IMPLEMENTAÇÃO DE INTERFACE PARA REALIZAÇÃO DE EXPERIMENTOS COM SISTEMAS DE RECONHECIMENTO DA FALA

Felipe Castro Vieira Martins

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Fernando Gil Vianna Resende Junior

Rio de Janeiro
Abril de 2013

IMPLEMENTAÇÃO DE INTERFACE PARA REALIZAÇÃO DE EXPERIMENTOS COM SISTEMAS DE RECONHECIMENTO DA FALA

Felipe Castro Vieira Martins

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. Fernando Gil Vianna Resende Junior, Ph.D.

Profª. Mariane Rembold Petraglia, Ph.D.

Prof. Amaro Azevedo de Lima, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
ABRIL DE 2013

Martins, Felipe Castro Vieira

Implementação de interface para realização de experimentos com sistemas de reconhecimento da fala/Felipe Castro Vieira Martins. – Rio de Janeiro: UFRJ/COPPE, 2013.

XVI, 87 p.: il.; 29, 7cm.

Orientador: Fernando Gil Vianna Resende Junior

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2013.

Referências Bibliográficas: p. 47 – 48.

1. Processamento de Sinais da Fala. 2. Reconhecimento de Voz. 3. Reconhecimento da Fala. 4. HMM. 5. Sphinx. 6. Interface de testes. I. Resende Junior, Fernando Gil Vianna. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Dedico este trabalho a todos que
confiaram em mim
verdadeiramente.

Agradecimentos

Agradeço primeiramente aos meus pais, Carlos e Nádia, pelo apoio constante e eterno dado a mim durante toda a vida. Fosse ele apoio moral, emocional ou financeiro.

Agradeço à minha namorada Isabela por ter me acompanhado durante toda esta jornada, me apoiando e me aconselhando. Mais do que isso, me estimulando em momentos de sufoco e compartilhando momentos felizes, sendo capaz de me alegrar ainda mais.

Agradeço ao meu irmão Fernando pelo companheirismo e por sempre acreditar em mim.

Agradeço aos meus familiares e amigos, os quais não cometerei a injustiça de citar nomes para não cometer o erro de esquecer de alguém.

Agradeço ao meu orientador Fernando Gil por ter me aceitado como seu aluno sem hesitar um momento sequer.

Agradeço à Professora Mariane Petraglia e ao Professor Amaro Lima por terem aceitado o convite para participar da banca examinadora.

Agradeço à FAPERJ pela bolsa Aluno Nota 10 contemplada a mim no segundo ano do Mestrado.

Agradeço à CAPES pela bolsa de Mestrado durante o primeiro ano do Mestrado.

Agradeço à COPPE-UFRJ pela oportunidade de ter estudado em uma das melhores instituições de ensino de pós-graduação do país.

E, por fim, agradeço a todos que contribuíram de alguma forma para a realização deste projeto.

Muito obrigado!

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

IMPLEMENTAÇÃO DE INTERFACE PARA REALIZAÇÃO DE EXPERIMENTOS COM SISTEMAS DE RECONHECIMENTO DA FALA

Felipe Castro Vieira Martins

Abril/2013

Orientador: Fernando Gil Vianna Resende Junior

Programa: Engenharia Elétrica

Apresenta-se, nesta dissertação, o desenvolvimento de um sistema de reconhecimento da fala contínua baseado no *toolkit* CMU Sphinx com a criação de uma interface para realização de testes e análises de resultados.

O objetivo desta interface é proporcionar ao projetista um estudo prático sobre a influência que determinados parâmetros causam num sistema de reconhecimento da fala sem a necessidade de qualquer conhecimento prévio sobre ferramentas auxiliares, como os *toolkits*.

Após a análise teórica da influência de cada um dos parâmetros pertinentes ao reconhecimento da fala, é realizado um detalhamento sobre o funcionamento e a manipulação do CMU Sphinx, e a implementação da interface.

Além disso, são apresentados resultados de testes qualitativos da ferramenta, que se mostrou satisfatória aos objetivos propostos. Utilizando uma base composta por um único locutor, contendo mil frases foneticamente balanceadas, obteve-se a melhor taxa de acurácia em $89,233\%$.

A dissertação também contém o tutorial para a montagem de um sistema de reconhecimento utilizando o CMU Sphinx, o manual de utilização da interface criada e exemplos de padrões de arquivos externos utilizados no sistema.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

IMPLEMENTATION OF INTERFACE TO REALIZE EXPERIMENTS WITH SPEECH RECOGNITION SYSTEMS

Felipe Castro Vieira Martins

April/2013

Advisor: Fernando Gil Vianna Resende Junior

Department: Electrical Engineering

This thesis presents the development of a complete system of continuous speech recognition based on the CMU Sphinx toolkit, including an interface for tests and analysis of results. The goal of this interface is to provide the user with a practical study of the configurable parameters' effect on a speech recognition system without the need of any prior knowledge of auxiliary tools, such as the toolkits.

After a theoretical analysis of the influence of each of the parameters related to the speech recognition, the operation and handling of the CMU Sphinx and the implementation of the interface are discussed.

In addition, the results of qualitative tests of the tool, which proved to be satisfactory, are presented. Using a base composed of a single speaker, containing one thousand phonetically balanced sentences, we obtained the best accuracy rate on 89.233%.

The document also contains a tutorial to aid in setting up a recognition system through the CMU Sphinx toolkit, the operation manual of the interface and external file pattern examples used in the system.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

A	Matriz de transição de estados, p. 11
B	Matriz de emissão de estados, p. 11
$H_{\text{pre}}(z)$	Resposta em frequência do filtro de pré-ênfase, p. 6
L	Tamanho da janela de liftragem cepstral, p. 8
M	Modelo HMM, p. 11
$N(.;\mu,\Sigma)$	Distribuição gaussiana, p. 12
P	Tamanho da janela, p. 6
S	Sequência de estados, p. 11
W	Sequência de palavras, p. 13
X	Sequência de vetores de características, p. 11
X_k	Transformada DCT, p. 8
$\Delta\Delta c_t$	Diferencial de segunda ordem dos coeficientes MFCC, p. 9
Δc_t	Diferencial de primeira ordem dos coeficientes MFCC, p. 9
Π	Matriz de probabilidade inicial de estados, p. 11
$\overline{x}$	Média aritmética, p. 36
σ	Desvio padrão populacional, p. 36
$\widehat{W}$	Estimativa de uma sequência de palavras, p. 14
a_n	n-ésimo coeficiente LPC, p. 8
a_{pre}	Coeficiente de pré-ênfase, p. 6
absprev	Valor absoluto de *prevlkhd*, p. 11

c_n	n-ésimo coeficiente MFCC, p. 8
$convgRatio$	Taxa de convergência, p. 11
del	Número de deleções, p. 14
ins	Número de inserções, p. 14
$l(n)$	Janela de liftragem cepstral, p. 8
$lkhdperframe$	Verossimilhança por quadro na interação corrente, p. 11
n	Índice da amostra, p. 6
$prevlkhd$	Verossimilhança por quadro na interação anterior, p. 11
sub	Número de substituições, p. 14
$w(n)$	Janela de Hamming, p. 6
w_i	i-ésima palavra, p. 13
x_t	Vetor de coeficientes, p. 9

Lista de Abreviaturas

CMU	Carnegie Mellon University, p. 3
CSR	Continuous Speech Recognition, p. 4
DCT	Discrete Cosine Transform, p. 8
FFT	Fast Fourrier Transform, p. 8
HMM	Hidden Markov Model, p. 4
HTK	Hidden Markov Model Toolkit, p. 2
HTML	HyperText Markup Language, p. 22
LPC	Linear Predictive Coding, p. 7
MFCC	Mel-Frequency Cepstral Coefficients, p. 7
PHP	Hypertext Preprocessor, p. 22
RTF	Real Time Factor, p. 16
URA	Unidade de Resposta Audível, p. 41
WER	Word Error Rate, p. 13
XML	eXtensible Markup Language, p. 22

Capítulo 1

Introdução

O desenvolvimento da tecnologia e o objetivo de tornar a interação homem-máquina mais parecida com a forma natural de interação entre seres humanos vêm fazendo com que diversos sistemas de automação e interfaces de comunicação se tornem cada vez mais simples. Quanto mais simples e intuitiva for a comunicação entre homens e máquinas, maior será a comodidade de utilização de sistemas e a popularização da tecnologia, visto que, com interfaces mais simples, menos conhecimento técnico é necessário para a utilização de sistemas. Como exemplo destas interfaces que vêm sendo desenvolvidas massivamente na última década, tem-se as telas sensíveis a toques e sistemas de reconhecimento e síntese de voz.

Além da intenção de facilitar a comunicação entre homem e máquina, a interface comunicadora por voz tem apelo para inclusão social de pessoas com deficiências visuais (sintetizador) e auditivas (reconhecedor).

Este trabalho se insere neste contexto de voz, especialmente na área de reconhecimento da fala. Sistemas de reconhecimento da fala estão disponibilizados em diversas formas por todo o mundo. Sejam em sistemas embarcados, sistemas *online*, *offline*, etc., eles são encontrados gratuitamente ou de maneira comercial. Países como Estados Unidos, Inglaterra e Japão são referências na área e possuem sistemas robustos e eficientes.

Dentre os principais *toolkits* para desenvolvimento de sistemas de reconhecimento de voz atualmente no mundo, destacam-se: o inglês *Hidden Markov Model Toolkit* (HTK) [1], desenvolvido na Universidade de Cambridge, hoje propriedade intelectual da Microsoft, que comprou seus direitos e mantém seu código aberto mas com restrições de comercialização; O japonês Julius [2], atualmente desenvolvido pela Universidade de Kyoto em parceria com o Instituto Tecnológico de Nagoya, criado sob a plataforma Unix / Linux com código aberto e livre, com documentação completa no idioma japonês e escassa no idioma inglês; E o norte-americano CMU Sphinx [3], desenvolvido pela Universidade de Carnegie Mellon, com código aberto e livre, com uma documentação melhor que o Julius e inferior à do HTK e com

fóruns de discussões e exemplos básicos de utilização. Por apresentar características interessantes de serem abordadas, ampla rede de colaboradores e pouca restrição de uso, optou-se neste trabalho por utilizar como ferramenta o CMU Sphinx. Este trabalho consiste na implementação de uma interface de teste e configuração de um sistema de reconhecimento de voz utilizando como base o *toolkit* CMU Sphinx.

1.1 Motivação

Sistemas de reconhecimento de voz apresentam uma teoria complexa. Por esta razão, os *toolkits* surgiram implementando o estado da arte em reconhecimento da fala e facilitando o desenvolvimento de sistemas mais específicos. Porém, uma das dificuldades destes *toolkits*, em especial do CMU Sphinx, é a quantidade de parâmetros a serem ajustados e a ligação entre arquivos, seja para treinamento ou testes do sistema. Além disso, os manuais e tutoriais da ferramenta ainda se encontram incompletos.

Não existem até hoje sistemas de reconhecimento da fala contínua para o Português Brasileiro baseado em Sphinx com uma interface gráfica simples e, além disso, sabe-se que sistemas de reconhecimento da fala são dependentes do idioma.

O principal foco deste trabalho é desenvolver uma ferramenta que possibilite ao usuário que deseja lidar com sistemas de reconhecimento de voz baseado no CMU Sphinx trabalhar em cima de testes e configurações do sistema sem a necessidade de conhecimento do *toolkit*, facilitando o desenvolvimento de novas técnicas e conclusões apenas com o estudo teórico da influência de cada parâmetro ou teste do sistema, podendo aperfeiçoar a tecnologia sem a necessidade de direcionar tempo ao aprendizado do *toolkit*.

Além disso, este projeto tem a intenção de facilitar o estudo da teoria de reconhecimento da fala, podendo ser aplicado como método didático de aprendizado.

A ferramenta desenvolvida neste trabalho é aplicável a qualquer idioma e os testes realizados estão focados na aplicação para o idioma Português Brasileiro, com sotaque do Rio de Janeiro.

1.2 Objetivos

O principal objetivo deste projeto consiste em elaborar um sistema que possua uma interface que permita ao usuário alterar de maneira simples e intuitiva todos os parâmetros do treinamento do modelo acústico e de linguagem, além da extração de características dos arquivos de áudio de treinamento e os testes de um sistema de reconhecimento de voz baseado em CMU Sphinx. Além disso, o sistema deve

ser capaz de gerar, ligar e gerenciar todos os arquivos necessários ao CMU Sphinx e mostrar resultados de testes.

De forma itemizada, os objetivos do trabalho são os seguintes:

- Implementar um sistema de reconhecimento da fala contínua para Português Brasileiro baseado no CMU Sphinx;
- Desenvolver uma interface baseada no sistema implementado no item acima;
- Realizar testes qualitativos do sistema de reconhecimento de voz como um todo, incluindo a interface criada;
- Escrever tutorial para desenvolvimento do sistema de reconhecimento da fala, escrever manual para utilização da interface.

1.3 Descrição

Este trabalho se encontra organizado da seguinte maneira: o Capítulo 2 descreve uma abordagem teórica e os fundamentos utilizados no trabalho. O Capítulo 3 descreve o sistema de reconhecimento da fala desenvolvido e a criação da interface. No Capítulo 4, são mostrados alguns testes qualitativos realizados com o sistema. Por fim, no Capítulo 5, são feitas as conclusões sobre o trabalho e uma projeção para trabalhos futuros. Posteriormente, nos Apêndices, seguem exemplos de arquivos externos utilizados no sistema, o tutorial de implementação de um sistema de reconhecimento da fala baseado em CMU Sphinx e o manual de utilização da interface.

Capítulo 2

Fundamentos

Este capítulo abordará todos os fundamentos teóricos necessários para a implementação de um sistema de reconhecimento da fala. O capítulo está organizado da seguinte maneira: primeiramente, na Seção 2.1, será dada uma visão geral sobre reconhecimento da fala; na Seção 2.2 serão abordados todos os fundamentos referentes à extração de características do sinal de voz para realização do treinamento e do reconhecimento; a Seção 2.3 descreverá os parâmetros necessários para o treinamento do modelo acústico do sistema, utilizando Modelos Ocultos de Markov, do inglês *Hidden Markov Models* (HMM); a Seção 2.4 descreverá os parâmetros e o treinamento referentes ao modelo de linguagem; e, por fim, na Seção 2.5, serão apresentadas a definição e a escolha de parâmetros referentes aos testes de execução de um sistema de reconhecimento da fala.

2.1 Visão geral sobre reconhecimento da fala

Basicamente, a definição de um sistema de reconhecimento da fala contínua, ou *Continuous Speech Recognition* (CSR), consiste na ideia de fazer uma máquina transcrever em texto o conteúdo de um sinal acústico de fala. De maneira simplificada, a Figura 2.1 demonstra, através de um diagrama, a ideia de funcionamento de um sistema de reconhecimento da fala.

Figura 2.1: Simplificação de um sistema de reconhecimento da fala

Um sinal de voz é captado pelo microfone e digitalizado. Posteriormente, o arquivo digital é entregue ao bloco do reconhecimento da fala. A saída do sistema é um arquivo texto contendo toda a transcrição do que foi falado no sinal de voz de entrada.

A implementação de um sistema de reconhecimento da fala pode ser dividido em duas etapas: o treinamento e a execução. O treinamento consiste em toda a preparação do sistema para reconhecer sinais de fala e a execução consiste no reconhecimento propriamente dito.

A Figura 2.2 demonstra melhor a divisão de etapas do treinamento e da execução. O treinamento compreende a extração de vetores de características do sinal, o treinamento do modelo acústico utilizando os vetores de características, calculando a verossimilhança de uma sequência de vetores dado um modelo, e o treinamento do modelo de linguagem, que mapeia através de probabilidades os relacionamentos entre palavras da base de texto. A execução compreende a utilização dos modelos acústicos e de linguagem treinados e do áudio a ser reconhecido no decodificador, através de reconhecimento de padrões e busca da sequência de palavras que melhor representa os vetores da entrada, para gerar o resultado em forma de texto.

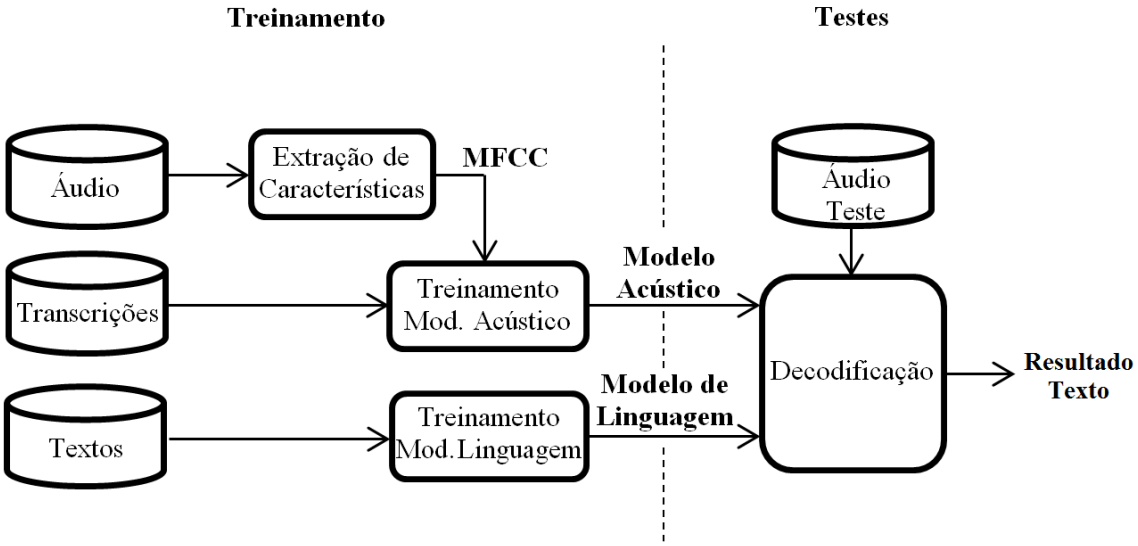


Figura 2.2: Diagrama de etapas de um sistema de reconhecimento da fala. Adaptado de [4]

2.2 Extração de características

A primeira etapa para proporcionar o funcionamento de um sistema CSR é a extração de características do sinal de voz. Existem diversos estudos e diferentes tipos de características que podem ser extraídas do sinal. A extração de características do sinal de voz por si só pode ser objeto de profundos estudos de pesquisa. Nesta seção

serão abordadas as características necessárias para o funcionamento do sistema CSR considerando o estado da arte desenvolvido nesta área.

O sinal elétrico de voz precisa ser transformado num sinal discretizado para que possa ser aplicado o processamento digital sobre ele. Então, para tal, é necessária a escolha da taxa de amostragem. Sabe-se da literatura que o sinal de voz humana não ultrapassa a faixa de 10 kHz e que, dependendo da aplicação, uma taxa de amostragem de 8 kHz é suficiente. Isso porque a maior parte da banda do sinal de voz se encontra até a faixa de 4 kHz (aplicação da taxa de Nyquist para escolha da taxa de amostragem que se deseja, levando em consideração o dobro da maior frequência do sinal emitido).

Outra opção é definir também se é desejado eliminar ou não o nível DC do sinal.

O próximo parâmetro a ser ajustado para extração de características é o de pré-ênfase. Este parâmetro é relativo ao filtro de pré-ênfase aplicado ao sinal digitalizado com o objetivo de compensar a atenuação natural de aproximadamente 20 dB por década que o sinal de voz tem devido a características fisiológicas. Para isso, utiliza-se um filtro com um ganho de aproximadamente mesmo módulo da atenuação para compensá-la [5]. O filtro de pré-ênfase apresenta a seguinte resposta em frequência:

$$H_{\text{pre}}(z) = 1 - a_{\text{pre}} z^{-1} \tag{2.1}$$

Tipicamente, os valores para o parâmetro a_{pre} variam entre $0,90$ e $1,0$. Para a maioria dos trabalhos na área de reconhecimento de voz, o valor utilizado para este parâmetro é de aproximadamente $0,95$ [6].

O próximo passo, então, para tratar o sinal para extração de características é o janelamento. Tipicamente, para sistemas de reconhecimento da fala, usa-se a janela de Hamming [7] para suavização do espectro de frequências, dada por

$$w\,[n] = \begin{cases} 0,54 - 0,46\cos[2\pi n/(P-1)], & 0 \leq n \leq P \\ 0, & \text{caso contrário} \end{cases} \tag{2.2}$$

onde P é o comprimento da janela.

O comprimento da janela pode ser variável. Para voz, considerando que seu sinal é estacionário durante um curto período de tempo, P pode ficar entre 15 ms e 30 ms.

O *frame rate* define a taxa de janelas por segundo. Essa taxa irá determinar o tamanho da sobreposição entre as janelas adjacentes. Geralmente, para processamento de voz, usa-se sobreposição de janelas a fim de se obter uma melhor representação do sinal, cobrindo, inclusive, as bordas de cada janela através das janelas adjacentes. Por exemplo, considerando uma janela de comprimento de 25 ms e uma taxa de janelas (*frame rate*) de 100 janelas por segundo, tem-se uma janela a cada 10 ms com

sobreposição de 15 ms. A Figura 2.3 ilustra o janelamento. O período do quadro é definido como o inverso do *frame rate*.

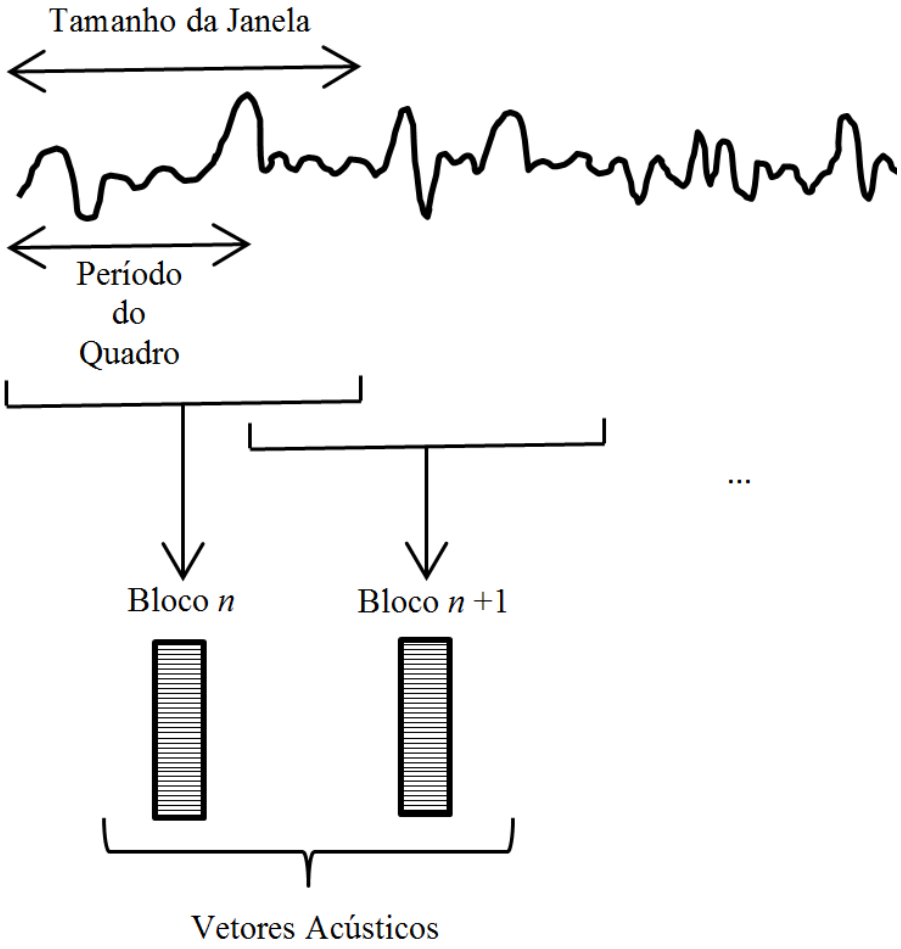


Figura 2.3: Diagrama do janelamento do sinal. Adaptado de [8]

De acordo com o estado da arte, a melhor forma de representação dos sinais da fala para reconhecimento é através dos coeficientes mel-cepstrais (MFCC - *Mel-Frequency Cepstral Coefficients*). Tais coeficientes são capazes de modelar de maneira eficiente o sistema auditivo humano, utilizando uma escala logarítmica para posicionar as bandas de frequência. Existem algumas maneiras para o cálculo dos coeficientes cepstrais. Eles podem ser calculados através de uma transformação direta dos coeficientes de predição linear (LPC) a partir das Equações (2.3) e (2.4) ou da utilização de bancos de filtros, para posterior obtenção dos MFCC. Este trabalho utiliza o método de obtenção dos MFCC através de banco de filtros.

$$c_1 = a_1 \tag{2.3}$$

$$c_n = \sum_{k=1}^{n-1} \left(1 - \frac{k}{n}\right) a_k c_{n-k} + a_n \tag{2.4}$$

Para realizar a extração dos coeficientes MFCC com um banco de filtros é aplicada a *Fast Fourier Transform* (FFT) à cada janela, e posteriormente seu módulo ao quadrado. Ou seja, na verdade tem-se o espectro de potência de curto termo do sinal. O espectro de potência é multiplicado pelo quadrado da resposta em frequência dos canais do banco de filtros, triangulares e linearmente espaçados na escala *mel*, realizando a convolução circular no tempo do sinal com as respostas impulsivas dos filtros do banco de filtros. A representação dessa forma consiste numa sequência de $N_{janelas}$ com N_{canais}, onde $N_{janelas}$ é o número de janelas e N_{canais} é o número de canais do banco de filtros [9]. Além do número de canais do banco de filtros a serem utilizados, é necessário definir também a frequência mínima e máxima de operação do banco de filtros. Tais frequências devem respeitar sempre o Teorema de Nyquist.

Além disso, deve ser definido o número de filtros que irão compor o banco. Como passo seguinte, após a aplicação de logaritmo, tem-se a utilização da transformada *Discrete Cosine Transform* (DCT) para levar os coeficientes de volta ao domínio do tempo para, por fim, obter-se os MFCC. A Figura 2.4 resume os passos para extração dos coeficientes. O *toolkit* CMU Sphinx permite a escolha do tipo de transformada DCT a ser utilizada para extração dos MFCC. Tem-se como opção a transformada DCT típica, denominada DCT-I, descrita pela Equação (2.5), a transformada HTK, utilizada pelo *toolkit* HTK, sendo a DCT-II usada, descrita pela Equação (2.6), e a transformada Legacy, implementada pelo próprio CMU Sphinx que utiliza uma pseudo-DCT, tendo uma ortogonalização aplicada a uma coluna ao invés de uma linha da matriz base.

$$X_k = \frac{1}{2}\left(x_0 + (-1)^k x_{N-1}\right) + \sum_{n=1}^{N-2} \cos\left[\frac{\pi}{N-1} nk\right] \quad , k = 0, ..., N-1 \tag{2.5}$$

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right] \quad , k = 0, ..., N-1 \tag{2.6}$$

Após a análise cepstral do sinal, é necessário realizar a liftragem cepstral. A liftragem consiste em reescalonar os coeficientes que tenham magnitudes similares de forma a se obter uma melhor representação do formante de um segmento de voz [10]. Este escalonamento é feito através de um método de filtragem simples ou simplesmente multiplicando, no tempo, o sinal por uma janela curta, de comprimento L. A Equação (2.7) demonstra um tipo de janela utilizada neste processo de liftragem.

$$l(n) = \begin{cases} 1 + \frac{L}{2}\sin\left(\frac{\pi n}{L}\right), & n = 0, 1, ...L \\ 0, & n > L \end{cases} \tag{2.7}$$

O número de coeficientes MFCC deve ser escolhido. Usualmente, para sistemas

de reconhecimento da fala e outras aplicações de voz, é indicado o uso de 13 coeficientes por janela. Tais coeficientes vêm acompanhados de suas diferenciais de primeira e segunda ordem, totalizando 39 coeficientes. A utilização das diferenciais de primeira e segunda ordens são necessárias para suprir a deficiência que os coeficientes MFCC básicos têm de não serem completamente descorrelacionados entre si, já que a modelagem acústica presume tal fato [11]. O vetor de coeficientes, x_t é, então, formado por:

$$x_t = \begin{pmatrix} c_t \\ \Delta c_t \\ \Delta\Delta c_t \end{pmatrix} \tag{2.8}$$

E os coeficientes Δc_t podem ser calculados de acordo com a Equação (2.9) e os $\Delta\Delta c_t$ pela Equação (2.10) [4].

$$\Delta c_t = \frac{\sum_{\theta=1}^{\Theta} \theta \left(c_{t+\theta} - c_{t-\theta}\right)}{2\sum_{\theta=1}^{\Theta} \theta^2} \tag{2.9}$$

$$\Delta\Delta c_t = \frac{\sum_{\theta=1}^{\Theta} \theta \left(\Delta c_{t+\theta} - \Delta c_{t-\theta}\right)}{2\sum_{\theta=1}^{\Theta} \theta^2} \tag{2.10}$$

2.3 Modelo acústico

O objetivo da modelagem acústica é definir uma maneira para se calcular a verossimilhança entre unidades sonoras de segmentos da fala. Como estado da arte na área de reconhecimento de voz, encontram-se os Modelos Ocultos de Markov (HMM). Atualmente, a maioria dos sistemas que apresentam melhor desempenho para o reconhecimento da fala utiliza como base os HMM's.

