UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Ferramentas e Recursos Livres para Reconhecimento e Síntese de Voz em Português Brasileiro

NOME DO AUTOR

Nelson Cruz Sampaio Neto

TD_05/2011

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2011

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

NOME DO AUTOR

Nelson Cruz Sampaio Neto

Ferramentas e Recursos Livres para Reconhecimento e Síntese de Voz em Português Brasileiro

TD_05/2011

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2011

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

NOME DO AUTOR

Nelson Cruz Sampaio Neto

Ferramentas e Recursos Livres para Reconhecimento e Síntese de Voz em Português Brasileiro

Tese de Doutorado submetida à Banca Examinadora do Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal do Pará para obtenção do Grau de Doutor em Engenharia Elétrica com ênfase em Computação Aplicada.

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2011

A gravidade explica os movimentos dos planetas, mas não pode explicar quem os colocou em movimento. Deus governa todas as coisas e sabe tudo que é ou que pode ser feito.

Isaac Newton.

Agradecimentos

Agradeço à DEUS pelas conquistas alcançadas.

Um agradecimento especial ao meu orientador Prof. Aldebaro Barreto da Rocha Klautau Jr. pela confiança depositada em mim e por ter me oferecido todo o suporte necessário para a realização deste estudo.

Agradeço também à todos os professores do PPGEE e ao programa de bolsa de estudo da FAPESPA, sem o qual seria impossível o andamento desta tese.

Aos meus pais Fernando e Vera pelo amor, pela paciência, pelo incentivo constante e por terem me proporcionado uma excelente educação, que se tornou a base de todo o meu crescimento pessoal, acadêmico e profissional.

À minha irmã Zenaide e à Ana Carolina por estarem sempre presentes em minha vida, torcendo por mim.

Por fim, agradeço à todos aqueles que direta ou indiretamente apoiaram e ajudaram para a conclusão deste trabalho, principalmente, aos amigos do LAPS pelo ambiente de verdadeira amizade e companheirismo e ao grupo de trabalho do Projeto FalaBrasil: Patrick Silva, Pedro Batista, Renan Moura e Igor Couto.

Resumo

Sistemas de reconhecimento e síntese de voz são constituídos por módulos que dependem da língua e, enquanto existem muitos recursos públicos para alguns idiomas (p.e. Inglês e Japonês), os recursos para Português Brasileiro (PB) ainda são escassos. Outro aspecto é que, para um grande número de tarefas, a taxa de erro dos sistemas de reconhecimento de voz atuais ainda é elevada, quando comparada à obtida por seres humanos. Assim, apesar do sucesso das cadeias escondidas de Markov (HMM), é necessária a pesquisa por novos métodos. Este trabalho tem como motivação esses dois fatos e se divide em duas partes. A primeira descreve o desenvolvimento de recursos e ferramentas livres para reconhecimento e síntese de voz em PB, consistindo de bases de dados de áudio e texto, um dicionário fonético, um conversor grafema-fone, um separador silábico e modelos acústico e de linguagem. Todos os recursos construídos encontram-se publicamente disponíveis e, junto com uma interface de programação proposta, têm sido usados para o desenvolvimento de várias novas aplicações em tempo-real, incluindo um módulo de reconhecimento de voz para a suíte de aplicativos para escritório OpenOffice.org. São apresentados testes de desempenho dos sistemas desenvolvidos. Os recursos aqui produzidos e disponibilizados facilitam a adoção da tecnologia de voz para PB por outros grupos de pesquisa, desenvolvedores e pela indústria. A segunda parte do trabalho apresenta um novo método para reavaliar (*rescoring*) o resultado do reconhecimento baseado em HMMs, o qual é organizado em uma estrutura de dados do tipo *lattice*. Mais especificamente, o sistema utiliza classificadores discriminativos que buscam diminuir a confusão entre pares de fones. Para cada um desses problemas binários, são usadas técnicas de seleção automática de parâmetros para escolher a representação paramétrica mais adequada para o problema em questão.

PALAVRAS-CHAVE: Reconhecimento automático da voz, síntese da voz.

Abstract

Automatic speech recognition and text-to-speech systems have modules that depend on the language and, while there are many public resources for some languages (e.g. English and Japanese), the resources for Brazilian Portuguese (BP) are still limited. Another aspect is that for many tasks the current speech recognition system error rate is still high, when compared to that obtained by humans. Thus, despite the success of hidden Markov models (HMM), it is necessary to investigate new methods. This work has these two facts as motivation and is divided into two parts. The first part describes the resources and free tools developed for BP speech recognition and synthesis, consisting of text and audio databases, phonetic dictionary, grapheme-to-phone converter, syllabification module, language and acoustic models. All of them are publicly available and, together with a proposed application programming interface, have been used for the development of several new real-time applications, including a speech module for the OpenOffice suite. Performance tests are presented for evaluating the developed systems. The resources make easier the adoption of BP speech technologies by other academic groups, developers and industry. The second part of this work presents a new method for rescoring the recognition result obtained via HMMs, with the result being organized as a lattice. More specifically, the system uses discriminative classifiers that aim at reducing the confusability between pairs of phones. For each of these binary problems, automatic feature selection techniques are used to choose the proper parametric representation for the specific problem.

KEYWORDS: Automatic speech recognition, text-to-speech.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

t	-	Quadro (ou *frame*)
$\mathbf{x}$	-	Vetor que representa um quadro após a parametrização
L	-	Dimensão do vetor $\mathbf{x}$ (número de parâmetros)
T	-	Número de quadros
$\mathcal{T}$	-	Sentença
w	-	Palavra dentro de uma sentença $\mathcal{T}$
P	-	Número de palavras que compõem uma sentença $\mathcal{T}$
B	-	Número de sentenças
$\mathbf{X}$	-	Matriz que representa uma sentença $\mathcal{T}$ de dimensão $L \times T$
$\mathcal{T}^*$	-	Sentença que maximiza a probabilidade posterior $p(\mathbf{X}\|\mathcal{T})p(\mathcal{T})$
$p(\mathcal{T})$	-	Probabilidade do modelo de linguagem
$p(\mathbf{X}\|\mathcal{T})$	-	Probabilidade do modelo acústico
R	-	Número de erros de substituição na sentença reconhecida
D	-	Número de erros de deleção na sentença reconhecida
$\mathbf{T}$	-	Conjunto composto por B sentenças
$p(\mathbf{T})$	-	Probabilidade do conjunto de sentenças $\mathbf{T}$
$W_{\mathbf{T}}$	-	Número de palavras presentes no conjunto de sentenças $\mathbf{T}$
$H_p(\mathbf{T})$	-	*cross-entropy* da sentença $\mathbf{T}$
I	-	Ganho de informação
PP	-	Perplexidade do modelo de linguagem
$\#$	-	Final da palavra na conversão grafema-fone

q	-	Sequência de estados que representa uma hipótese dentro da *lattice*
s_i	-	Escore total da sequência de palavras $\mathcal{T}_i$
$b_i(t)$	-	Escore acústico da sequência q_i no quadro t
g	-	Classificador do tipo g
f	-	Classificador do tipo f
$f(t)$	-	Escore de saída do classificador f no quadro t
α	-	Número positivo usado no re-escore das *lattices*
$b'_i(t)$	-	Novo escore acústico da sequência q_i no quadro t
$G(I, J)$	-	Grafo de uma *lattice*, com I nós e J arcos
$\mathcal{L}$	-	Lista com todas as hipóteses presentes na *lattice*
q_*	-	Hipótese correta dentro da *lattice*
s_*	-	Escore total da hipótese correta
$q_\dagger$	-	Hipótese reconhecida no processo de decodificação
$s_\dagger$	-	Escore total da hipótese reconhecida
$M_i(t)$	-	Margem acústica da i-ésima hipótese no t-ésimo quadro
M	-	Margem acústica total
$\mathcal{F}$	-	Conjunto de quadros que formam uma região de confusão
Q	-	Quantidade de quadros que formam o conjunto $\mathcal{F}$
γ	-	Limiar do valor da margem
$\mathcal{A}$	-	Classificador
$\mathcal{C}$	-	Conjunto de classificadores
A_g	-	Taxa de acerto dos classificadores g
A_f	-	Taxa de acerto dos classificadores f
N	-	Número de exemplos contidos no conjunto de treino do classificador
$\mathcal{X}$	-	Conjunto de instâncias $\mathbf{x}$ do classificador
Y	-	Conjunto de rótulos y do classificador
$\mathcal{K}$	-	Kernel do classificador

$\mathcal{H}_{\mathcal{K}}$	-	*Reproducing kernel Hilbert space* gerado pelo kernel $\mathcal{K}$
$p_{(+)}$	-	probabilidade de uma saída positiva do classificador
$p_{(-)}$	-	probabilidade de uma saída negativa do classificador
$p(y)$	-	Distribuição de probabilidade do rótulo y
$H(Y)$	-	Entropia da variável aleatória Y
$\mathcal{F}_e$	-	Região de confusão em *lattices* onde $q_* \neq q_\dagger$
$\mathcal{F}_i$	-	Região de confusão em *lattices* onde $q_* = q_\dagger$
t'	-	Quadro de menor margem da região de confusão $\mathcal{F}_e$
t_+	-	Quadro de maior margem da região de confusão $\mathcal{F}_i$
t_-	-	Quadro de menor margem da região de confusão $\mathcal{F}_i$
S	-	Conjunto de parâmetros
C	-	Fator de complexidade da SVM
H	-	Número de neurônios na camada oculta da MLP
m	-	Média dos parâmetros
d	-	Desvio padrão dos parâmetros
c	-	Coeficiente angular dos parâmetros

Capítulo 1

Introdução

Neste capítulo são discutidos alguns dos principais problemas enfrentados atualmente pelos pesquisadores e desenvolvedores que trabalham na área de tecnologia da fala (voz). Faz-se ainda uma revisão dos principais sistemas de reconhecimento e síntese de voz disponíveis para o Português, incluindo a variante europeia, com referência ao estado da arte das tecnologias empregadas. Por fim, é apresentado um resumo das principais metodologias utilizadas e dos objetivos traçados para a presente tese.

1.1 Organização do trabalho

Este trabalho se divide em duas partes. A primeira parte se situa no estado da arte em reconhecimento e síntese de voz para Português Brasileiro (PB), levando em conta as limitações dos recursos disponíveis e tentando superá-las. Um requisito importante para se desenvolver pesquisa em processamento de fala é ter à disposição os recursos necessários para se trabalhar com tarefas cuja dificuldade pode ser considerada no estado da arte. Dessa forma, o trabalho busca fornecer subsídios para tais pesquisas. Pretende-se também levar aos desenvolvedores de aplicativos e pesquisadores todos os recursos necessários para utilização de tecnologias de voz em seus trabalhos. Assim, foram construídos um sistema de reconhecimento automático de voz para grandes vocabulários (LVCSR, de "large vocabulary continuous speech recognition") e um sistema completo para geração de voz sintetizada na plataforma MARY, ambos específicos ao PB. Todos os recursos desenvolvidos encontram-se publicamente disponíveis [1].

Já a segunda parte do trabalho explora o paradigma *knowledge-rich* [2], com uma abordagem que pretende ir além dos limites atuais da tecnologia de reconhecimento de voz utilizando-se a reavaliação de *lattices*. Descreve-se na segunda parte como foram implemen-

tados classificadores binários baseados em aprendizado de máquina. O objetivo é diminuir a confusão entre fones na busca pela melhor hipótese dentre as existentes na *lattice*, e direcionar a busca para que a sentença correta seja reconhecida no final do processo.

A próxima seção apresenta uma descrição do atual contexto do processamento de fala.

1.2 Introdução ao problema

Processamento de voz inclui diversas tecnologias, sendo que o reconhecimento automático de voz (ASR, de "automatic speech recognition") [3, 4] e a síntese de voz a partir de texto (TTS, de "text-to-speech") [5, 6] são as mais proeminentes. Os sistemas TTS são compostos por módulos de *software* que convertem textos escritos em linguagem natural em voz sintetizada [7]. O ASR pode ser visto como o processo inverso ao TTS, no qual a voz digitalizada é convertida em texto. Apesar dos conhecidos problemas [8], como a queda de desempenho na presença de ruído ambiente, a tecnologia de reconhecimento de voz é vista atualmente como um importante instrumento para a comunicação natural humano-computador. A mesma é capaz de aumentar o número de cidadãos com acesso às tecnologias de informação, em especial as pessoas que possuem algum tipo de limitação física ou necessidade especial.

Hoje, a tecnologia de voz tem um mercado consolidado, que de acordo com a empresa de tecnologia Opus Research superou um bilhão de dólares pela primeira vez em 2006, e estima-se que atingiu US$ 3 bilhões em 2010, com nichos específicos como a geração automática de relatórios e consultas médicas. Uma pesquisa do Evans Group, realizada com 250 clientes da empresa Nuance em 2005, apontou que 83% dos usuários preferiam utilizar o reconhecimento de voz a discar; e que 74% preferiam atendimentos dirigidos com utilização de reconhecimento de voz a falar com um atendente. Dominado no passado por empresas especializadas, o mercado atualmente conta com participantes como a Microsoft e o Google, a investir fortemente no suporte de ASR e TTS no Windows [9] e no Chrome [10], por exemplo. Este trabalho situa-se em um contexto onde as tecnologias de fala estão se popularizando e visa apoiar a academia e a indústria de *software* no desenvolvimento de pesquisa e tecnologia de voz focada em PB.

Reconhecimento e síntese de voz são tecnologias guiadas pelos dados (*data-driven*) que requerem uma quantidade relativamente grande de informações rotuladas. Diante de tal dificuldade, os pesquisadores dependem fortemente de bases de dados (corpora) públicas e outros recursos específicos para expandir o estado da arte. Alguns grupos de pesquisa possuem corpora de áudio e texto proprietários [11–13], mas não é trivial estabelecer uma colaboração sustentável e a maioria desses corporas não é disponibilizado publicamente. Para o Português

Europeu (PE), os esforços destinados à coleta de recursos têm como alvo principal aplicações em *Broadcast News* (BN), visando a produção automática de legenda para a comunidade com deficiência auditiva. A base de dados (corpus) específica para o domínio BN contém cerca de 60 horas de áudio rotuladas manualmente e, mesmo com esse tamanho relativamente reduzido (quando comparado ao usado para outras línguas), já possibilitou o desenvolvimento de um sistema totalmente automático de produção de legendas [14] *on-line* no canal de televisão público, desde março de 2008. Outros corpora de áudio foram coletados para os mais variados domínios: BDPublico [15] (base de dados para PE equivalente ao corpus do Wall Street Journal [16]), CORAL [17] (corpus de diálogo etiquetado) e LECTRA [18] (corpus constituído por palestras em sala de aula universitária, atualmente composto por 27 horas de áudio).

Para PB, o *Spoltech* é uma das bases de dados mais utilizadas em atividades acadêmicas. O Spoltech é distribuído pelo Linguistic Data Consortium (LDC), cujo catálogo também oferta o corpus *West Point Brazilian Portuguese Speech*, que consiste de sentenças gravadas via microfone digital por falantes nativos e não-nativos na língua portuguesa. Esses dois corpora não são considerados suficientes para o desenvolvimento de um robusto sistema LVCSR em PB. Por exemplo, construir um sistema ASR considerando o critério de máxima informação mútua (MMI) [19] requer muitas horas de áudio para o treinamento dos modelos estatísticos, caso contrário, esse critério não será efetivo quando comparado ao convencional critério de máxima verossimilhança. Além da escassez de dados, não existem *scripts* (ou receitas) publicamente disponíveis para projetar sistemas de referência em PB. Essas receitas contribuem consideravelmente para encurtar o processo de desenvolvimento de tais sistemas.

1.3 Motivação

Apesar da reconhecida importância, as atividades em processamento de voz no Brasil, tanto na academia quanto na indústria (em especial na indústria de *software*), ainda não alcançaram a dimensão necessária para que as mesmas tragam benefícios significativos à sociedade. Por exemplo, tanto o reconhecimento quanto a síntese de voz são importantes tecnologias assistivas, que podem melhorar substancialmente a qualidade de vida de pessoas com necessidades especiais. Mas há poucas ações nesse sentido, com raras exceções como o projeto DOSVOX [20], que permite que pessoas cegas ou com deficiências motoras utilizem um microcomputador comum para desempenhar uma série de tarefas.

Diante do exposto, acredita-se que dois fatores são essenciais para a evolução das tecnologias de voz: bases de dados e *scripts*. Em resposta a essa necessidade, o projeto FalaBrasil [1] foi iniciado em 2009 e encontra-se intimamente vinculado à presente tese. Esse projeto

visa desenvolver e disponibilizar recursos e aplicativos específicos ao PB. Uma motivação para a composição e uso de bases públicas é a recente comprovação de que pesquisas cujos resultados são passíveis de reprodução produzem maior impacto [21]. Diante de aspectos como a crescente importância de pesquisas reproduzíveis, o projeto FalaBrasil alcançou boa visibilidade e é atualmente fomentado por uma ativa comunidade código-livre. Hoje, os recursos disponíveis permitem a composição de um sistema LVCSR e um sistema TTS completo para PB, que são o tema deste trabalho.

O presente trabalho seguiu dois requisitos para o aperfeiçoamento e rápida disseminação das tecnologias de processamento de voz:

- no âmbito da academia, para tornar mais eficiente o trabalho dos grupos de pesquisa: disponibilidade de recursos em domínio público para reconhecimento e síntese de voz em PB. Como citado, tais tecnologias são *data-driven* e dependem de grandes bases de dados, devidamente rotuladas, para o adequado desenvolvimento de sistemas no estado da arte; e

- no âmbito da indústria de *software*, para auxiliar programadores e empreendedores no desenvolvimento de aplicativos baseados em voz: disponibilidade de "engines" (ou seja, reconhecedores e sintetizadores de voz) gratuitos e com licenças que promovam seu uso comercial; e oferta de tutoriais e materiais de instrução visando profissionais sem formação específica em processamento de voz. Neste último caso, a existência de uma interface de programação (API, de "application programming interfaces") é crucial, haja vista a ausência de conhecimento teórico e prático em áreas como processamento digital de sinais e modelos ocultos de Markov por parte dos programadores em geral.

Como dito, o ponto de partida para implementar uma aplicação de voz é contar com um *engine* (ASR, TTS ou ambos). Existem soluções comerciais da Microsoft [22], Nuance [23] e outras companhias. Contudo, atualmente, nem todas as línguas são suportadas, especialmente as chamadas "underrepresented", como o Português. Em 2010, a Microsoft disponibilizou *engines* e ferramentas em versão beta para desenvolvimento de aplicativos específicos ao PB e PE [22]. Contudo, esses sistemas não encontram-se sob a licença código-livre. Portanto, um aspecto chave é ter *engines* disponíveis para o idioma alvo da aplicação.

É oportuno fazer uma breve descrição dos *engines* disponíveis. Com relação aos *engines* de código-livre, o projeto MBROLA [24] oferece sintetizadores de voz em 34 línguas, incluindo o PB. Já os *softwares* Julius [25] e Sphinx-4 [26] são exemplos de decodificadores código-livre que podem ser usados, em conjunto com os recursos requeridos, para construir *engines*

para ASR. Para aplicações embarcadas (e.g., *smartphones*), Julius e PocketSphinx [27] são os *softwares* mais populares.

Tendo um *engine* disponível, uma API facilita a tarefa de desenvolvimento de aplicações de voz para os usuários finais. Por exemplo, a API mais utilizada na industria é a SAPI, a API de voz da Microsoft [28]. Existem outras opções, igualmente disponíveis de forma gratuita, como a JSAPI (Java Speech API) da Sun Inc. Essas APIs especificam uma interface multi-plataforma para manipulação de reconhecedores de voz em aplicações de comando-e-controle, sistemas de ditado e sintetizadores de voz [29]. Essas interfaces não provêem apenas as requeridas funcionalidades de ASR e TTS, mas também inúmeros métodos e eventos que permitem ao programador acesso e controle sobre características operacionais do *engine*.

A maioria das pesquisas em ASR para PB são restritas a sistemas que utilizam pequenos vocabulários (p.e. [30,31]). O desenvolvimento de um sistema LVCSR independente de locutor para PB com um vocabulário com mais de 60 mil palavras é discutido em [32], onde os autores objetivaram a criação de um dicionário fonético mapeado e o aperfeiçoamento do modelo de linguagem. Os resultados foram obtidos com uma quantidade relativamente pequena de dados de áudio extraídos do corpus Spoltech.

Em [12], um corpus de áudio proprietário gravado por um único locutor (a quantidade de horas de áudio não foi divulgada) foi utilizado para treinar modelos acústicos estocásticos dependentes de locutor. Um corpus de texto também foi elaborado para treinar modelos de linguagem. A melhor taxa de acerto para 60 mil palavras obtida pelo sistema foi de 81%, quando reconhecendo sentenças contendo de 9 a 12 palavras, com perplexidade dos modelos de linguagem variando entre 250 e 350 e tempo de processamento menor que um minuto por sentença. Todos os testes foram executados em um computador com processador Dual Intel (XeonTM 3.0 MHz) e 2 GB de RAM.

