the timbre characteristics, they collected all detected notes with relative amplitude of their harmonics and then computed their histograms. Among others issues, their results show: i) an improvement when using perceptually motivated PGM instead of basic PGM, i.e. accuracy of 43% versus 40.6%; ii) training different NNs for each genre is better than training only one NN with all the genres being considered, which corresponds to an average accuracy of 49.3%.

In a previous work [1], the authors worked with entropies as parameters and tried several SVM architectures for a classifier. Full accuracy was reached with massive training, but and overall accuracy of 88% among 3 styles of music was obtained with little training, thus showing a strong capacity to generalize. Fractal dimensions were tested as parameters, worsening the results nevertheless.

What is shown is that a lot of work is being done in the area, but most of the approaches explore the timbre texture, the rhythmic content, the pitch content, or their combinations. As illustrated above, our work explores the use of entropies and lacunarities, thus, eliminating the use of musical information such as harmony, melody, beat and tempo. Information theory and fractals concepts are the basis of our approach.

3 The proposed approach

Our approach consists of a feature extraction stage followed by a classification step. These features are extracted directly from the digital music files. Each song is divided into frames of 1024 samples with 50 percent overlap between consecutive frames. Only the middle part of each song is used for the extraction. This procedure prevents the appearance of strange passages that could distort the results, such as those on the beginning and end of some tunes, and also saves a little computational effort.

Lacunarity of each frame was calculated based on the algorithm described in [3]. Then, after a wavelet transformation, entropy of each frame is

calculated via the energy approach, i.e.,

$$-\sum_{i=0}^{1023} p_i \log_2(p_i) \qquad ,$$

where

$$p_i = \frac{{x_i}^2}{\left(\sum_{j=0}^{1023} x_j^2\right)} \qquad ,$$

as described in [2]. This criterion was adopted because it turned out to be more stable than the amplitude and frequency approaches. As we have shown in [1], entropy values in frequency domain returned better classification results.

With the lacunarity and entropy value of each frame, we could form a ten-feature vector for each song, composed by: average entropy; standard deviation of the entropy values; maximum entropy; minimum entropy; maximum entropy difference among consecutive frames; average lacunarity; standard deviation of the lacunarity values; maximum lacunarity; minimum lacunarity; maximum lacunarity difference among consecutive frames. Finally, this feature vector feed a classifier based on Gaussian Mixture Models (GMMs), which are traditionally used in speech processing applications.

For all the tests, we used 80 examples of tunes equally divided on four distinct genres, namely Axé, Bossa Nova, Forró, and Samba. All the songs were ripped from CDs at 44.1 kHz sampling rate, 16-bit resolution, wave format. The entropy and lacunarity values were extracted directly from the wave files.

4 Tests and results

Three rounds of tests were carried out. First, 6 songs of each genre were used for training, and the other fourteen for testing purposes. For the second round, 10 tunes were used on training and 10 on tests. On the last round, 12 songs of each genre were reserved for training and the other 8 for tests. The results of each round are shown on table 1.

Table 1. Classification obtained; each line is a round of tests; the numbers on columns training and tests are for each genre. [X]: training, [Y]: tests, [A]: accuracy Axé, [B]: accuracy Bossa nova, [C]: accuracy Forró, [D]: accuracy Samba, [E]: Overall accuracy.

X	Y	A	B	C	D	E
6	14	71.43%	100%	71.43%	35.72%	69.65%
10	10	90%	90%	90%	30%	75%
12	8	87.5%	75%	100%	50%	78.13%

5 Conclusions

In this article, we described an algorithm for music genre classification based on entropies, lacunarities and GMMs. On some specific styles and training cases the classifier presented 90% of accuracy, reaching even 100% in one case. An overall accuracy of 69.65% was reached with a modest training of only 6 tunes of each genre, so the proposed approach demonstrated prominent results with a considerable ability to generalize.

We recall that only the middle part of the song files were used to extract the above-mentioned characteristics, according to some of our colleagues' reports listed in section 2. That saves a little computational effort and is sufficient in terms of information, once we are interested in genres general characteristics, not on specific parts of the songs.

Another importat fact to be mentioned is that there is no other work reported in the literature in which lacunarity is used for music genre classification.

References

[1] A. Goulart, R. Guido, C.D. Maciel. Exploring different approaches for Music Genre Classification, Expert systems with apllications, in press.

[2] T. Cover, J. Thomas; Elements of Information Theory, 2nd edition, John Wiley & Sons, 2006.

[3] M. Al-Akaidi; Fractal Speech Processing, Cambridge University Press, 2004.