Os HMM's são capazes de modelar quaisquer unidades de uma elocução. É possível modelar frases, palavras, unidades menores das palavras como fones, difones ou trifones, inseridos ou não em contextos diferentes.

Os HMM's podem ser vistos como máquinas de estados finitas. Cada estado emite um vetor acústico com uma densidade de probabilidade associada. Além disso, ocorrem transições nunca retrógradas entre estados, de forma que cada estado só pode transitar em direção ao estado seguinte ou permanecer nele mesmo, não havendo a possibilidade de retornar a estados anteriores.

A Figura 2.5 ilustra de maneira simplificada uma máquina de estados formada por HMM com três estados emissores. No início e no fim do modelo foram adicionados dois estados, respectivamente, não emissores para facilitar a união entre os modelos. Um HMM pode ser modelado de acordo com a Equação (2.11).

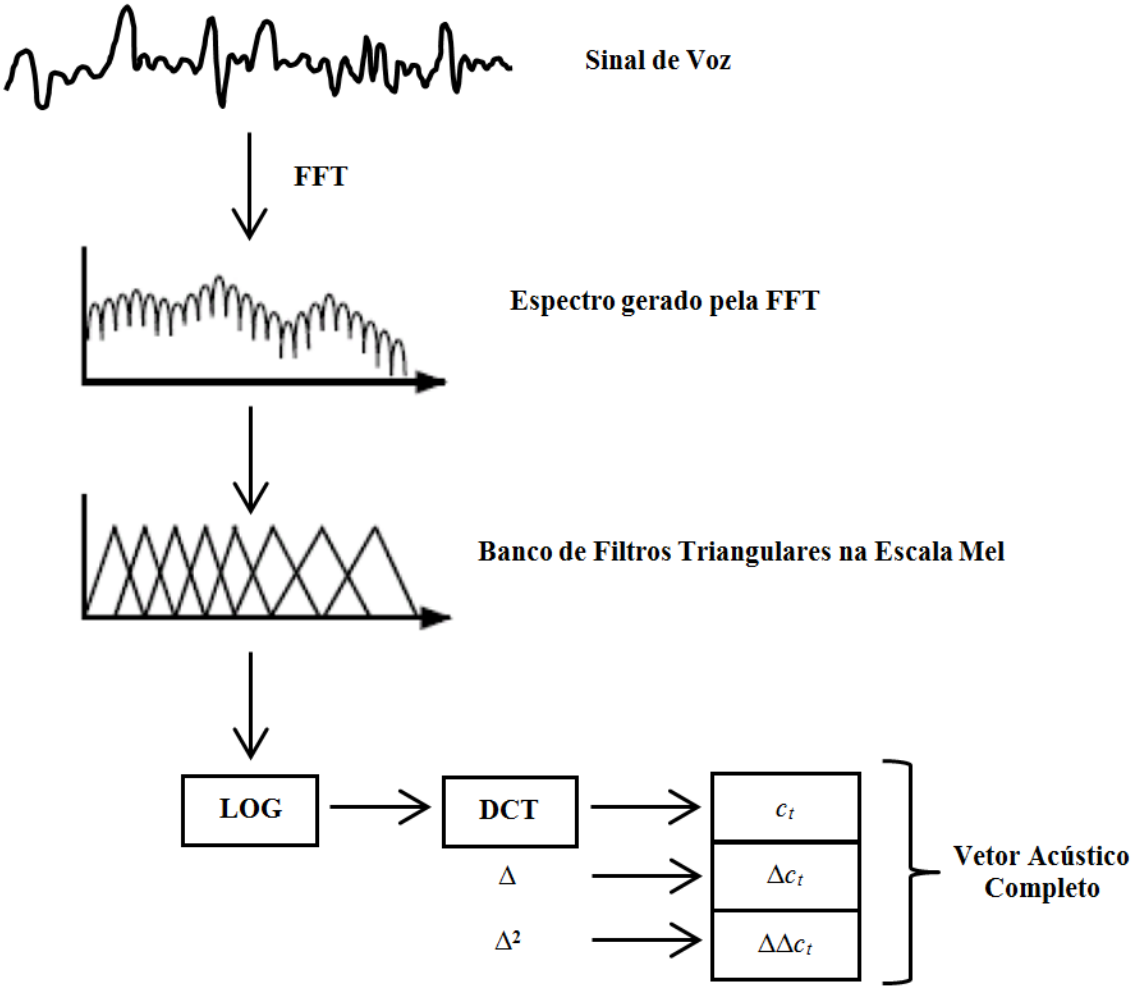


Figura 2.4: Diagrama de extração de coeficientes MFCC. Adaptado de [11]

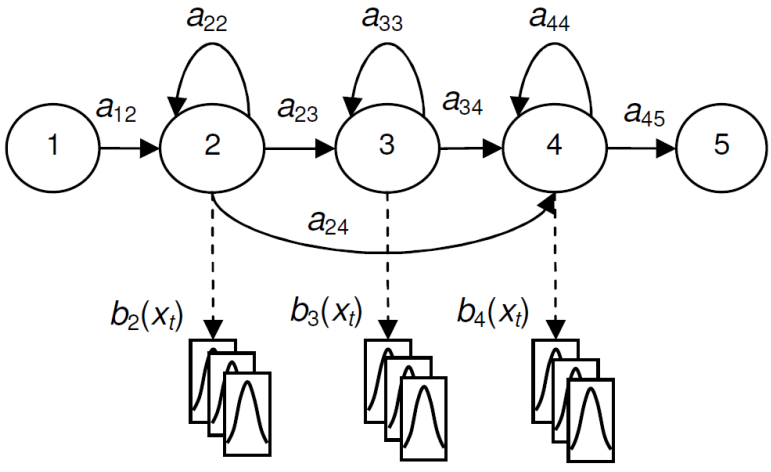


Figura 2.5: Diagrama de estados de um HMM com três estados emissores. Adaptado de [11]

$$M = A, B, \Pi = a_{ij}, b_i\left(x_t\right), \pi_i \quad , i, j = 1, ..., N \tag{2.11}$$

a_{ij} representa a probabilidade de transição do estado i para o estado j, $b_i(x_t)$ representa a probabilidade de emissão do vetor x_t em um estado i (neste caso representada por uma função densidade de probabilidade normal), π_i representa a probabilidade inicial de cada estado i.

O objetivo dessa modelagem é encontrar uma otimização de parâmetros para maximizar a verossimilhança de sequência de vetores acústicos. Essa modelagem é feita utilizando o algoritmo recursivo de reestimação de parâmetros denominado Baum-Welch [12] [13], também conhecido como *Forward-Backward*. Então, para este algoritmo é necessário definir, em termos práticos, os números mínimo e máximo de iterações do algoritmo. Além disso, é necessário definir a taxa de convergência do algoritmo. A definição da taxa de convergência aplicada pelo *toolkit* CMU Sphinx é dada por

$$convgRatio = \frac{(lkhdperframe - prevlkhd)}{absprev} \tag{2.12}$$

onde $convgRatio$ é a taxa de convergência de determinada iteração do algoritmo, $lkhdperframe$ é a verossimilhança por quadro obtida na iteração corrente, $prevlkhd$ é a verossimilhança por quadro obtida na iteração anterior e $absprev$ é o valor absoluto de $prevlkhd$.

A função de probabilidade conjunta de uma sequência de vetores X e uma sequência de estados S, dado um modelo M, é calculada de acordo com a Equação (2.13).

$$P(X, S|M) = a_{s(0)s(1)} \prod_{t=1}^{T} b_{s(t)}(x_t)\, a_{s(t)s(t+1)} \tag{2.13}$$

Como não é possível, através do modelo, saber a sequência de estados, fixa-se uma sequência de estados e calcula-se a probabilidade a priori, considerando-se todas as sequências de estados possíveis, de acordo com a Equação (2.14), somando todos os termos da Equação (2.13) para todos os estados.

$$P(X|M) = \sum_{S} P(X, S|M) \tag{2.14}$$

De forma alternativa, $P(X|M)$ pode ser aproximada pela maximização da Equação 2.13, utilizando o conhecido algoritmo eficiente para este cálculo, conhecido como Algoritmo de Viterbi. Este algoritmo é bastante útil na decodificação, onde é necessário apontar uma determinada sequência de estados mais provável para reconhecimento de uma sequência de palavras, fones ou trifones desconhecidos.

As probabilidades $b_{s(t)}(x_t)$, referentes à emissão de um vetor $x(t)$ em determinado estado são representadas por misturas de distribuições gaussianas, de acordo

com a Equação (2.15):

$$b_{s(t)}(x_t) = \sum_{m=1}^{M} c_{s(t)m} N\left(x_t; \mu_{s(t)m}, \Sigma_{s_t m}\right) \tag{2.15}$$

onde M representa o número de componentes na mistura, $c_{s(t)m}$ representa o peso do m-ésimo componente da mistura e $N(.; \mu, \Sigma)$ é uma gaussiana multivariada de acordo com a Equação (2.16):

$$N(x; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{2.16}$$

onde o vetor de médias é definido por μ, a matriz de covariâncias por Σ e o número de componentes por n. Então, para cada estado é necessário definir a quantidade de componentes da mistura gaussiana que irão compô-lo.

A fim de otimizar o modelo, utiliza-se o compartilhamento de estados para diminuir consideravelmente o número de variáveis a serem computadas. Essa técnica, chamada de *states-tying*, sugere agrupar estados que possuam as mesmas características fonéticas, ou seja, fonemas inseridos em diferentes contextos que não produzem uma variabilidade acústica suficientemente grande para serem modelados por HMM's diferentes. Estes fonemas são agrupados em um único modelo. A técnica de agrupamento de estados é realizada através de algoritmos de árvores de decisão onde, através de questões fonéticas (perguntas com a finalidade de encontrar semelhanças acústicas entre os fonemas), são agrupados os estados que caem no mesmo nó da árvore. Essas árvores de decisão podem realizar perguntas referentes ao próprio fone em questão ou relativas ao contexto da direita ou da esquerda (no caso de modelagem com trifones). Então, os fones que estiverem dentro de um mesmo nó folha serão agrupados, compartilhando um único estado. As Figuras 2.6 e 2.7 ilustram essa idéia de compartilhamento de estados através de exemplos [11]. Em termos práticos, é necessário definir o número máximo de estados compartilhados que serão permitidos, visto que mais estados compartilhados representam um sistema menos diversificado e menos estados compartilhados representam um sistema mais lento, com estados redundantes.

2.4 Modelo de linguagem

O modelo de linguagem consiste basicamente em calcular estatisticamente a ocorrência de palavras em determinadas sentenças. Esta estimação de probabilidades de ocorrência pode levar em conta o contexto no qual aparecem, ou seja, considerar sua ocorrência após uma determinada palavra ou determinado conjunto de palavras.

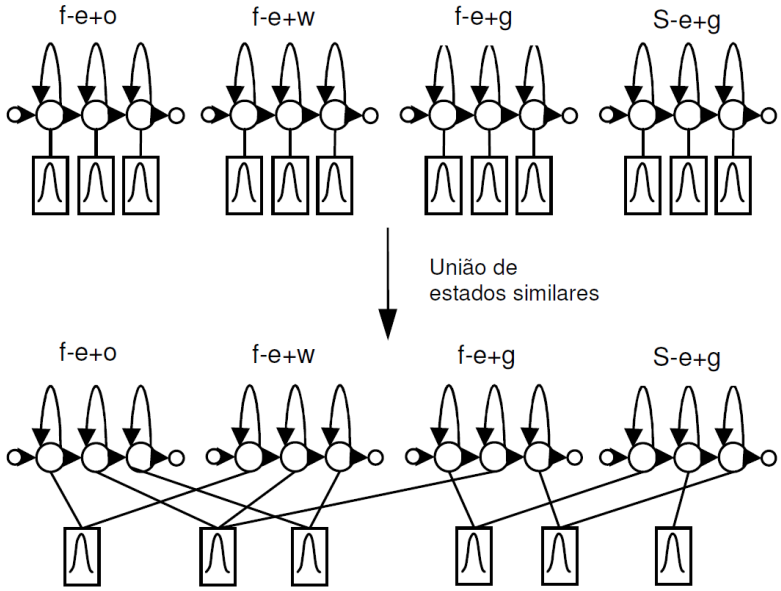


Figura 2.6: Compartilhamento de estados - *tied-states*. Estados com as mesmas características acústicas compartilham modelos para reduzir consideravelmente a quantidade de variáveis a serem determinadas. Adaptada de [11].

Considerando uma sentença W composta por um grupo de palavras, pode-se escrever a Equação (2.17):

$$\begin{aligned} P(W) &= P(w_1, w_2, ...w_n) \\ &= P(w_1)\, P(w_2|w_1)\, P(w_3|w_1, w_2) ...P(w_n|w_1, w_2, ..., w_{n-1}) \\ &= \prod_{i=1}^{n} P(w_i|w_1, w_2, ..., w_{n-1}) \end{aligned} \tag{2.17}$$

Essa técnica é denominada n-gramas, onde se presume que a ocorrência de determinada palavra w_i depende apenas das $n-1$ palavras antecessoras a ela. Ou seja, tem-se um bigrama se uma palavra depende apenas de sua antecessora para determinação de sua probabilidade ($P(w_i|w_{i-1})$) e tem-se um trigrama quando uma palavra depende somente das duas últimas palavras que a antecedem ($P(w_i|w_{i-2}, w_{i-1})$). Seguindo o mesmo raciocínio pode-se definir um unigrama como sendo unicamente a probabilidade de ocorrência de determinada palavra, independente do contexto ($P(w_i)$). Tipicamente, em sistemas de reconhecimento de voz, é utilizado o modelo trigrama.

O método mais comum para verificação do desempenho do modelo de linguagem é o *Word Error Rate* (WER), definido pela Equação (2.18), que é a taxa de erros de reconhecimento de palavras.

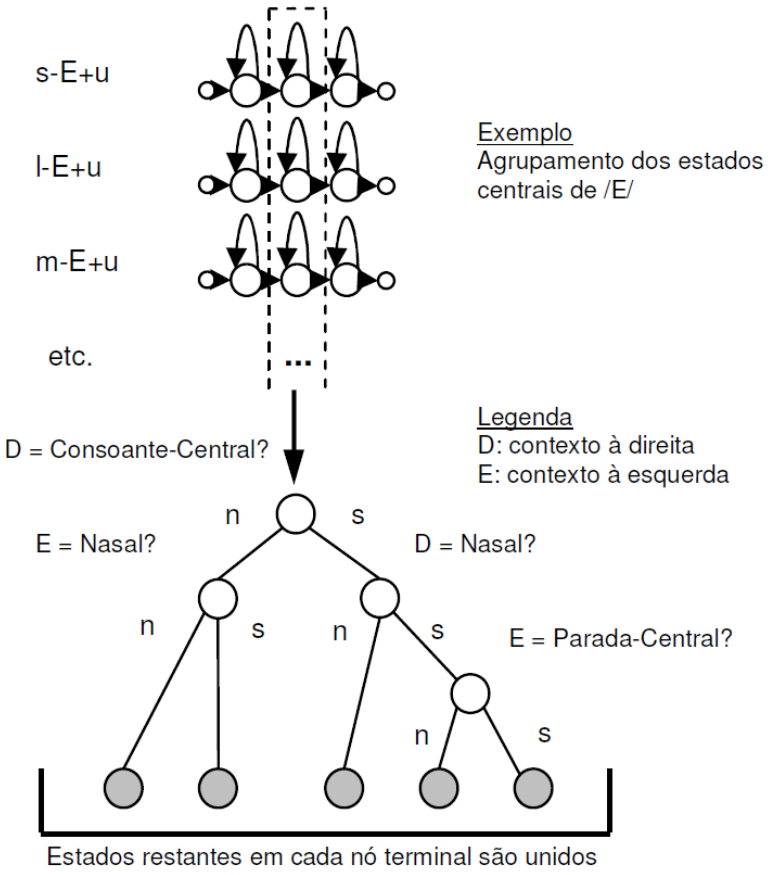


Figura 2.7: Questões fonéticas - árvore de decisão. Nós folha com as mesmas respostas para as questões fonéticas irão compartilhar o mesmo estado do HMM. Adaptada de [11].

$$WER = \frac{(sub + ins + del)}{\#palavras} * 100 \tag{2.18}$$

onde *sub* representa o número de palavras que foram substituidas erroneamente, *ins* é o número de inserções incorretas, *del* a quantidade de palavras que deveriam existir e foram excluídas e $\#palavras$ a quantidade total de palavras da sentença correta.

2.5 Decodificação

Um sistema de reconhecimento da fala transforma o sinal acústico da voz no texto correspondente. O modelo estatístico de distribuição conjunta, $P(X, W)$, onde W é a sequência de palavras pronunciadas e X uma sequência de informações acústicas observadas, busca uma estimativa $\widehat{W}$ da sequência de palavras pronunciadas a partir de características acústicas observadas X, usando a distribuição de probabilidades a *posteriori*, $P(W|X)$, utilizando a Equação (2.19).

$$\widehat{W} = \arg_w \max \left[P\left(W|X\right) \right] = \arg_w \max \left[\frac{P\left(W\right) P\left(X|W\right)}{P\left(X\right)} \right] \tag{2.19}$$

Após a aplicação do Teorema de Bayes [14] na Equação (2.19), $P\left(W|X\right)$ é decomposta em $P\left(W\right)$ (probabilidade a *priori* da sequência de palavras W - fornecida pelo modelo de linguagem) e $P\left(X|W\right)$ (probabilidade de observar a característica acústica X quando a sequência de palavras W é pronunciada - fornecida pelo modelo acústico).

O objetivo é encontrar, dentre todas as possíveis sequências de palavras W, a melhor estimativa que maximize $P\left(W|X\right)$.

Na Equação (2.19) pode-se omitir o termo $P\left(X\right)$ presente no denominador, já que, para todos os conjuntos de palavras testados, esse termo é comum.

Para encontrar a estimativa $\widehat{W}$ de sentenças pronunciadas mais provável, utiliza-se o algoritmo de Viterbi. Este é um algoritmo de busca síncrona que busca o estado mais provável a cada unidade de tempo. Então, conforme dito na Seção 2.3, ao invés de realização do somatório da Equação (2.14), é realizada a busca pela sequência de estados que maximize a função.

A decodificação pode se tornar um processo lento com o aumento do vocabulário existente no sistema, gerando espaços de busca muito grandes. O processo de decodificação é basicamente o único responsável pelo custo computacional de um sistema de reconhecimento de voz. Para melhorar o desempenho das buscas, o decodificador utiliza a busca em feixe, ou seja, armazena caminhos ótimos intermediários na busca, sem a necessidade de computá-los a cada novo caminho que os utilize, e poda determinados nós abaixo de um certo limiar, a fim de eliminar buscas por caminhos improváveis.

No CMU Sphinx, existem quatro variáveis que influenciam no desempenho do decodificador em termos de gasto de memória e velocidade de processamento e estas podem ser ajustadas de maneira a otimizar o sistema, seja em desempenho computacional, seja em aumento da acurácia no reconhecimento. São elas:

- *absoluteBeamWidth*: valor que define o tamanho da lista ativa, que é composta pelas densidades gaussianas (ou nós) que devem ser computadas pelo decodificador;

- *relativeBeamWidth*: valor definido em relação ao nó de maior pontuação a cada instante, ou seja, pontuação mínima que deve ser atingida por um nó para que ele possa permanecer na lista ativa;

- *languageWeight*: valor utilizado na decisão de importância relativa dada às probabilidades acústicas das palavras na hipótese;

- *wordInsertionPenalty*: valor utilizado como penalidade dada a uma nova palavra encontrada durante a busca.

O desempenho de processamento do decodificador pode ser medido pelo *Real Time Factor* (RTF), descrito pela Equação (2.20), que nada mais é que o tempo médio de processamento de uma sentença.

$$RTF = \frac{tempo\_processamento\_total}{tempo\_audio\_total} \tag{2.20}$$

Uma medida qualitativa de desempenho do decodificador é a chamada acurácia (ACC). A acurácia pode ser definida de acordo com a Equação (2.21):

$$ACC = \frac{\#palavras\_corretas}{\#palavras} * 100 \tag{2.21}$$

onde $\#palavras\_corretas$ representa o número de palavras reconhecidas corretamente e $\#palavras$ representa o número total de palavras da sentença de referência.

Capítulo 3

Implementação

Este capítulo contém uma descrição detalhada da implementação do sistema de reconhecimento da fala e da interface de teste e análise. Primeiramente, na Seção 3.1, é abordada a implementação do sistema de reconhecimento da fala utilizando o CMU Sphinx. Na Seção 3.2 será tratada a construção da ferramenta de testes e análise desde a implementação da interface até a comunicação com o sistema do CMU Sphinx.

3.1 CMU Sphinx

Conforme dito no Capítulo 1, o CMU Sphinx é um *toolkit* que implementa o método de reconhecimento da fala utilizando HMM's. Esta ferramenta proporciona o controle sobre diversos parâmetros necessários para configuração de um sistema de reconhecimento. A versão do decodificador utilizada neste trabalho foi o Sphinx-4. Esta versão é escrita na linguagem de programação JAVA, sendo, entretanto, multiplataforma.

O CMU Sphinx possui algumas ferramentas específicas para as diferentes etapas da criação do sistema de reconhecimento da fala. O SphinxTrain é a ferramenta utilizada para treinamento do modelo acústico. Dentro do SphinxTrain existe a ferramenta para extração de características do sinal de áudio, o chamado sphinx_fe. O CMUclmtk (*CMU-Cambridge Language Model Toolkit*) é a ferramenta utilizada para o treinamento do modelo de linguagem. O Sphinx-4, na realidade, consiste apenas no decodificador do sistema de reconhecimento, escrito em JAVA.

3.1.1 Preparação de dados e arquivos externos

Alguns arquivos externos utilizados pelo CMU Sphinx para treinamento do modelo acústico e de linguagem precisam estar no padrão definido por ele.

Para o treinamento acústico são necessários os arquivos de áudio (arquivos no formato PCM *MS Wav*), o arquivo com os nomes dos arquivos de áudio (arquivo texto com extensão `.fileids`), o arquivo de transcrições (arquivo texto com extenção `.transcription`) contendo os nomes dos arquivos de áudio com suas respectivas transcrições, o arquivo de dicionário (arquivo texto com extenção `.dic`) contendo um dicionário de palavras com suas respectivas transcrições fonéticas e o arquivo de fones (arquivo texto com extenção `.phone`) contendo a lista de todos os fones utilizados no sistema de reconhecimento a ser treinado. A Tabela 3.1 contém a lista de fonemas do Português Brasileiro com seus símbolos e exemplos [15]. Caso o desenvolvedor opte por utilizar questões fonéticas próprias para o agrupamento de estados (*tied-states*), ele deverá carregar este arquivo (arquivo texto com extenção `.tree_questions`). Caso o desenvolvedor opte por não utilizar questões fonéticas próprias, o sistema é capaz de gerar sozinho uma lista padrão de questões fonéticas para agrupamento de estados.

Para o treinamento do modelo de linguagem são necessários o arquivo compreendendo a base de texto (arquivo texto com extensão`.txt`) contendo uma frase por linha do arquivo e o arquivo de dicionário de palavras com suas respectivas transcrições fonéticas. Assim como no modelo acústico, este arquivo tem extensão `.dic` e contém todas as palavras que serão reconhecidas pelo sistema. Pode ser utilizado o mesmo arquivo de dicionário usado no treinamento do modelo acústico.

No Apêndice A são dados alguns exemplos destes arquivos externos.

3.1.2 Extração de parâmetros

O CMU Sphinx possibilita a extração dos coeficientes MFCC dos arquivos de áudio. A ferramenta utilizada para extração das características, tanto para o treinamento quanto para os testes, foi a sphinx_fe devido à sua flexibilidade de ajustes de parâmetros [4]. Os seguintes parâmetros podem ser ajustados através do arquivo de configuração `feat.params`, na ferramenta para a extração de características:

- Tipo de arquivo (*MS WAV* ou *RAW*);
- Taxa de amostragem;
- Parâmetro de pré ênfase - Alfa;
- Tamanho da janela;
- *Frame Rate*;
- Número de pontos da FFT;
- Número de filtros no banco de filtros;

Tabela 3.1: Lista com os fonemas do Português Brasileiro com símbolos e exemplos. Adaptada de [15]

SÍMBOLO	EXEMPLOS
VOGAIS ORAIS (7)	
a	lápis, jatobá, ábaco, capacete, cabeça, caça, lua, pedia
E	é, médico, pajé, épico, Pelé, pele, ferro, velho
e	capacete, resolver, respeito
i	justiça, pais, saia, lápis, idiota, aqueles, ele, pele
O	ópio, cópia, jogos, docas, sozinho, forte
o	resolver, jogo, golfinho, bolo, cor
u	baiacu, Raul, culpa, baú, cururu, logo, consolo, tijolo
VOGAIS NASAIS (5)	
a~	avião, campeão, andar, tampar, canção, cama
e~	então, consciência, tempo, bem, menos, dente
i~	ninho, tinta, latina, importa
o~	onda, campeões, somos, homem, fronha
u~	um, muito, umbigo
SEMI-VOGAIS (4)	
w	natal, fácil, voltar, eu, chapéu, quase, jaula
j	fui, pai, sei, foi, caracóis, hotéis, micróbio, pátria
w~	não, cão
j~	muito, bem, parabéns, compõe
FRICATIVAS NÃO VOZEADAS (3)	
f	festa, fanfarrão, afta, afluente
s	sapo, caçar, crescer, sessão, lápis, capaz, disco, casca, desço, excesso
S	chá, xaveco, cachorro
FRICATIVAS VOZEADAS (3)	
z	casa, coisa, quase, exato
v	vovó, vamos, avião
Z	geladeira, trovejar
AFRICATIVAS (2)	
tS	tia, pacote, constituinte, Tijuca
dZ	dia, cidade, disco
PLOSIVAS (6)	
b	barba, absinto
d	dados, cidade, dominar
t	todos, pato, constituinte
k	casa, casca, quero, quanto
g	guerra, gato, aguentar, agnóstico
p	papai, psicológico, apto, rapto
LÍQUIDAS (5)	
l	laranja, palafita, leitão
L	calhar, colheita, melhor
R	carro, rua, rato, germe
X	casar, certo, harpa, arco
r	carona, garoto, frango, graxa, por exemplo
CONSOANTES NASAIS (3)	
m	mamãe, ema, emancipar, marmota
n	nome, atenuar, encanação
J	casinha, galinha
SILÊNCIOS (2)	
sil	silêncio do início / fim de uma locução
sp	silêncio no interior de uma locução (pausa)

- Frequência mínima do banco de filtros;
- Frequência máxima do banco de filtros;
- Número de canais do banco de filtros;
- Tipo de transformada (DCT, HTK ou Legacy);
- Número de coeficientes MFCC;
- Coeficiente de Liftragem Cepstral;
- Remoção do nível DC (sim ou não).