Ditado é uma boa tarefa para acentuar os testes com sistemas LVCSR [4]. Existem muitos *softwares* comerciais que oferecem bom desempenho para aplicações de ditado em diversas línguas. Para PB, o único *software* comercial para *desktop* é o IBM ViaVoice, que foi descontinuado. Na academia, um sistema LVCSR para *broadcast news* originalmente desenvolvido para PE foi recentemente portado para PB e alcançou uma taxa de erro por palavra de aproximadamente 25% [33].

Os primeiros sistemas TTS completos para PB dentro da academia surgiram no final dos anos 90 na Pontifícia Universidade Católica do Rio de Janeiro (PUC-RJ) e Universidade de Campinas (UNICAMP), com síntese baseada em formantes [34] e síntese concatenativa [35], respectivamente. Atualmente, testes informais de audição indicam que o sistema TTS para PB mais maduro é o desenvolvido na Universidade Federal de Santa Catarina (UFSC) [36].

Em [37], um sistema baseado em cadeias escondidas de Markov é apresentado e os autores disponibilizaram a base de áudio e os *scripts* de treinamento utilizados no *software* HTS [38].

Em resumo, sistemas ASR e TTS no estado da arte são quase sempre construídos usando-se métodos *data-driven* probabilísticos, incrementados através da coleta de muitos dados de treino e re-estimação dos modelos. Daí a importância da disponibilidade de recursos. Além disso, para avançar o estado da arte, são importantes as investigações por novos paradigmas [2]. A próxima seção fornece mais detalhes acerca de cada uma dessas motivações.

1.4 Metodologia e contribuições

Os sistemas para reconhecimento e síntese de voz implementados neste trabalho buscam estabelecer uma referência, permitindo que outros centros de pesquisa usufruam dos recursos e comparem os resultados, promovendo a disseminação da tecnologia de voz para PB. Em resumo, as contribuições da primeira parte deste trabalho são:

- Recursos para os estágios de treino e teste necessários à construção de sistemas ASR: um corpus de texto baseado em dez jornais diários brasileiros, automaticamente formatados e coletados da Internet; dois corpora de áudio multi-locutor que correspondem juntos a 17,2 horas de áudio; e os *scripts* usados para a concepção dos modelos estatísticos acústico e de linguagem;
- Um conversor grafema-fone com determinação de vogal tônica para PB. O dicionário fonético resultante possui mais de 65 mil palavras;
- Uma API que abstrai do programador detalhes de baixo nível relacionados à operação do decodificador. A API implementada contém um conversor de gramáticas escritas na especificação Microsoft SAPI XML para facilitar o suporte ao ASR; e
- Recursos necessários à construção de um sistema TTS completo para PB usando a plataforma MARY, incluindo um separador silábico.

Esta pesquisa seguiu uma metodologia assente na proposição de regras linguísticas para a codificação do conversor grafema-fone e do separador silábico, partindo da consciência de que o Português é uma língua flexional, com grande regularidade fonética e cuja ortografia é principalmente de base fonológica [39]. Já para a construção dos modelos probabilísticos acústico e de língua, empregou-se uma abordagem *data-driven* (por aprendizagem através de corpora) refinada de forma iterativa [40].

A metodologia de avaliação e validação dos recursos desenvolvidos, usada ao longo da primeira etapa deste trabalho, consistiu inicialmente na construção de um corpus de referência para testes com características mais próximas da operação de um sistema de reconhecimento de voz em ambientes ruidosos. Em seguida, analisou-se a taxa de acerto e o tempo de resposta do sistema LVCSR implementado, comparando os resultados obtidos com outro decodificador. Já o sistema TTS implementado também foi comparado com outros sintetizadores, incluindo um *software* comercial. Os testes consistiram de avaliações subjetiva da naturalidade e objetiva da inteligibilidade das vozes produzidas.

A segunda parte deste trabalho apresenta uma proposta que aborda o problema de conflito entre fones em ASR, incorporando conhecimento extra explorado por classificadores binários baseados em aprendizado de máquina. Para cada um desses problemas binários, são usadas técnicas de seleção automática de parâmetros para escolher a representação paramétrica mais adequada para o problema em questão. O objetivo principal é identificar e corrigir possíveis erros obtidos com um sistema ASR convencional. O método proposto foi avaliado em um sistema para reconhecimento de dígitos e produziu resultados promissores.

O impacto positivo da aplicação dos sistemas e recursos livres para reconhecimento e síntese de voz em PB desenvolvidos tem sido demonstrado nos seguintes artigos publicados:

1. Nelson Neto, Carlos Patrick, Aldebaro Klautau e Isabel Trancoso. "Free Tools and Resources for Brazilian Portuguese Speech Recognition". In: Journal of the Brazilian Computer Society, Volume 17, Issue 1 (2011), Página 53.

2. Anderson Monte, Danielle Ribeiro, Nelson Neto, Regina Cruz and Aldebaro Klautau. "A Rule-based Syllabification Algorithm with Stress Determination for Brazilian Portuguese Natural Language Processing". In: The 17th International Congress of Phonetic Sciences, 2011.

3. Nelson Neto, Pedro Batista e Aldebaro Klautau. "Detection and Correction of Phone Confusability in Continuous Speech Recognition". In: International Workshop on Telecommunications, 2011.

4. Patrick Silva, Pedro Batista, Nelson Neto, Aldebaro Klautau. "An Open-Source Speech Recognizer for Brazilian Portuguese with a Windows Programming Interface". In: International Conference on Computational Processing of Portuguese, 2010.

5. Pedro Batista, Patrick Silva, Nelson Neto, Aldebaro Klautau. "A non-Visual Web-Browsing System using Speech Recognition for Brazilian Portuguese". In: International Conference on Computational Processing of Portuguese, Demos Session, 2010.

6. Igor Couto, Nelson Neto, Vincent Tadaiesky, Aldebaro Klautau, Ranniery Maia. "An Open Source HMM-based Text-to-Speech System for Brazilian Portuguese". In: 7th International Telecommunications Symposium, 2010.

7. Alberto Abad, Isabel Trancoso, Nelson Neto, Maria do Céu Ribeiro. "Porting an European Portuguese Broadcast News Recognition System to Brazilian Portuguese". In: Interspeech, 2009.

8. Patrick Silva, Nelson Neto, Aldebaro Klautau. "Novos Recursos e Utilização de Adaptação de Locutor no Desenvolvimento de um Sistema de Reconhecimento de Voz para o Português Brasileiro". In: XXVII Simpósio Brasileiro de Telecomunicações, 2009.

9. Jefferson Morais, Nelson Neto, Aldebaro Klautau. "Tecnologias para o Desenvolvimento de Sistemas de Diálogo Falado em Português Brasileiro". In: 7th Brazilian Symposium in Information and Human Language Technology, 2009.

10. Ana Carolina Siravenha, Nelson Neto, Valquiria Macedo, Aldebaro Klautau. "A Computer-assisted Learning Software Using Speech Synthesis and Recognition in Brazilian Portuguese". In: Interactive Computer Aided Blended Learning, 2009.

11. Ana Siravenha, Nelson Neto, Valquíria Macedo, Aldebaro Klautau. "Uso de Regras Fonológicas com Determinação de Vogal Tônica para Conversão Grafema-Fone em Português Brasileiro". In: 7th International Information and Telecommunication Technologies Symposium, 2008.

12. Nelson Neto, Patrick Silva, Aldebaro Klautau, Andre Adami. "Spoltech and OGI-22 Baseline Systems for Speech Recognition in Brazilian Portuguese". In: International Conference on Computational Processing of Portuguese Language, 2008.

13. Nelson Neto, Ana Siravenha, Valquiria Macedo, Aldebaro Klautau. "A Computer-Assisted Learning Software to Help Teaching English to Brazilians". In: International Conference on Computational Processing of Portuguese Language, Special Session, 2008.

14. Patrick Silva, Nelson Neto, Aldebaro Klautau, Andre Adami, Isabel Trancoso. "Speech Recognition for Brazilian Portuguese using the Spoltech and OGI-22 Corpora". In: XXVI Simpósio Brasileiro de Telecomunicações, 2008.

O *software* Coruja Navigator [41] usou os recursos desenvolvidos neste trabalho e foi premiado como o "Melhor Produto de Software do Pará - Categoria Acadêmica" no IX Simpósio Brasileiro de Qualidade de Software, 2010. Esse aplicativo permite navegação não visual na Web com reconhecimento de voz em Português Brasileiro.

1.5 Síntese dos conteúdos

A natureza híbrida do tema deste trabalho, situado na interseção entre as áreas da engenharia, da linguística e do processamento da linguagem natural, foi explicada neste capítulo. Os seis capítulos restantes foram redigidos de forma a discutir os seguintes conteúdos:

Capítulo 2. Tecnologias de voz. Apesar da grande flutuação existente no que se refere à arquitetura dos modernos sistemas de reconhecimento e síntese de voz, este capítulo fará uma descrição da arquitetura dos sistemas apresentados neste trabalho, com objetivo de contextualizar a aplicação dos módulos desenvolvidos. Por fim, serão apresentadas as principais figuras de mérito dos sistemas ASR e TTS, além dos pacotes de *software* não próprios utilizados nesta pesquisa.

Capítulo 3. Recursos desenvolvidos para ASR e TTS em PB. A fim de aumentar o número de recursos para reconhecimento e síntese de voz em PB, alguns recursos específicos foram construídos e disponibilizados. Neste capítulo, serão apresentados: o conversor grafema-fone; o dicionário fonético; o separador silábico; os corpora de áudio e texto; os modelos acústico e de linguagem; uma API para facilitar a operação do decodificador; e os recursos específicos construídos para gerar uma voz em PB na plataforma MARY.

Capítulo 4. Resultados experimentais. O objetivo deste capítulo será demonstrar a integração dos módulos apresentados previamente em um sistema de reconhecimento de voz em tempo-real. Experimentos comprovaram que técnicas de adaptação de locutor podem ser empregadas para combater descasamento acústico entre diferentes corpora de áudio. Por último, será feita uma comparação dos resultados obtidos entre o sistema TTS implementado neste trabalho e outros dois sintetizadores: Liane e Raquel.

Capítulo 5. Reconhecimento de voz baseado em classificadores binários. Este capítulo apresentará uma proposta que usa a saída de classificadores binários para ajudar algoritmos de decodificação tradicionais a resolver situações de conflito entre fones concorrentes.

Capítulo 6. Resultados experimentais com HMMs e classificadores. Este capítulo abordará os resultados para o método proposto no Capítulo 5. Serão apresentados experimentos concentrados no reconhecimento de uma sequência de dígitos.

Capítulo 7. Conclusões. Neste capítulo, será feita uma síntese do trabalho apresentado, bem como uma avaliação dos resultados atingidos em confronto com os objetivos inicialmente propostos. Ao final, será apresentada uma previsão e análise das principais linhas de ação que este trabalho abre para o futuro.

Capítulo 2

Tecnologias de voz

Primeiramente, este capítulo provê uma introdução aos sistemas ASR e TTS. Já os dois últimos tópicos do mesmo, apresentam os métodos de avaliação e as ferramentas de desenvolvimento usados neste trabalho.

2.1 Sistemas de reconhecimento automático de voz

Os sistemas ASR têm sido estudados desde os anos 50 [3, 4, 42]. Por exemplo, nos Laboratórios Bell nessa época, foi desenvolvido o primeiro reconhecedor de dígitos isolados com suporte a apenas um locutor. Na mesma década, foi introduzido o conceito de redes neurais, mas devido a muitos problemas práticos a ideia não foi seguida no âmbito de ASR. Nos anos 70, a técnica predominante era a *dynamic time-warping* (DTW) [43], que consiste em um algoritmo que mede a similaridade entre duas sequências após alinha-las ao longo do tempo. Com o passar dos anos, a pesquisa foi evoluindo e muitas limitações técnicas foram sendo superadas, além da globalização e popularização dos computadores, o que levou a um aumento no número de pesquisas na área de processamento de voz.

Nos anos 80, os sistemas ASR tentavam aplicar, inicialmente, um conjunto de regras gramaticais e sintáticas à fala [44]. Caso as palavras ditas caíssem dentro de um certo conjunto de regras, o programa poderia determinar quais eram aquelas palavras. Para isso, era preciso falar cada palavra separadamente (sistemas de palavras isoladas), com uma pequena pausa entre elas. Porém, características como sotaques, dialetos e as inúmeras exceções inerentes à língua dificultaram a disseminação dos sistemas baseados em regras. Foi então que vários pesquisadores iniciaram estudos para reconhecimento de palavras conectadas utilizando métodos estatísticos, com maior destaque para os modelos ocultos de Markov (HMMs) [45,46].

Atualmente, o estado da arte em reconhecimento de voz é representado pelo uso de HMMs, incrementado por técnicas como aprendizado discriminativo [47], seleção de misturas de gaussianas [48], maximização de margem [49], entre outras. Outra metodologia encontrada na literatura é o uso de HMMs e redes neurais artificiais em conjunto, obtendo assim, sistemas híbridos bastante robustos na tarefa de reconhecimento de fala contínua [50, 51].

Um sistema ASR é, tipicamente, composto por quatro blocos: *front-end*, modelo acústico, modelo de linguagem e decodificador, como indicado na Figura 2.1, que também mostra o dicionário fonético.

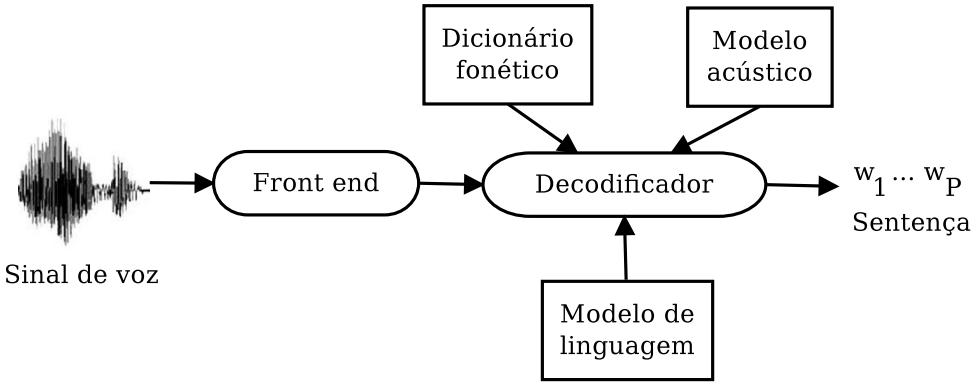


Figura 2.1: Principais blocos de um sistema de reconhecimento de voz.

No processo de *front-end* convencional, tem-se a segmentação do sinal de voz em segmentos curtos (janelas, ou "frames") de 20 a 25 milisegundos, com o deslocamento da janela de análise sendo tipicamente de 10 milisegundos. Então, cada *frame* é convertido em um vetor $\mathbf{x}$ de dimensão L (tipicamente, $L = 39$). É assumido aqui que T *frames* estão organizados em uma matriz $\mathbf{X}$ de $L \times T$, representando uma sentença completa.

Existem várias alternativas no que diz respeito à parametrização do sinal de voz [4,8,52]. Entretanto, a análise *Mel-frequency cepstral coefficients* (MFCCs) [53] tem provado ser eficiente e é geralmente empregada como entrada para os blocos de *back-end* que compõem um sistema ASR [4].

O modelo de linguagem provê a probabilidade $p(\mathcal{T})$ de observar a sentença $\mathcal{T} = [w_1, \ldots, w_P]$ com P palavras. Conceitualmente, o objetivo é encontrar a sentença $\mathcal{T}^*$ que maximiza a probabilidade posterior

$$\mathcal{T}^* = \arg\max_{\mathcal{T}} p(\mathcal{T}|\mathbf{X}) = \arg\max_{\mathcal{T}} \frac{p(\mathbf{X}|\mathcal{T})p(\mathcal{T})}{p(\mathbf{X})}, \tag{2.1}$$

onde $p(\mathbf{X}|\mathcal{T})$ é dada pelo modelo acústico. Como $p(\mathbf{X})$ não depende de $\mathcal{T}$:

$$\mathcal{T}^* = \arg\max_{\mathcal{T}} p(\mathbf{X}|\mathcal{T})p(\mathcal{T}). \tag{2.2}$$

Na prática, uma constante empírica é usada para ponderar a probabilidade do modelo de linguagem $p(\mathcal{T})$, antes da mesma ser combinada com a probabilidade do modelo acústico $p(\mathbf{X}|\mathcal{T})$.

Dado o grande volume de sentenças concorrentes, a Equação (2.2) não pode ser calculada independentemente para cada sentença candidata. Para esse fim, os sistemas ASR utilizam-se de estruturas de dados hierárquicas, como árvores léxicas, e do artifício de separar as sentenças em palavras, e as palavras em unidades básicas, que aqui chamaremos de fones [4]. A busca por $\mathcal{T}^*$ é chamada decodificação e, na maioria dos casos, hipóteses são descartadas ou podadas (de "pruning"). Em outras palavras, para tornar viável a busca pela "melhor" sentença, algumas candidatas são descartadas e a Equação (2.2) não é calculada para elas [54, 55].

Um dicionário fonético (também conhecido por modelo léxico) provê o mapeamento das palavras em unidades básicas (fones) e vice-versa.

Em termos de modelo acústico, no intuito de incrementar o desempenho, HMMs contínuas são geralmente empregadas, onde a distribuição de saída de cada estado é modelada por uma mistura de Gaussianas, como representado na Figura 2.2. A topologia "left-right" pode ser adotada, onde as únicas transições permitidas são de um estado para ele mesmo ou para o estado seguinte.

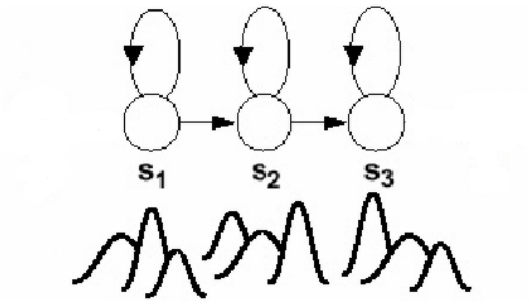


Figura 2.2: Representação de uma HMM contínua *left-right* com 3 estados e uma mistura de Gaussianas por estado.

O modelo acústico pode conter uma HMM por fone. Isso seria bastante razoável caso um fone pudesse ser seguido por qualquer outro, o que não é verdade, já que os articuladores do trato vocal não se movem de uma posição para outra imediatamente na maioria das transições de fones. Nesse sentido, durante o processo de criação de sistemas que modelam a fala fluente, busca-se um meio de modelar os efeitos contextuais causados pelas diferentes maneiras que alguns fones podem ser pronunciados em sequência [56]. A solução encontrada é o uso de HMMs dependentes de contexto, que modelam o fato de um fone sofrer influência dos fones

vizinhos. Por exemplo, supondo a notação do trifone *a-b+c*, temos que *b* representa o fone central ocorrendo após o fone *a* e antes do fone *c*.

Segundo [4], existem basicamente dois tipos de modelos trifones: *internal-word* e *cross-word*. As diferenças entre os mesmos é que no caso do *internal-word* as coarticulações que extrapolam as durações das palavras não são consideradas, sendo assim, menos modelos são necessários. Já no caso do *cross-word*, que considera a coarticulação entre o final de uma palavra e o início da seguinte, a modelagem é mais precisa, porém o número de modelos trifones gerados cresce muito, o que dificulta o trabalho do decodificador e gera uma necessidade de mais dados para treino. Alguns exemplos de transcrição podem ser conferidos na Tabela 2.1.

Tabela 2.1: Exemplos de transcrições com modelos independentes e dependentes de contexto.

Sentença	arroz com bife
Monofones	sil a R o s k o~ b i f i sil
Internal-Word	sil a+R a-R+o R-o+s o-s k+o~ k-o~ b+i b-i+f i-f+i f-i sil
Cross-Word	sil sil-a+R a-R+o R-o+s o-s+k s-k+o~ k-o~+b o~-b+i b-i+f i-f+i f-i+sil sil

Dois problemas clássicos com relação à modelagem acústica são: a inconstância dos fones devido à co-articulação e a insuficiência de dados para estimar os modelos. O método de compartilhamento de parâmetros (ou "sharing") visa combater esse último problema, melhorando a robustez dos modelos. Em muitos sistemas, o compartilhamento é implementado no nível de estado, ou seja, o mesmo estado pode ser compartilhado por HMMs diferentes.

Existem basicamente duas técnicas para compartilhamento de estados: *data-driven* e árvore de decisão fonética. Leva-se em consideração que o contexto do trifone não afeta o estado central do modelo e os estados centrais dos trifones devirados do mesmo monofone são compartilhados, conforme ilustra a Figura 2.3. No método *data-driven* os estados são compartilhados com base em informações estatísticas recolhidas durante o treino dos modelos. Uma desvantagem desse método é que trifones que não ocorrem na base de treino não podem ser adequadamente modelados.

A segunda técnica consiste no uso de uma árvore de decisão fonética para unir os estados que são acusticamente semelhantes. Esse método envolve a construção de uma árvore binária utilizando um procedimento de otimização sequencial *top-down*, onde questionamentos são anexados a cada nó [57]. As perguntas são relacionadas ao contexto fonético dos fones vizinhos ao fone analisado, como ilustra a Figura 2.4. Por exemplo, as questões são do tipo "o fonema à esquerda é nasal?", e dependendo da resposta (sim/não), uma das possíveis direções é seguida. Ao final do processo, todos os estados no mesmo nó folha são agrupados.