[4] Li Deng, Douglas O'shaugnessy, Deng Deng; Speech Processing: A Dynamic And Optimization-Oriented Approach, Marcel Dekker, 2003.

[5] Richard O. Duda, Peter E. Hart, David G. Stork; Pattern Classification, 2nd edition, John Wiley & Sons, 2001.

[6] McKay, C., Fujinaga, I; Musical genre classification: Is it worth pursuing and how can it be im-

proved. Proc. of the 7th Int. Conf. on Music Information Retrieval (ISMIR-06), 2006.

[7] J. H. Lee, and J. S. Downie; Survey of Music Information Needs, Uses, and Seeking Behaviours: Preliminary Findings. In Proceedings of the International Conference on Music Information Retrieval, 2004.

[8] A. C. North, and D. J. Hargreaves. Liking for Musical Styles, Music Scientae, vol. 1, no. 1, pp. 109-28, 1997.

[9] H. G. Tekman, and N. Hortacsu; Aspects of Stylistic Knowledge: What are Different Styles Like and Why Do We Listen to Them ?, Psychology of Music, vol. 30, no. 1, pp. 28-47, 2002.

[10] R. B. Dannenberg, B. Thom, and D. Watson. A Machine Learning Approach to Musical Style Recognition. In Proceedings of the International Computer Music Conference, 1997, pp. 344-7.

[11] G. Tzanetakis., and P. Cook. Musical Genre Classification of Audio Signals, IEEE Transactions on Speech and Audio Processing, vol. 10, no. 5, pp. 293-302, 2002.

[12] T. Li; M. Ogihara; Q. Li; A Comparative study on content-based Music Genre Classification, Proceedings of the 26th Annual International ACM SI-GIR Conference on Research and Development in Information Retrieval, Toronto, ACM Press, pages 282-289, 2003.

[13] H. Ezzaidi, and J. Rouat; Automatic Musical Genre Classification Using Divergence and Average Information Measures, M. World Academy of Science, Engineering and Technology, 15, 2006.

[14] C.N. Silla Jr., A.L. Koerich, C.A.A. Kaestner; A machine learning approach to automatic music genre classification, Journal of the Brazilian Computer Society, vol.14, no.3, Campinas, September 2008.

[15] Y. Panagakis, C. Kotropoulos, G. R. Arce; Music genre classification via sparse representations of auditory temporal modulations, 17th European Signal Processing Conference (EUSIPCO), 2009.

[16] A. Paradzinets, H. Harb, and L. Chen; Multiexpert system for automatic music genre classification, liris.cnrs.fr/Documents/Liris-4224.pdf, Research report, 2009.

Apêndice V - Matéria publicada na Folha de SP

Jornal Folha de São Paulo, Seção “tec”.
Quarta-feira, 16 de novembro de 2011.

Música na nuvem ganha espaço no cenário musical
por Leonardo Martins, em colaboração para a Folha.
Colaborou Rafael Capanema, de Sao Paulo.

Ouvir milhões de músicas via *streaming* no computador com a opção de baixá-las para seus aparelhos móveis, sem gastar muito, já foi sonho distante no Brasil.

Agora, com a chegada do Rdio e a consolidação do formato Terra Sonora, há opções práticas e com preço honesto para entrar no universo da música na nuvem.

Nos testes realizados pelha Folha, os dois serviços se mostraram capazes de substituir o *download* convencional, mesmo com omissões consideráveis no acervo.

O americano Rdio, que chega ao país em parceria com a Oi, afirma ter mais de 12 milhões de músicas.

Parte delas ainda não está liberada no Brasil, mas a empresa promete, sem definir uma data exata, que todo o acervo disponível nos EUA estará presente no Brasil - desde o início do mês, quando entrou no ar, o *site* e seus aplicativos do Rdio estão sendo atualizados constantemente.

O serviço fechou parcerias com gravadoras nacionais, como Deck e Som Livre, para abrasileirar a discoteca. O resultado é uma boa oferta de títulos nacionais dos anos 60 e 70 e um fraco catálogo para quem quer música brasileira mais atual.

E esse é o trunfo do Sonora, serviço criado pelo portal Terra e disponível desde 2006. Há um ano, a ferramenta se adaptou ao formato de *streaming* e hoje conta com 450 mil usuários pagantes.

Apesar do acervo menor, com mais de 3 milhões de músicas, o servico é bem adaptado ao público nacional. Com opções mais populares, foco em canções dos anos 90 e 2000 e rádios específicas.