3.1.3 Treinamento do modelo acústico

O treinamento do modelo acústico no CMU Sphinx é realizado pela ferramenta denominada SphinxTrain. Essa ferramenta é composta por ferramentas menores que realizam cada uma das etapas do treinamento.

Primeiramente, deve ser ajustado o arquivo de configuração `sphinx_train.cfg` com os parâmetros utilizados no treinamento do modelo acústico. Os parâmetros a serem ajustados neste arquivo são os seguintes:

- Número mínimo de iterações;
- Número máximo de iterações;
- Taxa de convergência;
- Número de componentes da mistura gaussiana do modelo;
- Número de estados compartilhados;
- Utilização de questões de fonéticas próprias (sim ou não).

O primeiro passo do treinamento acústico consiste no treinamento dos modelos independentes de contexto. Ele consiste basicamente no treinamento da matriz B da Equação (2.11). Este processo gera a inicialização dos parâmetros dos modelos através de estimação a partir dos vetores acústicos obtidos na extração de parâmetros. Após a inicialização, são realizadas uma série de iterações do algoritmo de Baum-Welch até que se atinja uma determinada taxa de convergência (*convgRatio* da Equação (2.12)) [16]. Deve-se tomar cuidado para que não sejam realizadas muitas iterações, com a possibilidade de ocorrer *overfitting*. O conjunto de *scripts* `20.ci_hmm` realiza esta etapa do treinamento.

O próximo passo é composto pelo treinamento dos modelos dependentes do contexto, ou seja, o treinamento dos trifones. A inicialização destes parâmetros consiste em utilizar os mesmos parâmetros determinados na etapa anterior, para posteriormente serem realizadas novas iterações para reestimação de parâmetros. Para esta etapa, o conjunto de *scripts* `30.cd_hmm_untied` é o responsável. Nesta etapa são treinados os parâmetros da matriz A da Equação 2.11. Nota-se que nesta etapa ainda não existem estados compartilhados.

O terceiro passo do treinamento acústico é justamente a geração da árvore de questões fonéticas para agrupamento de estados compartilhados e o próprio agrumento de estados em si. O CMU Sphinx é capaz de gerar as questões fonéticas automaticamente caso o projetista não deseje utilizar questões próprias. O conjunto de *scripts* `40.buildtrees` é responsável por esta etapa do treinamento. Juntamente com esta etapa, tem-se a utilização do conjunto de *scripts* `45.prunetree`, que realiza uma poda na árvore das questões fonéticas com a finalidade de eliminar nós para que seja atingido o número máximo de estados compartilhados definido pelo projetista e agrupar os estados compartilhados.

Como última etapa do treinamento acústico, realiza-se o re-treinamento de todos os estados, dessa vez, compartilhados. Este processo, realizado pelo conjunto de *scripts* `50.cd_hmm_tied`, distribui as gaussianas definidas no arquivo de configuração e retreina, recursivamente, todos os modelos.

3.1.4 Treinamento do modelo de linguagem

O treinamento do modelo de linguagem do CMU Sphinx é realizado pelo conjunto de ferramentas denominado CMUclmtk. Primeiramente, a ferramenta `text2idngram` calcula, a partir do arquivo de frases e do arquivo de vocabulário, as frequências de ocorrências das palavras e seus relacionamentos. A partir deste momento, é usada a ferramenta `idngram2lm` para gerar, finalmente, o arquivo com extensão `.arpa` contendo o modelo n-grama (definido pelo projetista) do modelo de linguagem.

3.1.5 Empacotamento do modelo acústico

O empacotamento do modelo acústico consiste basicamente em unir todos os modelos e parâmetros num arquivo compactado (com extensão `.jar`) para que o sistema possa ser utilizado. Para isso é necessário mover determinados arquivos gerados dentro das ferramentas do treinamento do modelo acústico e do treinamento do modelo de linguagem para diretórios específicos e o preenchimento de um arquivo de configuração com as definições dos modelos utilizados. Além disso, é necessário ajustar alguns parâmetros e referenciar os arquivos de dicionário e do modelo de linguagem num arquivo XML. Um detalhamento maior sobre esta etapa pode ser

encontrado no Apêndice B.

3.1.6 Decodificação

O decodificador utilizado neste projeto foi o Sphinx-4, o qual é dividido em diversos componentes. A escolha de quais componentes utilizar, assim como suas respectivas versões, é feita através do arquivo de configuração XML pelo *configurationManager*. Um exemplo de arquivo de configuração para o decodificador está no Apêndice A.

Conforme a Figura 3.1, depois que o sistema de reconhecimento é carregado, o *Recognizer* constrói o *Front-End*, que processa os dados de entrada, no caso os coeficientes MFCC extraídos dos arquivos de áudio de teste. Para este trabalho utiliza-se um padrão de testes com arquivos de áudio pré-gravados (*offline*) e extração de características externas à aplicação. O componente *Linguist* carrega os modelos acústico e de linguagem e o dicionário fonético para que o grafo de busca seja construído. O componente *Decoder* construirá o *Search Manager* que, por sua vez, construirá o *Active List*, que representa a lista de nós ativos durante a busca, o *Scorer*, que pontua cada nó e o *Pruner* que poda a lista ativa.

3.2 Sistema de análise e testes e interface gráfica

O sistema de interface gráfica para análise e testes proposto por este trabalho consiste em proporcionar e facilitar testes de um sistema de reconhecimento da fala, baseado no *toolkit* CMU Sphinx, sem qua haja a necessidade de lidar diretamente com arquivos de configuração, arquivos intermediários e execução direta de comandos. O objetivo é justamente focar a análise de testes e resultados, somente com ajustes dos parâmetros do reconhecimento, sem necessidade de dedicação à programação e aprendizado dos detalhes da ferramenta CMU Sphinx.

O sistema desenvolvido neste projeto foi escrito na linguagem PHP (acrônimo recursivo de *PHP Hypertext Preprocessor*, originalmente *Personal Home Page*) mesclada com *HyperText Markup Language* (HTML) para geração das páginas da interface. A utilização destas linguagens foi estimulada por serem compatíveis com praticamente todos os navegadores disponíveis no mercado, além de serem linguagens multiplataformas.

O sistema de interface desenvolvido pode ser dividido basicamente em 6 etapas. São elas:

- Interface Principal;
- Treinamento do Modelo Acústico;
- Treinamento do Modelo de Linguagem;

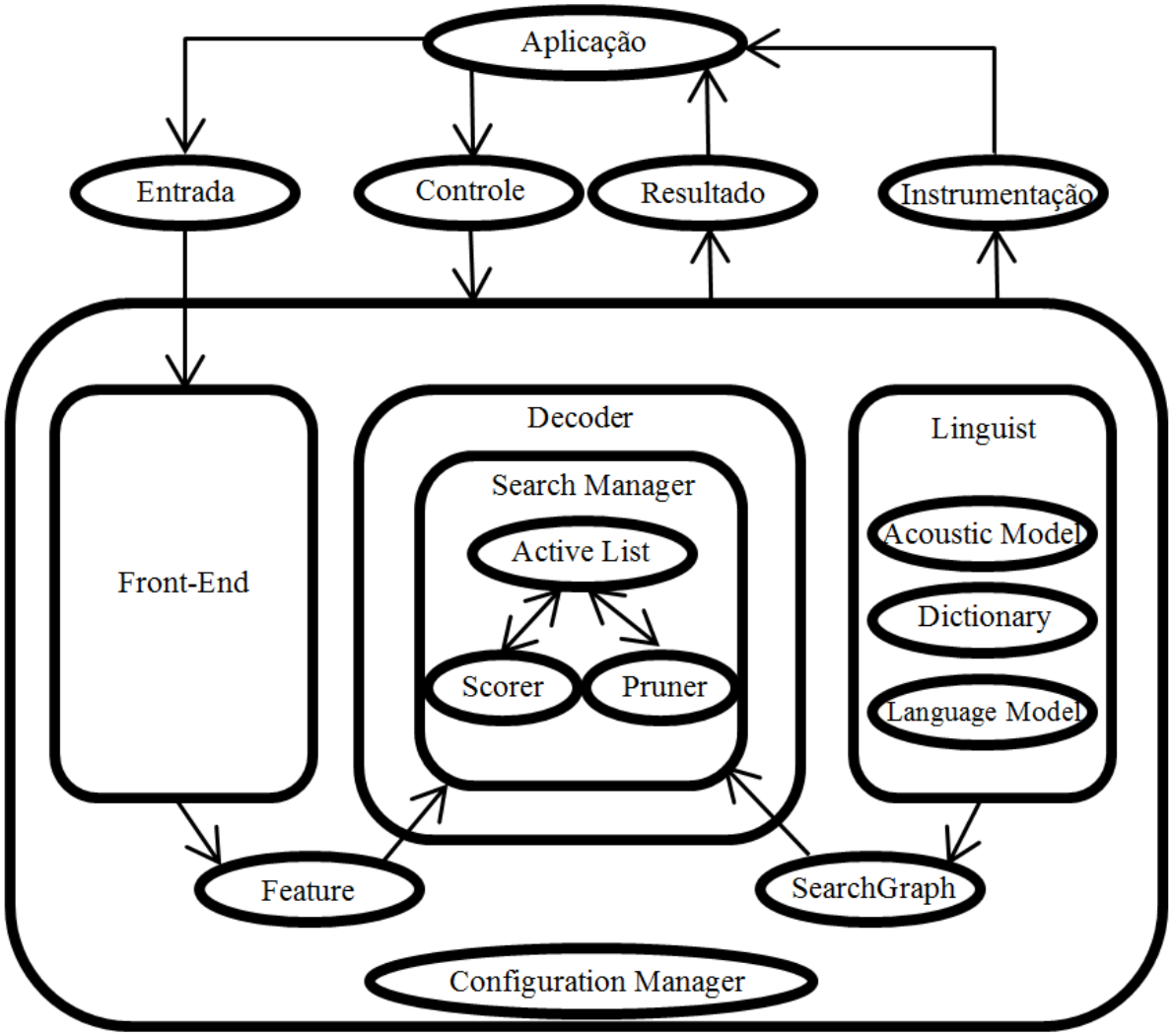


Figura 3.1: Decodificador Sphinx-4. Adaptada de [17].

- Testes;
- Limpar Sistema;
- Ajuda.

Cada uma destas etapas serão detalhadas nas seções seguintes.

3.2.1 Interface principal

A interface principal nada mais é do que a primeira página disponível ao usuário que contém o cabeçalho do projeto e o menu de opções sobre qual etapa selecionar. Esta página é composta pelo *frame* superior, contendo o título e os *links* de referência, pelo *frame* lateral esquerdo, que contém o menu com as opções das etapas do sistema e o *frame* central que contém, na página principal, o cabeçalho do projeto e que irá conter todas as telas intermediárias do sistema.

A Figura 3.2 mostra a Interface Principal.

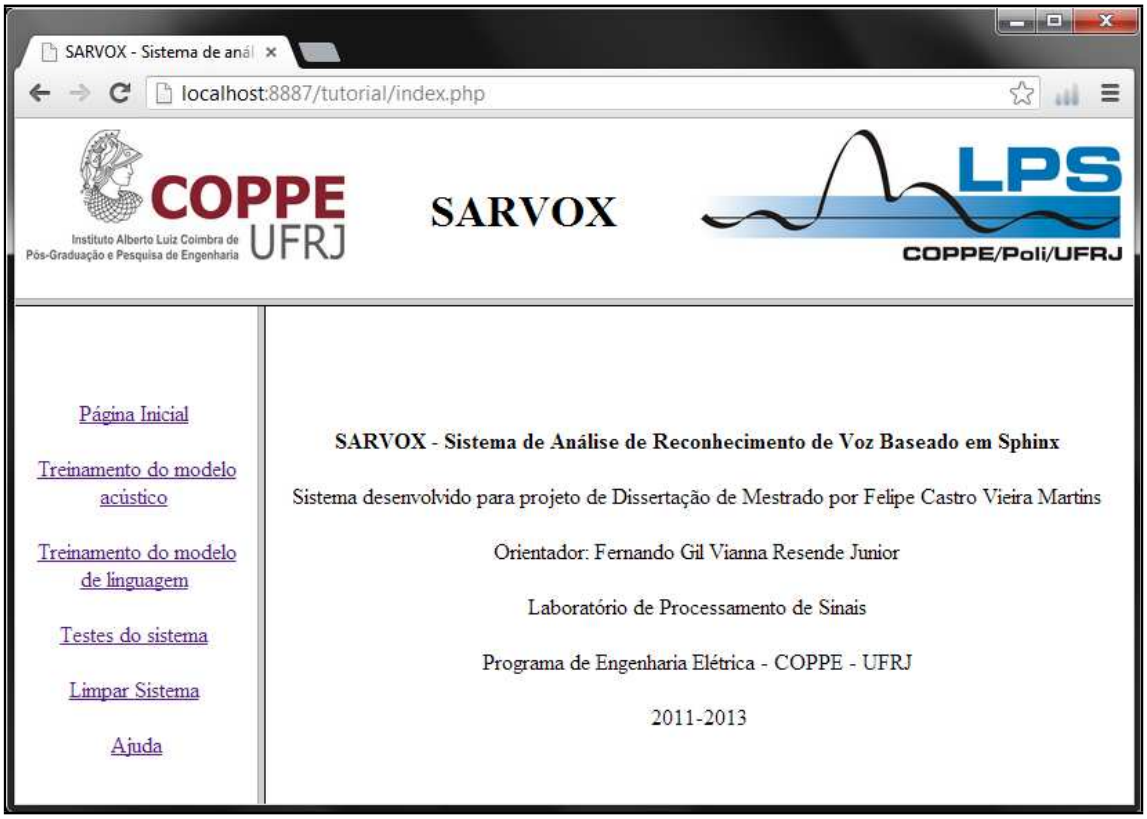


Figura 3.2: Interface principal

Esta interface principal consiste na elaboração de uma página HTML com 3 *frames*.

Todo o sistema criado neste projeto foi desenvolvido na plataforma Microsoft Windows, utilizando o navegador Google Chrome para testes.

3.2.2 Treinamento do modelo acústico

A etapa do treinamento do modelo acústico do sistema de análise e testes é composta por diversas páginas HTML e *scripts* PHP. Esta etapa realiza todo o processo do treinamento acústico do sistema de reconhecimento da fala, como o *upload* dos arquivos externos e de áudio, extração de características, treinamento do modelo acústico propriamente dito e empacotamento do modelo acústico.

Primeiramente, na primeira página do treinamento acústico, são solicitados os *uploads* dos arquivos externos necessários. São os arquivos de áudio, de transcrições, de dicionário e de fones. O *script* `uploads_treinamento_acustico.php` move, então, estes arquivos para seus respectivos diretórios, cria o arquivo `.fileids` automaticamente com os nomes dos arquivos de áudio utilizados e gera a próxima página, de extração de características.

Para a extração de características é solicitado o preenchimento do formulário contendo todos os parâmetros necessários. Por padrão, o formulário já se encontra previamente preenchido com os valores indicados em [4] de acordo com a Tabela 3.2. Após o envio deste formulário, o *script* `parametros_treinamento_acustico.php` cria o arquivo de configuração da extração de características (`feat.params`) e executa o comando de extração através de arquivo `.bat` de execução de comandos externos. Os arquivos com os coeficientes MFCC gerados para cada arquivo de áudio ficam, por padrão, no seu diretório de utilização.

Tabela 3.2: Tabela de parâmetros padrões para extração de características.

PARÂMETRO	**VALOR**
Tipo de Arquivo	MS Wav
Taxa de Amostragem	48000 Hz
Alfa - Pré Ênfase	0.97
Tamanho da Janela	0.025 s
Frame Rate	100 Janelas / s
Número de Pontos da FFT	2048
Número de Filtros no Banco	40
Frequência Mínima do Banco de Filtros	1 Hz
Frequência Máxima do Banco de Filtros	24000 Hz
Número de Canais	26
Tipo de Transformada	htk
Número de Coeficientes MFCC	13
Coeficiente de Liftragem	22
Remover Nível DC	Sim

Após a extração de características, tem-se o início do treinamento do modelo acústico propriamente dito. É solicitado, através de formulário, o ajuste dos parâmetros necessários para o treinamento acústico e o *upload* do arquivo de questões fonéticas caso o usuário deseje usar questões fonéticas próprias. Existem duas opções de envio deste formulário, seguido pela execução de dois *scripts* diferentes. A primeira opção realiza o treinamento somente da primeira etapa do treinamento do modelo acústico, necessitando que o usuário execute as etapas subsequentes sob seu comando. A segunda opção realiza o treinamento acústico completo, sendo executados todos os *scripts* do treinamento de maneira automática. O objetivo de permitir as duas formas de treinamento é que o usuário analise, etapa por etapa, se houve algum erro durante o treinamento ou que execute todo o treinamento através de um único comando.

O treinamento do modelo acústico passo a passo se inicia com a execução do *script* `treinamento_acustico.php`, que gera automaticamente o arquivo de configuração `sphinx_train.cfg` com os parâmetros setados pelo formulário e executa,

através de arquivo `.bat`, o treinamento dos fones independentes de contexto. Os valores destes parâmetros são previamente preenchidos de acordo com a Tabela 3.3, indicados em [4]. Posteriormente, é necessário que o usuário siga pelas etapas subsequentes (treinamento dos fones dependentes de contexto, geração de estados compartilhados e retreinamento com os estados compartilhados) através de seus comandos. Todas as etapas são individualmente realizadas através de arquivo `.bat`, executando comandos externos e imprimindo na tela os resultados de cada etapa do treinamento retornado por linha de comando pelo CMU Sphinx. Na última etapa, o sistema gera automaticamente o arquivo `.bat` do empacotamento do modelo acústico, o qual depende dos parâmetros escolhidos para geração do comando externo.

Tabela 3.3: Tabela de parâmetros padrões para o treinamendo do modelo acústico.

PARÂMETRO	**VALOR**
Número Mínimo de Iterações	1
Número Máximo de Iterações	5
Taxa de Convergência	0.001
Número de Gaussianas do Modelo	16
Número de Estados Compartilhados	300

O treinamento completo do modelo acústico consiste na execução do *script* `treinamento_acustico_completo.php`. Este *script* realiza todas as etapas do treinamento acústico, executando todos os arquivos `.bat` sequencialmente de forma automática, sem exibir os resultados das etapas intermediárias retornadas pelo CMU Sphinx.

Após a execução do treinamento acústico propriamente dito, tanto por passo a passo quanto pela maneira completa, é necessário que o usuário defina os parâmetros referentes à execução do sistema de reconhecimento da fala. Estes parâmetros definirão o desempenho do sistema em termos de custo × benefício. A Tabela 3.4 contém os valores indicados em [4] destes parâmetros, que são preenchidos previamente de forma padrão. O *script* `empacotamento_acustico.php` realizará o empacotamento do modelo acústico, gerando o arquivo `.jar` contendo todos os modelos acústicos encapsulados.

Tabela 3.4: Tabela de parâmetros padrões para a execução de testes do sistema.

PARÂMETRO	**VALOR**
$absoluteBeamWidth$	7000
$relativeBeamWidth$	1E-109
$languageWeight$	6.0
$wordInsertionPenalty$	0.0001

Terminado o empacotamento do modelo acústico, o processo de treinamento do modelo acústico está encerrado.

A Figura 3.3 ilustra o processo de treinamento acústico realizado pela interface.

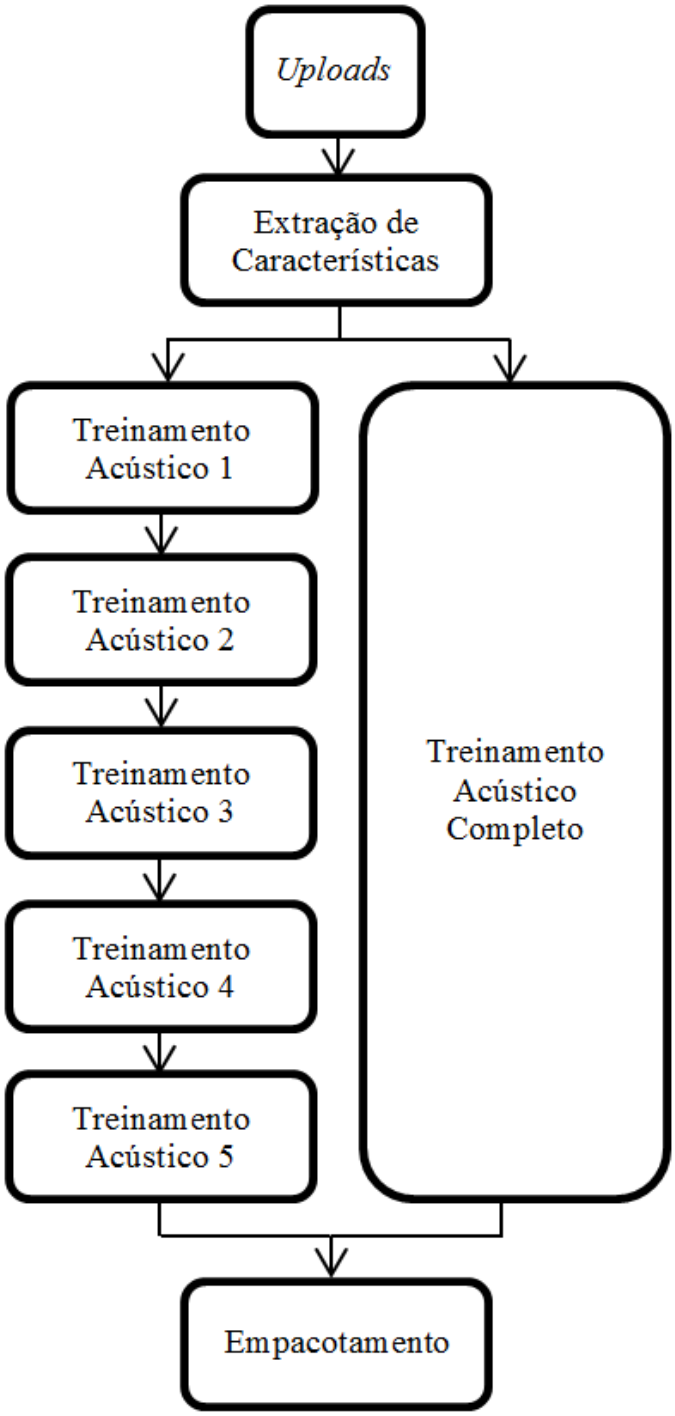


Figura 3.3: Interface - diagrama do treinamento do modelo acústico

Cada uma das cinco etapas do treinamento do modelo acústico propriamente dito corresponde a cada uma das etapas citadas na Seção 3.1.3.

3.2.3 Treinamento do modelo de linguagem

O treinamento do modelo de linguagem pelo sistema de interfaces é composto, primeiramente, pelo *upload* dos arquivos necessários para geração do modelo de linguagem (arquivo com a base de frases e arquivo de dicionário) e a escolha do tipo do modelo: unigrama, bigrama, trigrama ou quadrigrama. O *script* `uploads_treinamento_linguagem.php` move os arquivos para seus respectivos diretórios e gera a página seguinte, onde o usuário define se deseja realizar o treinamento completo do modelo de linguagem ou se deseja realizar o treinamento passo a passo.

O treinamento passo a passo contém somente dois passos e é realizado da mesma maneira que o treinamento do modelo acústico. É solicitada ação do usuário para cada passo do treinamento. O primeiro passo consiste no processamento dos dados de entrada com a geração das estatísticas e o segundo passo consiste na geração do modelo de linguagem. Após cada um dos passos, é possível analisar o resultado do treinamento retornado pelo CMU Sphinx. O *script* `treinamento_linguagem.php` cria e executa o arquivo `.bat` com base no modelo n-grama determinado pelo usuário. O *script* `treinamento_linguagem2.php` executa o segundo arquivo `.bat` na segunda etapa do treinamento, gerando, por fim, o arquivo `.arpa` contendo o modelo de linguagem.

O treinamento do modelo de linguagem completo é realizado pelo *script* `treinamento_linguagem_completo.php` com a criação e execução dos dois arquivos `.bat` necessários para as duas etapas do treinamento. Neste modo não é possível vizualizar os resultados intermediários retornados pelo CMU Sphinx.

Por padrão, o tipo de modelo de linguagem definido pelo sistema é o trigrama.

A Figura 3.4 ilustra o treinamento do modelo de linguagem realizado pela interface.

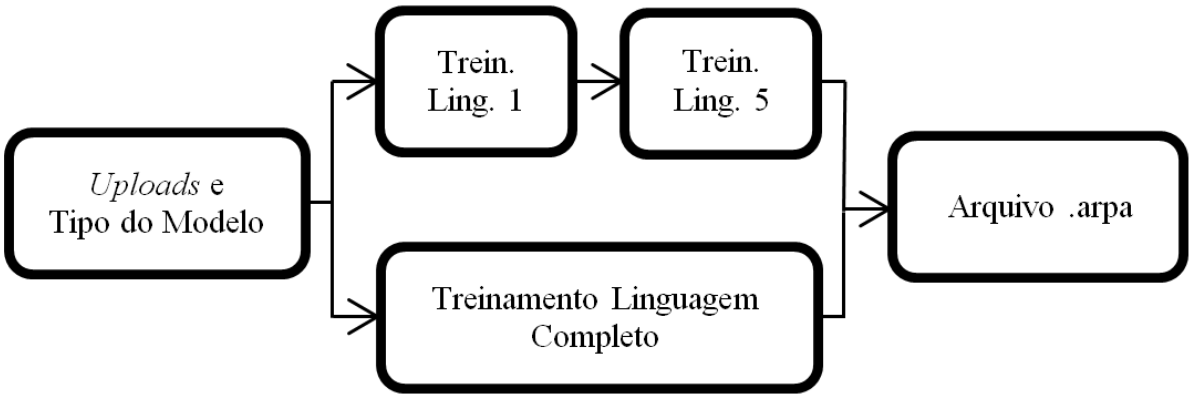


Figura 3.4: Interface - diagrama do treinamento do modelo de linguagem

Cada uma das duas etapas do treinamento do modelo de linguagem corresponde a cada uma das etapas citadas na Seção 3.1.4.

3.2.4 Testes

O sistema de testes só deve ser executado após a realização do treinamento do modelo acústico e do modelo de linguagem.

A primeira etapa do sistema de testes é a escolha do tipo de teste. Existem dois tipos básicos de testes: o teste com arquivo único, ou seja, o teste de reconhecimento com um único arquivo de áudio, e o teste com arquivos múltiplos, com a finalidade de geração de estatísticas de reconhecimento.

Caso a opção de teste com arquivo único seja escolhida, o usuário deverá realizar o *upload* do arquivo de áudio a ser reconhecido. Pode-se, então, escolher se o sistema retornará somente o resultado simples do reconhecimento ou se ele deve retornar uma estatística de reconhecimento deste arquivo único. Se essa opção for selecionada, o usuário deverá informar previamente a transcrição do arquivo de áudio através do formulário com a caixa de texto.

O *script* `teste_unico_simplificado.php` executa os arquivos `.bat` de extração de características (utilizando o mesmo arquivo de configuração gerado no processo de treinamento do modelo acústico) e de execução do sistema. O sistema é, então, carregado e o reconhecimento é feito, retornando ao usuário somente a transcrição do áudio solicitado.

O *script* `teste_unico_completo.php` age da mesma maneira que o anterior. A única diferença fica na exibição do resultado, onde, além da transcrição do áudio solicitado, o sistema mostra estatísticas referentes ao tempo de processamento, memória gasta, Acurácia, RTF e WER (caso a transcrição tenha sido informada na caixa de texto).

Caso a opção de teste com arquivos múltiplos seja escolhida, o processo se torna ligeiramente diferente. É necessário que o usuário faça o *upload* dos arquivos de áudio dos quais se deseja obter o reconhecimento e o arquivo contendo todas as transcrições dos respectivos áudios. Este arquivo deve seguir o padrão descrito na Seção A.6 do Apêndice A. Neste tipo de teste, o envio das transcrições (em forma de arquivo) não é facultativa, como é no teste simplificado para arquivo único, pois o objetivo de realização dos testes com arquivos múltiplos é levantar estatísticas sobre o sistema.