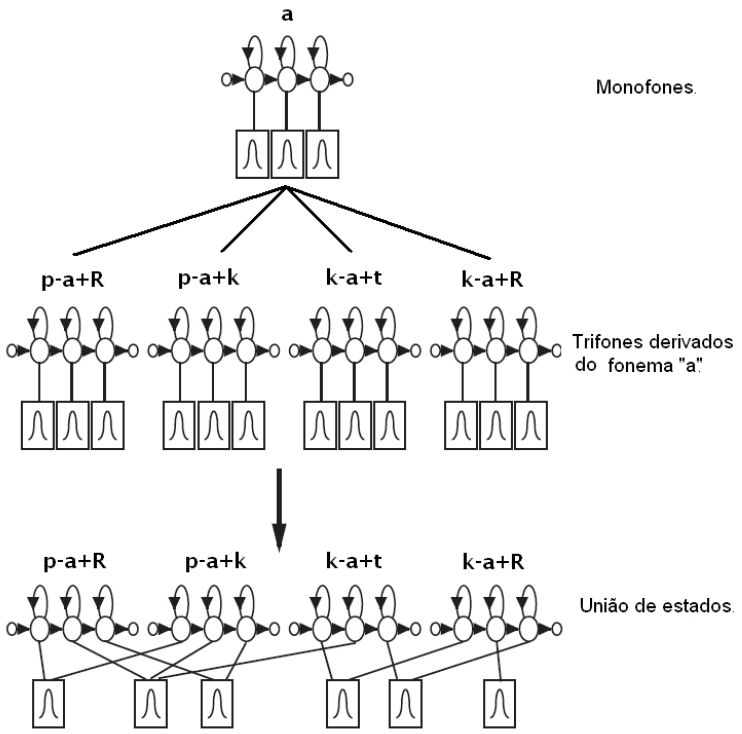


Figura 2.3: Compartilhamento de estados para fins de aumento da robustez da estimativa dos modelos HMM.

As perguntas são escolhidas de forma a maximizar a verossimilhança entre os dados de treino e o conjunto resultante da união dos estados, de forma que existam dados de treino suficientes para estimar de forma robusta os parâmetros das distribuições de probabilidade gaussiana. Além disso, trifones não observados nos dados de treino podem ser estimados realizando uma busca na árvore pelos respectivos nós terminais de seus estados.

A escassez de dados de treino também afeta a modelagem da língua que estima

$$P(\mathcal{T}) = P(w_1, w_2, \ldots, w_P) \tag{2.3}$$

$$= P(w_1)P(w_2|w_1)\ldots P(w_P|w_1, w_2, \ldots, w_{P-1}). \tag{2.4}$$

É impraticável estimar a probabilidade condicional $P(w_i|w_1, \ldots, w_{i-1})$, mesmo para valores moderados de i. Assim, o modelo de linguagem para sistemas ASR consiste de um modelo n-gram, que assume que a probabilidade $P(w_i|w_1, \ldots, w_{i-1})$ depende somente das $n-1$ palavras anteriores. Por exemplo, a probabilidade $P(w_i|w_{i-2}, w_{i-1})$ expressa um modelo de linguagem trigrama.

Resumindo, após o treinamento de todos os modelos estatísticos, um sistema ASR na etapa de teste usa o *front-end* para converter o sinal de entrada em parâmetros e o decodificador para encontrar a melhor sentença $\mathcal{T}$.

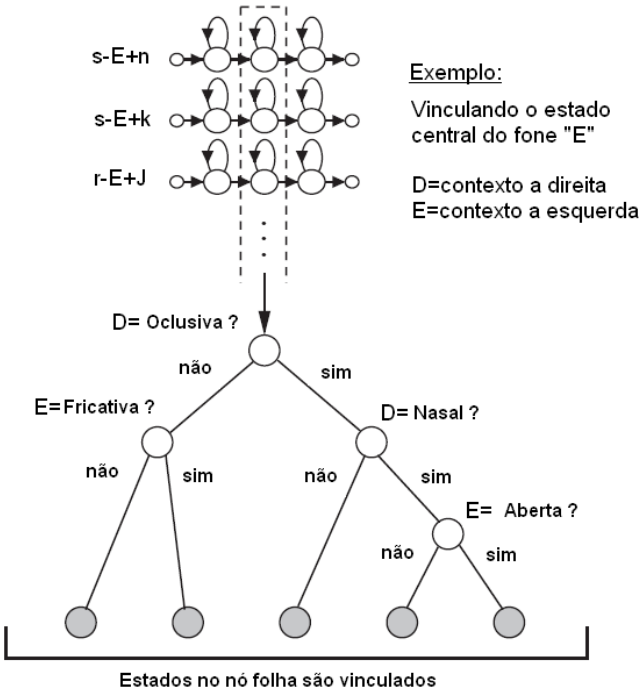


Figura 2.4: Compartilhamento de estados utilizando árvore de decisão fonética.

Os modelos acústico e de linguagem podem permanecer fixos durante toda a etapa de teste, mas adaptar um ou ambos pode incrementar consideravelmente o desempenho do sistema ASR. Por exemplo, um determinado tópico ou assunto pode ser estimado para a tarefa em questão e utilizado um modelo de linguagem específico para o mesmo. Isso é crucial para aplicações com vocabulário técnico, como a geração de relatório de Raios-X pelos médicos [58]. O processo de adaptação do modelo acústico também é importante [59].

O sistema ASR que usa modelos independentes de locutor são convenientes, contudo precisam ser capazes de reconhecer com boa precisão qualquer falante. À custa de solicitar que o usuário leia em voz alta algumas frases, técnicas de adaptação podem ajustar parâmetros das HMMs, como as médias e variâncias das Gaussianas, para um locutor alvo. Sendo assim, as técnicas utilizadas neste trabalho realizam adaptação somente nas médias das Gaussianas dos modelos HMM. Os parâmetros que não são adaptados são herdados do modelo independente do locutor, característica que diminuiu consideravelmente a carga computacional requerida no processo de adaptação [60].

A técnica de adaptação *maximum likelihood linear regression* (MLLR) estima um grupo de transformações lineares para os parâmetros das Gaussianas, que busca reduzir o descasamento entre o modelo inicial (modelo independente de locutor) e os dados de adaptação providos pelo usuário [61]. O efeito dessas transformações é de deslocar o componente das

médias no modelo original para que cada estado das HMMs tenha maior probabilidade de gerar dados adaptados à voz do novo locutor. A adaptação dos modelos também pode ser realizada usando a técnica *maximum a posteriori* (MAP), ou abordagem Bayesiana, onde as distribuições iniciais do modelo independente de locutor representam o conhecimento a priori e são atualizadas, através da regra de Bayes, para se alcançar um novo modelo dependente de locutor [61].

2.2 Sistemas de conversão texto-fala

A síntese de voz consiste em produzir a fala humana artificialmente, através da geração automática do sinal de voz a partir de texto [5]. Vários trabalhos em síntese de voz vêm sendo desenvolvidos há décadas e considerável avanço foi alcançado. Porém, a qualidade em termos da naturalidade da voz produzida ainda apresenta lacunas, principalmente no que tange às adaptações que a fala pode sofrer considerando aspectos como entonação e emoção, associados à expressividade do conteúdo a ser sintetizado.

O primeiro dispositivo considerado um sintetizador elétrico foi o VODER (*Voice Operating Demonstrator*), desenvolvido por Homer Dudley em 1939. Ele era composto por uma barra para selecionar o tipo de voz (vozeada ou não vozeada), um pedal para controlar a frequência fundamental e dez teclas que controlavam o trato vocal artificial. A estrutura básica do VODER é bastante similar aos sistemas baseados no modelo fonte-filtro. Atualmente, dentre as tecnologias envolvendo síntese de voz, destacam-se: por concatenação, articulatória, por formantes (regras) e mais recentemente baseada em HMMs [5].

Uma das aplicações de síntese de voz pode ser encontrada em sistemas TTS, os quais convertem um texto de entrada em uma voz artificial que seja inteligível e mais natural possível. A tarefa dos sistemas TTS é bastante complexa, pois envolve a imitação (*mimicking*) de como os seres humanos realizam a leitura de um texto. Segundo [6], esses sistemas são compostos por dois componentes principais: *front-end* e *back-end*, conforme mostra a Figura 2.5.

O módulo *front-end* é dependente de língua e realiza a análise do texto para que a informação de saída seja codificada de forma conveniente ao *back-end*. Por exemplo, o *front-end* executa a normalização (ou pré-processamento) do texto, convertendo símbolos, como números e abreviaturas, em palavras equivalentes escritas por extenso. Ele também implementa a conversão grafema-fone, silabação e determinação de sílaba tônica. Essas tarefas atribuem transcrição fonética à cada palavra e marcam o texto com informações prosódicas. Transcrições fonéticas e informações prosódicas compõem juntas a representação simbólica linguística que serve de entrada para o *back-end*.

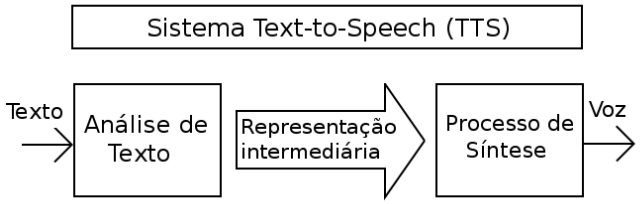


Figura 2.5: Diagrama funcional de um típico sistema TTS mostrando os módulos *front-end* e *back-end*, responsáveis pela análise de texto e síntese de voz, respectivamente.

O módulo *back-end* é tipicamente independente de língua e inclui o sintetizador, que é o bloco que efetivamente gera o som, convertendo a representação linguística em voz. Historicamente, os sistemas TTS evoluíram a partir de um paradigma baseado no conhecimento para uma abordagem pragmática (*data-driven*), como a técnica baseada em HMMs [62].

A síntese de voz baseada em HMMs vem se tornando bastante popular devido à sua flexibilidade em sintetizar voz considerando características como estilo e individualidade da fala do locutor, além da possibilidade de expressar aspectos emocionais na voz sem precisar de uma grande quantidade de amostras de dados [63].

Um sistema TTS baseado em HMMs funciona através da execução de duas etapas: treino e síntese, conforme mostra a Figura 2.6. Durante a fase de treinamento os parâmetros referentes ao espectro da voz e à excitação, tais como coeficientes MFCCs e frequência fundamental (F0), são extraídos de uma base de voz e modelados através de HMMs, considerando aspectos fonéticos, linguísticos e prosódicos da fala. Nesse caso específico, tem-se uma HMM para cada fone (monofone), porém o modelo estatístico pode ser construído baseado em unidades maiores, como difones, trifones, palavras, entre outras.

Como resultado dessa primeira etapa, tem-se um *framework* unificado, possuindo modelos dos espectros da excitação e duração da voz. Em seguida, tem-se o processo de síntese propriamente dito, onde um texto qualquer de entrada é transformado em uma sequência de fones, onde cada HMM correspondente a um fone é concatenada com a HMM do fone anterior e posterior (dependente do contexto). Após o processo de concatenação, cada HMM emite um conjunto de observações (parâmetros MFCC, F0, entre outros). Por fim, uma forma de onda da voz é sintetizada diretamente a partir dos parâmetros estimados.

Com relação à síntese de voz, o objetivo principal deste trabalho é a construção e disponibilização de um módulo *front-end* para PB. Detalhes técnicos de implementação do módulo *back-end* não serão abordados, visto que, para isso, utilizou-se o *framework* MARY, que abstrai do desenvolvedor esses detalhes, como será mostrado nas seções futuras.

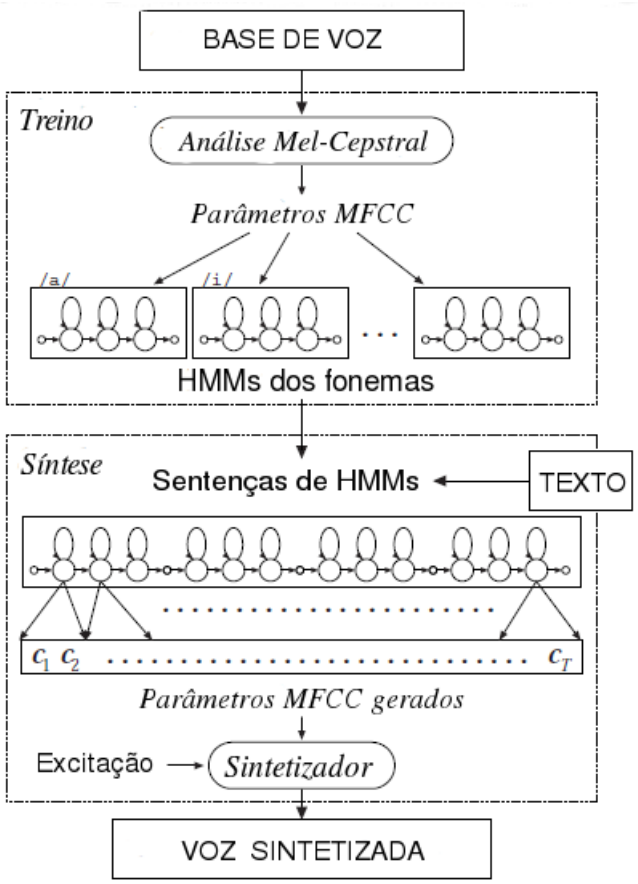


Figura 2.6: Síntese de voz baseada em HMMs.

2.3 Métricas de avaliação

Na maioria das aplicações que usam reconhecimento de voz (inclusive para ditado) a medida de desempenho utilizada é a taxa de erro por palavra (WER). Dado que geralmente as transcrições correta e reconhecida possuem diferente número de palavras, elas são alinhadas através de programação dinâmica [64]. Dessa forma, a WER empregada neste trabalho foi definida como

$$WER = \frac{D + R}{P} \times 100\%, \tag{2.5}$$

onde P é o número de palavras na sentença de entrada, R e D são o número de erros de substituição e deleção na sentença reconhecida, respectivamente, quando comparada com a transcrição correta.

Outra métrica de avaliação é o fator de tempo-real (xRT). O fator xRT é calculado dividindo o tempo que o sistema despende para reconhecer uma sentença pela sua duração. Assim, quanto menor for o fator xRT, mais rápido será o reconhecimento.

Durante o processo de construção dos modelos de linguagem, existem várias características que os diferenciam, como o número de palavras distintas e o número de sentenças usadas para estimar os modelos. No entanto, a métrica mais comum para avaliar os modelos de linguagem é a probabilidade $p(\mathbf{T})$ que o modelo atribui para alguns dados de teste $\mathbf{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_B\}$ compostos de B sentenças. Independência entre as sentenças é assumida, o que leva ao produto das probabilidades $p(\mathbf{T}) = p(\mathcal{T}_1) \ldots p(\mathcal{T}_B)$. Duas medidas são derivadas dessa probabilidade: perplexidade e entropia cruzada (*cross-entropy*) [4]. A *cross-entropy* $H_p(\mathbf{T})$ é definida como

$$H_p(\mathbf{T}) = -\frac{1}{W_\mathbf{T}} \log_2 p(\mathbf{T}), \tag{2.6}$$

onde $W_\mathbf{T}$ é o número de palavras em $\mathbf{T}$.

A perplexidade (PP) representa o número médio de palavras que podem seguir uma dada palavra e está relacionada com a *cross-entropy* $H_p(\mathbf{T})$ por

$$PP = 2^{H_p(\mathbf{T})}. \tag{2.7}$$

Para uma dada tarefa (por exemplo, com o tamanho do vocabulário fixado), baixos valores de perplexidade e *cross-entropy* indicam menor incerteza na predição da próxima palavra e, tipicamente, revelam um melhor modelo de linguagem.

Com relação à síntese de voz, a qualidade da voz pode ser medida através de dois fatores: inteligibilidade e naturalidade. O nível de inteligibilidade pode ser dividido em dois segmentos: primeiramente a precisão com que as palavras pronunciadas são recebidas pelo usuário; e em seguida de que forma elas foram compreendidas dentro do contexto proposto. A precisão pode ser calculada de maneira semelhante à WER, comparando o que foi recebido pelo usuário com as mensagens de referência. Já a compreensão pode ser avaliada através de questões e tarefas requisitadas ao receptor, que deve mostrar que conseguiu absorver o significado da mensagem.

A naturalidade pode ser medida através da coleta de opiniões de um considerável número de usuários e usando, por exemplo, o protocolo *mean opinion score* (MOS) [65]. O MOS é calculado pela média dos resultados de um conjunto de testes subjetivos, onde um número de ouvintes avalia a qualidade do áudio gerado pelo *software* TTS. Note que, inteligibilidade e naturalidade estão relacionadas, mas não diretamente atreladas. Isso porque sistemas TTS com alto grau de inteligibilidade podem não oferecer uma voz que soe naturalmente, tornando o uso prolongado desses sistemas pouco agradável aos ouvidos dos usuários.

2.4 Ferramentas

O pacote de ferramentas HTK [66] foi usado para construir e adaptar os modelos acústicos discutidos neste trabalho. Já o SRI Language Modeling Toolkit (SRILM) foi usado para construir os modelos de linguagem n-gram no formato ARPA. O SRILM [67] é um *toolkit* para construção e manipulação de modelos estatísticos de linguagem. Esse pacote de ferramentas possibilita o uso de várias técnicas de suavização das probabilidades referentes aos modelos de linguagem.

Para decodificação, o sistema LVCSR desenvolvido neste trabalho adotou o Julius rev.4.1.5 [25], que é um decodificador capaz de operar em aplicações de tempo-real e é código-livre. Experimentos também foram realizados com o decodificador HDecode (parte do HTK). A atual versão do HDecode [66] suporta modelos de linguagem n-gram (até trigramas), mas não oferece suporte para operações em tempo-real. Nesta pesquisa, o HDecode foi usado apenas como parâmetro de comparação.

A motivação para usar a plataforma MARY neste trabalho deveu-se à mesma ser código-livre, completamente escrita na linguagem de programação Java e suportar síntese de voz baseada em HMMs [68]. Sua arquitetura modular permite a inserção de novas línguas e, consequentemente, a criação de novas vozes. Para isso, no entanto, é necessário a implementação de um módulo de processamento de texto, externo à plataforma, para a língua alvo. Atualmente, a plataforma MARY oferece suporte às línguas alemã, inglesa e tibetana.

O próximo capítulo descreve os recursos específicos para PB desenvolvidos ao longo deste trabalho, além das bases não-proprietárias usadas para executar os experimentos.

Capítulo 3

Recursos desenvolvidos para ASR e TTS em PB

Com o intuito de viabilizar os trabalhos futuros em nosso grupo de pesquisa, além de promover e disseminar o uso da tecnologia de voz em PB, alguns recursos específicos foram construídos e disponibilizados. Neste capítulo, serão descritos primeiramente os recursos linguísticos construídos, que se resumem a um dicionário fonético, um separador silábico, um corpus de texto e dois corpora de áudio. Os corpora Spoltech e West Point também serão descritos, já que foram utilizados ao longo deste trabalho. Esses corpora são protegidos por direitos autorais e podem ser adquiridos através do LDC. As revisões e correções realizadas nesses corpora de áudio estão documentadas em [1]. Em seguida, serão apresentados os processos de desenvolvimento dos modelos acústico e de linguagem. Por fim, a interface de programação implementada a fim de operar com mais facilidade o decodificador Julius e os recursos desenvolvidos especificamente para a criação de uma voz sintetizada na plataforma MARY serão apresentados.

3.1 UFPAdic: Um dicionário fonético para PB

A elaboração de um dicionário de pronúncias (ou fonético), que mapeia cada palavra do léxico em uma ou mais transcrições fonéticas (pronúncias), é uma etapa fundamental no processo de desenvolvimento de sistemas que empregam técnicas de processamento de voz. Por exemplo, para implementar um sistema LVCSR, é essencial que se tenha um dicionário fonético, que fornece a correspondência entre a ortografia e a(s) pronúncia(s). Na prática, construir um dicionário fonético para ASR é muito similar ao desenvolvimento de um módulo

grafema-fone (G2P) para sistemas TTS [35]. Na verdade, um dicionário fonético pode ser construído invocando um módulo G2P pré-existente. No entanto, a tarefa de conceber um módulo G2P não é trivial.

Os conversores G2P podem ser organizados em métodos baseados em regras e *data-driven*. Em [69], a combinação dos dois métodos foi implementada pela compilação de regras fonológicas e de determinação de vogais tônicas para PE, usando a flexibilidade de integração de múltiplas fontes de informação dos transdutores de estados finitos. O conjunto de regras para conversão G2P na língua portuguesa é extremamente amplo. Em função disso, diversos estudos têm enfatizado o uso de algoritmos de aprendizagem de máquina para obtenção automática de regras para conversão G2P, como redes neurais de múltiplas camadas [70] e algoritmos genéticos [71]. Por exemplo, o trabalho de [72] descreve um conversor G2P baseado em métodos de aproximação por memória e transformação, sendo que a aproximação híbrida retornou o melhor resultado. Tal pesquisa exalta a importância da informação silábica e do tamanho do corpus de treino.

Abordando a influência das variantes léxicas do PB em sistemas de reconhecimento de fala, [73] esclarece os processos fonológicos, próprios das línguas naturais, como inserção ou apagamento de segmentos, assimilação segmental e estruturação silábica. Os resultados obtidos são comparados a um *software* desenvolvido com base em regras que descrevem as variantes para as palavras do léxico de um sistema de reconhecimento de voz para PB, com as transcrições geradas manualmente. Essa última pode ser, eficientemente, substituída pelas transcrições obtidas automaticamente.