O sistema de catalogação e a disposição por gêneros do Sonora deixam a desejar. É comum encontrar obras de Miles Davis na seção rock, enquanto Justin Bieber figura na área dedicada à MPB.

Além da experiência *on-line*, o Sonora oferece pacotes com a opção de *downloads* mensais com boa qualidade de áudio que podem ser transferidos para qualquer aparelho ou

software.

Para o usuário, é como se, além da audição via *streaming* - limitada apenas ao *site* e a aplicativos do Sonora -, ele pagasse pela compra de um álbum completo todos os meses, sem restrições.

Já a Oi Rdio oferece sistema de sincronia entre aparelhos móveis (*smartphones* e *tablets*) e a coleção ouvida no PC. Com apenas dois sistemas de plano, seu valor máximo é de R$14,90 mensais.

Os servicos têm aplicativos para Android, iOS e BlackBerry. O Oi Rdio oferece ainda um programa para *desktop* que substitui o iTunes ou o Windows Media Player.

Saiba mais

Streaming se popularizou com rádio

O crescimento dos servicos de *streaming* comecou no início dos anos 2000, com as rádios *on-line*. Com a evolução da Internet banda larga e acordos com grandes gravadoras, serviços personalizáveis e completos, como o Pandora e o Rhapsody, tornaram-se populares nos EUA e na Europa.

Música na nuvem ganha espaço no cenário nacional

Rdio chega prometendo 12 milhões de músicas; Sonora, do Terra, tem 450 mil usuários e mais de 3 milhões de faixas

Serviços funcionam bem em desktops, smartphones e tablets; apesar de buracos, acervos se completam

LEONARDO MARTINS
COLABORAÇÃO PARA A **FOLHA**

Ouvir milhões de músicas via streaming no computador com a opção de baixá-las para seus aparelhos móveis, sem gastar muito, já foi sonho distante no Brasil.

Agora, com a chegada do Rdio e a consolidação do formato no Terra Sonora, há opções práticas e com preço honesto para entrar no universo da música na nuvem.

Nos testes realizados pela **Folha**, os dois serviços se mostraram capazes de substituir o download convencional, mesmo com omissões consideráveis no acervo.

O americano Rdio, que chega ao país em parceria com a Oi, afirma ter mais de 12 milhões de músicas.

Parte delas ainda não está liberada no Brasil, mas a empresa promete, sem definir uma data exata, que todo o acervo disponível nos EUA estará presente no Brasil —desde o início do mês, quando entrou no ar, o site e seus aplicativos do Rdio estão sendo atualizados constantemente.

O serviço fechou parcerias com gravadoras nacionais, como Deck e Som Livre, para abrasileirar a discoteca.

O resultado é uma boa oferta de títulos nacionais dos anos 60 e 70 e um fraco catálogo para quem quer música brasileira mais atual.

E esse é o trunfo do Sonora, serviço criado pelo portal Terra e disponível desde 2006. Há um ano, a ferramenta se adaptou ao formato de streaming e hoje conta com 450 mil usuários pagantes.

Apesar do acervo menor, com mais de 3 milhões de músicas, o serviço é bem adaptado ao público nacional, com opções mais populares, foco em canções dos anos 90 e 2000 e rádios específicas.

O sistema de catalogação e a disposição por gêneros do Sonora deixam a desejar. É comum encontrar obras de Miles Davis na seção rock, enquanto Justin Bieber figura na área dedicada à MPB.

Além da experiência on-line, o Sonora oferece pacotes com a opção de downloads mensais com boa qualidade de áudio e que podem ser transferidos para qualquer aparelho ou software.

Para o usuário, é como se, além da audição via streaming —limitada apenas ao site e a aplicativos do Sonora—, ele pagasse pela compra de um álbum completo todos os meses, sem restrições.

Já o Oi Rdio oferece sistema de sincronia entre aparelhos móveis (smartphones e tablets) e a coleção ouvida no PC. Com apenas dois sistemas de plano, seu valor máximo é de R$ 14,90 mensais.

Os serviços têm aplicativos para Android, iOS e BlackBerry. O Oi Rdio oferece ainda um programa para desktop que substitui o iTunes ou o Windows Media Player.

Colaborou **RAFAEL CAPANEMA**, de São Paulo.

› SAIBA MAIS

STREAMING SE POPULARIZOU COM RÁDIO

O crescimento dos serviços de streaming começou no início dos anos 2000, com as rádios on-line. Com a evolução da internet banda larga e acordos com grandes gravadoras, serviços personalizáveis e completos, como o Pandora e o Rhapsody, tornaram-se populares nos EUA e na Europa.