O *script* `teste_varios.php` move os arquivos de áudio e de transcrição para seus respectivos diretórios e cria o arquivo `.fileids` automaticamente, baseado nos nomes dos arquivos de áudio. Além disso, este *script* também executa o arquivo `.bat` para extração de características da mesma maneira que os anteriores. Então, o *script* `teste_varios2.php` executa o arquivo `.bat` de teste, carrega o sistema de reconhecimento e mostra, ao final do reconhecimento, uma página contendo o resultado de todas as transcrições de todos os arquivos solicitados, além de todas as

estatísticas citadas anteriormente para cada arquivo de áudio individualmente e um resumo contendo a estatística média de todos os arquivos.

A Figura 3.5 ilustra o processo de testes.

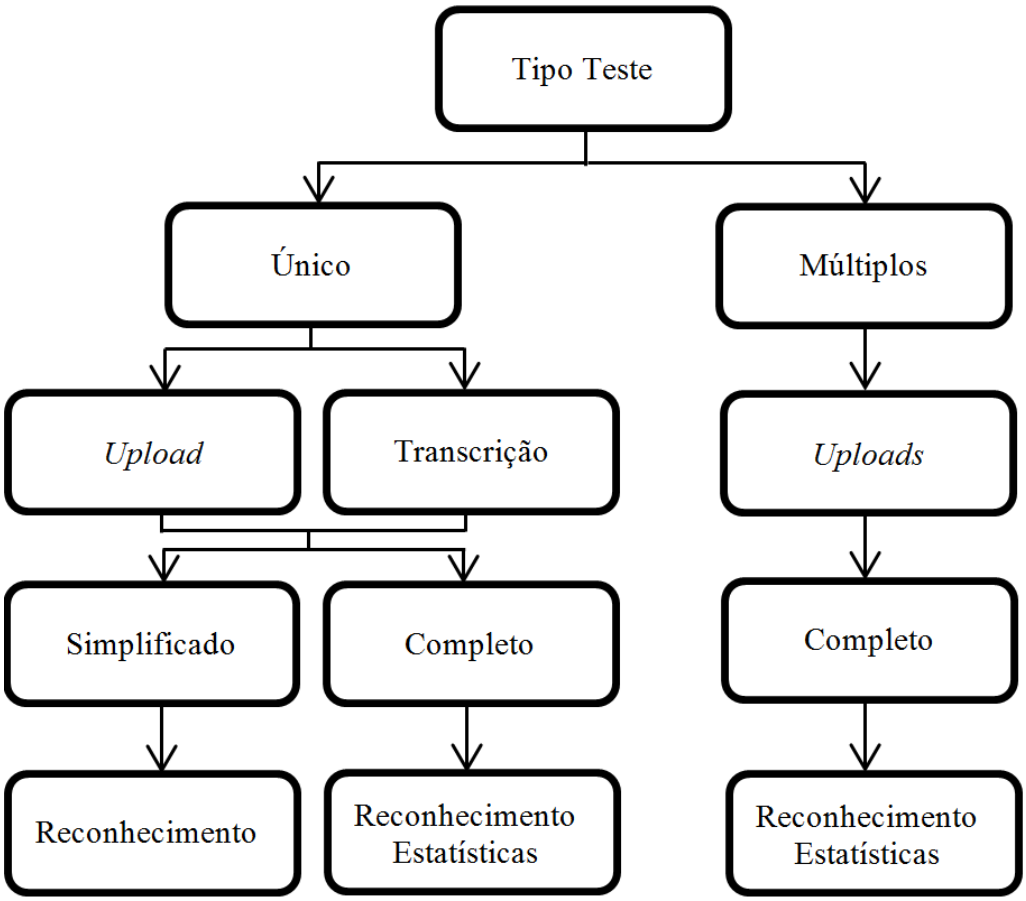


Figura 3.5: Interface - diagrama de testes

3.2.5 Limpar sistema

A ferramenta de limpeza do sistema consiste em eliminar todos os arquivos transmitidos para o sistema de reconhecimento via *uploads* e todos os arquivos intermediários criados durante o processo. O *script* `limpar.php` executa o arquivo `.bat` de limpeza, que executa todos os comandos relativos à exclusão dos arquivos e à limpeza do sistema.

O objetivo desta ferramenta é garantir que, a cada novo sistema de reconhecimento a ser montado, não se possua qualquer resquício de sistemas previamente testados. Recomenda-se que antes da execução de qualquer novo treinamento se execute a ferramenta de limpeza do sistema.

3.2.6 Ajuda

A ferramenta de ajuda exibe as informações do manual do usuário do sistema. Esta função está subdividida em Página Inicial, Treinamento do Modelo Acústico, Treinamendo do Modelo de Linguagem, Testes, Limpar Sistema e Ajuda, da mesma maneira que a divisão de subseções desta seção. O Apêndice C contém o mesmo conteúdo da seção de Ajuda da ferramenta, com o manual de utilização e todas as capturas de telas possíveis do sistema.

Capítulo 4

Testes comparativos

Este capítulo abordará testes realizados, analisará os resultados e proporá outros testes que podem ser feitos com um sistema de reconhecimento de voz. Conforme dito no capítulo anterior, o objetivo do sistema criado é permitir a elaboração de testes com o sistema de reconhecimento da fala baseado no *toolkit* CMU Sphinx, focando na abordagem mais técnica e teórica dos testes, sem a necessidade de haver preocupações com o aspecto de entendimento e manipulação do *toolkit*.

Este capítulo está organizado da seguinte maneira: primeiramente, na Seção 4.1 será exposta a quantidade de parâmetros disponíveis pelo sistema passíveis de alterações; posteriormente, na Seção 4.2 os parâmetros serão divididos em categorias de forma a se analisar a influência de cada um deles sobre o sistema de reconhecimento da fala; na Seção 4.4 serão definidos os princípios de decisão para realização de testes; por fim, na Seção 4.5 serão expostos os resultados dos testes.

4.1 Parâmetros manipuláveis

Um sistema genérico de reconhecimento da fala possui uma quantidade de parâmetros manipuláveis muito grande. Este fato torna a questão de otimização de parâmetros bastante complexa para estes sistemas. Este problema não é diferente com o *toolkit* CMU Sphinx. A Tabela 4.1 mostra a quantidade de parâmetros presentes no CMU Sphinx divididos por categorias.

Tabela 4.1: Quantidade de parâmetros por categoria

Tipo	**Quantidade**
Extração de Características	13
Modelo Acústico	6
Modelo de Linguagem	1
Testes	4
Total	24

Além da grande quantidade de parâmetros, existe uma ampla faixa de valores que determinados parâmetros podem assumir. Na próxima seção, serão detalhados todos os parâmetros disponíveis para manipulação do sistema e suas divisões por categorias.

4.2 Organização dos parâmetros e suas influências

Os parâmetros passíveis de alterações podem ser divididos em sub-categorias, de acordo com as seções a seguir. Dependendo do tipo de análise do sistema, é possível verificar o impacto de cada categoria no desempenho do sistema.

Vale lembrar que, além dos testes dos parâmetros, usar diferentes arquivos externos (dicionário, base de áudio, base de texto e etc.), com diferentes conteúdos, também representa um aumento considerável na complexidade de otimização de parâmetros.

4.2.1 Digitalização do sinal

Esta categoria não compreende competência inteiramente da ferramenta CMU Sphinx, mas, por sua vez, não deixa de ser importante. Este processo é realizado sempre previamente à utilização do sistema. A digitalização do áudio pode influenciar bastante no desempenho qualitativo de um sistema de reconhecimento de voz. Um dos principais parâmetros para a digitalização do áudio é a frequência de amostragem. Além deste parâmetro, a remoção de nível DC e o valor do coeficiente de pré-ênfase, de competência do CMU Sphinx, também devem ser definidos. Resumindo, os parâmetros referentes à digitalização do sinal e sua preparação para a extração de características são os seguintes:

- Taxa de Amostragem;
- Remoção de Nível DC;
- Coeficiente de Pré-Ênfase.

4.2.2 Janelamento

A catergoria referente ao janelamento do sinal de áudio é uma categoria bastante importante na extração de características. Principalmente porque o sinal de voz é considerado estacionário durante cada janela, a escolha do tamanho da janela e a sobreposição entre janelas devem ser cuidadosamente feitas. Então, os parâmetros referentes ao janelamento do sinal de áudio são os seguintes:

- Tamanho da Janela;

- *Frame Rate*.

4.2.3 Extração de coeficientes

Os parâmetros desta categoria compreendem a extração dos coeficientes MFCC modeladores do trato vocal. Dentre os principais parâmetros desta seção, está o número de coeficientes a serem extraídos. Os parâmetros referentes à extração dos coeficientes de modelagem do trato vocal são os seguintes:

- Número de Pontos da FFT;
- Número de Filtros no Banco de Filtros;
- Frequência Mínima do Banco de Filtros;
- Frequência Máxima do Banco de Filtros;
- Número de Canais do Banco de Filtros;
- Tipo de Transformada;
- Número de Coeficientes MFCC;
- Coeficiente de Liftragem Cepstral.

4.2.4 Modelo acústico

O treinamento do modelo acústico, apesar de complexo, não possui tantos parâmetros de análise. Porém, a importância destes parâmetros na qualidade do modelo treinado é bastante relevante. Basicamente, os parâmetros referentes ao treinamento do modelo acústico são os seguintes:

- Número Mínimo de Iterações;
- Número Máximo de Iterações;
- Taxa de Convergência;
- Número de Componentes Gaussianas;
- Número de Estados Compartilhados.

4.2.5 Modelo de linguagem

O treinamento do modelo de linguagem consiste basicamente em definir somente o tipo de modelo a ser utilizado. Neste caso, mais importantes são a base de texto usada para treinamento do modelo e o arquivo de dicionários. Então, para o modelo de linguagem só há um parâmetro a ser definido:

- Tipo de Modelos (Unigrama, Bigrama, Trigrama ou Quadrigrama).

4.2.6 Execução de testes

A execução de testes possui alguns parâmetros variáveis que determinam basicamente a forma como o computador irá processar as informações, priorizando a qualidade em detrimento da velocidade de processamento ou vice-versa. Os parâmetros referentes à execução de testes são os seguintes:

- Tamanho da lista ativa;
- Nó de maior pontuação a cada instante;
- Importância relativa de palavras;
- Penalidade a cada nova palavra.

4.3 Base de dados e arquivos externos

A base de dados utilizada para a implementação de testes do sistema de reconhecimento da fala neste trabalho consiste na elocução de mil frases foneticamente balanceadas [18], pronunciadas por um único locutor e separadas uma frase por arquivo de áudio tipo *MS Wav* com taxa de amostragem de 48 kHz, junto com suas respectivas trancrições.

Para o modelo de linguagem foi utilizada uma base de texto com 150 mil frases extraídas da base de dados pública do CETENFolha [19].

Também foi utilizado um dicionário fonético contendo uma lista com 3528 palavras (palavras distintas encontradas na base das 1000 frases) com suas respectivas transcrições fonéticas. As transcrições fonéticas foram obtidas automaticamente a partir de algoritmos publicados em [20] e revisadas manualmente.

O arquivo de fones utilizado contém os 39 fones presentes no Português Brasileiro (incluindo o fone representativo de silêncio). O conteúdo deste arquivo pode ser encontrado na Seção A.4 do Apêndice A.

Quando optado por utilizar arquivo de questões fonéticas próprias, utilizaram-se as questões fonéticas propostas em [11].

4.4 Princípios para decisão dos testes comparativos

Conforme dito anteriormente, o problema de otimização dos parâmetros do reconhecimento de voz é bastante complexo. Por isso, neste trabalho, optou-se por realizar somente alguns testes dos parâmetros julgados mais impactantes no desempenho do sistema de cada categoria. Dessa forma, para o teste de cada parâmetro de cada categoria, mantiveram-se todos os outros parâmetros com os valores definidos por padrão em [4].

Como a base de áudio é composta por 1000 frases, a mesma regra de teste adotada em [4] foi utilizada: 750 arquivos foram utilizados para treinamento do sistema e 250 arquivos para testes. Variando estes grupos a fim de se obter homogeneidade nos testes, tem-se 4 etapas de testes, cada um com diferentes grupos de treinamento e teste. O diagrama da Figura 4.1 ilustra a divisão da base para cada etapa de teste. Após a realização das quatro etapas de testes, é calculada a média a fim de se obter um resultado mais homogêneo do sistema. Este processo é também chamado de validação cruzada.

4.5 Testes e resultados

Todos os testes realizados neste trabalho foram feitos sob a plataforma *Microsoft Windows 7 Professional 64 bits*, Processador *Intel Core 2 Quad* $2,4\ GHz$, $4\ GB$ de Memória RAM e navegador *Google Chrome*.

Para efeito de análise, foram utilizadas as médias aritméticas dos resultados para os quatro conjuntos de testes para cada medida utilizada: Acurácia, WER e RTF. Além disso, também foi utilizado o respectivo desvio padrão populacional para cada medida. A média aritmética é definida por:

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.1}$$

onde x_i representa a i-ésima amostra e n representa o total de amostras. Já o desvio padrão populacional pode ser definido como:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (x_i - \overline{x})^2} \tag{4.2}$$

onde x_i representa a i-ésima amostra, n representa o total de amostras e $\overline{x}$ a média aritmética.

O primeiro teste realizado, e considerado o teste padrão, corresponde aos

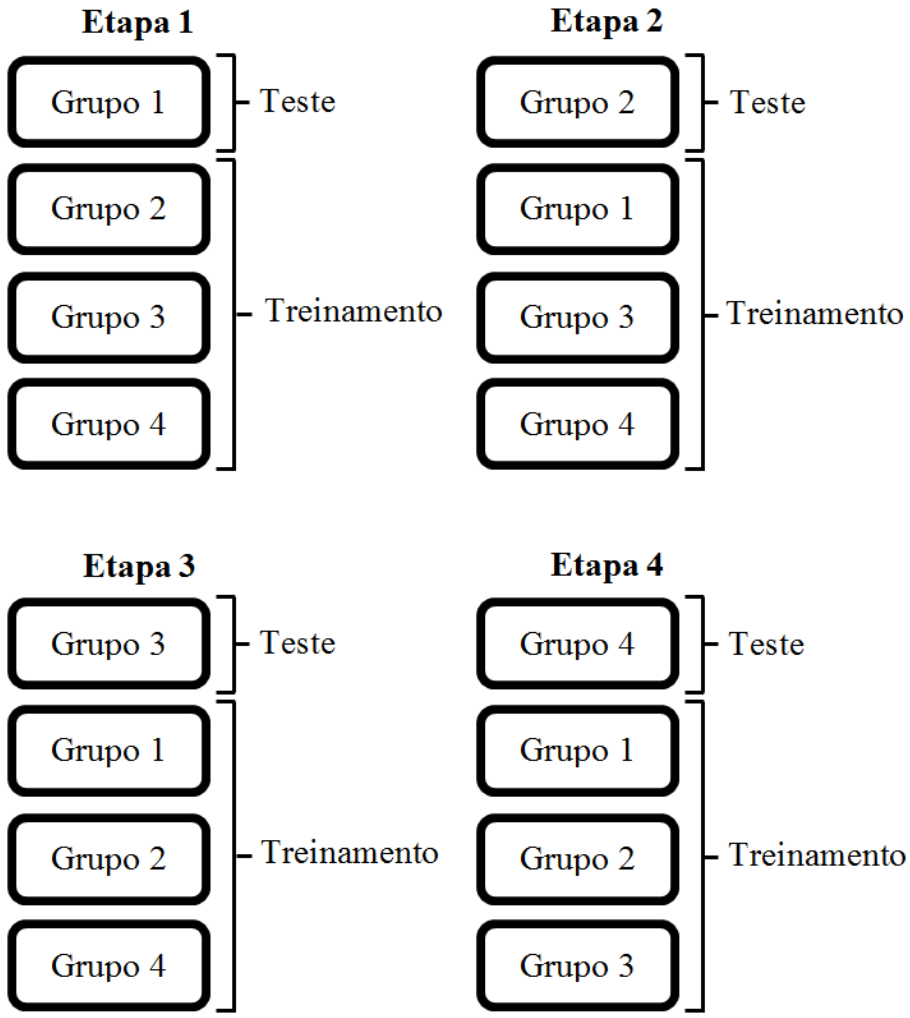


Figura 4.1: Diagrama dos grupos de testes. Cada grupo é composto por um conjunto de 250 arquivos de áudio. Para cada etapa tem-se 250 arquivos de áudio destinados ao teste e 750 arquivos destinados ao treinamento. Após a realização das 4 etapas, tira-se a média para obter o resultado completo do teste.

parâmetros setados com os valores sugeridos por [4] e anteriormente apresentados nas Tabelas 3.2, 3.3 e 3.4, utilizando modelo de linguagem do tipo trigrama. A Tabela 4.2 mostra os resultados para este teste. Todos os outros resultados obtidos serão comparados com este.

Tabela 4.2: Resultado padrão

Acurácia	$\sigma_{\mathbf{AC}}$	**WER**	$\sigma_{\mathbf{WER}}$	**RTF**	$\sigma_{\mathbf{RTF}}$
88,454	1,592	14,510	1,939	0,875	0,135

Com relação à extração de parâmetros, primeiramente foi testada a influência da remoção ou não do nível DC do sinal. A Tabela 4.3 mostra estes resultados.

A remoção do nível DC do sinal, para este conjunto de testes, mostrou influenciar muito pouco o desempenho do sistema, seja em termos de tempo de processamento,

Tabela 4.3: Resultado da remoção do nível DC

Remover Nível DC	**Acurácia**	$\sigma_{\mathbf{AC}}$	**WER**	$\sigma_{\mathbf{WER}}$	**RTF**	$\sigma_{\mathbf{RTF}}$
Sim	88,454	1,592	14,510	1,939	0,875	0,135
Não	88,404	2,146	14,520	2,323	0,870	0,119

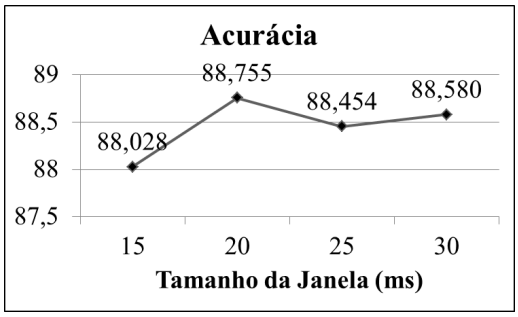
seja em termos de análises qualitativas.

O próximo teste com relação à extração de parâmetros foi o do tamanho da janela utilizada. Foram testados os tamanhos de 15 ms, 20 ms, 25 ms e 30 ms. Os resultados estão na Tabela 4.4.

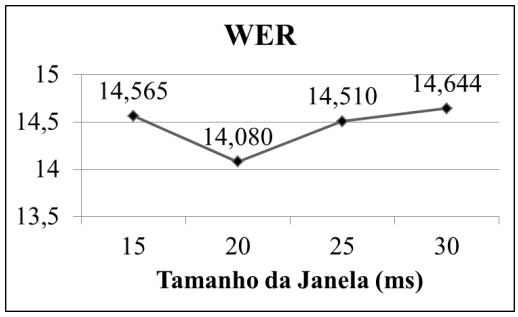
Tabela 4.4: Resultado do tamanho da janela

Tamanho da Janela	**Acurácia**	$\sigma_{\mathbf{AC}}$	**WER**	$\sigma_{\mathbf{WER}}$	**RTF**	$\sigma_{\mathbf{RTF}}$
15 ms	88,028	2,041	14,565	2,178	0,960	0,135
20 ms	88,755	1,984	14,080	2,034	0,948	0,163
25 ms	88,454	1,592	14,510	1,939	0,875	0,135
30 ms	88,389	2,070	14,896	2,218	0,694	0,238

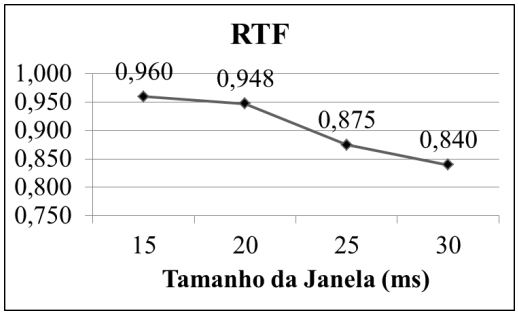
Os gráficos das Figuras 4.2a, 4.2b e 4.2c ilustram a evolução da Acurácia, WER e RTF, respectivamente, para os diferentes tamanhos de janelas testados.



(a) Acurácia - Janelamento



(b) WER - Janelamento



(c) RTF - Janelamento

Figura 4.2: Gráficos da Acurácia, WER e RTF para diferentes tamanhos de janelas

Nota-se que o tempo de processamento é inversamente proporcional ao tamanho da janela. Com a diminuição do tamanho da janela, mais janelas precisarão ser

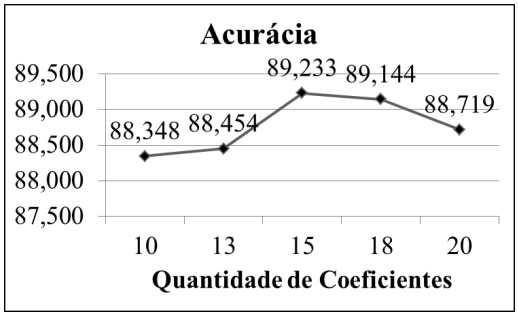
computadas, naturalmente. Para a janela de 20 ms, o sistema se mostrou levemente melhor na média em termos qualitativos.

Ainda com relação à extração de parâmetros, foram realizados testes com diferentes quantidades de coeficientes MFCC. A Tabela 4.5 mostra estes resultados.

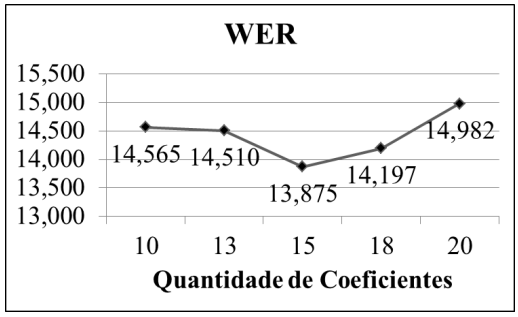
Tabela 4.5: Resultado da quantidade de coeficientes

Número de Coeficientes	**Acurácia**	$\sigma_{\mathbf{AC}}$	**WER**	$\sigma_{\mathbf{WER}}$	**RTF**	$\sigma_{\mathbf{RTF}}$
10	88,348	1,107	14,565	1,098	0,930	0,060
13	88,454	1,592	14,510	1,939	0,875	0,135
15	89,233	1,411	13,875	1,427	0,765	0,137
18	89,144	1,420	14,197	1,688	0,164	0,678
20	88,719	1,731	14,982	2,153	0,640	0,202

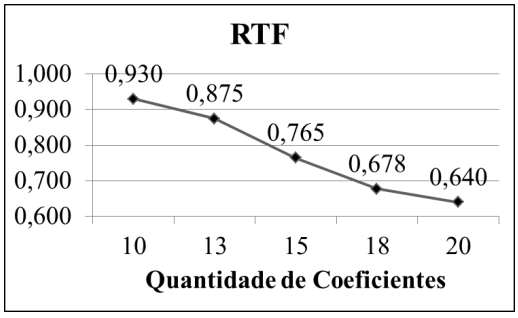
Os gráficos das Figuras 4.3a, 4.3b e 4.3c ilustram a evolução da Acurácia, WER e RTF, respectivamente, para diferentes quantidades de coeficientes MFCC testados.



(a) Acurácia - Coeficientes MFCC



(b) WER - Coeficientes MFCC



(c) RTF - Coeficientes MFCC

Figura 4.3: Gráficos da Acurácia, WER e RTF para diferentes quantidades de coeficientes MFCC

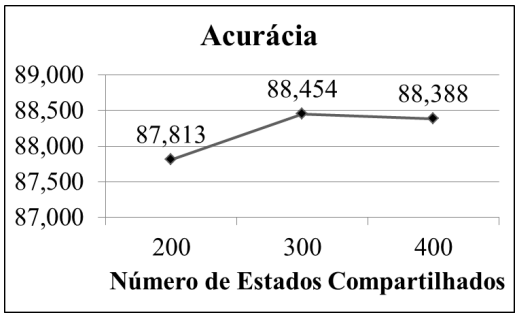
Com os testes realizados, o melhor desempenho qualitativo médio foi com a utilização de 15 coeficientes MFCC, embora haja um empate estatístico. Como a base de voz utilizada neste trabalho consiste apenas de um único locutor, sugere-se que sejam realizados mais testes com este parâmetro, possivelmente utilizando uma base com mais locutores e analisando o impacto do aumento do número de coeficientes extraídos. Percebe-se também que o RTF médio cai com o aumento da quantidade

de coeficientes. Esse comportamento pode ser explicado pela maior diferenciação entre vetores de comprimentos maiores, reduzindo o tempo de busca no reconhecimento. Quanto maior a quantidade de coeficientes por vetor, mais informações são disponibilizadas sobre cada bloco, facilitando as buscas por similaridades.

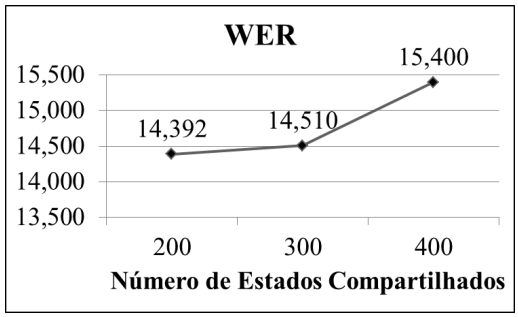
Com relação ao modelo acústico, foram realizados testes com a alteração do número de estados compartilhados e a quantidade de gaussianas do modelo. As Tabelas 4.6 e 4.7 mostram, respectivamente, estes resultados. As Figuras 4.4 e 4.5 ilustram, respectivamente, a evolução da Acurácia, do WER e do RTF para a variação do número de estados compartilhados e da quantidade de gaussinas do modelo.

Tabela 4.6: Resultado da quantidade de estados compartilhados

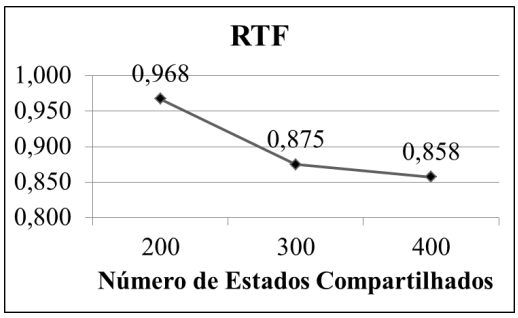
Estados Compartilhados	**Acurácia**	$\sigma_{\mathbf{AC}}$	**WER**	$\sigma_{\mathbf{WER}}$	**RTF**	$\sigma_{\mathbf{RTF}}$
200	87,813	1,609	14,392	1,525	0,968	0,099
300	88,454	1,592	14,510	1,939	0,875	0,135
400	88,388	2,239	15,400	2,565	0,858	0,105



(a) Acurácia - Estados Compartilhados



(b) WER - Estados Compartilhados



(c) RTF - Estados Compartilhados

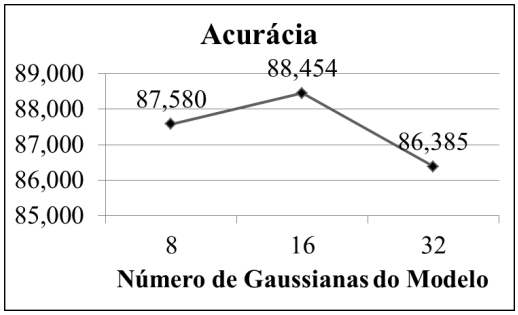
Figura 4.4: Gráficos da Acurácia, WER e RTF para diferentes quantidades de estados compartilhados

Nota-se que, com 200 estados compartilhados, é obtido o menor índice médio de WER. Porém, com 300 estados compartilhados, a Acurácia média obtida é melhor. Então, é necessário definir, nesse caso, qual medida é mais importante para a aplicação desejada. Determinadas aplicações, como reconhecimento de fala contínua, por exemplo, apreciam mais uma menor taxa de WER, enquanto aplicações como

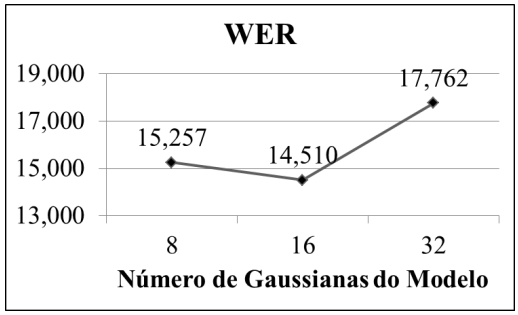
Unidades de Resposta Audível (URA), utilizadas em *call centers*, privilegiam a utilização de medidas de Acurácia, visto que o objetivo destes sistemas é reconhecer palavras-chave dentro do contexto da elocução. Com relação ao RTF, quanto mais estados compartilhados, mais rápido será o sistema, visto que mais estados compartilharão o mesmo modelo, tornando o sistema mais eficiente e menos preciso.