Esta atual pesquisa contribui com um algoritmo para conversão G2P com determinação de vogal tônica para PB. O conversor G2P proposto é uma codificação do conjunto de regras descritas em [74], com algumas adaptações. As regras não focam um dialeto específico da língua portuguesa. Uma vantagem dos conversores baseados em regras é que o alinhamento lexical não se faz necessário, visto que o conversor não precisa ser treinado para gerar suas próprias regras [75]. Em outras palavras, as regras de conversão, baseadas em critérios fonológicos pré-estabelecidos, são fornecidas ao sistema de acordo com a língua a qual o aplicativo se destina.

A arquitetura do conversor G2P implementado neste trabalho é *self-contained*, ou seja, não carece de estágios intermediários, nem depende de outros algoritmos, para realizar análises específicas, como separação silábica ou identificação de pluralidade. Também é importante ressaltar que existe uma ordem obrigatória para aplicação das regras de conversão. Isto é, primeiro são analisadas as regras consideradas mais específicas e, por último, a regra, ou caso geral, que finaliza a análise. O formato geral para cada entrada do dicionário sugerido pelo

software HTK [66] é ilustrado pelo exemplo abaixo:

```
leite l e j tS i sp
```

portanto, o conversor G2P proposto lida apenas com palavras isoladas e não implementa análise co-articulatória entre palavras.

As regras são especificadas em um conjunto de expressões regulares usando a linguagem de programação C#, onde expressões regulares também são permitidas na definição de símbolos não-terminais (p.e. #abacaxi#). As regras do conversor G2P proposto estão organizadas em três fases. São elas:

- um procedimento que insere o símbolo não-terminal # antes e depois de cada palavra.
- a determinação da vogal tônica da palavra através de 29 regras.
- o núcleo do sistema, composto por 140 regras, que converte os grafemas (incluindo a marcação de vogal tônica) em 38 fones do alfabeto fonético SAMPA [76].

Como dito, antes da conversão G2P é preciso identificar a vogal tônica da palavra analisada. Para isso, primeiramente, um objeto "Regex" é criado com o padrão a ser encontrado dentro da palavra analisada. Já o objeto "Match" recebe a resposta da comparação do padrão com a palavra apresentada. Sendo verdadeira, a vogal tônica é determinada, caso contrário, outras regras são testadas até que se esgotem as possibilidades e o caso geral seja aplicado. Por exemplo, o código abaixo implementa a regra 8 aplicada para determinação de vogal tônica e é explicado a seguir.

```
Regex rule_8=new Regex("[^aeiou][iu][#]");
Match m8 = rule_8.Match(word);
if(m8.Success) {
   index_strVw = m8.Index + 1;
   strVw = word.Substring(index_strVw,1);
   break;
}
```

O final da palavra é indicado pelo símbolo #. Então, o grafema i ou u é o último caracter da palavra. Ainda de acordo com a estrutura, o penúltimo caracter não pode ser uma vogal. O objeto "Match" sempre indexa o ponteiro para o primeiro componente do padrão analisado, nesse caso, o caracter que não é uma vogal. Finalmente, o último caracter é definido

como a vogal tônica. Por exemplo, a análise da palavra "abacaxi", que retorna a vogal *i* como tônica, é feita pela regra descrita acima.

De posse dessa informação, o próximo passo é a conversão G2P propriamente dita, que segue a ordem sequencial da palavra (*left-to-right*). O exemplo abaixo apresenta uma das regras aplicadas para a transcrição da palavra "abacaxi", onde a letra *x* é convertida no correspondente fone *S*.

```
letter = word.Substring(index,1);
Regex idX = new Regex("x");
Match gX = idX.Match(letter);
if(gX.Success) {
   phone[index] = "S";
   index++;
}
```

É importante atentar que a ordem de precedência escolhida para o funcionamento do algoritmo não é casual. As regras mais específicas precisam ser aplicadas primeiro. Além disso, a presença de uma vogal tônica muda a interpretação da conversão, p.e.:

```
(e(V_ton))(l)(C-h,Pont)  - [E]
(e(V_aton))(l)(C-h,Pont) - [e]
```

onde a vogal (*e*) é convertida para fones distintos, dependendo se ela assume a característica aberta [*E*] ou fechada [*e*].

A conversão é temporariamente armazenada em um arranjo de *strings* até o último passo do processo, que remove os grafemas a fim de produzir a desejada sequência de fones. Por fim, a palavra e a sua respectiva conversão G2P são escritas na forma abaixo:

```
abacaxi a b a k a S i sp
```

que é o formato sugerido pelo *software* HTK, onde o fone delimitador *sp* precisa ser adicionado no final de cada pronúncia.

Durante a pesquisa, constatou-se a necessidade de adaptação de algumas regras propostas por [74]. Um resumo das regras adicionadas e modificadas pode ser visto na Tabela 3.1. Originalmente, algumas normas que trabalham a nasalidade dos grafemas *a* e *u* não analisam se o grafema seguinte é uma vogal ou consoante. No entanto, verificou-se que essa distinção é importante para o processo de transcrição. Por exemplo, a palavra "adotando", onde a vogal tônica nasal *a* é seguida da consoante *d*, deveria ser originalmente convertida para:

Tabela 3.1: Novas regras para os grafemas [i, a, u, x].

Letra	Regra	Sequência para o algoritmo	Fone
a	3	... (a(V_ton))(m,n)(V,h)...	[a∼]
i	1	Exception: gratuito(a)	[j]
u	2	...(u(m,n))(C-h)...	[u∼]
u	4	...(u)(m,n)(V,h)...	[u∼]
u	5	...(V-u)(u)...	[w]
u	6	...(q,g)(u)(a)...	[w]
u	7	...(g)(u)(o)...	[w]
x	1	...(V,C-f,m)(i)(x)...	[S]
x	2	...(f,m)(i)(x)...	[k s]
x	3	...((W_bgn)e,ê)(x)(V,C_v)...	[z]
x	4	...((W_bgn)ine)(x)(o,C_v)...	[k s]
x	5	...((W_bgn)ine)(x)(a,e,i)...	[z]
x	6	...((W_bgn)(e,ê,ine))(x)(C_uv)...	[s]
x	7	...((W_bgn)e)(x)(Hf)(V,C_v)...	[z]
x	8	...((W_bgn)e)(x)(Hf)(C_uv)...	[s]
x	9	...(V-e)(x)(Hf)(Ltr)...	[k z]
x	10	...(V-e)(x)(V)...	[k s]
x	11	...(b,f,m,p,v)(e)(x)(V)...	[S]
x	12	...(V)(e)(x)(V)...	[z]
x	13	...(C-b,f,m,p,v)(e)(x)(V)...	[k s]
x	14	...((W_bgn)x)...	[S]
x	15	...(e,é,ê)(x)(C)...	[s]
x	16	...(x)(Pont)...	[k s]
x	17	...(x)...	[S]

```
adotando a d o t a~ n d u sp
```

onde a transcrição do fone *n* não foi desconsiderada. Já usando a regra adaptada, descrita na primeira linha da Tabela 3.1, a palavra foi convertida para:

```
adotando a d o t a~ d u sp
```

Ainda sobre o grafema *u*, observou-se a inexistência de regras que o classifique como uma semivogal, representada aqui pelo fone *w*. Com isso, três novas regras foram elaboradas

e incluídas antes da regra geral do grafema *u*. Por exemplo, a palavra "quatro" deveria ser originalmente convertida para:

```
quatro k u a t r u sp
```

onde o ditongo formado pela sequência $< qu >$ é transcrito para o difone $< ku >$. Já usando a regra 6, mostrada na Tabela 3.1, a palavra foi convertida para:

```
quatro k w a t r u sp
```

Além da exceção feita à palavra "gratuito(a)", que não segue a primeira regra do grafema *i*, outro ponto que se mostrou carente de análise foi o grafema *x*. Segundo [77], a letra *x* representa o som consonantal mais variável da língua portuguesa. Dado que [74] usa uma lista de exceções para converter o grafema *x*, este trabalho contribui com 17 novas regras. O suporte técnico para elaboração dessas regras foi extraído de [77].

Com relação às regras para determinação de vogal tônica, fez-se necessário a inserção de apenas uma norma, no caso para o monossílabo "que", que assim como a palavra "quem", não se encaixa em nenhuma das regras originalmente propostas.

Então, usando o conversor G2P proposto, um dicionário fonético específico para PB foi construído. Ele tem 65.532 palavras e é chamado de UFPAdic. As palavras foram selecionadas entre as mais frequentes no corpus CETENFolha [78], que é um corpus composto por texto extraídos do jornal Folha de S. Paulo e compilado pelo Núcleo de Linguística Computacional da Universidade de São Paulo, Brasil. O conversor G2P (arquivo executável) e o UFPAdic encontram-se publicamente disponíveis [1].

3.2 Separador silábico

Apesar de determinar a vogal tônica, o conversor G2P não executa separação silábica e identificação de sílaba tônica. Contudo, esses dois blocos são fundamentais dentro do módulo *front-end* de um sistema TTS. Para qualquer língua, é essencial realizar o mapeamento das sílabas quando tratamos de sistemas TTS, tanto para transcrição fonética, como para a geração de prosódia, já que em uma sílaba tônica há um aumento de duração e intensidade.

Comprovadamente, o uso de regras linguísticas é uma boa escolha para sistemas TTS, considerando que o Português é uma língua fonologicamente regular. Além disso, a solução para separação silábica e determinação da tonicidade por regras apresenta duas vantagens em relação, por exemplo, à solução por dicionário: pouca utilização de memória e a capacidade de sempre poder ler uma nova palavra.

Assim, tais tarefas foram implementadas através de um *software* escrito na linguagem de programação Java. O separador silábico proposto é uma codificação do conjunto de regras descritas em [79], com algumas modificações. A ideia principal desse algoritmo é que todas as sílabas possuem uma vogal como núcleo, que pode ser cercado por consoantes ou outras vogais (semi-vogais ou glide). Então, é possível localizar a vogal que compõe o núcleo da sílaba e isolar as consoantes e as semi-vogais.

A arquitetura do separador silábico implementado é *self-contained* e as regras são dispostas hierarquicamente desde a primeira até a penúltima, sendo a última regra o caso geral que finaliza a análise. As regras são baseadas na busca das vogais existentes em cada palavra, seguida de análise dos caracteres existentes à esquerda e à direita, para então decidir qual ação deve ser tomada. Como as vogais são a base da sílaba, a ação tomada é no sentido de unir ou não a referida vogal ao conjunto de grafemas que a cerca.

As regras responsáveis pela separação silábica são especificadas em um conjunto de expressões regulares, seguindo a mesma estratégia usada no conversor G2P e descrita na Seção 3.1. O núcleo do sistema é composto por 20 regras. Já a identificação da sílaba tônica provou ser uma tarefa fácil, beneficiada pelo fato do conversor G2P implementado já ser capaz de identificar a vogal tônica de uma dada palavra. Dessa forma, após receber o resultado da separação silábica, torna-se trivial identificar a sílaba que contém a vogal tônica.

Durante a pesquisa, observou-se que as regras propostas por [79] não consideram a análise da vogal tônica durante o processo de separação silábica. Contudo, existem palavras cuja separação silábica torna-se bastante complicada (e muitas vezes falha) sem a análise da vogal tônica, especialmente quando as mesmas possuem ditongos. Para contornar essa dificuldade, este trabalho contribui com duas novas regras, que foram adicionadas ao conjunto original. A primeira regra aborda os ditongos decrescentes ("vogal + glide"), enquanto a segunda regra trata dos ditongos que variam como hiato ("glide + vogal"). Dado que, os ditongos necessitam desse tratamento especial, definiu-se que as novas regras, descritas abaixo, devem ser analisadas antes das 20 regras originais.

```
Ditongos decrescentes (<a>,<e>,<o>) + (<i>,<u>):

Se (<i>,<u>) for vogal tônica,
 então (<a>,<e>,<o>) são separadas de (<i>,<u>).
   exemplos: sa-í-da, gra-ú-go.
 senão (<a>,<e>,<o>) e (<i> or <u>) ficam na mesma sílaba.
   exemplos: cãi-bra, mai-se-na.
```

```
Ditongos que variam como hiato (<i>,<u>) + (<a>,<e>,<o>):

Se (<i>,<u>) + (<a>,<e>,<o>) não for final de palavra,
  então (<i>,<u>) são separadas de (<a>,<e>,<o>).
    exemplos: bi-o-ma.

Se (<i>,<u>) + (<a>,<e>,<o>) for final de palavra, então
  Se (<i>,<u>) ou (<a>,<e>,<o>) for vogal tônica,
    então (<i>,<u>) são separadas de (<a>,<e>,<o>).
      examplos: de-mo-cra-ci-a, ta-man-du-á.
    senão (<i> or <u>) e (<a>,<e>,<o>) ficam na mesma sílaba.
      examplos: só-cio, cí-lio.
```

Com relação às regras originais, apenas uma alteração foi julgada necessária. Antes, na regra 19, o núcleo da sílaba era unido ao próximo grafema da direita e separado dos subsequentes. Agora, o núcleo é separado do grafema seguinte. Essa nova versão da regra 19 consertou erros que ocorriam na separação de algumas palavras, como "teólogo", por exemplo.

Para avaliar o comportamento das duas regras adicionadas e da alteração sugerida, o resultado da separação silábica de 10.000 palavras da língua portuguesa foi analisado, tomando como referência a separação silábica fornecida pelo *Dicionário WEB*.[1] Os algoritmos com as regras originais e com as modificações obtiveram 2.044 e 737 erros, respectivamente, comprovando a eficiência das novas regras propostas. O pacote do separador silábico, incluindo o código-fonte, encontra-se publicamente disponível [1].

3.3 LapsNews: Um corpus de texto

Os modelos de linguagem são tipicamente construídos utilizando-se de modelos interpolados de transcrições ortográficas dos corpora de áudio e textos extraídos de jornais. Nosso primeiro corpus com textos de jornal foi o CETENFolha [78], que vem sendo expandido através de um processo totalmente automatizado de coleta e formatação diária de dez jornais brasileiros disponíveis na Internet. Após a obtenção dos arquivos de texto, alguns exemplos das operações de formatação são:

- Retirada de pontuação e *tags* ([ext], [t], [a], entre outras).

[1] http://www.dicionarioweb.com.br/

- Conversão para letras minúsculas.
- Expansão de números e acrônimos.
- Correção gramatical de palavras escritas incorretamente. Tarefa feita manualmente.

Um exemplo do resultado dessas operações é dado abaixo:

```
Antes: O Senado Federal tem uma <<caixa preta>> de R$ 1 milhão
Depois: o senado federal tem uma caixa preta de um milhão de reais
```

Hoje, o corpus de texto resultante tem aproximadamente 672 mil sentenças formatadas e é conhecido como LapsNews. O corpus LapsNews encontra-se publicamente disponível [1].

3.4 LapsStory: Um corpus de áudio

Dentre as dificuldades encontradas em se produzir grandes corpora de áudio, têm-se a coleta de dados (voz) e a transcrição ortográfica. Visando contornar tais problemas, foi construído um novo corpus de áudio, LapsStory, baseado em *audiobooks* (ou livros falados). Com os arquivos de áudio e suas respectivas transcrições (os próprios livros), uma redução considerável nos recursos humanos despendidos pode ser alcançada.

Os arquivos de áudio originais foram manualmente segmentados em arquivos menores, com aproximadamente 30 segundos de duração cada, e re-amostrados de 44.100 Hz para 16.000 Hz com 16 bits. Atualmente, o corpus LapsStory é composto por 8 locutores, sendo 5 do sexo masculino e 3 do sexo feminino. Os arquivos totalizam 16 horas e 17 minutos de áudio.

Infelizmente, o corpus de áudio LapsStory não pode ser completamente disponibilizado em função de alguns *audiobooks* serem protegidos por direitos autorais. Portanto, somente uma parte do corpus LapsStory encontra-se publicamente disponível, que corresponde a 9 horas de áudio [1].

Uma característica presente nos *audiobooks* é que o ambiente de gravação utilizado é bastante controlado, sendo assim, os arquivos não possuem ruído audível, ou seja, têm alta relação sinal/ruído. Dessa forma, quando tais arquivos são usados para treinar um sistema que irá operar em ambiente ruidoso, tem-se um problema com o descasamento acústico, que será abordado na Seção 4.2.

3.5 LapsBenchmark: Um corpus de referência

Com o intuito de obter uma boa avaliação de desempenho e possibilitar a comparação de resultados com outros grupos de pesquisas, foi construído o corpus de áudio LapsBenchmark. Busca-se aqui criar um corpus de referência com características mais próximas da operação de um sistema ASR em ambientes ruidosos. Isso distingue o corpus LapsBenchmark do LapsStory, previamente apresentado.

Para construção do corpus LapsBenchmark, foram utilizadas as sentenças descritas em [80]. Atualmente, o corpus possui 35 locutores (homens e mulheres) com 20 frases cada, que corresponde a 54 minutos de áudio. Todas as gravações foram realizadas em computadores usando microfones comuns de *desktop*. A taxa de amostragem utilizada foi de 16.000 Hz e cada amostra foi representada com 16 bits. Como mencionado, o ambiente não foi controlado, existindo a presença de ruído nas gravações. O corpus LapsBenchmark encontra-se publicamente disponível [1].

É sabido que o corpus LapsBenchhmark precisa ter seu tamanho consideravelmente aumentado para ser utilizado plenamente na realização de experimentos considerados como LVCSR. Nesse trabalho, usa-se uma estratégia que busca imitar a operação de um sistema LVCSR: o modelo de linguagem possui mais de 60 mil palavras, e o decodificador precisa lidar com alta perplexidade e descasamento acústico. Obviamente, tal estratégia permite avaliar aspectos importantes mas possui limitações. Uma dessas limitações, inerente à pouca quantidade de dados para teste, é a falta de robustez das estimativas de taxa de erro, visto que o conjunto de teste (corpus LapsBenchhmark) é relativamente reduzido.

Diferentemente dos anteriores, os próximos dois corpora não foram desenvolvidos nesta pesquisa. Contudo, os corpora de áudio Spoltech e West Point serão descritos por terem sido usados nos experimentos, após passarem por um processo de revisão manual.

3.6 Corpus Spoltech

O corpus de áudio Spoltech [81] foi criado pela Universidade Federal do Rio Grande do Sul, Brasil, pela Universidade Federal de Caxias do Sul, Brasil, e pelo Oregon Graduate Institute, EUA. O corpus está incluído no catálogo do LDC (LDC2006S16).

O corpus Spoltech consiste de gravações via microfone de 477 locutores de múltiplos gêneros e várias regiões do Brasil com suas respectivas transcrições fonéticas e ortográficas. As gravações consistem tanto de leituras de frases curtas quanto de respostas a perguntas (no

intuito de modelar a fala espontânea). No total, o corpus é composto por 8.080 arquivos de voz digitalizada (extensão wav), 2.540 arquivos com transcrições no nível de palavra (arquivos de texto sem alinhamento temporal, com extensão txt) e 5.479 arquivos com transcrições no nível de fone (com alinhamento temporal e extensão phn).

O ambiente de gravação não foi controlado. Assim, algumas gravações foram feitas em estúdio e outras em ambientes ruidosos (feiras, escolas, etc). Os dados foram gravados a uma taxa de amostragem de 44.100 Hz (mono, 16-bit). Embora útil, o corpus Spoltech possui vários problemas. Alguns arquivos de áudio não possuem suas correspondentes transcrições ortográfica e fonética, e vice-versa. Outro aspecto problemático é que suas transcrições possuem muitos erros. Assim, uma trabalhosa e manual correção dos arquivos de áudio e texto foi realizada [82]. No presente trabalho, apenas uma parte do corpus foi usada, consistindo de 477 locutores e totalizando 4,3 horas de áudio. Além disso, para utilização do mesmo de forma compatível aos demais corpora citados, os arquivos de áudio foram re-amostrados para 16.000 Hz.

3.7 Corpus West Point

O West Point Brazilian Portuguese Speech [83] é um corpus de gravações digitais para PB criado pelo governo dos EUA no intuito de desenvolver modelos acústicos para sistemas de reconhecimento de voz. O corpus é distribuído pelo LDC (LDC2008S04) e consiste de sentenças lidas por 60 mulheres e 68 homens, nativos e não-nativos. As sentenças gravadas via microfone se resumem a 296 frases e expressões.

Vale a pena ressaltar que o corpus West Point possui algumas restrições, como ausência de transcrições fonéticas e ortográficas para algumas gravações, e nenhuma com alinhamento temporal. Outro aspecto problemático é a existência de arquivos de áudio com falhas, como ruídos e fala não clara. Assim, uma etapa de pré-processamento foi realizada e 7.920 arquivos com locutores nativos foram selecionados, que correspondem a 8 horas de áudio. Uma parte do corpus é amostrada em 22.050 Hz e outra em 11.025 Hz, com 16 bits.