Tabela 4.7: Resultado da quantidade de gaussianas

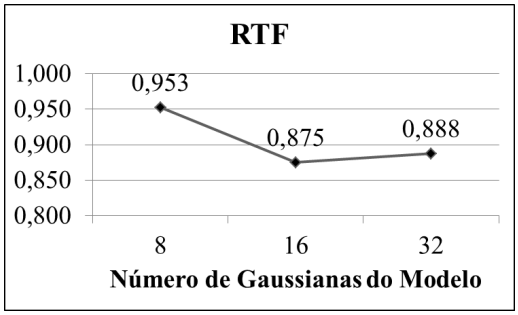
Gaussianas	**Acurácia**	$\sigma_{\mathbf{AC}}$	**WER**	$\sigma_{\mathbf{WER}}$	**RTF**	$\sigma_{\mathbf{RTF}}$
8	87,580	1,296	15,257	1,673	0,953	0,074
16	88,454	1,592	14,510	1,939	0,875	0,135
32	86,385	3,597	17,762	3,884	0,888	0,164



(a) Acurácia - Gaussianas



(b) WER - Gaussianas



(c) RTF - Gaussianas

Figura 4.5: Gráficos da Acurácia, WER e RTF para diferentes quantidades de gaussianas do modelo

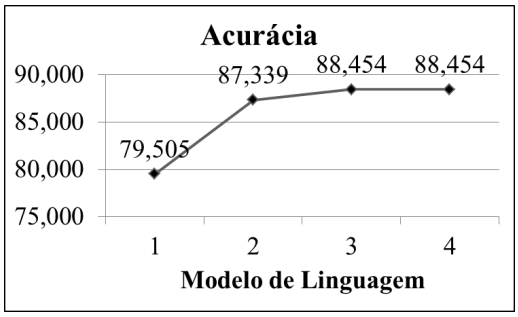
Com relação à quantidade de gaussianas, o melhor resultado médio obtido em termos qualitativos foi com 16 gaussianas. A piora da média da Acurácia e da média do WER com a utilização de 32 gaussianas pode ser explicada pela escassez de dados para estimação dos parâmetros de cada gaussiana. Para uma base de dados maior, a tendência é que, com o aumento do número de gaussianas, o sistema fique mais preciso.

Os testes realizados com os modelos de linguagem propuseram a comparação entre a utilização do modelo unigrama, bigrama, trigrama e quadrigrama. A Tabela 4.8

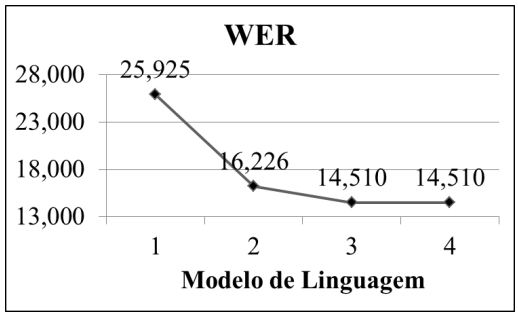
mostra os resultados obtidos dos testes do modelo de linguagem e a Figura 4.6 mostra o gráfico da evolução da Acurácia, do WER e do RTF para os diferentes tipos de modelo de linguagem.

Tabela 4.8: Resultado do modelo de linguagem

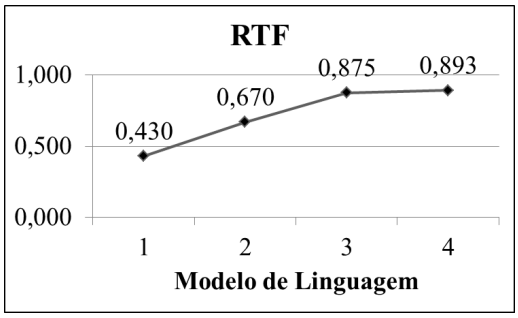
Tipo	**Acurácia**	$\sigma_{\mathbf{AC}}$	**WER**	$\sigma_{\mathbf{WER}}$	**RTF**	$\sigma_{\mathbf{RTF}}$
Unigrama	80,092	2,541	24,491	2,804	0,325	0,179
Bigrama	87,339	1,631	16,226	1,931	0,670	0,139
Trigrama	88,454	1,592	14,510	1,939	0,875	0,135
Quadrigrama	88,454	1,592	14,510	1,939	0,893	0,135



(a) Acurácia - Modelo de Linguagem



(b) WER - Modelo de Linguagem



(c) RTF - Modelo de Linguagem

Figura 4.6: Gráficos da Acurácia, WER e RTF para os diferentes tipos de modelo de linguagem

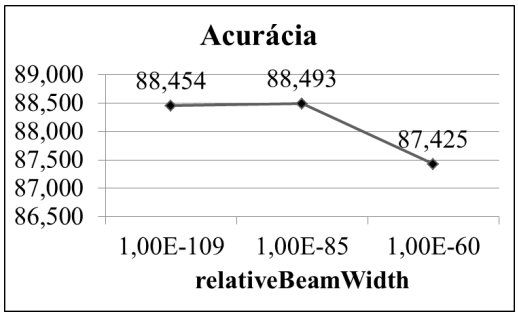
Como esperado, o modelo de linguagem do tipo trigrama se mostra superior aos outros para o reconhecimento. Porém, no tempo de processamento, o modelo unigrama se mostra superior. Isso se dá devido à complexidade do modelo de linguagem. O modelo unigrama é o mais simples possível, considerando apenas a probabilidade de ocorrência de cada palavra individualmente. Neste sentido, é necessário avaliar a diminuição do tempo de processamento em detrimento da qualidade ou o aumento da qualidade em detrimento do tempo de processamento para escolha de qual modelo utilizar. Nota-se que o modelo quadrigrama não melhorou o desempenho do sistema, mantendo-o, em termos qualitativos, exatamente igual ao modelo trigrama. Porém, o tempo de processamento médio aumentou ligeiramente, tornando o modelo

trigrama superior. Com este resultado, pode-se chegar à conclusão que o aumento do número de palavras do contexto, acima de dois, para este modelo de linguagem utilizado, não agrega nenhuma nova informação relevante que auxilie no reconhecimento. Sugere-se que sejam testados o modelo quadrigrama, comparado ao trigrama, com novas bases de texto, novos dicionários de palavras e nova base de áudio.

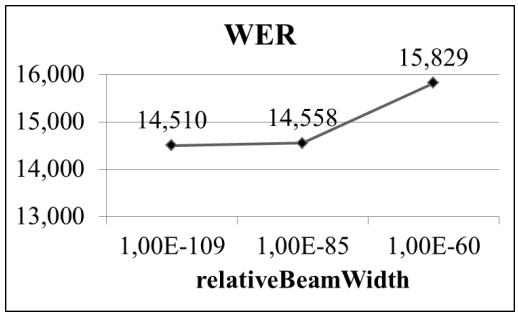
Com relação aos parâmetros de execução, foi testada a influência do *relativeBeamWidth*. Este parâmetro se mostrou eficiente em termos de tempo de processamento, com pequena perda no reconhecimento. A Tabela 4.9 mostra os resultados deste teste e a Figura 4.7 mostra os gráficos com a evolução da Acurácia, do WER e do RTF.

Tabela 4.9: Resultado do *relativeBeamWidth*

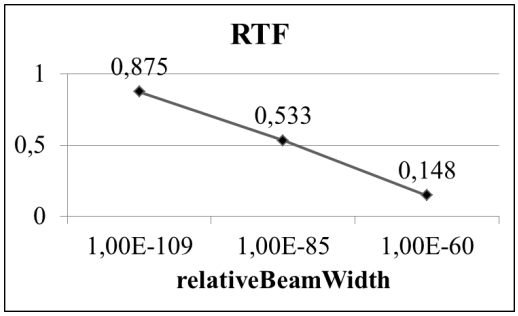
relativeBeamWidth	**Acurácia**	$\sigma_{\mathbf{AC}}$	**WER**	$\sigma_{\mathbf{WER}}$	**RTF**	$\sigma_{\mathbf{RTF}}$
10^{-109}	88,454	1,592	14,510	1,939	0,875	0,135
10^{-85}	88,493	1,681	14,558	2,086	0,533	0,164
10^{-60}	87,425	1,262	15,829	1,696	0,148	0,042



(a) Acurácia - relativeBeamWidth



(b) WER - relativeBeamWidth



(c) RTF - relativeBeamWidth

Figura 4.7: Gráficos da Acurácia, WER e RTF para os diferentes valores do relativeBeamWidth

Este teste do parâmetro relativeBeamWidth de execução merece atenção devido ao grande ganho no tempo de processamento comparado às perdas na Acurácia e no WER.

A partir dos testes realizados, utilizando os parâmetros com melhores desempenho em termos qualitativos, realizou-se um teste com os parâmetros otimizados para este trabalho. Os parâmetros alterados em relação ao teste padrão proposto foram: diminuição da janela para 20 ms e 15 coeficientes MFCC. O resultado obtido está descrito na Tabela 4.10.

Tabela 4.10: Resultado final

Acurácia	$\sigma_{\mathbf{AC}}$	**WER**	$\sigma_{\mathbf{WER}}$	**RTF**	$\sigma_{\mathbf{RTF}}$
88,757	1,326	14,457	1,247	0,875	0,142

Nota-se que, embora os parâmetros escolhidos para o tamanho da janela e o número de coeficientes tenham sido os com melhores resultados médios dos testes deste trabalho, quando utilizados em conjunto demonstraram uma piora no desempenho do sistema se comparado com os resultados individuais dos testes. Por exemplo, utilizando 15 coeficientes e mantendo o tamanho da janela em 25 ms, tem-se uma Acurácia média de 89,233 e um WER médio de 13,875 (conforme visto na Tabela 4.5), que são resultados melhores que o teste final. Portanto, torna-se bastante razoável a realização de testes conjuntos com variações de parâmetros, o que dificulta ainda mais a obtenção de um conjunto de parâmetros ótimos para um sistema de reconhecimento da fala.

Capítulo 5

Conclusões e trabalhos futuros

Este trabalho apresentou o desenvolvimento de um sistema de reconhecimento da fala baseado no *toolkit* CMU Sphinx e a criação de uma interface de análise e testes. Primeiramente, toda a abordagem teórica do sistema de reconhecimento foi apresentada. Posteriormente, foi realizada toda a implementação do sistema do CMU Sphinx. Por último, foi desenvolvida a interface de comunicação entre o projetista e o sistema.

O desenvolvimento da interface em PHP mostrou-se eficiente e compatível com diversos navegadores. Um ponto bastante importante para se destacar neste trabalho é o longo tempo de processamento necessário para o treinamento e execução de testes. Este tempo faz com que o servidor demore a retornar os resultados, necessitando a alteração de parâmetros no servidor. Foram necessárias alterações no tempo máximo de resposta permitida pelo servidor, no tamanho máximo dos arquivos para *upload*, na quantidade de arquivos de *upload*, além de alterações de parâmetros de gerenciamento de memória pelo servidor.

Por estar desenvolvido em JAVA (e o sistema de interface estar em PHP), o sistema completo é multiplataforma, sendo necessárias somente alterações nos arquivos de execução de comandos (`.bat`) para adaptação para outro sistema operacional diferente do Microsoft Windows.

Os testes da interface se mostraram eficientes, de forma que não foram encontrados *bugs* ou qualquer problema de outra natureza. A comunicação com o CMU Sphinx se mostrou muito adequada e sem perda de tempo de processamento intermediário. Vale ressaltar que os maiores e únicos processamentos da interface se encontram na manipulação dos arquivos de *upload*, onde há a necessidade de movê-los para seus respectivos diretórios. Os outros tempos de processamento decorrem unicamente da execução do CMU Sphinx.

Os testes do sistema como um todo mostraram que, para a base utilizada, a diminuição do tamanho da janela até 20 ms e o aumento do número de coeficientes MFCC até 15 melhorou o reconhecimento do sistema. O tempo de processamento

se manteve exatamente igual ao do teste padrão. O parâmetro de execução *absoluteBeamWidth* se mostrou importante no tempo de execução do sistema. Os testes se mostraram coerentes com a base teórica.

Como trabalhos futuros, propõe-se a implementação deste sistema num servidor com alto poder de processamento e acesso remoto. Esta implementação proporcionaria um treinamento de um sistema de reconhecimento num tempo mais hábil, sem que houvesse sobrecarga do computador de trabalho do projetista. Também propõe-se a realização de estudos a respeito dos testes qualitativos com a finalidade de se tentar realizar uma melhor otimização dos parâmetros do sistema, além de realização de testes com diferentes arquivos de dicionário, base de dados, e questões fonéticas. Além disso, propõe-se uma maior variação dos parâmetros testados e a realização de testes conjuntos para avaliação do impacto da variação de determinados parâmetros sobre outros. Este tipo de necessidade de teste ficou bastante claro nos últimos resultados, referenciados na Tabela 4.10.

A base de dados utilizada neste trabalho, principalmente a base de áudio, restringe muito o sistema por ser dependente do locutor e por possuir um número limitado de elocuções. Além disso, o impacto da alteração de cada parâmetro sobre o sistema depende da base de dados sobre a qual é aplicada. Assim, propõe-se também o estudo de um sistema independente de locutor com uma quantidade maior de elocuções para treinamento do sistema.

Referências Bibliográficas

[1] *HTK Speech Recognition*. http://htk.eng.cam.ac.uk/, 2013.

[2] *Open-Source Large Vocabulary CSR Engine Julius*. http://julius.sourceforge.jp/en_index.php, 2013.

[3] *CMU Sphinx - Speech Recognition Toolkit*. http://cmusphinx.sourceforge.net/, 2013.

[4] DE FRANÇA OLIVEIRA, V. *Reconhecimento de Fala Contínua para o Português Brasileiro Baseado em HTK e Sphinx*. Projeto final de graduação, Universidade Federal do Rio de Janeiro, Brasil, 2010.

[5] OLIVEIRA, M. P. B. *Verificação Automática do Locutor, Dependente do Texto, Utilizando Sistemas Híbridos MLP/HMM*. Dissertação de mestrado, Instituto Militar de Engenharia, Praça General Tibúrcio, 80, Praia Vermelha, Rio de Janeiro - RJ, Brasil, 2001.

[6] DE ALENCAR, V. F. S. *Atributos e Domínios de Interpolação Eficientes em Reconhecimento de Voz Distribuído*. Dissertação de mestrado, Pontifícia Universidade Católica do Rio de Janeiro, Brasil, 2005.

[7] PICONE, J. "Signal Modeling Techiques in Speech Recognition", *Proceedings of the IEEE*, v. 81, n. 9, pp. 1215–1246, set. 1993.

[8] YOUNG, S., EVERMANN, G., GALES, M., et al. *The HTK Book (for HTK Version 3.4)*. http://htk.eng.cam.ac.uk/docs/docs.shtml, 2009.

[9] DOS SANTOS PERDIGÃO, F. M. *Modelos do Sistema Auditivo Periférico no Reconhecimento Automático de Fala*. Tese de D.Sc., Faculdade de Ciências e Tecnologia - Universidade de Coimbra, Coimbra, Portugal, 1997.

[10] KUTWAK, A. B. *Análise da Codificação LPC para Sinais de Fala*. Projeto final de graduação, Universidade Federal do Rio de Janeiro, Brasil, 1999.

[11] TEVAH, R. T. *Implementação de um Sistema de Reconhecimento de Fala Contínua com Amplo Vocabulário para o Português Brasileiro*. Dissertação de mestrado, Universidade Federal do Rio de Janeiro, Brasil, 2006.

[12] DUGAD, R., DESAI, U. B. *A Tutorial on Hidden Markov Models*. Technical Report SPANN-96.1, 1996.

[13] RABINER, L. R., JUANG, B. H. "An Introduction to Hidden Markov Models", *IEEE ASSP Magazine*, pp. 4–16, jan. 1986.

[14] PEYTON Z. PEEBLES, J. *Probability, Random Variables and Random Signal Principles*. 4 ed. New York, McGraw-Hill, Inc., 2000.

[15] DA SILVA MAIA, R. *Speech synthesis and phonetic vocoding for Brazilian Portuguese based on parameter generation from hidden Markov models*. Ph.D. thesis, Nagoya Institute of Technology, Japan, 2006.

[16] *SphinxTrain and Sphinx-3 Documentation*. http://www.speech.cs.cmu.edu/sphinxman/, 2013.

[17] WALKER, W., LAMERE, P., KWOK, P., et al. "Sphinx-4: A Flexible Open Source Framework for Speech Recognition", *WhitePaper SMLI Sun Microsystems Inc*, ago. 2004.

[18] CIRIGLIANO, R. J. R., MONTEIRO, C., DE F. BARBOSA, F. L., et al. "Um Conjunto de 1000 Frases Foneticamente Balanceadas para o Português Brasileiro Obtido Utilizando a Abordagem de Algoritmos Genéticos", *Anais do XXII Simpósio Brasileiro de Telecomunicações - SBrT*, set. 2005.

[19] TEVAH, R. T., RESENDE, F. G. "Implementation of a Large Vocabulary Continuous Speech Recognition System for Brazilian Portuguese", *Journal of Communication and Information Systems*, v. 21, n. 3, pp. 204–218, jan. 2006.

[20] SILVA, D., DE LIMA, A., MAIA, R., et al. "A rule-based grapheme-phone converter and stress determination for Brazilian Portuguese natural language processing", *Proceedings of International Telecommunications Symposium*, v. 1, pp. 992–996, set. 2006.

Apêndice A

Arquivos externos utilizados no CMU Sphinx

A.1 Arquivo de identificação dos arquivos de áudio para o treinamento do modelo acústico

O arquivo de indentificação dos arquivos de áudio deve conter os nomes dos arquivos de áudio sem a extensão, um por linha, que serão utilizados no treinamento. Este arquivo deve seguir o seguinte padrão:

```
M011311
M011714
M011313
M011715
M011716
M011716
M011717
M011318
M011319
M011320
...
```

A.2 Arquivo de transcrições para o treinamento do modelo acústico

O arquivo de transcrições para o treinamento acústico deve conter a transcrição seguido pelo nome do arquivo de áudio correspondente da seguinte maneira:

```
<s> SEGUNDO ELE <sil> A POLÍCIA NÃO IRIA CEDER A EXIGÊNCIAS </s>  (M011311)
<s> <sil> ONTEM <sil> MAIS SEIS EXILADOS VOLTARAM À FAIXA DE GAZA </s>  (M011714)
<s> O TREINO DE HOJE SERÁ DAS TREZE ÀS DEZESSETE HORAS EM BRASÍLIA </s>  (M011313)
```

```
<s> A PACIENTE <sil> DE VINTE E DOIS ANOS <sil> PASSA BEM <sil> </s>  (M011715)
<s> <sil> O DINHEIRO É ARRECADADO PELOS TEMPLOS ATRAVÉS DO DÍZIMO </s>  (M011716)
<s> O BRASIL TAMBÉM É COR DE ROSA E CARVÃO </s>  (M014315)
<s> TEM-SE UMA RECEITA MENSAL DE TREZENTOS E QUARENTA MIL DÓLARES </s>  (M011717)
<s> O QUE CONTA AGORA É O QUE O CANDIDATO TEM A DIZER </s>  (M011318)
<s> <sil> NÃO BASTARÁ QUE LULA EXIBA OS DENTES DIANTE DAS CÂMERAS </s>  (M011319)
<s> AGORA ELE DESPENCA DO TETO EM DIREÇÃO AO PISO </s>  (M011320)
...
```

A.3 Arquivo de dicionário para o treinamento do modelo acústico ou de linguagem

O arquivo de dicionário deve conter uma lista de palavras com as suas respectivas transcrições fonéticas. Este arquivo deve seguir o mesmo padrão, tanto no treinamento do modelo acústico, quanto no treinamento do modelo de linguagem, da seguinte maneira:

```
A a
ABAIXO a b a j S u
ABALADO a b a l a d u
ABANDONAR a b a~ d o n a X
ABANDONOU a b a~ d o n o w
ABASTECER a b a s t e s e X
ABASTECIMENTO a b a s t e s i m e~ t u
ABATIMENTO a b a tS i m e~ t u
ABERNÉSSIA a b e R n E s i a
ABERTO a b e X t u
...
```

A.4 Arquivo de fones

O arquivo de fones contém uma lista com todos os fonemas a serem utilizados pelo sistema, incluindo o símbolo que representa silêncio. O arquivo deve seguir o seguinte padrão:

```
E
J
L
O
R
S
X
Z
a
```

```
a~
b
d
dZ
e
e~
f
g
i
i~
j
j~
k
l
m
n
o
o~
p
r
s
SIL
t
tS
u
u~
v
w
w~
z
```

A.5 Arquivo de questões fonéticas

Se o desenvolvedor optar por utilizar questões fonéticas próprias, será necessário o *upload* do arquivo de questões fonéticas. Este arquivo irá conter um conjunto de perguntas com respostas *booleanas* do tipo "sim"ou "não"a fim de se criar uma árvore onde cada nó terminal é composto por estados que possuem as mesmas respostas para o conjunto de questões, a fim de agrupá-los no *tied states*. O arquivo pode conter perguntas sobre o próprio fone analisado ou sobre seus vizinhos imediatos da direita (identificado por `R_`) ou da esquerda (identificado por `L_`). No arquivo, após cada pergunta, na mesma linha, devem haver os fones que correspondem à determinada pergunta. Deve haver uma pergunta por linha do arquivo e cada pergunta é identificada por uma espécie de *tag*. O arquivo de questões fonéticas deve seguir o seguinte padrão:

```
L_Silence_or_Pause SIL
L_Voiced           a E O e i o u a~ e~ i~ o~ u~ w j w~ j~ b d g m n J z v Z l L r R X
L_Continuant       a E O e i o u a~ e~ i~ o~ u~ s S z f v Z m n J
L_Noncontinuant    w j w~ j~ R r X l L p b t d k g
L_Vowel            a E O e i o u a~ e~ i~ o~ u~
L_Anterior_vowel   E e i e~ i~
...
Silence_or_Pause   SIL
Vowel              a E O e i o u a~ e~ i~ o~ u~
Voiced             a E O e i o u a~ e~ i~ o~ u~ w j w~ j~ b d g m n J z v Z l L r R X
...
R_Silence_or_Pause SIL
R_Voiced           a E O e i o u a~ e~ i~ o~ u~ w j w~ j~ b d g m n J z v Z l L r R X
R_Continuant       a E O e i o u a~ e~ i~ o~ u~ s S z f v Z m n J
R_Noncontinuant    w j w~ j~ R r X l L p b t d k g
R_Vowel            a E O e i o u a~ e~ i~ o~ u~
...
```

A.6 Arquivo *batch* de teste com arquivos múltiplos

Se o desenvolvedor optar por realizar testes com arquivos múltiplos de áudio a serem reconhecidos, será necessário o *upload* do arquivo contendo os nomes dos arquivos de áudio e suas respectivas transcrições. O arquivo de *batch* deve seguir o seguinte padrão:

```
M010101.wav PESQUISA É UMA COISA QUE MUDA A TODA HORA
M010102.wav NO TOTAL SERÃO CHAMADOS VINTE E SEIS MIL CANDIDATOS
M010103.wav O NÚMERO DE CONVOCADOS POR VAGA É DE DOZE CANDIDATOS
M010306.wav OS DOIS TIMES PERDERAM UMA CHANCE ATRÁS DA OUTRA
M010105.wav SANDRA REGINA MACHADO ACHO QUE ELA ENFIM CRIOU JUÍZO
M010307.wav A GERAÇÃO DE ZICO NÃO SERIA CAMPEÃ DO MUNDO
M010308.wav O CURSO É GRATUITO E TEM A DURAÇÃO DE SEIS MESES
M010108.wav NO TOTAL SETE MÍSSEIS FORAM DISPARADOS CONTRA O ENCRAVE
M010310.wav O MERCADO DE AÇÕES VIVE MOMENTOS DE PLENA EUFORIA
M010110.wav AS SITUAÇÕES DITAS EMBARAÇOSAS SÃO RESOLVIDAS COM OS DADOS
M010405.wav O MERCADO FICA DE ALTO RISCO A CURTO PRAZO
...
```

A.7 Arquivo de frases do modelo de linguagem

O arquivo com a base de texto utilizado no treinamento do modelo de linguagem deve conter uma frase por linha do arquivo, iniciadas pela *tag* `<s>` e finalizadas pela *tag* `</s>`, de acordo com o exemplo a seguir:

```
<s> NEM LULA NEM O PARTIDO AINDA ENCONTRARAM UM DISCURSO PARA SE DIFERENCIAR </s>
<s> ELES SE DIZEM OPOSIÇÃO MAS AINDA NÃO INFORMARAM O QUE VÃO COMBATER </s>
<s> AQUI SÓ JOGA QUEM ESTÁ BEM </s>
<s> ELE SÓ NÃO JOGAVA PORQUE NÃO ESTAVA BEM </s>
<s> OS TRÊS ESTÃO PRESOS DESDE TRINTA DE JULHO DE NOVENTA E TRÊS </s>
<s> ESSA DIVISÃO GERA ALGUMAS DISTORÇÕES TERRÍVEIS </s>
<s> O PANORAMA SOFRE PREJUÍZOS DEMAIS EM FAVOR DA TESE </s>
<s> TALVEZ ISTO SEJA MUITO BARULHO POR NADA </s>
<s> SEGUNDO O MÉDICO O CASO NÃO PREOCUPA </s>
<s> NESTE CASO O REGISTRO DA CANDIDATURA SERIA CANCELADO PELA JUSTIÇA </s>
...
```

A.8 Arquivo de empacotamento do modelo aústico

Para empacotamento do modelo acústico é necessária a criação do arquivo de configuração que deve possuir o mesmo nome do arquivo `.jar`. Segue abaixo um exemplo deste arquivo de configuração do empacotamento:

```
description = Modelo acustico do sistema de fala continua
de amplo vocabulario para o portugues brasileiro

modelClass = edu.cmu.sphinx.model.acoustic.
MYACMODEL_16gau_300s_13dCep_48k_40mel_1Hz_24000Hz.Model

modelLoader = edu.cmu.sphinx.model.acoustic.
MYACMODEL_16gau_300s_13dCep_48k_40mel_1Hz_24000Hz.ModelLoader

dataLocation = cd_continuous_16gau

modelDefinition = etc/MYACMODEL_16gau_300s_13dCep_48k_40mel_1Hz_24000Hz.300.mdef

isBinary = true
featureType = 1s_c_d_dd
vectorLength = 39
sparseForm = false

numberFftPoints = 2048
numberFilters = 40
gaussians = 16
minimumFrequency = 1
maximumFrequency = 24000
sampleRate = 48000
```

A.9 Arquivo *filler*

O arquivo *Filler* é um arquivo que contém a lista de eventos de não fala, como silêncio, ínício e fim de frase. O arquivo utilizado neste projeto foi o seguinte:

```
<s> SIL
<sil> SIL
</s> SIL
```