3.8 Um modelo acústico para PB

Esta seção descreve o desenvolvimento de um modelo acústico específico para PB (veja Figura 3.1). Estimar um bom modelo acústico é considerada a etapa mais difícil dentro do projeto de um sistema ASR. Para treinar um modelo acústico é necessário um corpus com voz

digitalizada, transcrito no nível de palavra (ortografia) e/ou no nível de fone. Alguns aspectos relacionados ao desenvolvimento do modelo acústico serão discutidos a seguir.

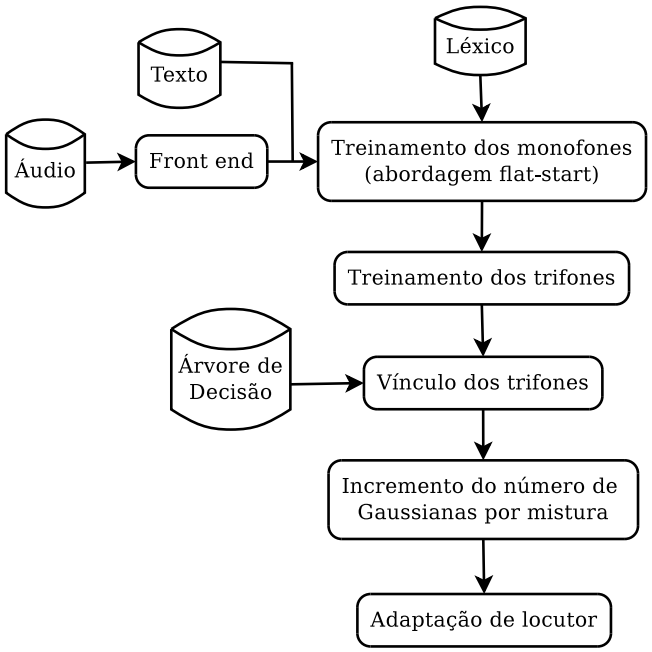


Figura 3.1: O processo de desenvolvimento do modelo acústico.

A atual versão do decodificador Julius trabalha apenas com os *front-ends* MFCC e, portanto, a abordagem MFCC foi empregada por conveniência. Mais especificamente, o *front-end* consiste do amplamente utilizado 12 coeficientes MFCCs [53], usando C0 como componente de energia e computado a cada 10 milisegundos (ou seja, 10 ms é o deslocamento do *frame*) para um *frame* de 25 ms. Em seguida, foram extraídas a primeira e segunda derivadas, compondo um vetor com 39 parâmetros para cada *frame*. Finalmente, a técnica *cepstral mean subtraction* foi usada para normalizar os coeficientes MFCCs [66].

O modelo acústico foi refinado de forma iterativa [40]. A abordagem conhecida como *flat-start* foi adotada [66]. Iniciando com modelos baseados em monofones e com uma Gaussiana por mistura, as HMMs foram gradualmente expandidas de forma a ter múltiplas Gaussianas por mistura com trifones a modelar as distribuições de saída. Em todo o processo de treino, o algoritmo de Baum-Welch [84] foi usado para re-estimar os modelos.

No total, foram utilizados 39 fones (38 monofones e um modelo de silêncio) com HMMs compostas por 3 estados na estrutura *left-to-right*. O modelo *short-pause* (sp), com apenas um estado emissor, foi construído através da cópia do estado central do modelo de silêncio. Depois disso, modelos trifones dependentes de contexto (*cross-word*) foram criados a partir dos monofones. As matrizes de transição dos trifones que possuem o mesmo fone base foram compartilhados.

Com um conjunto de categorias (ou "questões"), uma árvore de decisão fonética específica para PB foi elaborada para compartilhar os trifones com características fonéticas similares, conforme descrito na Seção 2.1. Para ilustrar, seguem abaixo algumas questões para classificação de vogais e consoantes usadas na construção da árvore de decisão:

```
...
QS "R_V-Fechada"    { *+i,*+e,*+o,*+u }
QS "R_V-Anterior"   { *+i,*+E,*+e }
QS "R_C-Palatal"    { *+S,*+Z,*+L,*+J }
QS "L_V-Posterior"  { u-*,o-*,O-* }
QS "L_V-Aberta"     { a-*,E-*,O-* }
...
```

Nota-se que, para um sistema trifone, é importante a inclusão de questões referentes aos contextos direito e esquerdo do fone. As questões devem avançar a partir de características gerais (como consoantes, vogais, ditongos, etc.) até particularidades mais específicas de cada fone. Teoricamente, o conjunto de questões carregado através dos comandos *QS* do *software* HTK deveria incluir todas as características linguísticas ou fonéticas que possam influenciar acusticamente o contexto de um determinado fone.

Após o processo de vínculo, o número de componentes por mistura nas distribuições foi gradualmente incrementado até 14 Gaussianas por mistura, finalizando o processo de treino. Os *scripts* usados para construir o modelo acústico e a árvore de decisão, específicos para PB, encontram-se publicamente disponíveis [1].

3.9 Um modelo de linguagem para PB

O treinamento de um modelo de linguagem requer o dicionário (léxico) escolhido e um arquivo com as sentenças de onde a contagem das palavras será extraída [85]. As palavras que não fazem parte do dicionário não serão contabilizadas, mesmo que estejam presentes nas sentenças de treino. A Figura 3.2 ilustra a forma geral dos processos de treino e teste do modelo de linguagem.

Os modelos n-gram são facilmente construídos exceto pela questão da suavização, ou seja, técnicas usadas para estimar as probabilidades na presença de poucos dados de treino. Várias são as técnicas disponíveis na literatura que podem ser usadas para suavizar os modelos n-gram [86]. Neste trabalho, aplicou-se a técnica Kneser-Ney [87].

A técnica de suavização Kneser-Ney é uma extensão do modelo de desconto absoluto e adota a heurística de que a probabilidade unigrama é proporcional não ao número de vezes que a palavra foi observada no contexto, mas sim ao número de diferentes contextos nos quais foi observada, produzindo dessa forma uma grande melhoria de desempenho [85].

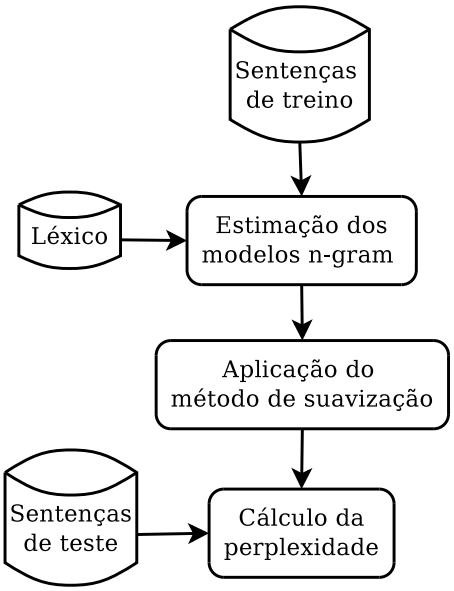


Figura 3.2: O processo de desenvolvimento do modelo de linguagem.

Uma descrição mais detalhada do processo de desenvolvimento dos modelos de linguagem testados pode ser vista na Seção 4.3. A seguir, é apresentada a API desenvolvida com o propósito de facilitar o uso do Julius, que é um decodificador código-livre de alto desempenho para reconhecimento de voz para grandes vocabulários.

3.10 Uma interface de programação de aplicativos

Ao buscar promover o amplo desenvolvimento de aplicações baseadas em reconhecimento de voz, observou-se que não era suficiente apenas tornar os recursos disponíveis, tais como modelos de linguagem. Esses recursos são úteis para pesquisadores, mas o que a maioria dos programadores desejam é a praticidade oriunda de uma API. Por isso, foi necessário complementar o código que faz parte do pacote de distribuição do Julius [25].

Recentemente, foi disponibilizada uma versão do pacote Julius totalmente SAPI 5.1 *compliant*, mas somente na língua japonesa. Logo, é extremamente complicado o uso do Julius com SAPI no caso de outras línguas, como o Português. O Julius também não suporta a especificação JSAPI, contudo possui sua própria API, implementada na linguagem de programação C/C++.

Neste contexto, o trabalho incluiu o desenvolvimento de uma API para facilitar a operação do decodificador Julius. A API proposta busca flexibilidade com relação à linguagem de programação. Além disso, o objetivo é prover suporte às plataformas Windows e Linux. Diante de tal cenário, a solução encontrada foi desenvolver uma API simples, com as funcionalidades necessárias para operar o Julius e implementações para C++ no Linux e plataforma Microsoft .NET no Windows. O suporte à plataforma Windows é baseado na especificação *common language runtime*, que permite comunicação entre as linguagens suportadas pela plataforma .NET (C#, Visual Basic, J#, entre outras).

A API desenvolvida permite o controle em tempo-real do decodificador Julius e da interface de áudio do sistema. Como pode ser visto na Figura 3.3, as aplicações interagem com o Julius através da API. Basicamente, a API abstrai do programador detalhes de baixo nível relacionados à operação do *engine*.

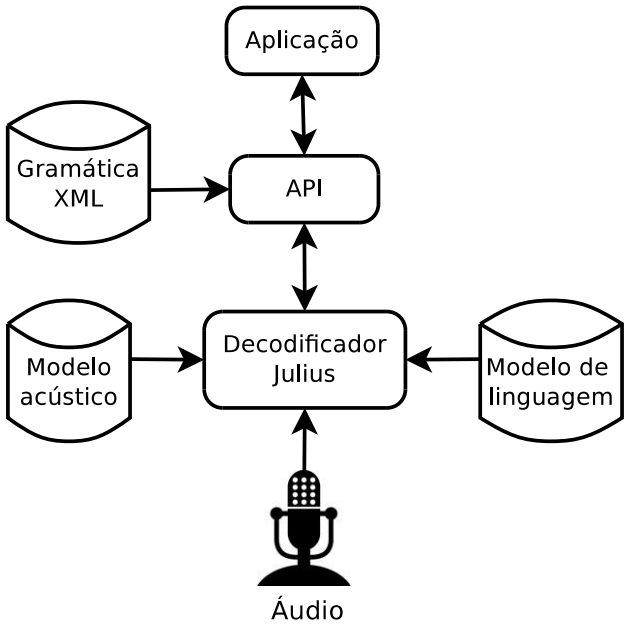


Figura 3.3: A API desenvolvida para facilitar a tarefa de operar o decodificador Julius.

Visto que a API suporta objetos compatíveis com o modelo de automação COM (*component object model*), é possível acessar e manipular (i.e. ajustar propriedades, invocar métodos) objetos compartilhados que são exportados por outras aplicações. Do ponto de vista da programação, a API consiste de uma classe principal denominada *SREngine*. Essa classe expõe à aplicação o conjunto de métodos e eventos descritos na Tabela 3.2.

A classe *SREngine* permite que a aplicação controle aspectos do decodificador Julius. É através dessa classe que a aplicação carrega os modelos acústico e de linguagem, inicia e para o reconhecimento, recebe eventos e resultados do processo de reconhecimento de voz.

Tabela 3.2: Principais métodos e eventos da API.

Métodos e Eventos	Descrição Básica
SREngine	Método para configurar e controlar o reconhecedor
loadGrammar	Método para carregar gramática SAPI XML
addGrammar	Método para carregar gramática nativa do Julius
startRecognition	Método para iniciar o reconhecimento
stopRecognition	Método para parar o reconhecimento
OnRecognition	Evento chamado quando alguma sentença é reconhecida
OnSpeechReady	Evento chamado quando o reconhecimento é ativado

Através do método *loadGrammar* é possível carregar uma gramática livre de contexto[2] especificada no formato SAPI XML. Para tornar isso possível, um conversor gramatical foi desenvolvido e integrado ao método *loadGrammar*. Essa ferramenta permite que o sistema converta automaticamente uma gramática de reconhecimento especificada no padrão SAPI Text Grammar Format [88] para o formato suportado pelo decodificador Julius.[3] O procedimento de conversão usa as regras gramaticais SAPI para encontrar as conexões permitidas entre as palavras, usando o nome das categorias como nós terminais. Isso também define as palavras candidatas em cada categoria, juntamente com as suas respectivas pronúncias.

É importante salientar que a atual versão do conversor não suporta regras gramaticais do tipo recursivas, facilidade suportada pelo decodificador Julius. Para esse tipo de funcionalidade deve-se carregar a gramática nativa do Julius através do método *addGrammar*. A especificação para esse tipo de gramática pode ser encontrada em [25].

O método *startRecognition*, responsável por iniciar o processo de reconhecimento de voz, basicamente ativa as regras gramaticais e abre o *stream* de áudio. Similarmente, o método *stopRecognition* desativa as regras e fecha o *stream* de áudio.

Alguns eventos também foram implementados. O evento *OnSpeechReady* sinaliza que o *engine* está ativado para reconhecimento. Em outras palavras, ele surge toda vez que o método *startRecognition* é invocado. Já o evento *OnRecognition* acontece sempre que o resultado do reconhecimento encontra-se disponível, juntamente com o seu nível de confiança. Assim, o programador tem a facilidade de aceitar ou rejeitar o resultado do reconhecimento em função do seu nível de confiança. A sequência de palavras reconhecidas e o seu nível de confiança são

[2] A gramática livre de contexto age como o modelo de linguagem, provendo ao reconhecedor regras que definem as palavras e frases que podem ser ditas.

[3] O Julius suporta tanto modelos n-gram como gramáticas livre de contexto.

passados da API para a aplicação através da classe *RecoResult*.

Através do conjunto reduzido de métodos e eventos apresentados acima é viável construir compactas aplicações com reconhecimento de voz usando o decodificador Julius. A Listagem 3.1 apresenta um código simples que reconhece a partir de uma gramática livre de contexto no formato XML e mostra o resultado na tela. O código-fonte e as bibliotecas da API, assim como alguns aplicativos, encontram-se publicamente disponíveis [1].

Listagem 3.1: Exemplo de código C# usando a API implementada.

```
namespace Test {
  public partial class Form1 : Form {
    private SREngine engine = null;
    public Form1() {
      SREngine.OnRecognition += handleResult;
    }
    public void handleResult(RecoResult result) {
      Console.Writeline(result.getConfidence() + " | "
        + result.getUterrance() + "\n");
    }
    private void but_Click(object sender, EventArgs e) {
      engine=new SREngine(@".\LaPSAM1.5.jconf");
      engine.loadGrammar(@".\grammar.xml");
      engine.startRecognition();
    }
  }
}
```

3.11 Recursos específicos à plataforma MARY

No intuito de construir um sistema TTS completo para PB na plataforma MARY, alguns recursos foram desenvolvidos e procedimentos executados, de acordo com os tutoriais descritos em [68]. Usando a nomenclatura adotada no *framework* MARY, a tarefa de prover suporte à uma nova língua pode ser dividida em duas partes: criação de um módulo para processamento de texto e produção da voz. A primeira foca no tratamento do texto em PB,

por exemplo, executar a conversão G2P, separação silábica e tonicidade. Já a criação de uma voz corresponde ao treinamento das HMMs usando o *software* HTS.

Para o módulo *front-end*, a plataforma MARY requer um conjunto de arquivos específicos para a língua alvo. Este trabalho adotou a receita sugerida por [68] para treinar transdutores de estados finitos (FST) [89]. O processo de treinamento desses FST necessita de um alfabeto fonético e uma lista com as palavras mais frequentes na língua em questão. Esses dois arquivos, construídos especificamente para o PB, são descritos abaixo:

- allophones.pt_BR.xml: é o alfabeto fonético, que precisa descrever as características distintivas de cada fone empregado, como vozeado ou não-vozeado, vogal ou consoante, entre outras. Esse arquivo foi construído usando 38 fones extraídos do alfabeto fonético SAMPA [76]. O alfabeto fonético elaborado pode ser conferido no Apêndice A.

- pt_BR.dic: é uma lista que contém todas as palavras planejadas (o sistema também é capaz de sintetizar as palavras fora do vocabulário) com suas respectivas transcrições fonéticas com base no alfabeto fonético, descrito anteriormente. Essas transcrições são, então, separadas em sílabas, com indicação de tonicidade, e etiquetadas foneticamente. Uma amostra de entrada do dicionário é exemplificada abaixo:

  ```
  acabou a-ka-'bow functional
  ```

Nota-se que a etiquetação padrão "functional" foi adotada em todas as palavras, dado que o bloco de etiquetação não foi implementado neste trabalho. A etiquetação (ou *speech tagging*) permitiria associar as palavras a diferentes classes e potencialmente obter melhores resultados.

Após construir o módulo *front-end*, o HTS foi usado para criar o módulo *back-end* baseado em HMMs. No intuito de facilitar o processo, o *framework* MARY disponibiliza a ferramenta *VoiceImport*, ilustrada na Figure 3.4, que possui uma rotina automática para treinamento das HMMs, ou seja, detalhes de implementação do HTS são estrategicamente abstraídos do desenvolvedor.

Assim, para o treinamento das HMMs, é necessário um corpus rotulado com as respectivas transcrições dos arquivos de áudio. Neste trabalho, utilizou-se a base de voz disponibilizada no pacote para PB do HTS [37], com 221 arquivos e aproximadamente 20 minutos de áudio. As transcrições no nível de palavra não foram encontradas no referido pacote, sendo confeccionadas manualmente.

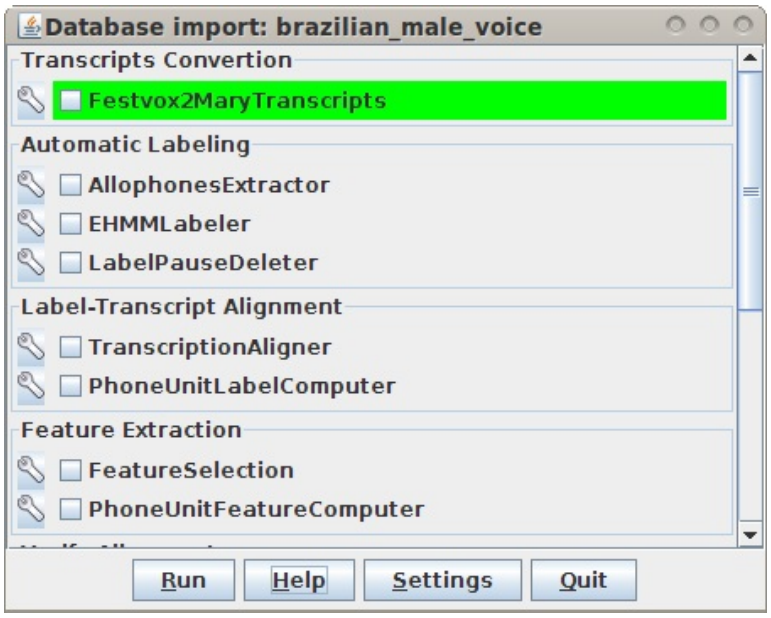


Figura 3.4: Interface principal da ferramenta *VoiceImport*.

Após a etapa de treinamento, o sistema TTS para PB encontra-se suportado pela plataforma MARY e a voz pode ser gerada a partir de um texto qualquer de entrada. Todos os arquivos construídos e procedimentos adotados, por conveniência para outros usuários, encontram-se publicamente disponíveis [1].

Tendo apresentado um resumo dos mais importantes recursos desenvolvidos para PB, o próximo capítulo discute alguns dos principais resultados obtidos.

Capítulo 4

Resultados experimentais dos sistemas baseados em HMM e TTS

Primeiramente, este capítulo apresenta os resultados experimentais que validam a integração dos recursos desenvolvidos em um sistema ASR em tempo-real, incluindo o impacto das técnicas de adaptação. Os modelos acústicos e de linguagem usados nos experimentos foram construídos seguindo os procedimentos descritos nas Seções 3.8 e 3.9, respectivamente. Por fim, o desempenho do sistema TTS implementado é comparado com outros dois sintetizadores, incluindo um *software* comercial. Todos os testes foram executados em um computador com processador Core 2 Duo Intel (E6420 2.13 GHz) e 1 GB de RAM.

A seguir, é descrito o primeiro experimento realizado, que avalia a eficiência do conversor G2P e a influência do dicionário fonético no desempenho de um sistema LVCSR.

4.1 Avaliação do conversor G2P e do UFPAdic

Três dicionários fonéticos foram comparados: o primeiro dicionário foi o UFPAdic; o segundo dicionário foi fielmente implementado de acordo com as regras fonológicas descritas em [74], ou seja, sem considerar as alterações propostas neste trabalho; e o terceiro dicionário seguiu a abordagem descrita em [75], que usou um classificador de árvore de decisão J4.8 para obtenção automática de regras para conversão G2P. Esse último dicionário adota um alfabeto fonético com apenas 34 fones, enquanto os demais usam 38 fones.

Com a posse de um conjunto de dados rotulados e corretamente transcritos, o mesmo pode ser usado para testes, fornecendo a resposta correta. No entanto, esta abordagem pode

ser questionada com relação à própria correção da base de dados rotulada. Em alternativa, este trabalho avalia os dicionários, observando seu impacto sobre a WER, que é um valor considerado como figura de mérito para os sistemas ASR.

Os modelos acústicos foram construídos usando somente o corpus West Point, que foi dividido em dois conjuntos distintos: treino e teste. Nos experimentos realizados, a base de treino foi composta por 6.334 arquivos, que corresponde a 384 minutos de áudio, e a base de teste com os restantes 1.586 arquivos de áudio, com um total de 96 minutos de áudio. Os arquivos com taxa de amostragem igual a 22.050 Hz foram re-amostrados para 11.025 Hz.

O vocabulário foi composto pelas 679 palavras presentes nas transcrições do corpus West Point. Modelos de linguagem bigramas foram usados, com o número de sentenças para treinamento variando entre 1.000 e 180.000. Para o cálculo da perplexidade, foram utilizadas 1.000 frases não vistas na fase de treino. Todas as sentenças foram extraídas de forma gradual do corpus CETENFolha. É importante observar que os corpora West Point e CETENFolha podem ser considerados disjuntos com relação ao conteúdo das sentenças. A Tabela 4.1 mostra as perplexidades encontradas em cada modelo de linguagem. Como esperado, a perplexidade diminui à medida que o número de sentenças usadas no treinamento aumenta [12].

Tabela 4.1: Perplexidade para diferentes tamanhos da base de treino.

	Número de sentenças							
	1k	10k	30k	60k	90k	120k	150k	180k
Perplexidade	44,9	33,3	26,2	21,7	19,2	16,4	15,7	14,8

Dado que não há interesse em operação em tempo-real neste experimento, a ferramenta HDecode foi usada para efetuar os testes de decodificação e os resultados podem ser visualizados na Figura 4.1, com a WER diminuindo à medida que o número de sentenças usadas para treinar o modelo de linguagem aumenta. Contudo, observou-se que os resultados permaneceram praticamente constantes em alguns intervalos. A razão para isso pode estar relacionada com uma saturação do modelo de linguagem, onde quase todas as sequências comuns de palavras já foram observadas, entretanto, as sequências mais raras permanecem fora do âmbito da base de treino. Assim, a tarefa de estimar corretamente a probabilidade desses pares de palavras é bem mais complexa, já que adiciona-se uma quantidade reduzida de dados novos, sendo insuficiente para melhorar o desempenho do modelo de linguagem.

A comparação entre as diferentes abordagens deve considerar o tamanho dos transdutores resultantes e outras características que também podem ser bastante relevantes [69], como o fato da abordagem por aprendizado de máquina requerer alinhamento lexical, diferentemente

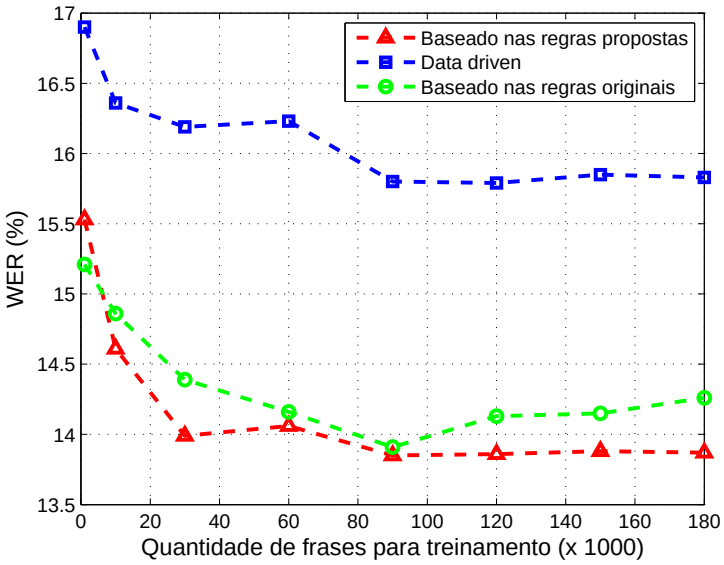


Figura 4.1: Uma comparação entre os dois dicionários baseados em regras e o que se baseia em aprendizado de máquina (*data-drive*).

da abordagem baseada em regras pré-estabelecidas. Como esperado, os melhores resultados foram obtidos com os sistemas baseados em regras, contudo deve-se levar em conta a questão da elaboração manual do dicionário fonético usado como conjunto de treino rotulado do classificador empregado em [75]. Diante de um modelo de linguagem independente do contexto do corpus West Point, ficou nítido que as mudanças sugeridas às regras de [74] melhoraram o desempenho do decodificador.

O próximo experimento mostra como técnicas de adaptação de locutor podem ser utilizadas para tratar o problema de descasamento acústico entre diferentes corpora de áudio.

4.2 Adaptação de locutor vs descasamento acústico

Esse experimento foi realizado em duas etapas. Inicialmente, três modelos acústicos foram criados a partir dos corpora de áudio Spoltech e LapsStory, sendo dois com as bases individuais e um combinando as duas bases de dados. Um modelo de linguagem trigrama, com perplexidade igual a 166, foi criado a partir das sentenças extraídas dos corpora Spoltech, West Point, OGI22 [90], LapsStory, CETENFolha e LapsNews. O dicionário fonético utilizado para a construção dos modelos estatísticos foi o UFPAdic.

As simulações foram realizadas utilizando o decodificador HDecode e o corpus de referência LapsBenchmark foi usado como conjunto de teste. Com isso, busca-se avaliar como

o sistema se comporta em aplicações com descasamento acústico. A Tabela 4.2 mostra os resultados obtidos com os diferentes modelos acústicos.

Tabela 4.2: Resultados obtidos com os modelos acústicos do LapsStory e Spoltech.

Modelo acústico (peso 1.0)	**WER(%)**	**xRT**
Spoltech	34,8	6,0
LapsStory	52,5	10,0
LapsStory+Spoltech	40,0	9,0
Modelo acústico (peso 2.0)	**WER(%)**	**xRT**
Spoltech	44,3	1,5
LapsStory	55,9	3,0
LapsStory+Spoltech	48,5	1,6

Nos testes realizados usando os modelos treinados individualmente, verificou-se altas taxas de erro, principalmente no caso do corpus LapsStory, já que o mesmo foi produzido em um ambiente acústico totalmente diferente do corpus LapsBenchmark. O modelo acústico criado a partir da combinação das bases de dados também apresentou resultados ruins, mesmo quando se realizou a normalização da média e da variância dos parâmetros MFCCs. Nessa primeira simulação, o sistema também apresentou um alto valor de xRT, o que impossibilita a utilização dos mesmos em aplicações de tempo-real. De forma a aumentar a velocidade do sistema, modificou-se o peso do modelo acústico para 2.0 no processo de decodificação (parâmetro *-a* do HDecode), o qual acelera o processo de busca. Essa alteração tornou o sistema até 4 vezes mais rápido, porém acarretou em perda de precisão, como mostrado na segunda parte da Tabela 4.2.

Na segunda etapa de testes, foram adotadas as técnicas de adaptação de locutor MLLR e MAP na tentativa de melhorar o desempenho do sistema. Porém, diferente de uma adaptação de locutor convencional, realizou-se uma adaptação de ambiente (ou "environment adaptation"). Partindo do modelo acústico produzido exclusivamente com o corpus LapsStory, fez-se adaptações com todo o corpus Spoltech usando o *software* HTK. Assim, o modelo acústico criado a partir do corpus LapsStory, treinado com uma maior quantidade de dados, com poucos locutores e alta relação sinal/ruído, foi modificado de maneira a melhor modelar ambientes ruidosos e a fala de diversos locutores. Os resultados das simulações usando novamente o HDecode e o corpus LapsBenchmark são exibidos na Tabela 4.3.

A combinação das técnicas de adaptação MLLR e MAP apresentou os melhores resultados. Trabalhos relacionados apontam vantagens no uso combinado das duas técnicas [61].

Tabela 4.3: Resultados obtidos com as técnicas de adaptação de locutor. Na segunda parte da tabela, modificou-se o peso do modelo acústico para 2.0 no processo de decodificação.

Técnica de adaptação (peso 1.0)	**WER(%)**	**xRT**
MLLR	30,2	9,5
MAP	29,3	7,0
MLLR+MAP	25,5	8,3
Técnica de adaptação (peso 2.0)	**WER(%)**	**xRT**
MLLR	34,2	2,2
MAP	35,6	2,0
MLLR+MAP	29,6	2,4

De forma geral, a técnica MLLR trabalha com agrupamento de Gaussianas com características semelhantes via matrizes de transformação. Isso permite que a mesma seja eficiente mesmo para pequenos conjuntos de dados, mas não obtém desempenho ótimo quando o conjunto de dados é grande. Em contraste, a MAP utiliza as Gaussianas dos trifones individualmente. Isso possibilita incremento de desempenho ao custo de um maior conjunto de dados para treino. Como visto na segunda parte da Tabela 4.3, a queda na WER com o uso das duas técnicas de adaptação em conjunto é de mais de 14% quando comparado com o modelo acústico não adaptado do corpus Spoltech, que teve o melhor desempenho na primeira etapa de teste.

4.3 Avaliação do sistema LVCSR para PB

Dado que os resultados das seções anteriores foram obtidos com uma configuração simplificada, os próximos experimentos empregaram todos os recursos desenvolvidos para reconhecimento de voz em PB. O modelo acústico independente de locutor foi inicialmente treinado com os corpora LapsStory e West Point, totalizando 21,65 horas de áudio. Do corpus West Point, foram utilizados apenas os arquivos com taxa de amostragem igual a 22.050 Hz, que foram re-amostrados para 16.000 Hz. O dicionário fonético utilizado foi o UFPAdic.

Em seguida, o *software* HTK foi usado para adaptar o modelo acústico, usando as técnicas MLLR e MAP com o corpus Spoltech em treinamento supervisionado (*offline*). No treinamento supervisionado a transcrição da voz é conhecida, ou seja, tem-se conhecimento do que está sendo falado. O corpus de referência LapsBenchmark foi usado para avaliar os sistemas, em termos de WER, em todos os testes descritos a seguir.

4.3.1 Avaliação do modelo de linguagem

Esse experimento avalia a perplexidade do modelo de linguagem em função do número de sentenças usadas para treiná-lo (veja Figura 4.2). O *toolkit* SRILM foi usado para construir os modelos de linguagem trigrama no formato ARPA. O número de sentenças usadas para treinar os modelos variou entre 255.830 e 1.534.980, extraídas dos corpora CETENFolha e LapsStory. O vocabulário foi mantido constante com as 65.532 palavras do UFPAdic.

As sentenças usadas para mensurar a perplexidade foram separadas em três conjuntos de teste com 10.000 frases cada um, não vistas durante a fase de treino. O primeiro conjunto de teste foi composto apenas com sentenças extraídas do corpus CETENFolha; o segundo conjunto foi construído com sentenças do corpus LapsNews; e o terceiro englobou sentenças presentes nos corpora CETENFolha e LapsNews. Note que, em termos de sentenças, os corpora CETENFolha e LapsNews podem ser considerados disjuntos.

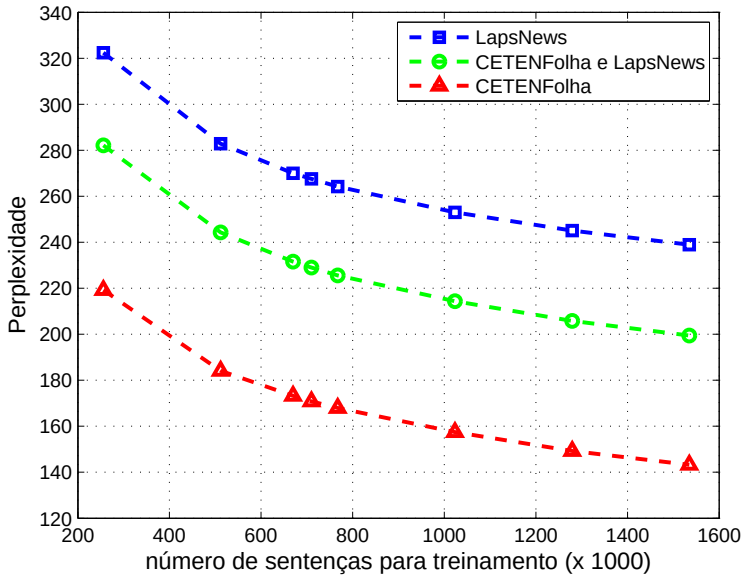


Figura 4.2: Perplexidade em função do número de sentenças de treino.

Como esperado, a perplexidade tende a diminuir com o aumento do número de sentenças de treino. Isso está relacionado ao fato de que as estatísticas dos modelos treinados melhoram à medida que mais ocorrências de trincas de palavras são registradas na base de dados de treino. É importante observar, na Figura 4.2, a diferença entre as perplexidades calculadas para os diferentes conjuntos de teste. A perplexidade é maior quando as sentenças usadas para avaliar o modelo de linguagem são coletadas de um corpus de texto diferente do usado para treiná-lo. Isso pode ser visualizado nas curvas do CETENFolha e LapsNews, dado que o último não foi usado para treinar os modelos. Em mil sentenças essa diferença é de 37%.

Outro experimento analisou a influência dos modelos de linguagem treinados anteriormente no desempenho do sistema LVCSR. As Figuras 4.3 e 4.4 mostram a evolução do fator xRT e WER, respectivamente, em função do número de sentenças de treino com os decodificadores HDecode e Julius. Apenas uma Gaussiana foi usada para modelar as distribuições de saída das HMMs.

Note que, quanto maior o número de sentenças de treino, maior número de trincas de palavras serão incorporadas ao modelo de linguagem (o mesmo aumenta em tamanho). O impacto de um modelo de linguagem maior, e eventualmente melhor estimado, com o xRT não é trivial. A Figura 4.3 indica que o Julius foi relativamente insensível ao aumento (e melhor estimativa) do modelo de linguagem, enquanto o HDecode sofreu ligeiro aumento do xRT.

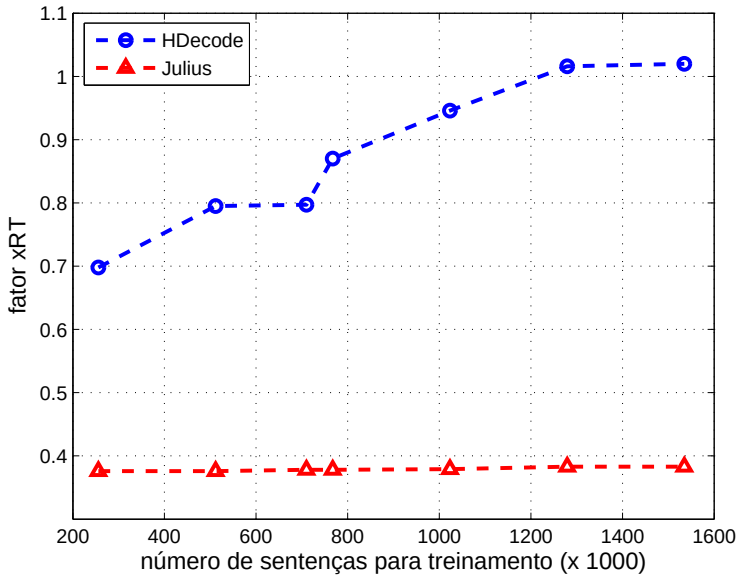


Figura 4.3: Fator xRT em função do número de sentenças de treino.

Através da Figura 4.4 verifica-se que a WER parou de decrescer após 700.000 sentenças. Em contraste, a Figura 4.2 indica que a perplexidade continua caindo mesmo após o limiar de 700 mil sentenças. Isso pode ser entendido pelo fato da perplexidade na Figura 4.2 ser estimada com um conjunto de teste distinto das sentenças do corpus LapsBenchmark. Em função disso e do fato do HDecode ter apresentado ligeiro aumento do xRT, nos testes seguintes, onde o modelo de linguagem é fixo, somente o modelo treinado com 710.000 sentenças foi considerado.

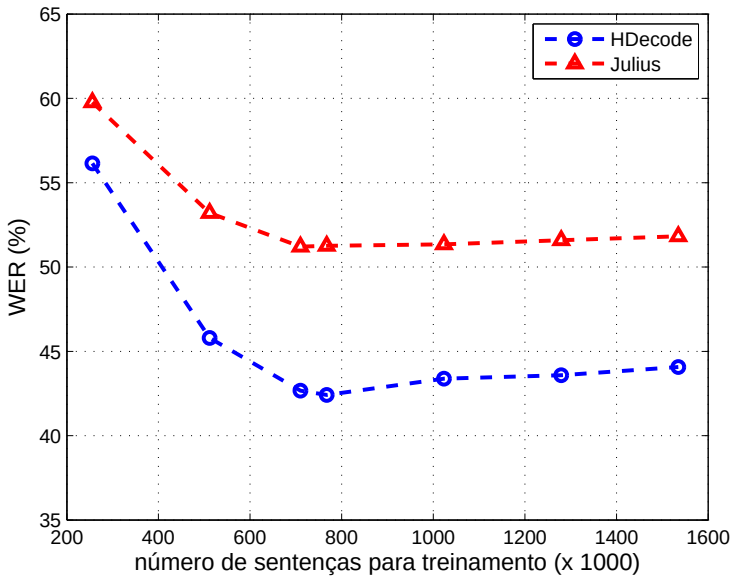


Figura 4.4: WER em função do número de sentenças de treino.

4.3.2 Aumentando o número de Gaussianas

O objetivo desse experimento é avaliar o comportamento do sistema LVCSR à medida que o número de Gaussianas usadas para modelar as distribuições de saída é incrementado. Assim, o número de Gaussianas foi gradativamente aumentado de uma única Gaussiana até 20 Gaussianas por mistura, como mostrado nas Figuras 4.5 e 4.6.

Observou-se que, o custo computacional adicionado para incrementar o número de Gaussianas é compensado por um melhor desempenho dos decodificadores. No entanto, a WER parou de decrescer após 18 Gaussianas por mistura, por isso os próximos testes consideraram esse número como padrão.

4.3.3 Avaliando os parâmetros de decodificação

Durante os experimentos, constatou-se que o *pruning* é o procedimento que mais influência no desempenho dos decodificadores. O processo de *pruning* é realizado a cada intervalo de tempo, registrando as melhores hipóteses e eliminando todas as hipóteses que possuem os *logs* das probabilidades de transição e de emissão dos estados das HMMs abaixo do limiar definido pelo parâmetro *beam-width*.

Configurar o parâmetro *beam-width* é, então, um compromisso entre velocidade e a tentativa de evitar erros de busca pela melhor concorrente, como mostrado na Figura 4.7.

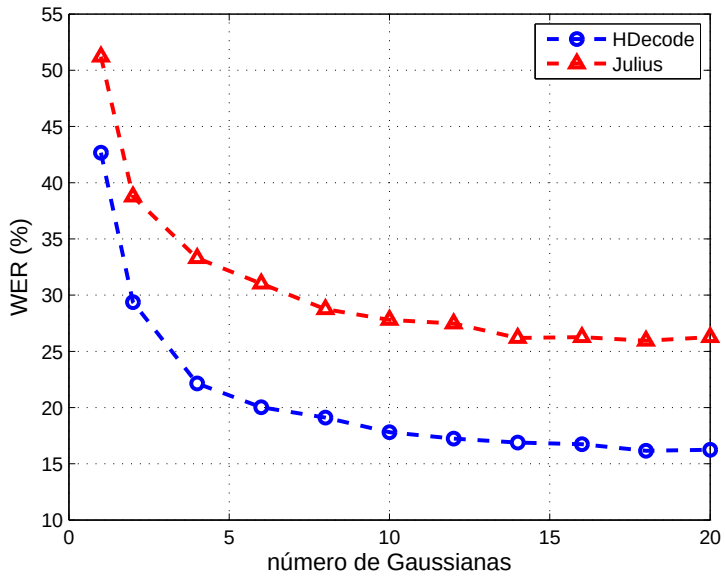


Figura 4.5: WER em função do número de Gaussianas usado para treinar o modelo acústico.

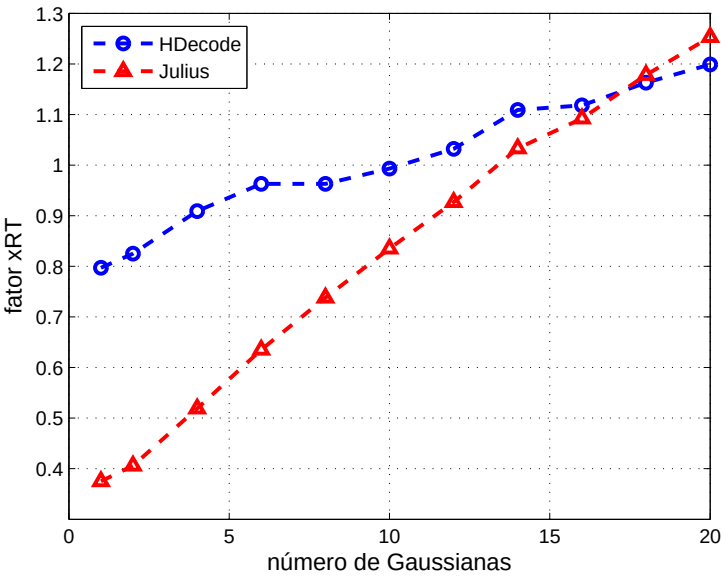


Figura 4.6: xRT em função do número de Gaussianas usado para treinar o modelo acústico.

O Julius é um decodificador de dois passos (*forward* e *backward*) com modelos bigrama e trigrama no primeiro e segundo passos, respectivamente. Os parâmetros de configuração do Julius, como o *beam-width*, podem ser ajustados para ambos os passos. Assim, o valor do parâmetro *beam-width* foi variado no primeiro passo e fixado em 200 no segundo passo do processo de decodificação.