A.10 Arquivo de configuração do decodificador do Sphinx-4

O arquivo `config.xml` é utilizado para definir quais componentes serão utilizados no decodificador, assim como quais versões dos respectivos componentes serão usados. Segue um exemplo deste arquivo:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- ************************************************************ -->
<!--                   Sphinx-4 Configuration File                -->
<!--                   by Felipe Martins                          -->
<!-- ************************************************************ -->

<config>

<!-- ************************************************************ -->
<!--                  Global Properties                           -->
<!-- ************************************************************ -->

        <property name="frontend" value="myFrontEnd"/>
        <property name="recognizer" value="myRecognizer"/>
        <property name="logmath" value="myLogMath"/>

        <property name="relativeBeamWidth" value="1E-109"/>
        <property name="absoluteBeamWidth" value="7000"/>
        <property name="languageWeight" value="6.0"/>
        <property name="wordInsertionProbability" value="0.0001"/>

        <property name="logLevel" value="OFF"/>


<!-- ************************************************************ -->
<!--                        Batch                                 -->
<!-- ************************************************************ -->
```

```
  <component name="batch"
        type="edu.cmu.sphinx.tools.batch.BatchModeRecognizer">
        <propertylist name="inputDataProcessors">
            <item>streamCepstrumSource</item>
        </propertylist>
        <property name="skip" value="0"/>
        <property name="recognizer" value="\${recognizer}"/>
        <property name="logLevel" value="INFO"/>
  </component>


<!-- ************************************************************ -->
<!-- ************************************************************ -->
<!--                     My Decoder                               -->

  <component name="myDecoder"
         type="edu.cmu.sphinx.decoder.Decoder">
         <property name="searchManager" value="searchManager"/>
  </component>

<!-- ************************************************************ -->
<!--                     My Recognizer                            -->
<!-- ************************************************************ -->

   <component name="myRecognizer"
                type="edu.cmu.sphinx.recognizer.Recognizer">
    <property name="decoder" value="myDecoder"/>
    <propertylist name="monitors">
      <item>accuracyTracker</item>
      <item>speedTracker</item>
      <item>memoryTracker</item>
      <item>recognizerMonitor</item>
      <item>beamFinder</item>
    </propertylist>
  </component>

<!-- ************************************************************ -->
<!--                     Simple Search Manager                    -->
<!-- ************************************************************ -->

<component name="searchManager"
         type="edu.cmu.sphinx.decoder.search.SimpleBreadthFirstSearchManager">
         <property name="logMath" value="myLogMath"/>
         <property name="linguist" value="lexTreeLinguist"/>
         <property name="pruner" value="trivialPruner"/>
         <property name="scorer" value="threadedScorer"/>
         <property name="activeListFactory" value="activeList"/>
```

```
  </component>


<!-- ************************************************************ -->
<!--                 Word Pruning Search Manager                  -->
<!-- ************************************************************ -->

  <component name="wordPruningSearchManager"
       type="edu.cmu.sphinx.decoder.search.WordPruningBreadthFirstSearchManager">
        <property name="linguist" value="lexTreeLinguist"/>
        <property name="pruner" value="simplePruner"/>
        <property name="scorer" value="threadedScorer"/>
        <property name="logMath" value="myLogMath"/>
        <property name="growSkipInterval" value="0"/>
        <property name="activeListManager" value="activeListManager"/>
        <property name="checkStateOrder" value="false"/>
        <property name="buildWordLattice" value="false"/>
        <property name="acousticLookaheadFrames" value="2.0"/>
        <property name="keepAllTokens" value="true"/>
        <property name="relativeBeamWidth" value="\${relativeBeamWidth}"/>
  </component>

<!-- ************************************************************ -->
<!--                          Linguist                            -->
<!-- ************************************************************ -->

<component name="lexTreeLinguist"
        type="edu.cmu.sphinx.linguist.lextree.LexTreeLinguist">
        <property name="acousticModel" value="myAcModel"/>
        <property name="unitManager" value="myUnitManager"/>
        <property name="logMath" value="\${logmath}"/>
        <property name="languageModel" value="myNgramModel"/>
        <property name="dictionary" value="myDictionary"/>
        <property name="cacheSize" value="0"/>
        <property name="addFillerWords" value="true"/>
        <property name="generateUnitStates" value="false"/>
        <property name="wantUnigramSmear" value="false"/>
        <property name="wordInsertionProbability" value="\${wordInsertionProbability}"/>
        <property name="languageWeight" value="\${languageWeight}"/>
  </component>


<!-- ************************************************************ -->
<!--                       Acoustic Model                         -->
<!-- ************************************************************ -->

<component name="myAcModel" type="edu.cmu.sphinx.linguist.acoustic
```

```
.tiedstate.TiedStateAcousticModel">
        <property name="loader" value="sphinx3Loader"/>
        <property name="unitManager" value="myUnitManager"/>
  </component>

<!-- ************************************************************ -->
<!--                       Sphinx3Loader                          -->
<!-- ************************************************************ -->

<component name="sphinx3Loader" type="edu.cmu.sphinx.linguist.acoustic
.tiedstate.Sphinx3Loader">
        <property name="logMath" value="\${logmath}"/>
        <property name="unitManager" value="myUnitManager"/>
        <property name = "location" value =
           "resource:/MYACMODEL_16gau_300s_13dCep_48k_40mel_1Hz_24000Hz"/>
        <property name = "modelDefinition" value = "etc/tutorial.300.mdef"/>
        <property name = "dataLocation" value = "cd_continuous_300_16/"/>
  </component>


<!-- ************************************************************ -->
<!--                       Language Model                         -->
<!-- ************************************************************ -->

  <component name="myNgramModel"
        type="edu.cmu.sphinx.linguist.language.ngram.SimpleNGramModel">
        <property name="location" value="frases.trigram.arpa"/>
        <property name="logMath" value="\${logmath}"/>
        <property name="dictionary" value="myDictionary"/>
        <property name="maxDepth" value="3"/>
  </component>


<!-- ************************************************************ -->
<!--                         Dictionary                           -->
<!-- ************************************************************ -->

<!--                       Fast Dictionary                         -->

 <component name="fastDictionary"
type="edu.cmu.sphinx.linguist.dictionary.FastDictionary">
        <property name="dictionaryPath" value="resource:
         /MYACMODEL_16gau_300s_13dCep_48k_40mel_1Hz_24000Hz/dict/treinoma1.dic"/>
        <property name="fillerPath" value="resource:
         /MYACMODEL_16gau_300s_13dCep_48k_40mel_1Hz_24000Hz/dict/treinoma1.filler"/>
        <property name="addSilEndingPronunciation" value="false"/>
        <property name="allowMissingWords" value="true"/>
```

```
        <property name="unitManager" value="myUnitManager"/>
    </component>


<!--                       Full Dictionary                             -->

    <component name="myDictionary"
          type="edu.cmu.sphinx.linguist.dictionary.FullDictionary">
        <property name="dictionaryPath" value="resource:
      /MYACMODEL_16gau_300s_13dCep_48k_40mel_1Hz_24000Hz/dict/treinoma1.dic"/>
        <property name="fillerPath" value="resource:
         /MYACMODEL_16gau_300s_13dCep_48k_40mel_1Hz_24000Hz/dict/treinoma1.filler"/>
        <property name="addSilEndingPronunciation" value="false"/>
        <property name="wordReplacement" value="\&lt;sil\&gt;"/>
        <property name="allowMissingWords" value="true"/>
        <property name="unitManager" value="myUnitManager"/>
    </component>


<!-- ************************************************************ -->
<!--                            Pruner                            -->
<!-- ************************************************************ -->

<component name="trivialPruner"
          type="edu.cmu.sphinx.decoder.pruner.SimplePruner">
</component>


<!-- ************************************************************ -->
<!--                            Scorer                            -->
<!-- ************************************************************ -->

    <component name="threadedScorer"
         type="edu.cmu.sphinx.decoder.scorer.ThreadedAcousticScorer">
        <property name="numThreads" value="0"/>
        <property name="scoreablesKeepFeature" value="true"/>
        <property name="frontend" value="\${frontend}"/>
        <property name="isCpuRelative" value="true"/>
        <property name="minScoreablesPerThread" value="10"/>
    </component>

<!-- ************************************************************ -->
<!--                          Log Math                            -->
<!-- ************************************************************ -->

  <component name="myLogMath"
        type="edu.cmu.sphinx.util.LogMath">
```

```
        <property name="logBase" value="1.0001"/>
        <property name="useAddTable" value="true"/>
  </component>


<!-- ************************************************************ -->
<!--                        Unit Manager                          -->
<!-- ************************************************************ -->

<component name="myUnitManager"
        type="edu.cmu.sphinx.linguist.acoustic.UnitManager">
</component>


<!-- ************************************************************ -->
<!--                      Active List Manager                     -->
<!-- ************************************************************ -->

  <component name="activeListManager"
        type="edu.cmu.sphinx.decoder.search.SimpleActiveListManager">
        <propertylist name="activeListFactories">
           <item>unitExitActiveList</item>
           <item>wordActiveList</item>
           <item>wordActiveList</item>
           <item>activeList</item>
           <item>activeList</item>
           <item>activeList</item>
        </propertylist>
  </component>

<!-- ************************************************************ -->
<!--                      unitExitActiveList                      -->
<!-- ************************************************************ -->

  <component name="unitExitActiveList"
        type="edu.cmu.sphinx.decoder.search.PartitionActiveListFactory">
        <property name="absoluteBeamWidth" value="-1"/>
        <property name="logMath" value="\${logmath}"/>
        <property name="relativeBeamWidth" value="\${relativeBeamWidth}"/>
  </component>

<!-- ************************************************************ -->
<!--                        wordActiveList                        -->
<!-- ************************************************************ -->

  <component name="wordActiveList"
        type="edu.cmu.sphinx.decoder.search.WordActiveListFactory">
```

```
        <property name="absoluteBeamWidth" value="21"/>
        <property name="logMath" value="\${logmath}"/>
        <property name="relativeBeamWidth" value="1E-25"/>
  </component>


<!-- ************************************************************ -->
<!--                          activeList                          -->
<!-- ************************************************************ -->

  <component name="activeList"
        type="edu.cmu.sphinx.decoder.search.PartitionActiveListFactory">
        <property name="absoluteBeamWidth" value="\${absoluteBeamWidth}"/>
        <property name="logMath" value="\${logmath}"/>
        <property name="relativeBeamWidth" value="\${relativeBeamWidth}"/>
  </component>


<!-- ************************************************************ -->
<!--                         My Front End                         -->
<!-- ************************************************************ -->

  <component name="myFrontEnd"
        type="edu.cmu.sphinx.frontend.FrontEnd">
        <propertylist name="pipeline">
            <item>streamCepstrumSource</item>
            <item>batchCMN</item>
            <item>featureExtraction</item>
        </propertylist>
  </component>

<component name="streamCepstrumSource"
        type="edu.cmu.sphinx.frontend.util.StreamCepstrumSource">
        <property name="frameSizeInMs" value="25"/>
        <property name="frameShiftInMs" value="10"/>
        <property name="cepstrumLength" value="13"/>
        <property name="sampleRate" value="48000"/>
  </component>

  <component name="batchCMN"
        type="edu.cmu.sphinx.frontend.feature.BatchCMN">
  </component>

  <component name="featureExtraction"
        type="edu.cmu.sphinx.frontend.feature.DeltasFeatureExtractor">
  </component>
```

```
  <!-- ************************************************************ -->
  <!--                    Instrumentation                           -->
  <!-- ************************************************************ -->

  <!-- Accuracy -->

  <component name="accuracyTracker"
        type="edu.cmu.sphinx.instrumentation.BestPathAccuracyTracker">
        <property name="recognizer" value="\${recognizer}"/>
        <property name="showAlignedResults" value="true"/>
        <property name="showRawResults" value="false"/>
  </component>

  <component name="speedTracker"
        type="edu.cmu.sphinx.instrumentation.SpeedTracker">
        <property name="recognizer" value="\${recognizer}"/>
        <property name="frontend" value="\${frontend}"/>
  </component>

  <component name="memoryTracker"
        type="edu.cmu.sphinx.instrumentation.MemoryTracker">
        <property name="recognizer" value="\${recognizer}"/>
  </component>

  <component name="recognizerMonitor"
        type="edu.cmu.sphinx.instrumentation.RecognizerMonitor">
        <property name="recognizer" value="\${recognizer}"/>
        <propertylist name="allocatedMonitors">
        </propertylist>
  </component>

  <component name="beamFinder"
        type="edu.cmu.sphinx.instrumentation.BeamFinder">
        <property name="showDetails" value="true"/>
        <property name="logMath" value="\${logmath}"/>
        <property name="showSummary" value="true"/>
        <property name="recognizer" value="\${recognizer}"/>
        <property name="enable" value="false"/>
  </component>

</config>
```

A.11 Arquivo de execução de teste

O arquivo `build.xml` é o arquivo para execução de testes. Segue um exemplo deste arquivo:

```
<?xml version="1.0" encoding="UTF-8"?>

<project basedir="."  name="My Tests">

<description>
by Felipe Martins
</description>

    <!-- ********************************************************** -->
    <!-- *                                                        * -->
    <!-- * Properties common to all tests                         * -->
    <!-- *                                                        * -->
    <!-- ********************************************************** -->

        <property name="top_dir" value=".."/>
        <path id="libs">
            <fileset dir="lib" includes="${top_dir}/**/jsapi.jar"/>
       </path>

       <property name="build_dir" value="${top_dir}/bld"/>
       <property name="classes_dir" value="${build_dir}"/>

        <property name="classpath1" value="${classes_dir}:${top_dir}/lib/
            MYACMODEL_16gau_300s_13dCep_48k_40mel_1Hz_24000Hz.jar"/>

       <property name="skip" value="0"/>
       <property name="logger_props" value=""/>

       <property name="initial_heap_size" value="200m"/>
       <property name="maximum_heap_size" value="1024m"/>
       <property name="jit" value="client"/>
       <property name="gc_log_file" value="gc.txt"/>

       <property name="batch_main" value=
          "edu.cmu.sphinx.tools.batch.BatchModeRecognizer"/>

<!-- ********************************************************** -->
<!--                      Config Files                          -->
<!-- ********************************************************** -->

     <property name="config1" value="config1.xml"/>
```

```
<!-- ************************************************************ -->
<!--                        Batch Files                           -->
<!-- ************************************************************ -->

    <property name="batch1" value="wavlist1.batch"/>

<!-- ********************************************************** -->
<!-- *                                                        * -->
<!-- * Teste                                                  * -->
<!-- *                                                        * -->
<!-- ********************************************************** -->

        <target name="my_trigramtest1"
        description="teste">
           <java classpath="${classpath1}"
           classname="${batch_main}"
               fork="true">
               <jvmarg value="-da"/>
               <jvmarg value="-${jit}"/>
               <jvmarg value="-ms${initial_heap_size}"/>
               <jvmarg value="-mx${maximum_heap_size}"/>
               <sysproperty key="batch[skip]" value="${skip}"/>
               <arg value="${config1}"/>
               <arg value="${batch1}"/>
           </java>
        </target>
</project>
```

Apêndice B

Tutorial CMU Sphinx

B.1 *Downloads*

Para começar a trabalhar com o ambiente do CMU Sphinx é necessário realizar os *downloads* das seguintes ferramentas:

- SphinxBase
- SphinxTrain
- CMUclmtk
- Sphinx-4

Além disso, devem estar instaladas na máquina as ferramentas JAVA e interpretador de Perl.

B.2 Preparação do sistema

Após a instalação de todos os componentes citados na Seção B.1 num mesmo diretório, denominado neste trabalho como `tutorial`, deve-se executar o seguinte comando Perl para preparação da árvore de diretórios do SphinxTrain:

```
perl SphinxTrain/scripts_pl/setup_SphinxTrain.pl -task tutorial
```

B.3 Arquivos externos

Após a construção da árvore de diretórios, os seguintes arquivos devem ser colocados no diretório `tutorial/etc`:

- `tutorial_train.fileids` - arquivo com os nomes dos arquivos de áudio que serão usados no treinamento;
- `tutorial_train.transcription` - transcrição dos arquivos de áudio descritos no arquivo `tutorial_train.fileids`;
- `tutorial.dic` - arquivo com o dicionário de palavras e suas respectivas transcrições fonéticas;
- `tutorial.filler` - arquivo com dicionários de eventos que não são falas - silêncio, por exemplo;
- `tutorial.phone` - lista de todos os fones possíveis.

Os arquivos de áudio que serão utilizados no treinamento (descritos no arquivo `tutorial_train.fileids`) deverão ser colocados no diretório `tutorial/wav`.

B.4 Extração de características

O arquivo `feat.params` localizado no diretório `tutorial/etc` contém todos os parâmetros referêntes à extração de características (coeficientes MFCC) dos sinais de áudio para treinamento e teste.

Para extrair os coeficientes, o seguinte comando deve ser executado:

```
"sphinxbase/bin/Release/sphinx_fe" -c etc/tutorial_train.fileids -di
wav -do feat -ei wav -eo mfc -argfile etc/feat.params
```

Este comando deverá gerar os arquivos com extensão `.mfc` localizados no diretório `tutorial/feat`. Após a execução deste comando, deve haver o mesmo número de arquivos `.mfc` que arquivos `.wav` utilizados para treinamento.

B.5 Treinamento do modelo acústico

Caso o projetista desejar utilizar questões fonéticas próprias, o arquivo de questões fonéticas, denominado `tutorial.tree_questions`, deve ser colocado no diretório `tutorial/model_architecture`.

O arquivo `sphinx_train.cfg`, localizado no diretório `tutorial/etc` contém as variáveis referentes aos parâmetros do treinamendo do modelo acústico. As principais variáveis deste arquivo a serem alteradas são as seguintes:

- `CFG_MAKE_QUESTS` - variável *booleana* que define ou não a utilização de questões fonéticas próprias. Caso seja setada com valor "no", será utilizado o arquivo de questões fonéticas próprias. Se setada com o valor "yes", o arquivo de questões fonéticas será gerado automaticamente.
- `CFG_N_TIED_STATES` - representa o número de estados compartilhados.
- `CFG_FINAL_NUM_DENSITIES` - representa o número de componentes gaussianas no modelo.
- `CFG_CONVERGENCE_RATIO` - representa a taxa de convergência do algoritmo de treinamento.
- `CFG_MIN_ITERATIONS` - representa o número mínimo de iterações do algoritmo.
- `CFG_MAX_ITERATIONS` - representa o número máximo de iterações do algoritmo.

Para realizar o treinamento do modelo acústico, basta executar os seguintes comandos em ordem:

`perl scripts_pl/20.ci_hmm/slave_convg.pl` - Treina os fones independentes de contexto.
`perl scripts_pl/30.cd_hmm_untied/slave_convg.pl` - Treina os fones dependentes de contexto, sem estados compartilhados.
`perl scripts_pl/40.buildtrees/slave.treebuilder.pl` - Gera árvore de decisão para compartilhamento de estados.
`perl scripts_pl/45.prunetree/slave.state-tying.pl` - Poda a árvore de decisão e agrupa os estados que serão compartilhados.
`perl scripts_pl/50.cd_hmm_tied/slave_convg.pl` - Retreina os fones dependentes e independentes de contexto com os estados compartilhados.

Se todos os passos não obtiverem nenhum `Fatal Error`, o treinamento foi realizado com sucesso.

B.6 Treinamento do modelo de linguagem

Para o treinamento do modelo de linguagem, primeiramente, é necessário colocar o arquivo da base de texto e o arquivo de dicionário no diretório `tutorial/cmuclmtk` e executar os seguintes comandos:

```
text2idngram -vocab dict.vocab -idngram frases.trigram.idngram -n
3 < frases.txt
```

```
idngram2lm -idngram frases.trigram.idngram -vocab dict.vocab -arpa
frases.trigram.arpa -context frases.ccs -absolute -n 3
```

Estes comandos geram o modelo do tipo trigrama. Para gerar modelos do tipo bigrama, basta substituir os números "3"por "2"em cada um dos comandos.

Se não houver retorno de mensagem de erro, o modelo de linguagem foi criado corretamente e deve haver um arquivo denominado `frases.trigram.arpa` no próprio diretório. Este arquivo deve ser copiado para o diretório de testes.

B.7 Empacotamento do modelo acústico

Depois que o treinamento do modelo acústico é realizado, é necessário realizar a cópia de alguns arquivos para os diretórios corretos. Primeiramente, os seguintes diretórios devem ser criados:

- `sphinx4-1.0beta5/models/acoustic/my_model/cd_continuous_300_16`
  Para este diretório deve-se copiar todos os arquivos do diretório: `tutorial/model_parameters/tutorial.cd_cont_300_16`

- `sphinx4-1.0beta5/models/acoustic/my_model/dict`
  Para este diretório deve-se copiar os arquivos `.dic` e `.filler`.

- `sphinx4-1.0beta5/models/acoustic/my_model/etc`
  Para este diretório deve-se copiar os arquivos de extensão *.mdef* do diretório `tutorial/model_architecture`.

Posteriormente, deve ser criado o arquivo de configuração da Seção A.8 do Apêndice A com o nome `MYACMODEL_16gau_300s_13dCep_48k_40mel_1Hz_24000Hz`.

O arquivo `build.xml` do diretório `/sphinx4-1.0beta5` deve ser alterado da seguinte maneira:

- Depois do comentário "*For generating the WSJ and TIDIGITS models.*", seguindo o padrão, adicionar:

```
<property name="mymodel_name" value=
   "MYACMODEL_16gau_300s_13dCep_48k_40mel_1Hz_24000Hz"/>
<property name="mymodel_data_dir" value="models/acoustic/my_model"/>
```

- Depois do comentário "*Builds the TIDIGITS and WSJ acoustic model files*", seguindo o padrão, adicionar:

```
<antcall target="create_my_model">
  <param name="my_model_data_dir" value="${mymodel_data_dir}"/>
  <param name="my_model_name" value="${mymodel_name}"/>
</antcall>
```

- Depois do comentário "*Deletes all build output and *~ file droppings*", seguindo o padrão, adicionar:

  ```
  <delete file = "${lib_dir}/${mymodel_name}.jar" failonerror = "false"/>
  ```

  Por fim, no mesmo diretório, para gerar o arquivo compactado `.jar`, basta executar o comando `ant`.

B.8 Preparação e execução de testes

Para a execução de testes deve ser criado o diretório de testes. Este diretório deve conter os arquivos `config.xml`, `build.xml`, `frases.trigram.arpa` e o arquivo `wavlist1.batch`. Os arquivos devem seguir todos os padrões descritos no Apêndice A.

Para rodar o teste definido pelo arquivo `build.xml`, basta executar o comando `ant`.

Apêndice C

Manual Sarvox

C.1 Sobre

SARVOX - Sistema de Análise de Reconhecimento de Voz Baseado em CMU Sphinx

O SARVOX foi desenvolvido como projeto de Dissertação e Pesquisa de Mestrado por Felipe Castro Vieira Martins com o objetivo de possibilitar e facilitar a realização de testes e análises de sistema de reconhecimento de voz baseado em CMU Sphinx, possibilitando facilmente o ajuste de parâmetros sem necessidade de conhecimento prévio da ferramenta.

Este projeto teve a orientação do Professor Fernando Gil Vianna Resende Junior.

Laboratório de Processamento de Sinais

Programa de Engenharia Elétrica - COPPE - UFRJ

2011-2013

C.2 Página principal

A página principal do sistema consiste basicamente em três elementos:

- Barra superior;
- Menu lateral;
- Quadro principal.

A Figura C.1 mostra a página principal e suas subdivisões.

C.2.1 Barra superior

A barra superior possui unicamente a função de conter o título do sistema e os *links* da instituição de ensino e do laboratório em questão. Esta barra permanece constante durante toda a operação do sistema.

Figura C.1: Página principal

C.2.2 Menu lateral

O menu lateral contém os acessos para a utilização de cada uma das ferramentas disponíveis. São elas:

- Página Inicial;
- Treinamento do Modelo Acústico;
- Treinamento do Modelo de Linguagem;
- Testes do Sistema;
- Limpar Sistema;
- Ajuda.

As seções seguintes irão detalhar cada uma das ferramentas disponíveis. O menu lateral permance constante durante toda a operação do sistema.

C.2.3 Quadro principal

O quadro principal compreende a maior área de visualização do sistema. Este quadro irá conter todas as informações, formulários e botões referentes à operação de cada

ferramenta do sistema. Este quadro apresenta um conteúdo variável de acordo com a sua utilização.

C.3 Página inicial

Ao clicar no menu da Página Inicial, o sistema voltará à tela principal de sua inicialização, mostrando as informações básicas sobre o sistema. Ao clicar neste botão, nenhuma informação será perdida ou alterada.

C.4 Treinamendo do modelo acústico

O menu referente ao treinamento do modelo acústico irá direcioná-lo para todas as etapas referentes. A primeira etapa do treinamento do modelo acústico são os *uploads* de todos os arquivos externos ao sistema. A Figura C.2 mostra esta tela com os respectivos campos de *upload*. Basta fazer o *upload* dos arquivos selecionados, escolher se deseja utilizar as questões fonéticas próprias e seguir para a próxima etapa.

A segunda etapa consiste no ajuste dos parâmetros referentes à extração de características. O formulário contido nesta etapa é o da Figura C.3. Após o ajuste dos parâmetros, basta clicar no botão `Extrair Características`. Dependendo da quantidade de arquivos de áudio carregados, este processo pode demorar alguns minutos.

A terceira etapa é o ajuste dos parâmetros referentes ao modelo acústico. Além disso, caso tenha sido escolhido utilizar questões fonéticas próprias, nesta etapa deverá ser feito o *upload* do arquivo de questões fonéticas. Após o ajuste dos parâmetros, o projetista tem a opção de escolher entre treinar a primeira etapa do treinamento ou realizá-lo por completo. Se for escolhido realizar a etapa passo a passo, será possível o acompanhamento das mensagens de retorno do CMU Sphinx para cada etapa de treinamento, sendo possível a visualização de eventuais erros durante o treinamento. No caso do treinamento completo, não será exibida nenhuma informação referente a erros das etapas intermediárias. A Figura C.4 mostra o formulário referente à terceira etapa. E as Figuras C.5, C.6, C.7, C.8 e C.9 mostram as páginas referentes ao treinamento passo a passo, com exemplos de mensagens retornadas pelo CMU Sphinx. O treinamento acústico é um processo demorado e pode durar vários minutos ou até algumas horas.

A quarta etapa consiste no ajuste dos parâmetros de execução do sistema e a realização do empacotamento acústico. Esta página surgirá logo após o término das etapas passo a passo do treinamento acústico ou após o treinamento completo. Depois de ajustados os parâmetros referentes à execução de testes, o projetista deve

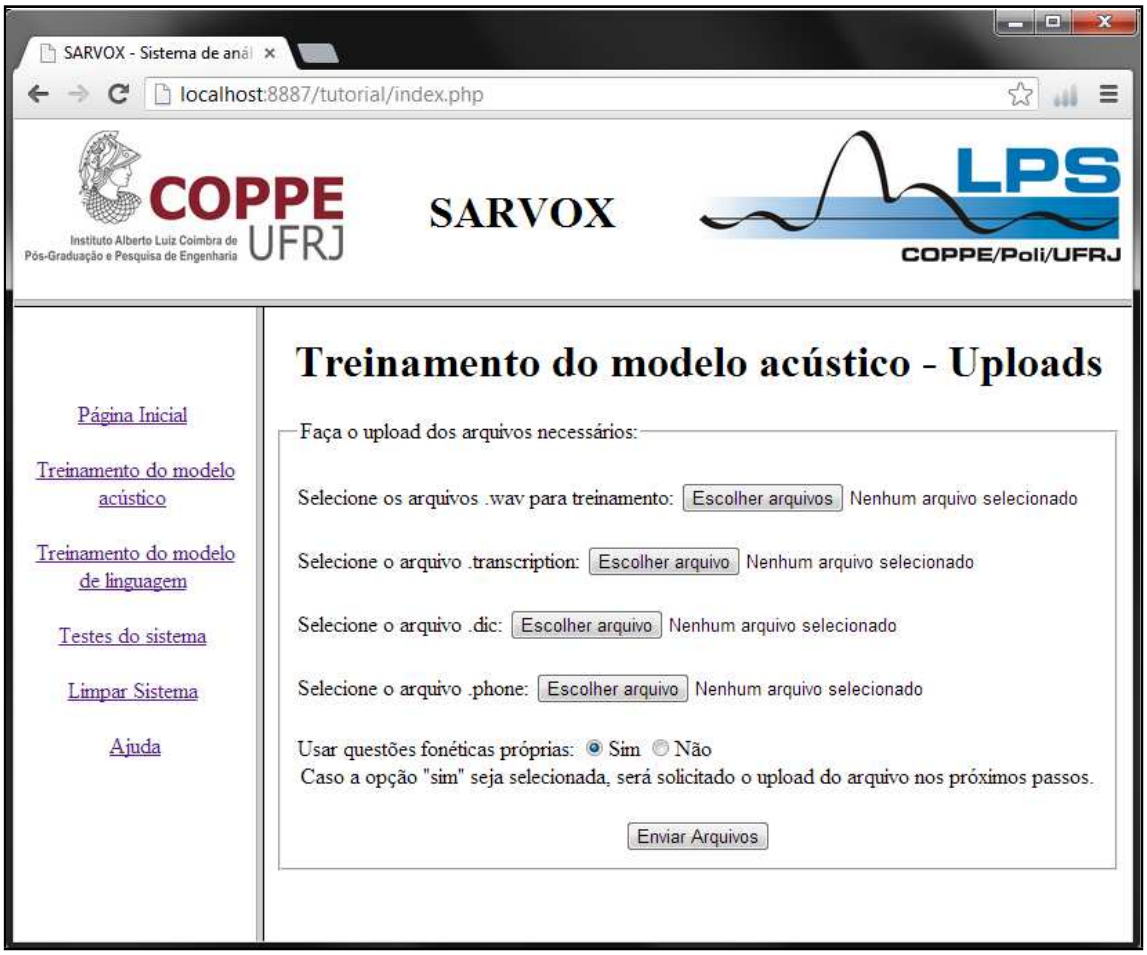


Figura C.2: *Uploads* do treinamento do modelo acústico

clicar no botão para realizar o empacotamento do modelo acústico. A Figura C.9 mostra esta etapa e a Figura C.10 mostra a finalização do processo. Este processo pode demorar vários segundos.

C.5 Treinamendo do modelo de linguagem

O treinamento do modelo de linguagem é bastante simples e não exige muitos passos. O primeiro passo consiste no *upload* dos arquivos necessários e a escolha do tipo do modelo de linguagem: unigrama, bigrama ou trigrama. A Figura C.11 mostra o formulário de *uploads*.

Após o envio dos arquivos, da mesma maneira que o treinamento do modelo acústico, pode-se optar por realizar o treinamento de cada uma das etapas individualmente, visualizando os resultados intermediários, ou de maneira completa. A Figura C.12 mostra as opções de treinamento e as Figuras C.13 e C.14 mostram os passos intermediários, concluindo o processo.

O treinamento do modelo de linguagem pode demorar vários segundos.

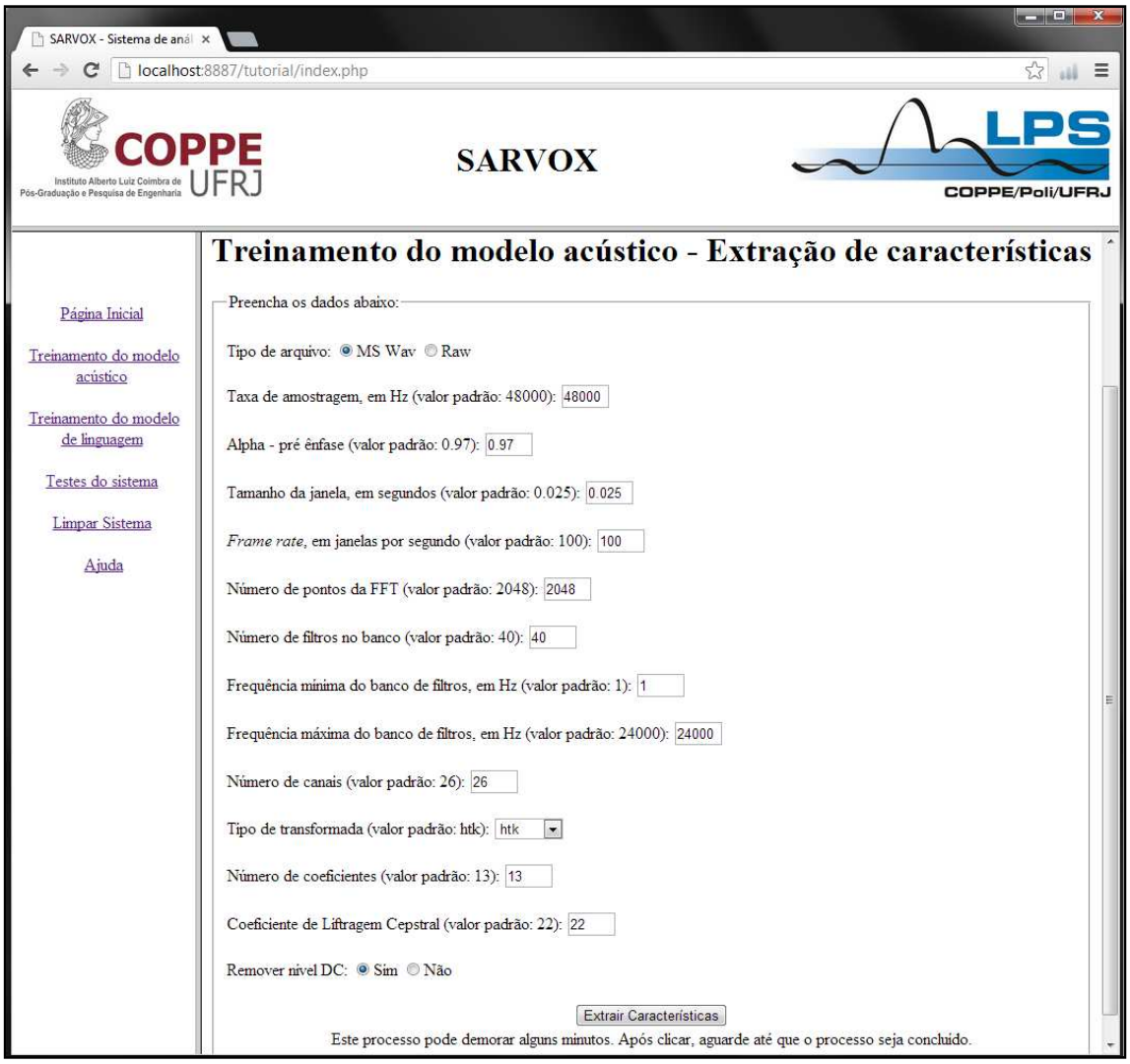


Figura C.3: Formulário para extração de características do treinamento do modelo acústico

C.6 Testes

O menu de testes começa com a realização da escolha do tipo de testes a ser executado. Podem ser realizados testes com arquivo único ou com diversos arquivos. A Figura C.15 mostra a primeira página do menu de testes.

C.6.1 Arquivo único

O teste com arquivo único tem o objetivo de permitir testes simples com o intuito de fazer uma análise mais pontual sobre o reconhecimento. Se escolhido o teste com arquivo único, o projetista deve fazer o *upload* do arquivo de áudio e digitar a transcrição deste áudio no campo correspondente. A transcrição do áudio é facultativa, visto que o objetivo da transcrição é unicamente para que o sistema gere uma pequena estatística de reconhecimento e mostre o tempo de processamento.

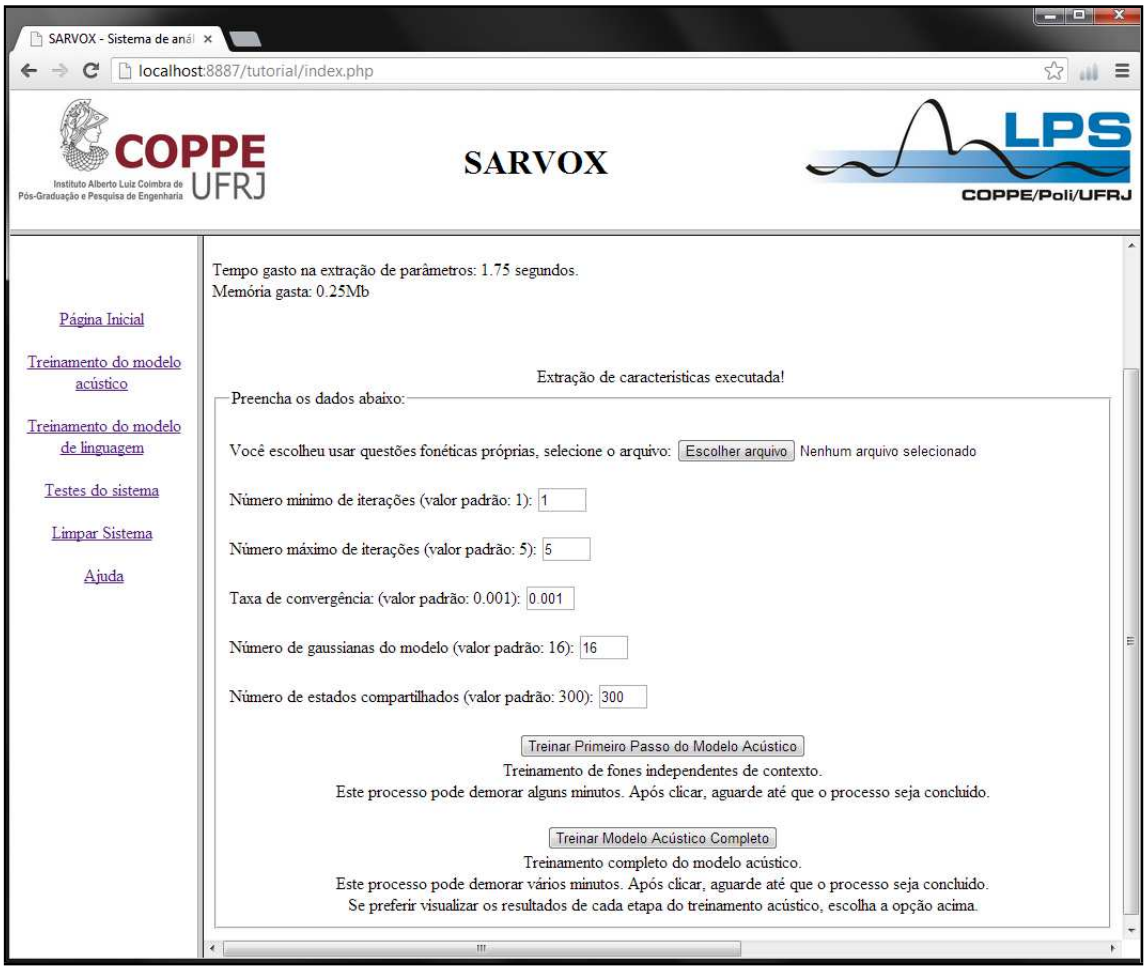


Figura C.4: Formulário para treinamento do modelo acústico

Mas o projetista pode optar por realizar um teste simples cujo único objetivo seja obter o resultado da transcrição do arquivo de áudio em questão. A Figura C.16 mostra o formulário para teste com arquivo único, a Figura C.17 mostra o resultado do reconhecimento simples e a Figura C.18 mostra o resultado com a geração da estatística. Este teste com arquivo único pode demorar alguns segundos.

C.6.2 Arquivos múltiplos

O teste com arquivos múltiplos, também chamados de *batch*, corresponde a um teste com objetivo mais robusto. Este tipo de teste gera uma estatística individual e acumulada para o reconhecimento de diversos arquivos de áudio, utilizando um único teste. Para isso, é necessário realizar o *upload* dos arquivos de áudio a serem reconhecidos e do arquivo que contém a transcrição de todos estes áudios. A Figura C.19 mostra o formulário para testes com arquivos múltiplos. Posteriormente, deve-se, então, realizar o teste propriamente dito. A Figura C.20 mostra esta etapa. E, por fim, a visualização do resultado obtido com suas estatísticas, conforme a Figura C.21.

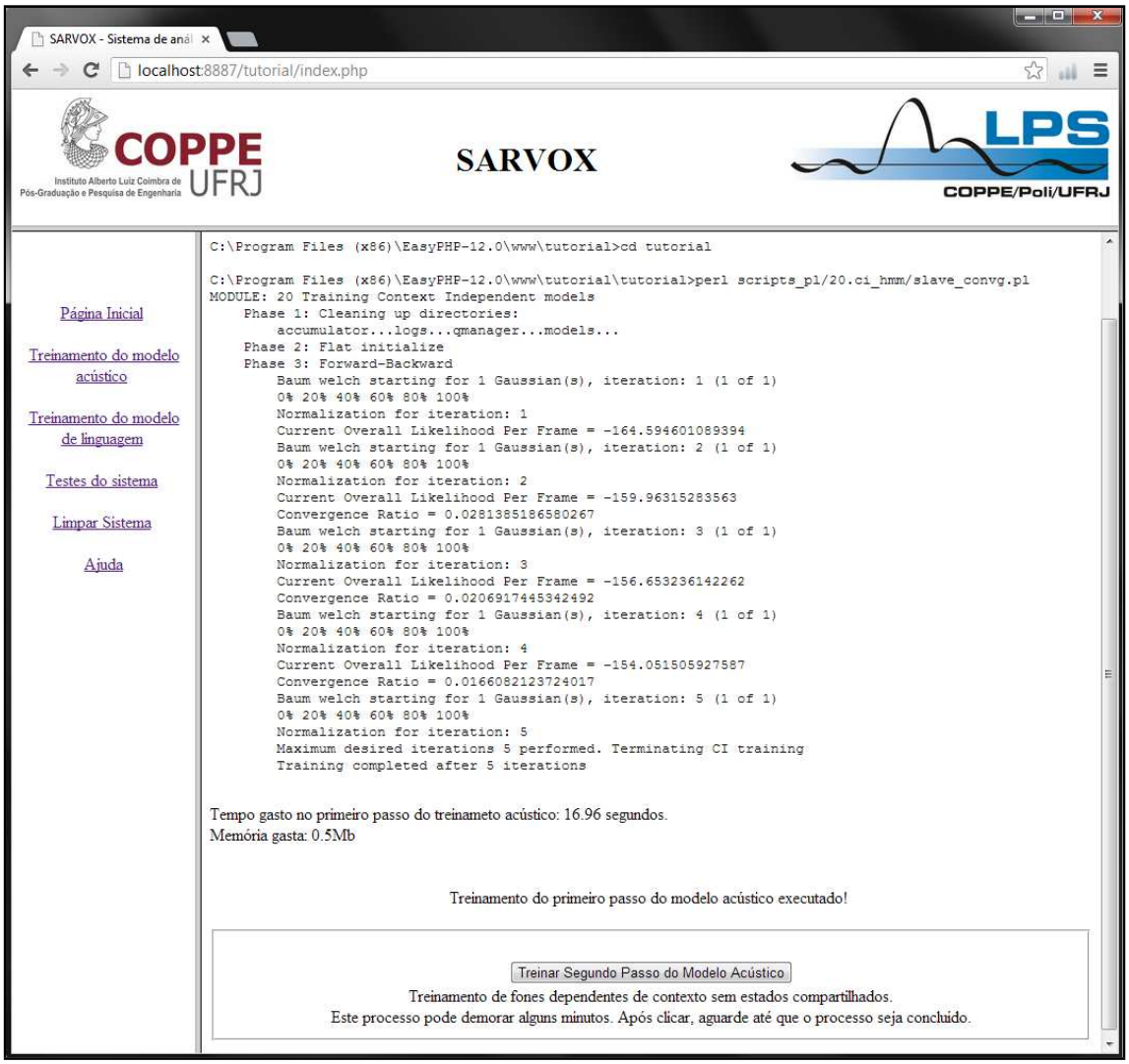


Figura C.5: Primeiro passo do treinamento do modelo acústico

Nota-se que neste teste com arquivos múltiplos as transcrições dos áudios não são facultativas, visto que o objetivo deste teste é justamente obter uma estatística sobre o reconhecimento do sistema.

C.7 Limpar sistema

O menu para limpar o sistema consiste em eliminar todos os arquivos enviados por *upload* e todos os arquivos intermediários criados. A finalidade desta ferramenta é limpar completamente o sistema para que o mesmo esteja preparado para ser retreinado. A seleção deste menu implica apenas em uma confirmação para exclusão dos dados de acordo com a Figura C.22.

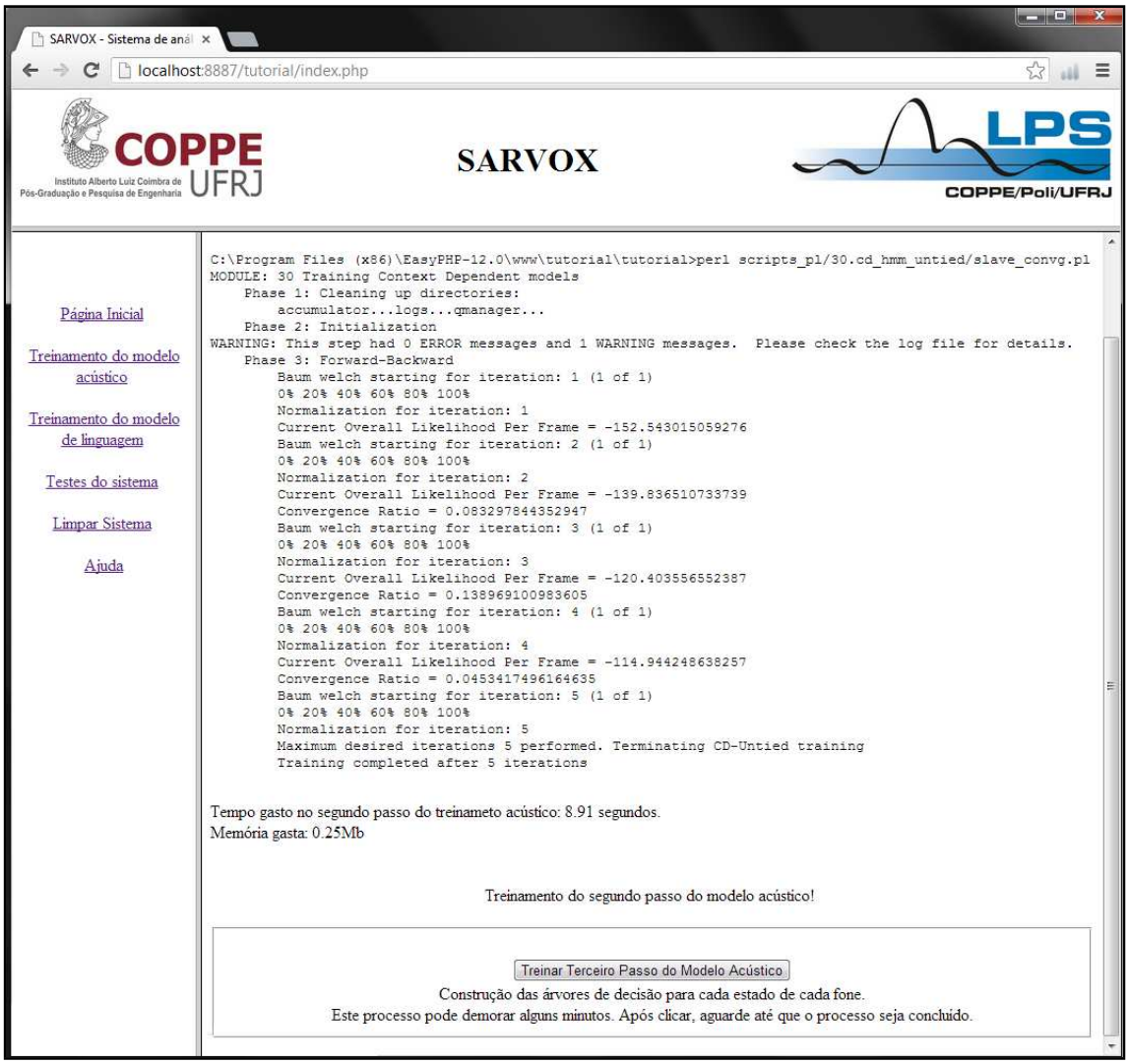


Figura C.6: Segundo passo do treinamento do modelo acústico

C.8 Ajuda

O menu de ajuda contém todas as informações necessárias para utilização do sistema.

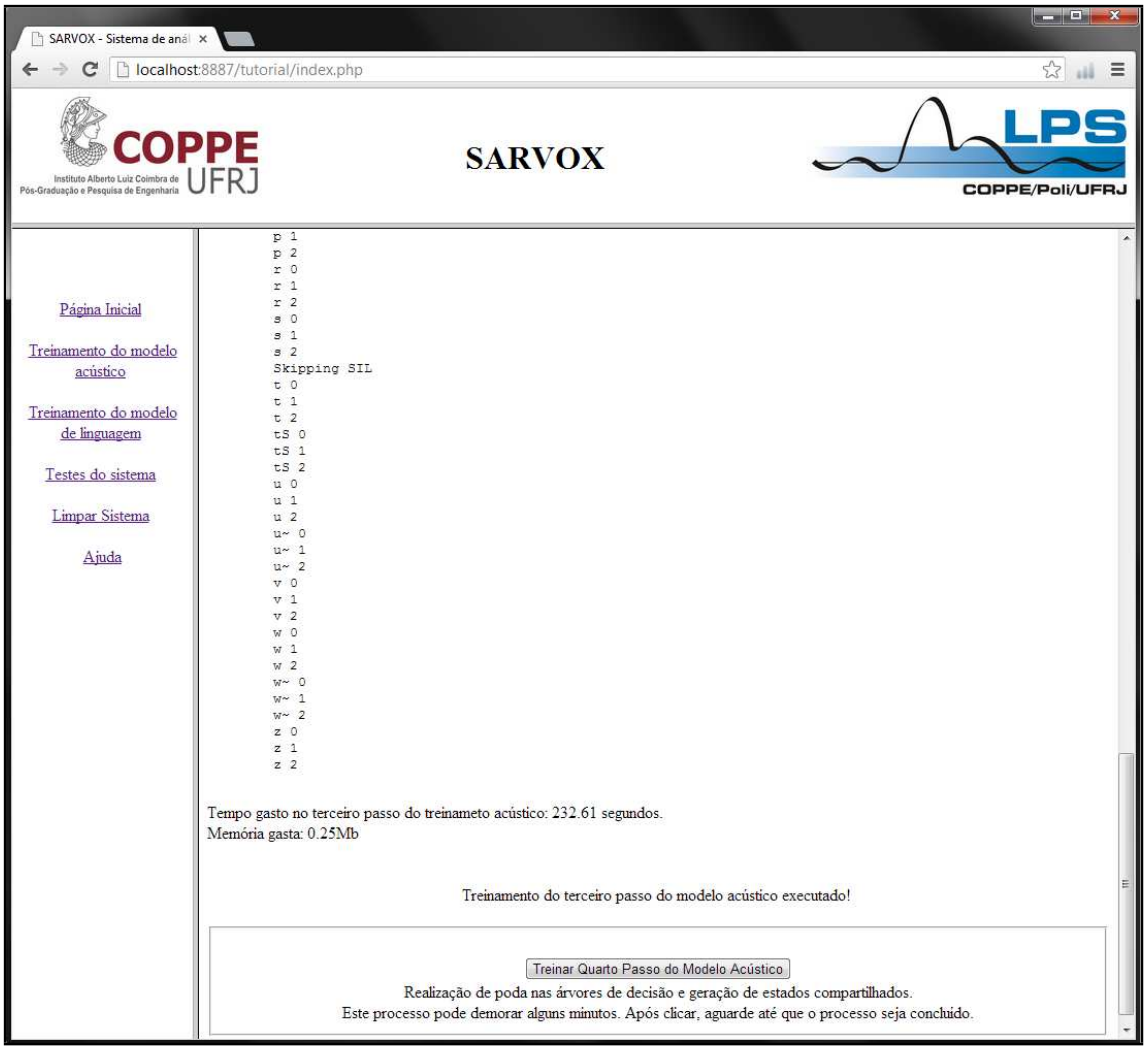


Figura C.7: Terceiro passo do treinamento do modelo acústico

Figura C.8: Quarto passo do treinamento do modelo acústico

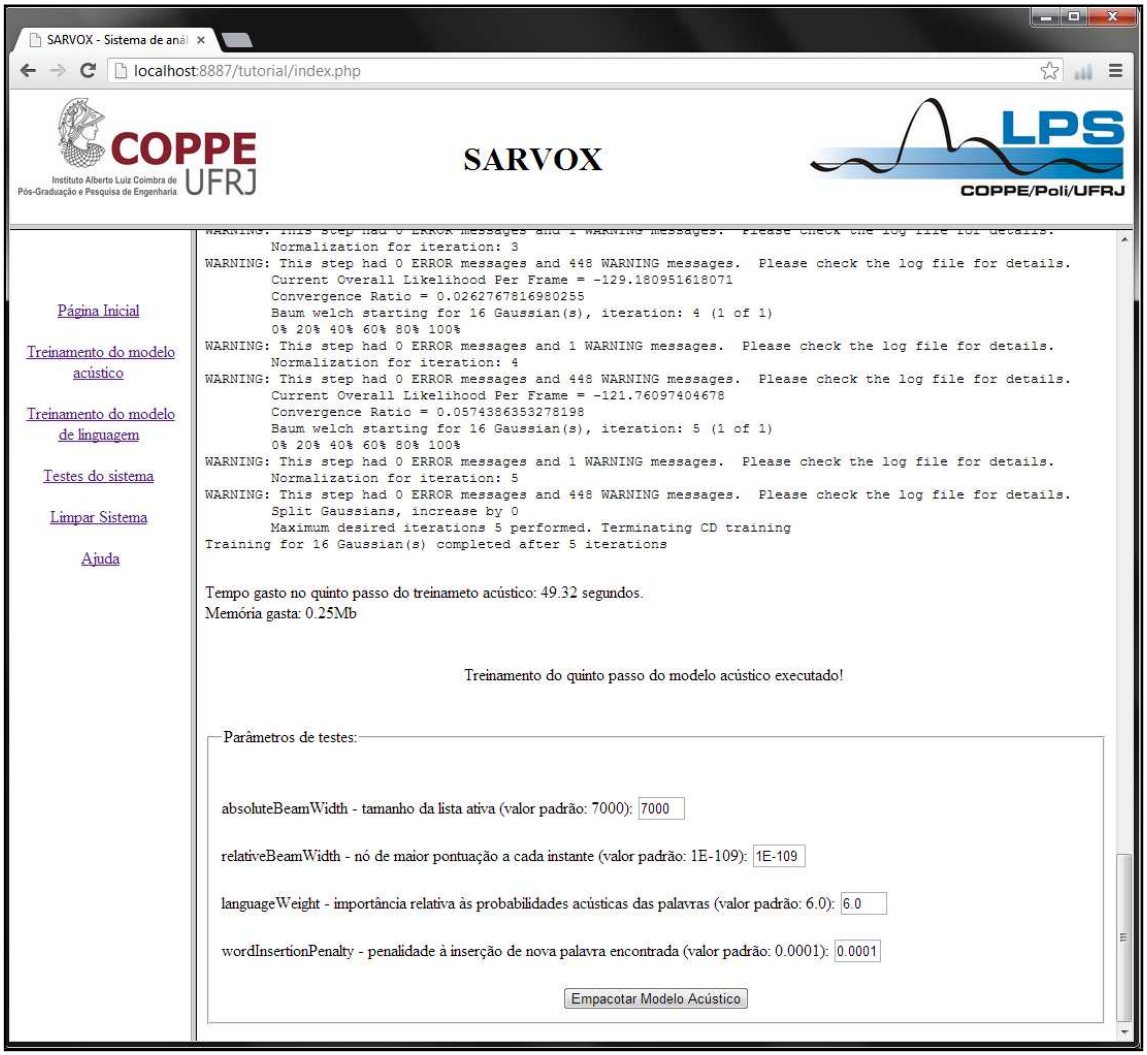


Figura C.9: Quinto passo do treinamento do modelo acústico e empacotamento do modelo acústico

Figura C.10: Finalização do empacotamento do modelo acústico

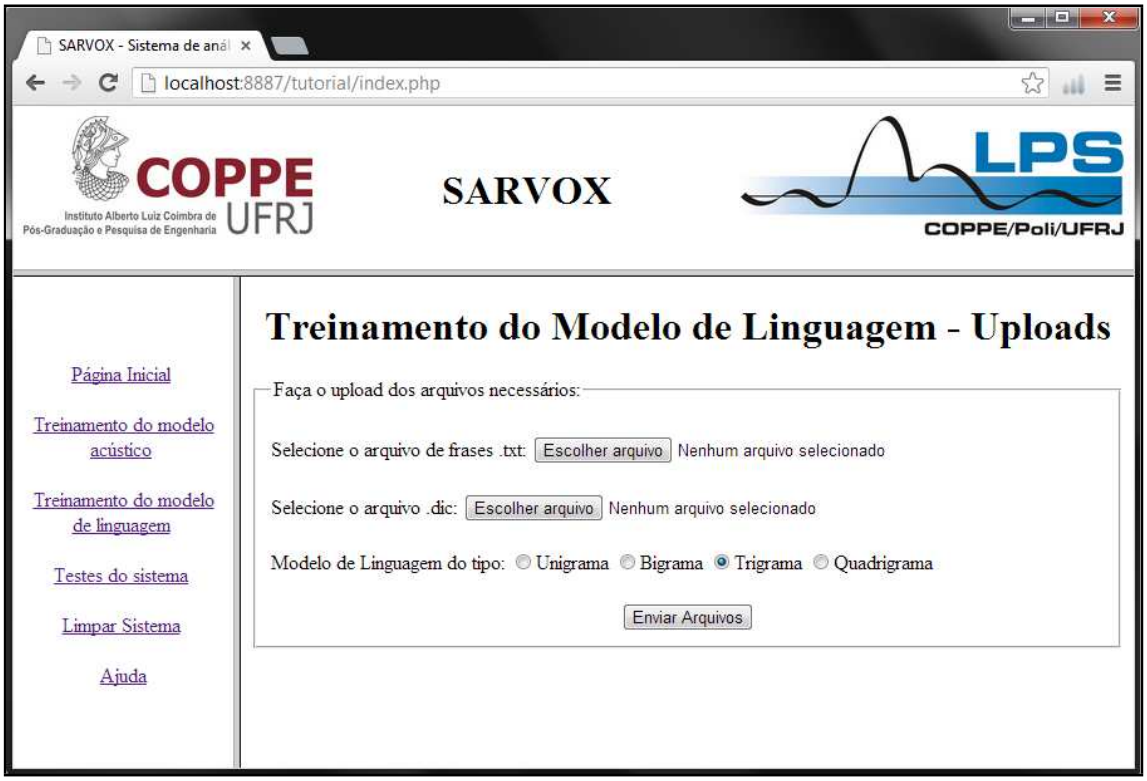


Figura C.11: Formulário de *uploads* do treinamento do modelo de linguagem

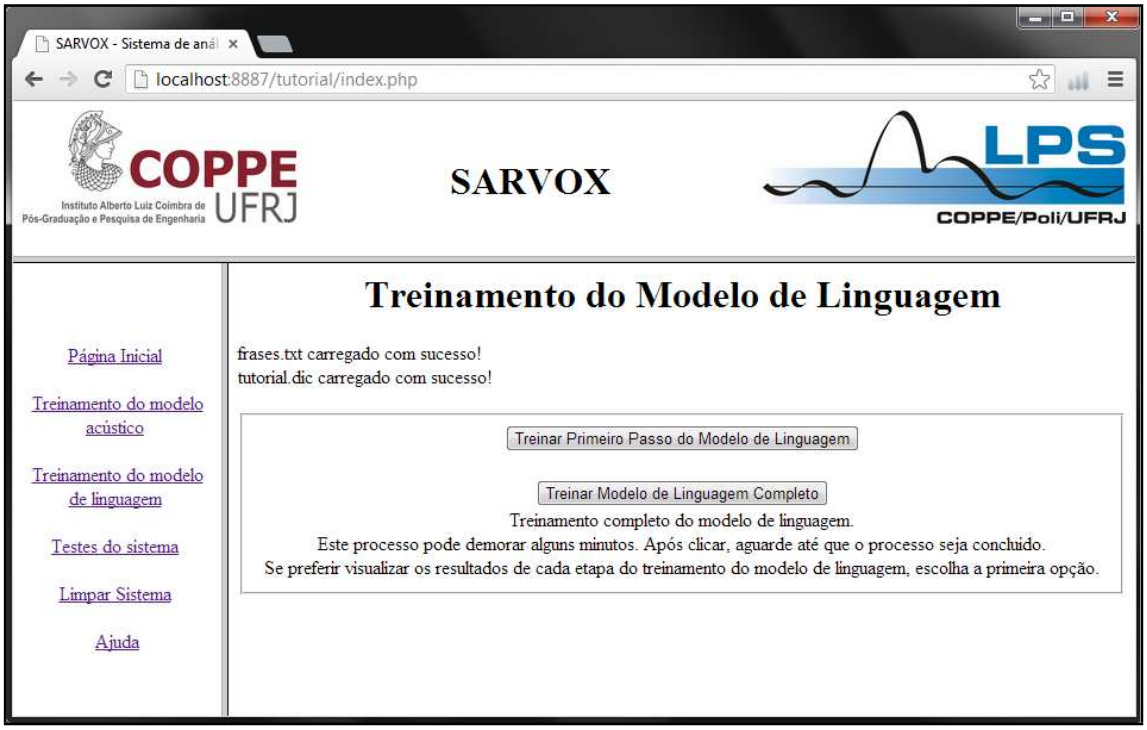


Figura C.12: Opções do treinamento do modelo de linguagem

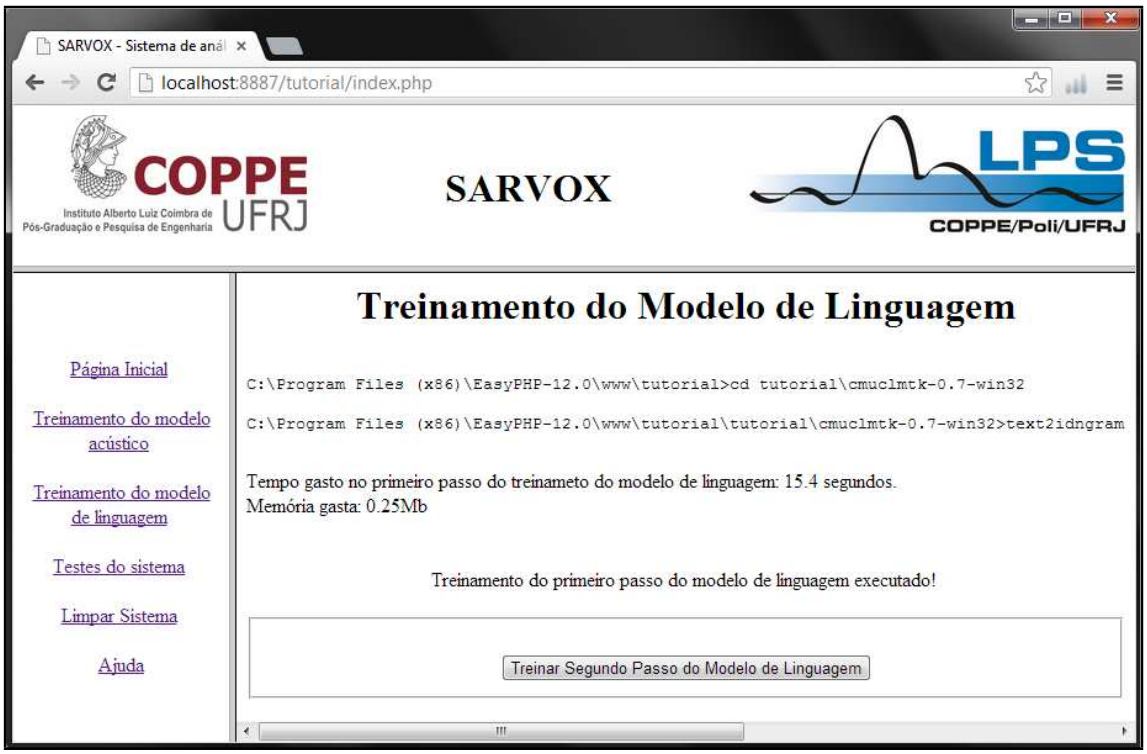


Figura C.13: Primeiro passo do treinamento do modelo de linguagem

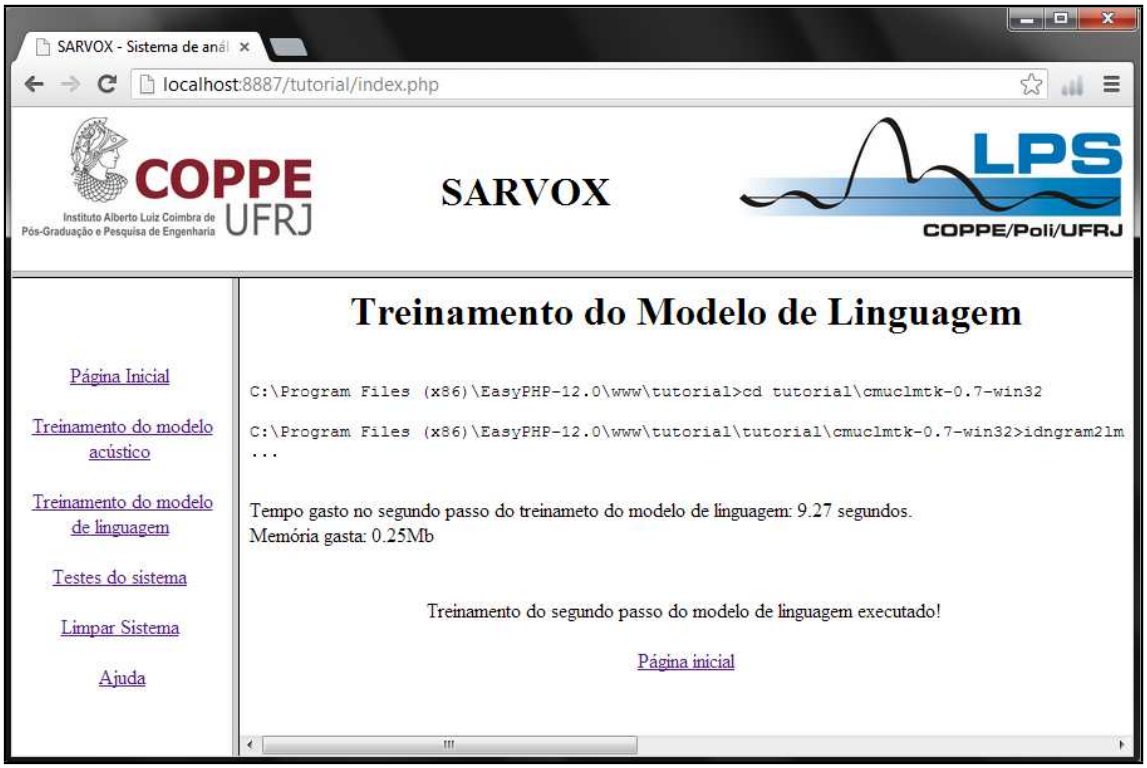


Figura C.14: Segundo passo do treinamento do modelo de linguagem

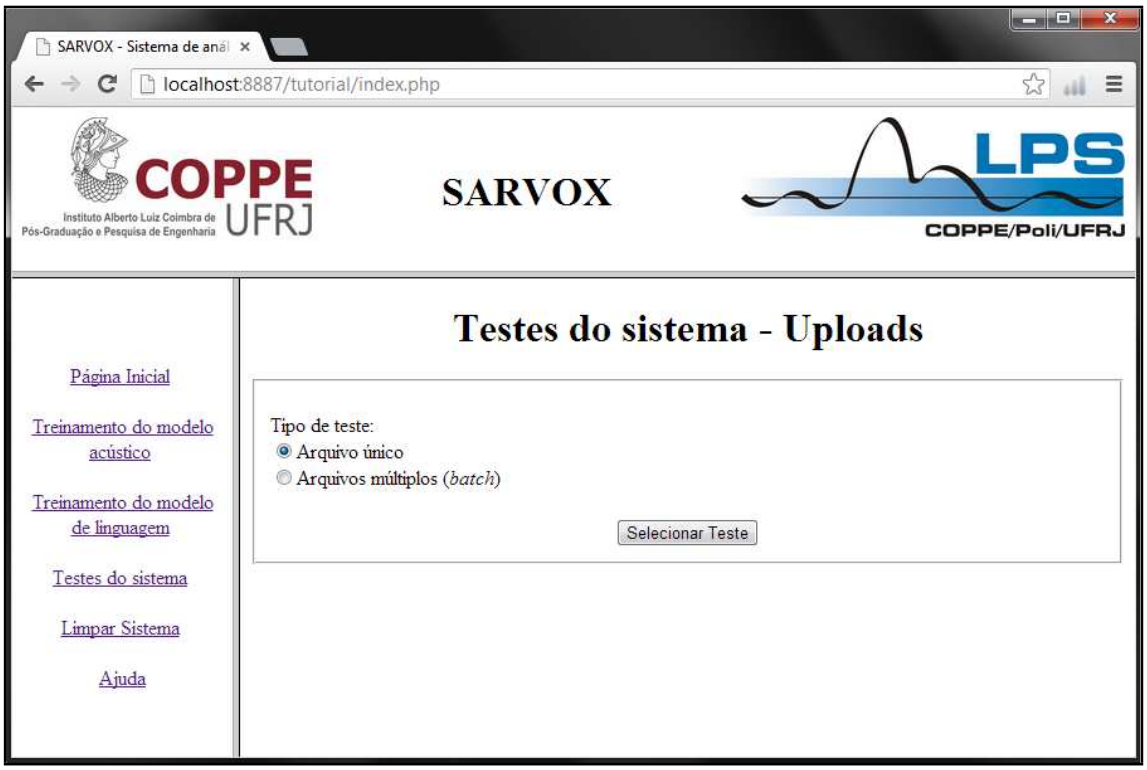


Figura C.15: Página principal de testes

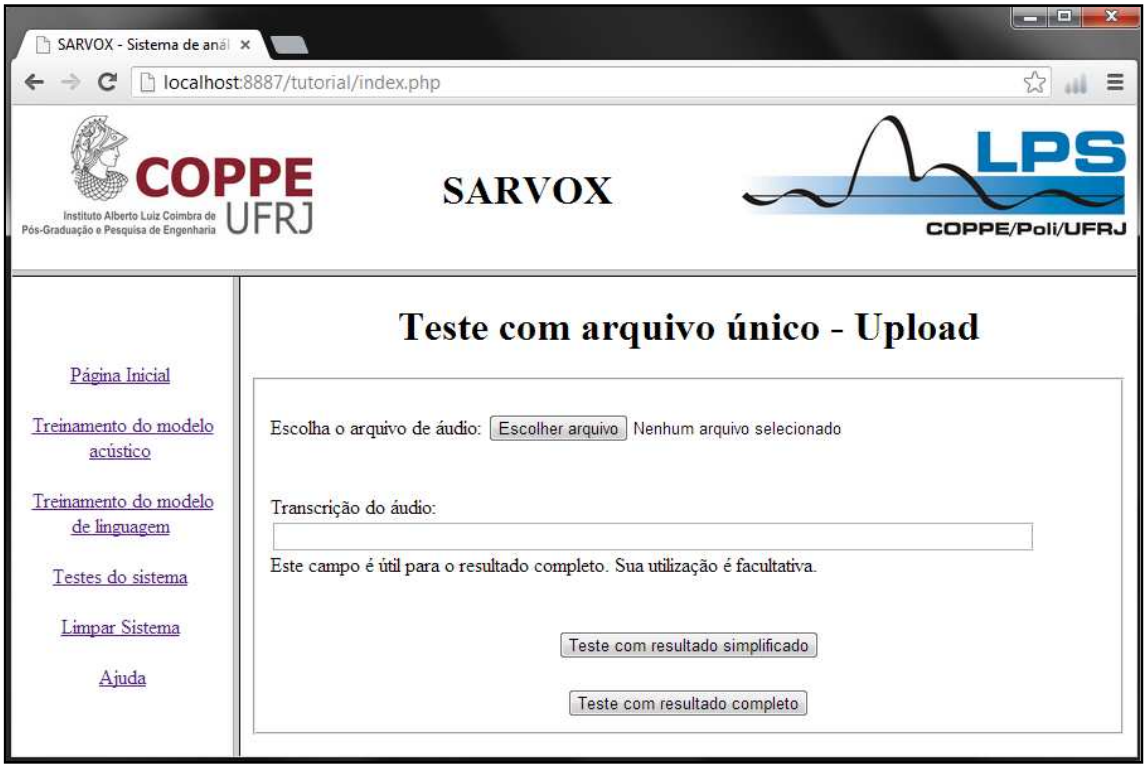


Figura C.16: Teste com arquivo único

Figura C.17: Resultado simplificado do teste com arquivo único

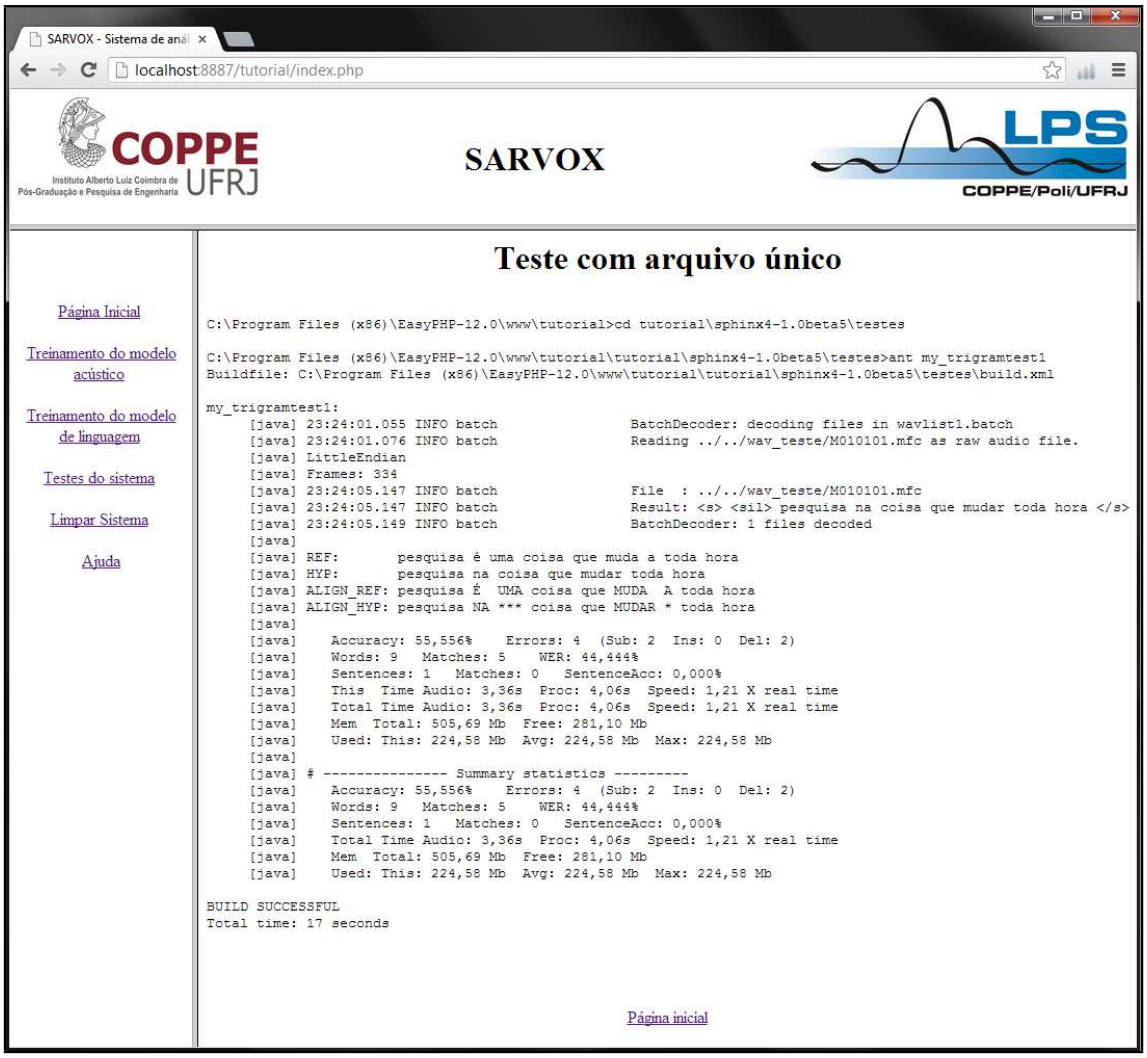


Figura C.18: Resultado completo do teste com arquivo único

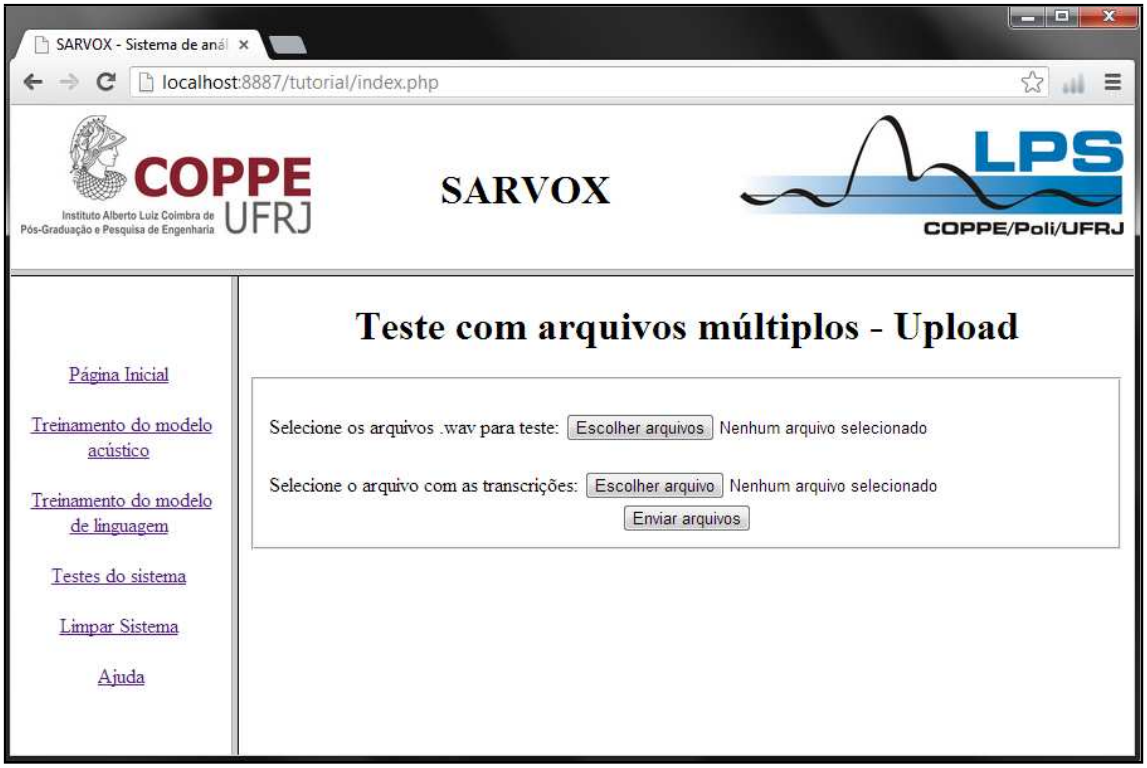


Figura C.19: *Uploads* teste com arquivos múltiplos

Figura C.20: Teste com arquivos múltiplos

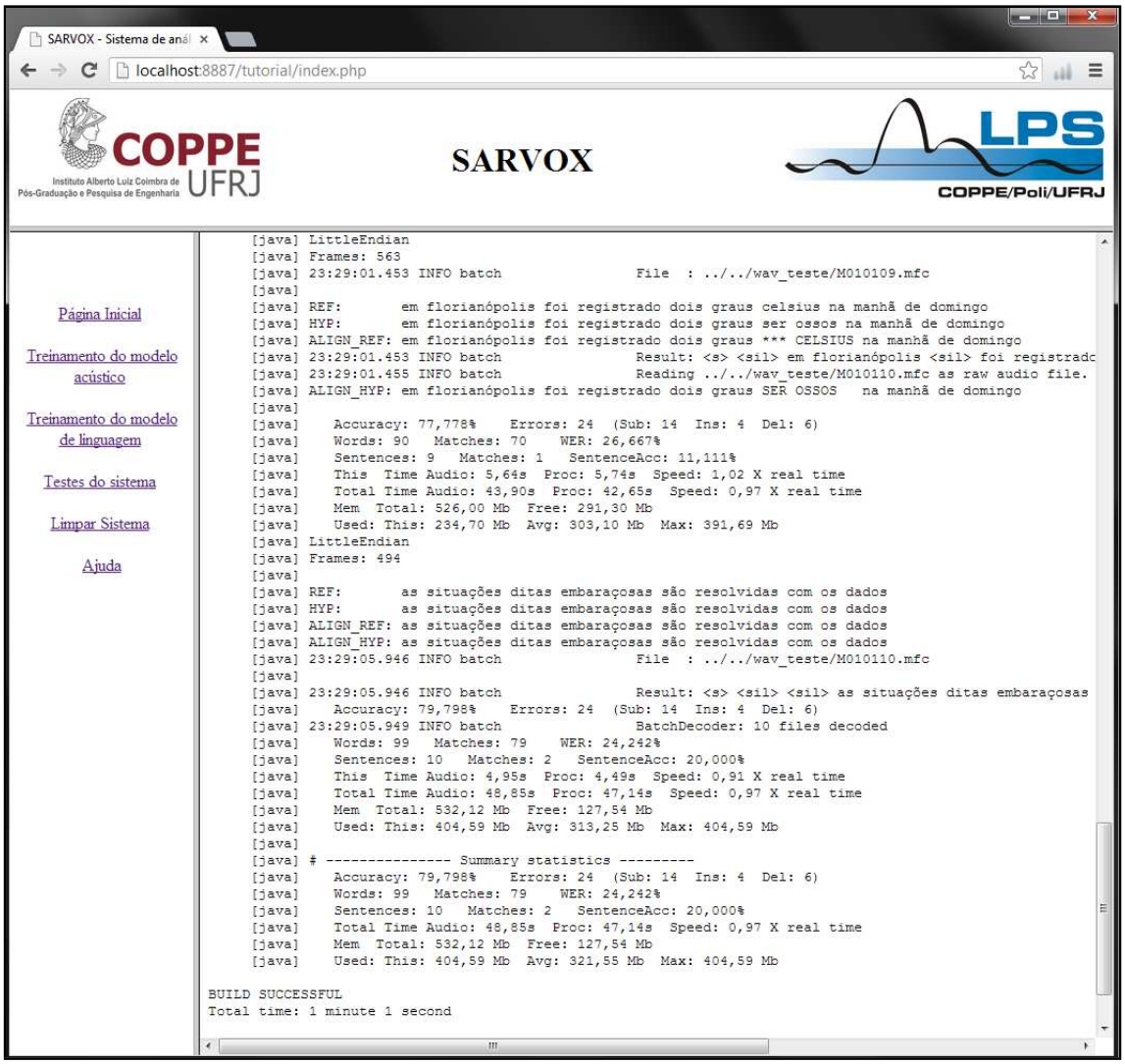


Figura C.21: Resultado do teste com arquivos múltiplos

Figura C.22: Limpar sistema