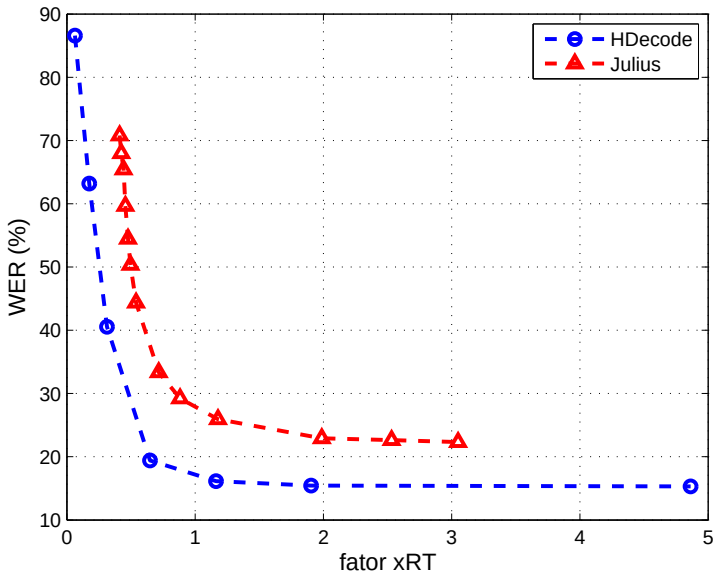


Figura 4.7: WER em função do fator xRT usando os decodificadores Julius e HDecode.

Os testes foram realizados variando o valor do parâmetro *beam-width* de 100 até 400 e 10.000 no HDecode e Julius, respectivamente. Isso porque a WER parou de decrescer, enquanto que o fator xRT continuou a aumentar significativamente. Isto é, com o incremento do parâmetro *beam-width* tem-se uma considerável redução na WER, pois quanto maior for o valor do parâmetro *beam-width*, menos hipóteses serão descartadas durante a busca, aumentando as chances de acerto por parte do decodificador. Porém, o aumento do espaço de busca através do incremento do parâmetro *beam-width* torna os decodificadores mais lentos.

O Julius mostrou-se capaz de implementar um processo de *pruning* mais agressivo que o HDecode, sem um crescimento significativo no fator xRT. Em contrapartida, o Julius não foi capaz de obter a mesma WER atingida pelo HDecode. Então, os melhores valores de WER encontrados para o HDecode e Julius foram 16,15% e 25,94%, respectivamente. Por conveniência, o valor do fator xRT foi mantido em torno de um. Um resumo dos parâmetros de decodificação utilizados são descritos nas Tabelas 4.4 e 4.5. A descrição detalhada dos parâmetros do HDecode e Julius pode ser encontrada em [66] e [25], respectivamente.

Com os resultados obtidos, podemos concluir que é possível construir sistemas LVCSR para PB usando os recursos produzidos neste trabalho. O modelo acústico independente de locutor e o modelo de linguagem usado nos testes encontram-se publicamente disponíveis [1].

Tabela 4.4: Parâmetros utilizados nos testes com o HDecode.

Parâmetro	Valor
Pruning beam-width	250.0
Language model scale factor	20.0
Word insertion penalty	22.0
Word end beam-width	100.0
Number of tokens per state	8
Acoustic scale factor	1.5

Tabela 4.5: Parâmetros utilizados nos testes com o Julius.

Parâmetro	Valor
Pruning beam width for the first pass	2000.0
Pruning beam width for the second pass	200.0
Language model weight for the first and second passes	15.0
Word insertion penalties for the first and second passes	10.0
Score envelope width	300.0

4.3.4 Modelo acústico dependente de locutor

O objetivo desse experimento é avaliar o comportamento do sistema LVCSR quando o modelo acústico é adaptado a voz de um locutor específico. Primeiramente, o modelo acústico independente de locutor, adotado até então, foi testado com parte do corpus LapsBenchmark, que corresponde a 1,55 minutos de áudio gravados exclusivamente com a voz de um locutor do sexo masculino. Os resultados podem ser conferidos na Figura 4.8.

Em seguida, um modelo dependente de locutor foi gerado a partir do modelo independente de locutor. Para isso, o mesmo locutor do teste anterior contribuiu com 10,8 minutos de sua voz. Então, o áudio coletado foi utilizado para adaptar o modelo acústico independente de locutor, usando as técnicas de adaptação MLLR e MAP, de acordo com o tutorial descrito em [66]. Como esperado, o processo de adaptação de locutor melhorou o desempenho dos decodificadores, o que pode ser comprovado comparando os resultados das Figuras 4.8 e 4.9.

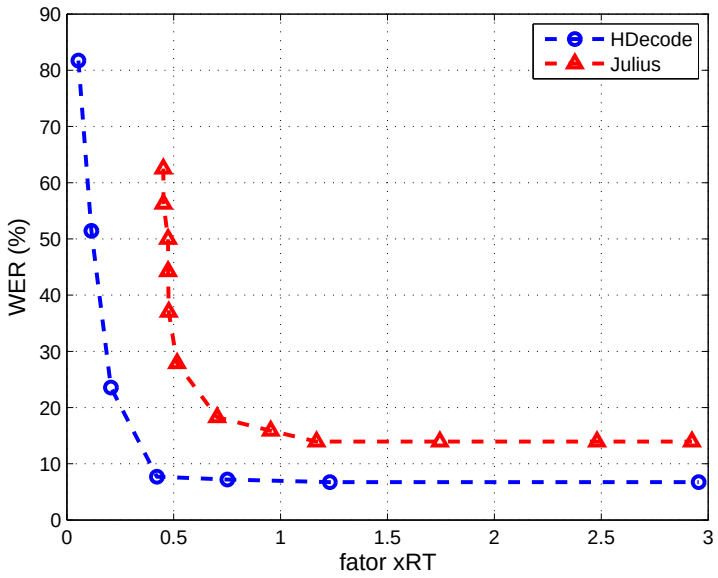


Figura 4.8: WER e xRT observados nos testes com reconhecimento independente de locutor.

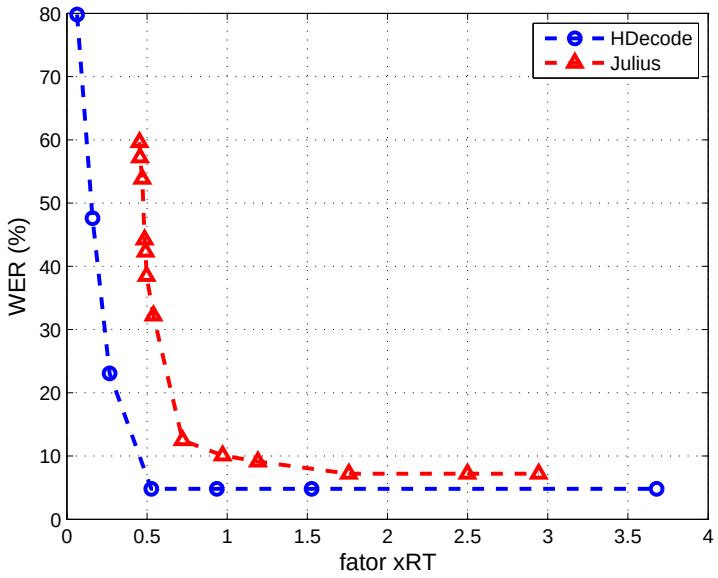


Figura 4.9: WER e xRT observados nos testes com reconhecimento dependente de locutor.

4.4 Avaliação do sistema TTS para PB

Esse experimento avalia o sistema TTS implementado. Testes comparativos foram realizados com outros dois sintetizadores para PB: Liane e Raquel. Liane é um sintetizador MBROLA [24] desenvolvido pelo Serviço Federal de Processamento de Dados (SERPRO/PA) e Raquel é um *software* comercial da Nuance [23]. Ambos são sintetizadores concatenativos

(difones) e foram usados apenas para fins de comparação preliminar, ou seja, não há ambição de se obter conclusões definitivas em relação a quanto um TTS é melhor que o outro.

O processo de avaliação foi dividido em dois estágios: inteligibilidade e naturalidade. A naturalidade da voz foi analisada através do protocolo MOS [65] e a WER foi usada para mensurar a inteligibilidade. Vinte sentenças (com 5 segundos de duração cada uma) foram coletadas na Internet e usadas nos testes. Então, os sessenta arquivos de áudio foram aleatoriamente tocados e os ouvintes (participantes) convidados a dar uma nota (veja Tabela 4.6) para a naturalidade da voz e repetir o que escutaram. No total, dez pessoas, sem formação na área de processamento de voz, foram entrevistadas.

Tabela 4.6: Esquema de classificação do protocolo MOS.

MOS	Qualidade	Imparidade
5	Excelente	Agradável
4	Bom	Pouco agradável
3	Razoável	Ligeiramente irritante
2	Pobre	Irritante
1	Ruim	Muito irritante

Os resultados dos experimentos podem ser conferidos nas Figuras 4.10 e 4.11. Na avaliação subjetiva da naturalidade, o sistema TTS e Liane foram considerados ligeiramente irritantes, enquanto Raquel foi avaliado como um bom sintetizador. Observou-se que, a voz gerada pelos sistemas mais evoluídos ainda soa menos natural que a voz humana.

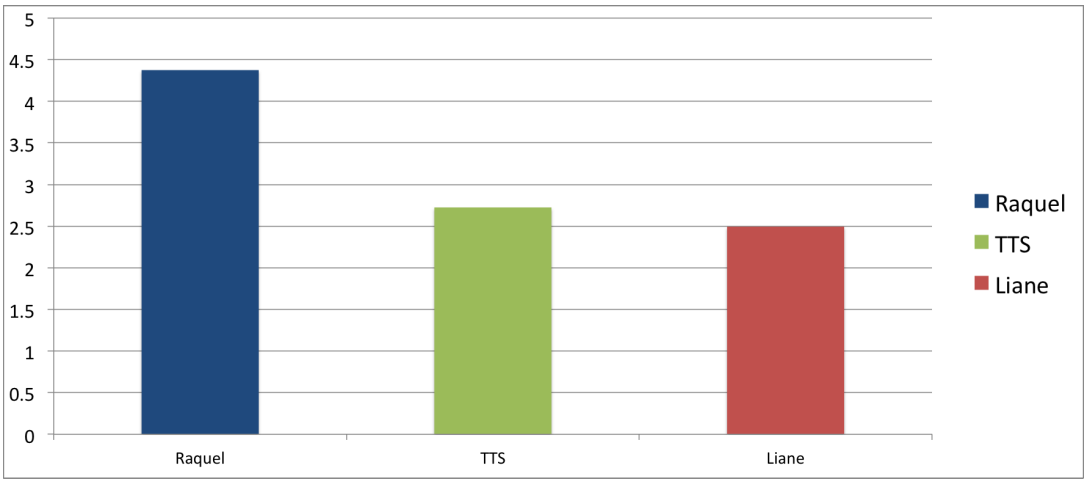


Figura 4.10: A média da naturalidade para cada sintetizador avaliado.

No teste objetivo para avaliar a inteligibilidade, o sistema TTS (com WER em torno de 10%) superou o sintetizador Liane. Já o *software* Raquel obteve novamente o melhor desempenho, com WER próxima de zero.

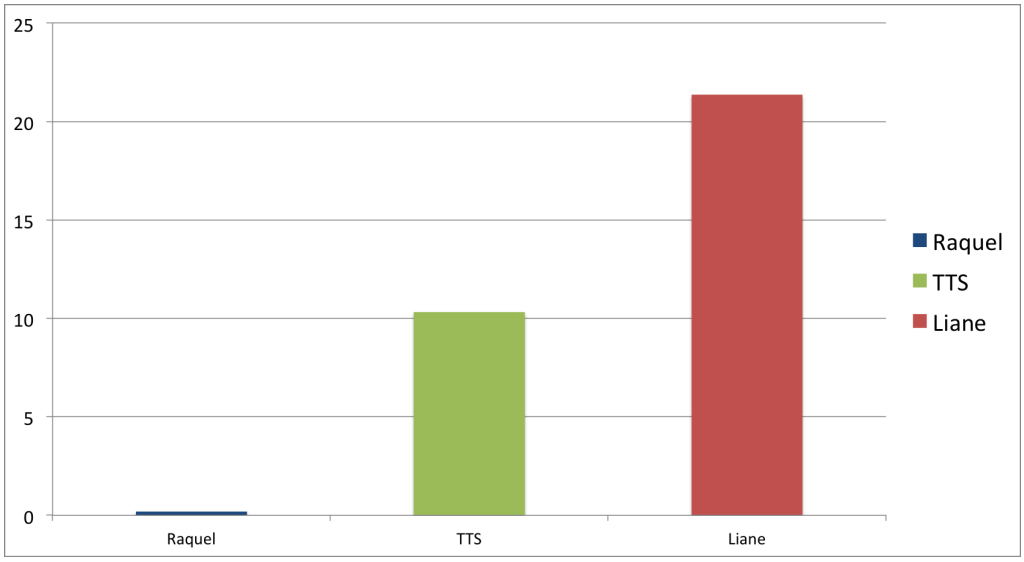


Figura 4.11: WER observada para cada sintetizador avaliado.

O próximo capítulo apresenta uma proposta que usa a saída de classificadores binários para ajudar algoritmos de decodificação tradicionais a resolver situações de conflito entre pares de fones concorrentes. Para cada um desses problemas binários, são usadas técnicas de seleção automática de parâmetros para escolher a representação paramétrica mais adequada para o problema em questão.

Capítulo 5

Reconhecimento de voz baseado em classificadores binários

Um sistema ASR no estado da arte é quase sempre construído usando-se métodos *data-driven* probabilísticos, como os modelos ocultos de Markov [45], incrementados através da coleta de muitos dados de treino e re-estimação dos modelos. A integração de fontes de conhecimento extras é benéfica ao sistema ASR. Contudo, o arcabouço probabilístico representa uma restrição em como pode ser feita essa integração. Por exemplo, quando se combina os escores oriundos dos modelos de linguagem e acústico, é necessário usar um "peso" escolhido empiricamente para que essa combinação funcione bem na prática.

O presente trabalho apresenta uma proposta para ASR que busca diminuir a *confusão* entre pares de fones através de classificadores ou detectores binários baseados em técnicas de aprendizado de máquina [91–93]. O objetivo é identificar instantes (ou, mais especificamente, "frames" ou quadros) com confusão entre fones dentro de uma lista com as melhores sentenças candidatas (ou hipóteses), provida pelo sistema ASR convencional baseado em HMMs. Um classificador binário provê um escore que é então combinado com o escore fornecido pelo sistema HMM convencional, atingindo-se com isso um melhor desempenho global do sistema.

No intuito de avaliar a proposta, um sistema para reconhecimento de dígitos independente de locutor e baseado em modelos monofones foi construído usando a base de dados TIDIGITS [94]. Apesar dessa tarefa não apresentar o apelo de LVCSR, em função do imenso número de graus de liberdade para desenvolver um sistema completo, é importante utilizar uma base pequena para tornar o custo computacional razoável. Sabe-se contudo, que algumas técnicas que funcionam bem para bases pequenas acabam apresentando problemas de escala quando aplicadas a grandes bases. Com essa preocupação em mente, o sistema foi desen-

volvido partindo-se da premissa que poderá ser adaptado a bases maiores. Antes de descrever a proposta em si, faz-se a seguir a devida conexão da mesma a pesquisas prévias.

A proposta pode ser vista como uma adaptação do sistema apresentado em [95], no qual classificadores binários foram usados a cada quadro, ou seja, o reconhecimento não utilizou *lattices*. Tal estratégia implica em ter os escores oriundos dos classificadores binários influenciando na decodificação das HMMs através do algoritmo Viterbi [96], visto serem os mesmos incorporados a cada quadro. Em contraste, na presente proposta, o sistema HMM trabalha de forma autônoma, independente dos classificadores binários, os quais são consultados apenas ao se obter uma *lattice* oriunda exclusivamente do sistema baseado em HMMs.

A estratégia para definir os classificadores binários apoia-se no conceito de **margem**, que é amplamente empregado em aprendizado de máquina [97]. A técnica de maximização de margem foi primeiramente aplicada em classificadores de aprendizado, como *support vector machines* [98], e mais recentemente em HMMs [49, 99, 100]. Apesar de assumir definições distintas, dependendo do algoritmo de aprendizagem adotado, a margem é positiva se o exemplo é rotulado corretamente e negativa se o exemplo é rotulado incorretamente.

Em [2], o autor apresenta o paradigma *Automatic Speech Attribute Transcription* (ASAT) para reconhecimento de voz como uma nova forma de agregar conhecimento aos sistemas baseados em métodos *data-driven*. O conceito de ASAT tem similaridade com a proposta apresentada no que se refere à utilização de novas informações, não apenas as utilizadas pelos sistemas baseados em HMMs. Por exemplo, em [101, 102], conhecimentos adquiridos a partir de características acústicas e fonéticas da fala foram incorporados ao *front-end* de um sistema de reconhecimento de voz baseado em HMMs.

Aplicações do conceito ASAT em sistemas ASR podem ser encontradas [103, 104]. Em [103], características articulatórias adquiridas através de classificadores baseados em algoritmos de aprendizado de máquina [105] serviram como informações complementares que foram combinadas aos escores acústicos providos pelas HMMs para reduzir as taxas de erro por fone e palavra usando a técnica de "re-escore das *lattices*". A WER foi reduzida de 4,54% para 4,03% em um sistema ASR para reconhecimento de dígitos. Em [104], uma estratégia de *pruning* foi implementada e usada como fonte de conhecimento na tentativa de reduzir falsos alarmes de detecção e consequentemente decrementar a WER em sistemas ASR.

Nos trabalhos citados, os escores fornecidos pelo modelo de linguagem não foram levados em consideração e os testes se limitaram a pequenos vocabulários. De outra forma, em [106], a contribuição do modelo de linguagem foi considerada no processo de re-escore das *lattices* em um sistema LVCSR. O corpus utilizado foi o Wall Street Journal (WSJ) e os pesos de interpolação foram empiricamente estimados. Os experimentos realizados mostraram que

o método foi capaz de corrigir erros em tarefas de reconhecimento de voz. Já em [107], classificadores foram usados para treinar diretamente um típico sistema ASR baseado em HMMs, produzindo um padrão de erro diferente de um sistema convencional baseado em coeficientes cepstrais, mas nenhuma melhoria foi relatada.

Uma análise da literatura recente em ASR indica o interesse em métodos para re-escore de *lattice* [108], e a presente proposta segue essa linha. É fato que a latência do sistema aumenta quando se adiciona essa fase de pós-processamento, mas o foco do trabalho foi na diminuição da taxa de erro, em detrimento das questões de custo computacional e latência (ou, equivalentemente, xRT). A próxima seção descreve em detalhes o método proposto.

5.1 O método proposto

5.1.1 Princípio de funcionamento

Um sistema de ASR convencional baseado em HMMs é utilizado para gerar uma série de hipóteses, as quais são organizadas sob a forma de uma *lattice*. Registra-se que a *lattice* é apenas uma estrutura de dados que permite representar as hipóteses de uma maneira eficiente. Para fins de entendimento, pode-se assumir que o sistema de HMMs gera uma *lattice* que é convertida em uma lista com as melhores hipóteses (ou *N-best list*).

Uma hipótese é aqui representada por uma sequência q de T estados, onde T é o número de *frames* (ou quadros, ou ainda segmentos) da sentença. Uma sequência de palavras $\mathcal{T}$ é convertida em fones através do dicionário fonético e depois, associando-se cada fone a sua correspondente HMM, encontra-se a respectiva sequência de estados q das HMMs. Por exemplo, $q = \{v[2], v[2], s[1], s[2]\}$ exprimiria que para uma hipótese com $T = 4$ quadros, os dois primeiros foram associados ao estado 2 do fone v e os dois últimos quadros aos estados 1 e 2 do fone s.

Para cada sequência de palavras $\mathcal{T}_i$, é calculado o escore total $s_i = p(\mathbf{X}|\mathcal{T}_i)p(\mathcal{T}_i)$. A parcela devida ao modelo acústico $p(\mathbf{X}|\mathcal{T}_i)$ é obtida pela soma dos escores acústicos $b_i(t), t = 1, \ldots, T$, de cada um dos T quadros de q_i.

Após as hipóteses serem obtidas pelo sistema baseado em HMMs e seus respectivos escores estarem disponíveis, o sistema proposto utiliza classificadores binários g que buscam detectar potenciais "confusões" (descritas formalmente adiante) a serem remediadas por outro conjunto de classificadores binários f. O processo é feito para cada quadro t. Para um dado t, caso um detector de confusões g seja ativado por ter estimado haver confusão entre as hipóteses

i e j, o respectivo classificador f é invocado e decide se a hipótese a ser favorecida é i ou j. Caso f indique que i é a vencedora, seu escore de saída $f(t)$ é combinado de forma ponderada com o escore acústico $b_i(t)$ de forma a aumentá-lo. Caso j seja a vencedora, $f(t)$ é usado para aumentar $b_j(t)$. A cada t, vários detectores g podem ser ativados, e várias hipóteses terem seus escores originais aumentados.

A combinação dos escores das HMMs com os dos classificadores f é feita de forma heurística, com o novo escore $b'(t)$ sendo dado por

$$b'(t) = b(t) + \alpha f(t), \tag{5.1}$$

onde α é um número positivo encontrado através de experimentos. Note que, o uso dessa expressão é equivalente a varrer todos os quadros $t = 1, \ldots, T$ e, para cada hipótese, somar o fator $\alpha f(t)$ à respectiva hipótese vencedora, de maneira que a somas acumuladas desses fatores sejam adicionadas aos escores b oriundos das HMMs. É possível utilizar um valor de α distinto para cada classificador f, mas os resultados preliminares de heurísticas com esse objetivo não foram satisfatórios e adota-se o mesmo valor de α para todos os classificadores.

Descrito o funcionamento global do sistema, resta definir como são projetados os detectores de confusão g e os classificadores f.

5.1.2 Margem em *lattices*

Uma *lattice* contém todas as hipóteses geradas durante o processo de decodificação de uma dada sentença. Como se usa *pruning* em ASR, várias hipóteses são descartadas e a *lattice* final pode não incluir a hipótese correta. Formalmente, uma *lattice* pode ser definida como um grafo, $G(I, J)$, onde I é o número de nós e J o número de arcos. A informação de tempo é incorporada ao nó, enquanto que os arcos carregam o símbolo do fone juntamente com o escore associado. A ideia básica por trás das *lattices* é que elas conseguem representar um grande volume de dados de uma forma compacta, proporcionando um maior espaço de busca. O grafo da Figura 5.1 exemplifica a estrutura de uma *lattice* com duas hipóteses. Uma hipótese formada pela palavra "three" e a outra pela palavra "eight". O valor associado a cada arco significa o escore acústico de cada palavra. Os arcos *send-start* e *send-end* representam o silêncio inicial e final da fala, respectivamente.

Através de um treinamento supervisionado (*offline*), é possível estudar as *lattices* e definir **regiões de confusão**. Essas regiões podem ser identificadas por intervalos de tempo onde duas sentenças candidatas não possuem o mesmo escore. Como o treinamento é supervisionado, sabe-se qual a sentença correta e se pode definir o conceito de região de confusão com base na margem, como explicado a seguir.

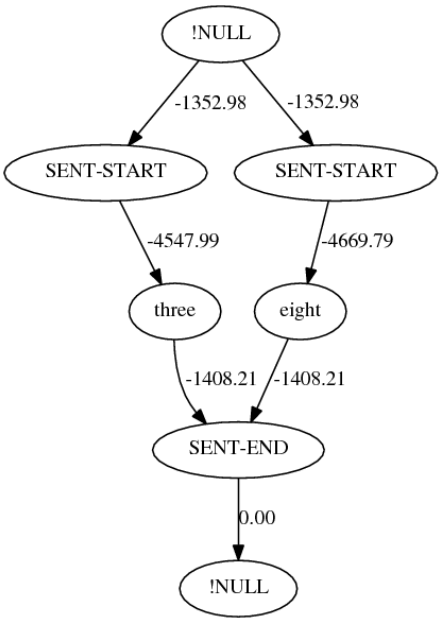


Figura 5.1: Exemplo de uma *lattice*.

Durante a fase de treino, assume-se que a hipótese correta q_* é conhecida e está na *lattice*, ou seja, a *lattice* sempre contém a transcrição correta (o que não ocorre na fase de teste). Ainda no treinamento, as *lattices* são analisadas e suas hipóteses ordenadas em uma lista $\mathcal{L}$ de tal forma que $s_i \geq s_j$, se $i < j$, onde s_i e s_j são os escores totais das hipóteses q_i e q_j, respectivamente. Para fins de explicação, assume-se que a hipótese correta q_* é excluída da lista ordenada $\mathcal{L}$, que armazena assim apenas as hipóteses concorrentes (e erradas). Por exemplo, caso a *lattice* represente 20 hipóteses (incluindo q_*), a lista $\mathcal{L}$ contém 19 hipóteses, com escores s_1 a s_{19}.

A hipótese reconhecida $q_\dagger$ é aquela que corresponde ao maior escore entre s_* e s_1. Quando a correta e reconhecida coincidem, i.e., $s_\dagger = s_*$, todas as palavras na sentença foram corretamente reconhecidas. Um erro de sentença (esse erro é distinto da WER, que contabiliza os erros por palavra) ocorre quando $s_1 > s_*$, ou seja, ao menos uma hipótese possui escore maior do que o da sentença correta s_*.

A margem acústica (ou simplesmente margem) $M_i(t)$ da i-ésima hipótese no t-ésimo quadro é definida neste trabalho como

$$M_i(t) = b_*(t) - b_i(t), \tag{5.2}$$

onde $b_*(t)$ e $b_i(t)$ são os escores acústicos (sem considerar o escore do modelo de linguagem) da correta e i-ésima (ordenada) hipótese para o quadro $\mathbf{x}_t$, respectivamente. Apesar da margem ser definida com base apenas no escore acústico, o escore do modelo de linguagem é utilizado na decodificação e, assim, influencia no processo. Note também que a sequência de estados

q_i informa o modelo HMM e o estado correspondente que devem ser associados ao t-ésimo quadro da hipótese i.

A notação de margem $M_i(t)$ de uma hipótese pode ser estendida para a margem $M(t)$ de uma *lattice* (é assumido que q_* sempre está presente na *lattice*, o que pode ser facilmente reforçado durante o treinamento supervisionado). Portanto, a margem de uma *lattice* para o quadro t pode ser definida baseando-se somente em q_* e q_1:

$$M(t) = b_*(t) - b_1(t), \tag{5.3}$$

com a margem acústica total sendo definida como

$$M = \sum_{t=1}^{T} M(t). \tag{5.4}$$

Como M leva em consideração apenas os escores acústicos, o resultado do reconhecimento expresso pela *lattice* pode ser a transcrição correta, isto é, $s_* > s_1$. Contudo, a margem M também pode ser negativa. Nesse caso, o modelo de linguagem ajuda a guiar o decodificador para encontrar a transcrição correta, enquanto que se fosse usado apenas o escore acústico, se teria um erro de sentença.

Dados esses conceitos básicos, a tarefa passa a ser definir um procedimento para obter os classificadores que fazem a detecção de confusão. Inicialmente, discutem-se possíveis definições.

Várias definições podem ser adotadas para região de confusão. Uma alternativa é usar um limiar (*threshold*) γ do valor da margem. Nesse caso, os Q quadros $f_1, f_2, \ldots, f_Q$, que podem ser considerados como parte de uma região de confusão, correspondem ao conjunto $\mathcal{F} = \{f_1, f_2, \ldots, f_Q\}$, onde cada elemento f_t é o índice de um quadro $\mathbf{x}_t$ que tem margem menor que γ, isto é,

$$M(t) < \gamma, \tag{5.5}$$

sendo essa verificação realizada para todos os quadros $f_t, t = 1, \ldots, T$. Note que, $Q \leq T$.

Por exemplo, a Figura 5.2 mostra a evolução do escore acústico no tempo de duas hipóteses competidoras b_* e b_1 dentro de uma *lattice*, onde duas regiões de confusão podem ser observadas. Considerando que, "eight two" (b_*) corresponde à sentença correta e "five six" (b_1) é uma das sentenças candidatas, podemos dizer que os quadros localizados nas proximidades do 30-ésimo quadro possuem margem positiva. Dependendo do *threshold* γ, alguns quadros do conjunto $\mathcal{F} = \{f_{45}, f_{46}, \ldots, f_{54}\}$ podem compor uma região de confusão, dada a margem negativa dos seus quadros. Em outras palavras, decrementando continuamente o valor de γ irá excluir cada vez mais quadros de $\mathcal{F}$.

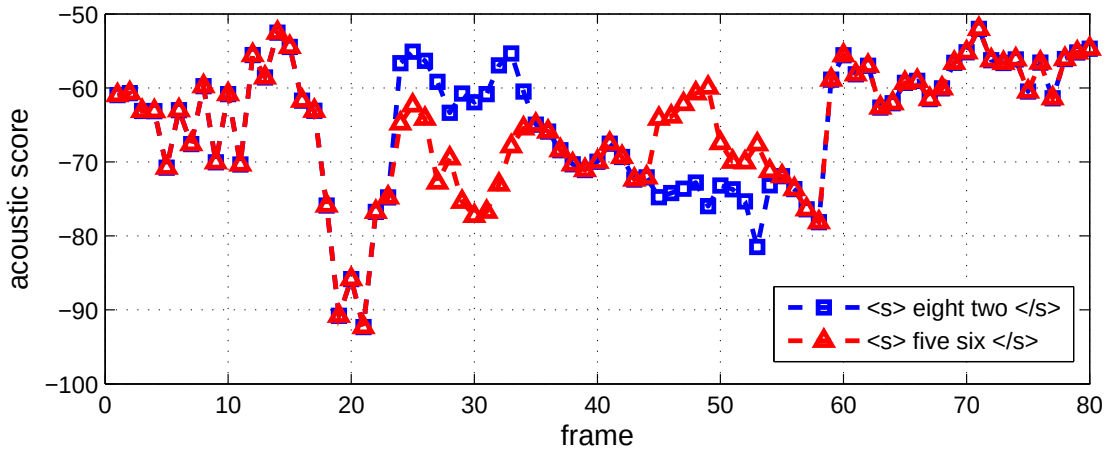


Figura 5.2: Desempenho de duas sentenças candidatas dentro de uma *lattice*.

A definição de regiões de confusão pode ser aperfeiçoada impondo, por exemplo, um número mínimo de quadros contínuos. Então, dada a definição de região de confusão, as *lattices* obtidas a partir do conjunto de treino podem informar quais são os fones e estados que lideram o *rank* de confusão. A próxima seção descreve como essa informação foi utilizada.

5.1.3 Escolhendo pares de estados que suscitam confusão

Após calcular a margem $M(t)$ para cada *lattice* do conjunto de treino, pode-se obter, para cada quadro t, o par de estados de HMMs (um de b_* e outro de b_1) que estão envolvidos nas regiões de confusão identificadas. Um histograma desses pares pode ser calculado, possibilitando a identificação dos pares que apresentam maior número de ocorrências. Note que, essa estratégia é diferente de "estaticamente" comparar modelos HMMs como feito, por exemplo, em [109], dado que o histograma adotado aqui leva em consideração o processo real de decodificação e as *lattices* refletem as hipóteses concorrentes.

Como citado, os escores providos pelo modelo de linguagem são usados quando o decodificador gera a *lattice*. Observa-se que, em função da tarefa a ser usada neste trabalho para testar a proposta ser reconhecimento de dígitos (vide Capítulo 6), não existe modelo de linguagem. Mas para grandes vocabulários, um modelo de linguagem n-gram é essencial para ajudar o decodificador no processo de *pruning* e conjectura-se que o mesmo irá auxiliar a construção de um histograma que efetivamente reflita a confusão que mais impacta o desempenho.

Em suma, durante o treino, as margens das *lattices* foram usadas para se escolher pares de estados distintos que apresentavam margem menor do que γ. Nota-se que, durante o teste,

classificadores g devem indicar que a respectiva confusão está ocorrendo. Em função da tarefa de reconhecimento de dígitos ser relativamente simples e haver um pequeno número de fones ativos na *lattice* para um dado quadro t, optou-se pelo uso de uma heurística simplificada. Se um classificador g deve disparar quando o par de fones (x, y) apresentar confusão, o classificador não é criado e o correspondente disparo ocorre todas as vezes que, em um dado t, ambos x e y estiverem ativos na *lattice*. Em outras palavras, o sistema de pós-processamento varre a *lattice* observando se os pares escolhidos estão ativos ao mesmo tempo e, caso positivo, o respectivo classificador f é acionado. Essa heurística será chamada de *simplificada*.

Nota-se que é possível calcular a taxa de acerto A_g dos classificadores g ao se definir que eles devem disparar quando, para um dado t, um dos estados do par respectivo (por exemplo, x ou y) for o estado da hipótese correta q_*. Por exemplo, um dado classificador g deve disparar quando houver "confusão" entre os pares de estados $v[2]$ e $f[4]$. Se a hipótese correta q_* corresponder a um desses dois estados no instante t, então um erro é contabilizado no cômputo de A_g caso g não dispare. Com essa definição, para cada t, idealmente apenas um classificador g deveria disparar. Contudo, usando-se a heurística simplificada é possível que mais de um g disparem simultaneamente.

A taxa de acerto A_f dos classificadores f é calculada levando-se em conta apenas os casos onde o respectivo classificador g disparou adequadamente, ou seja, um dos dois estados envolvidos pertence à hipótese correta q_*. Dessa forma, pode-se definir que os classificadores binários realizaram uma intervenção correta com uma taxa de A_g A_f. Por exemplo, se em 1000 quadros os classificadores g dispararam 200 vezes, sendo que 80 desses disparos foram indevidos e para os 120 restantes, os classificadores f decidiram 90 vezes acertadamente (pelo estado da hipótese q_*), as taxas são $A_g = 120/200$ e $A_f = 90/120$. Nesse caso, a taxa de acerto global é 90/200, ou seja, para 45% dos quadros nos quais os classificadores binários foram usados, melhorou-se a margem.

5.2 Algoritmos para treinar classificadores

O problema de classificação convencional usando treinamento supervisionado exige que se tenha um *conjunto de treino* $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ contendo N *exemplos*, os quais são amostras independentes e identicamente distribuídas (iid) de uma distribuição $P(\mathbf{x}, y)$ fixa mas desconhecida. Cada exemplo $(\mathbf{x}, y)$ consiste de uma instância $\mathbf{x} \in \mathcal{X}$ e um rótulo (ou classe) $y \in \{1, \ldots, Y\}$. A entrada $\mathbf{x}$ é um vetor de dimensão L. Um *classificador* é um mapeamento $F : \mathcal{X} \to \{1, \ldots, Y\}$. De especial interesse são classificadores binários, para os quais $Y = 2$, e por conveniência matemática, às vezes os rótulos são $y \in \{-1, 1\}$ ou $y \in \{0, 1\}$.

Este trabalho utilizou dois algoritmos para aprendizado de classificadores, os quais são brevemente discutidos a seguir.

5.2.1 Redes neurais artificiais

Neste trabalho foi usada a rede neural artificial (ANN) chamada de perceptron multicamada (*multi-layer perceptron* ou MLP) treinada com o algoritmo *backpropagation* [91, 110]. A principal motivação para usar ANN é que elas tem sido usadas com sucesso em ASR [111–113]. Nota-se que muitas das estratégias sugeridas em [111] para fazer uma ANN substituir HMMs na modelagem acústica podem ser aplicadas também no contexto proposto.

5.2.2 Classificadores baseados em kernels

Dentre os vários classificadores modernos baseados em kernels [114], o mais popular é a máquina de vetores de suporte ou SVM (de *support vector machine*) [98]. Esses classificadores são relacionados ao método de estimação de funções em espaço de Hilbert usando-se regularização (ou RKHS de *reproducing kernel Hilbert space*) [115]. Em essência, o objetivo é achar uma função F que minimiza

$$\frac{1}{N}\sum_{n=1}^{N} L(F(\mathbf{x}_n), y_n) + \lambda ||F||^2_{\mathcal{H}_\mathcal{K}}, \tag{5.6}$$

onde $\mathcal{H}_\mathcal{K}$ é o RKHS gerado pelo kernel $\mathcal{K}$; $F = h + b$; $h \in \mathcal{H}_\mathcal{K}$; $b, \lambda \in \mathbb{R}$; e $L(F(\mathbf{x}_n), y_n)$ é uma função de penalização (*loss function*).

Antes de tentar um classificador com um kernel não-linear, é útil verificar se um kernel linear, definido por

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}', \tag{5.7}$$

permite atingir a precisão desejada. Um kernel linear pode ser convertido para um *perceptron*, o que permite evitar o armazenamento dos vetores de suporte e economizar em termos de custo computacional. Dentre os kernels não-lineares, a RBF (*radial-basis function*) Gaussiana é dos mais competitivos em termos de precisão [116]. O kernel Gaussiano é dado por

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') = e^{-\gamma||\mathbf{x}-\mathbf{x}'||^2}. \tag{5.8}$$

A solução do problema descrito na Equação 5.6 é dada por [117]:

$$F(\mathbf{x}) = \sum_{n=1}^{N} \omega_n \mathcal{K}(\mathbf{x}, \mathbf{x}_n) + b. \tag{5.9}$$

Esta expressão indica que uma SVM e os classificadores relacionados são baseados em exemplos [114]: F é dado em termos dos exemplos de treino $\mathbf{x}_n$. Em outras palavras, assumindo-se um kernel Gaussiano, a média da Gaussiana é restrita a ser uma instância de treino $\mathbf{x}_n$.

Alguns exemplos $\mathbf{x}_n$ podem não ser usados na solução final (nesses casos, o procedimento de treino do classificador atribuiu pesos $\omega_n = 0$). São chamados *vetores de suporte* os exemplos que são efetivamente usados na solução final. Para economizar memória e cálculos no estágio de teste, é conveniente gerar um F esparso, com poucos vetores de suporte. Pode-se obter soluções esparsa apenas se a função de penalização L é zero em algum intervalo [114]. SVM atinge uma solução esparsa usando

$$L(F(\mathbf{x}_n), y_n) = (1 - F(\mathbf{x}_n)y_n)_+, \tag{5.10}$$

onde $(z)_+ = \max\{0, z\}$.

Os classificadores ANN e SVM produzem como saída escores de **confiança**, que podem ser convertidos para probabilidades $p_{(+)}$ e $p_{(-)}$, onde $p_{(+)}$ e $p_{(-)}$ são as probabilidades de uma saída positiva e negativa, respectivamente, e $p_{(+)} + p_{(-)} = 1$. Essa conversão não foi utilizada neste trabalho, mas pode eventualmente simplificar a combinação dos escores provenientes dos classificadores com os das HMMs.

Em função desses classificadores não serem restritos ao arcabouço probabilístico imposto pelos modelos HMMs, seus escores são combinados com os escores das HMMs através de heurísticas, como informado. Nesse sentido, essa estratégia é similar ao peso que é usado para combinar os escores dos modelos acústico e de linguagem. A ausência de uma matemática elegante para combinar os escores das HMMs com os dos classificadores binários pode ser compensada por um grau extra de liberdade: o espaço de entrada dos classificadores pode ser diferente do usado nos modelos HMMs. Por exemplo, a frequência fundamental F0 pode ser usada para distinguir sons vozeados de não-vozeados [56]. A próxima seção discute esse aspecto.

5.3 Seleção de parâmetros heterogêneos

É possível considerar que os modelos HMMs e classificadores binários propostos compõem um sistema que usa um *front-end* heterogêneo, que adiciona algumas vantagens ao modelamento acústico [118]. Ainda considerando o exemplo de F0, pode não ser vantajoso modelar as HMMs incorporando essa *feature* à parametrização MFCC, porém para ajudar a distinguir específicas confusões, F0 pode ser eficaz.

5.3.1 Parâmetros heterogêneos

Os *front-ends* em ASR são tipicamente baseados em MFCC [53] ou PLP (*perceptual linear predictive*) [119], propostos em 1980 e 1990, respectivamente. Contudo, conjectura-se que um único conjunto homogêneo de parâmetros não pode ser o mais apropriado para todas as classes fonéticas. Por exemplo, como discutido em [118], os sons nasais e plosivos apresentam requisitos conflitantes. Os sons nasais requerem maior resolução em frequência enquanto os sons plosivos maior resolução no tempo. Contudo, implementações de MFCC, PLP, etc., usam tipicamente uma janela com aproximadamente 25 ms. Isso indica que seria possível obter conjuntos de parâmetros especificamente sintonizados para determinadas classes fonéticas como em [118].

A integração de parâmetros heterogêneos e modelos probabilísticos (e.g., HMMs) tipicamente requer esforço extra. Por exemplo, o algoritmo Baum-Welch foi modificado para dar suporte a parâmetros heterogêneos em [120]. Essa complexidade extra pode ser a razão para a baixa popularidade de parâmetros heterogêneos quando comparados ao uso, por exemplo, de métodos *multi-stream* [121].

Nota-se que um *front-end* típico faz seu processamento em quadros, convertendo segmentos t de voz digitalizada em vetores de parâmetros $\mathbf{x} = (x_1, \ldots, x_L)$, onde $x_i \in \mathcal{X}_i$ e $\mathbf{x} \in \mathcal{X}^L = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_L$.

Denota-se o conjunto de parâmetros $\mathcal{X}^L$ como *homogêneo*, com respeito a um determinado sistema de classificadores, se os L parâmetros compõem o conjunto de treino de todos os classificadores usados pelo sistema. O conjunto $\mathcal{X}^L$ é chamado *heterogêneo* com respeito ao sistema de classificadores se há no mínimo dois subconjuntos distintos de $\mathcal{X}^L$ que são usados para treinar classificadores. De forma similar, quando o modelo acústico é baseado em HMMs, $\mathcal{X}^L$ é heterogêneo com respeito a esse sistema se subconjuntos distintos de $\mathcal{X}^L$ são usados como espaço de entrada das distribuições de saída das HMMs (e.g., quando os parâmetros dependem da classe fonética do modelo HMM). Note que é feita clara distinção entre o sistema *multi-stream* [121] e parâmetros heterogêneos. Por exemplo, mesmo com os parâmetros sendo obtidos de diferentes *front-ends* em [122], o sistema é considerado homogêneo porque todos os parâmetros são usados pelo modelo acústico, independente da classe fonética.

Uma pergunta pertinente é o motivo de não se usar sempre todos os L parâmetros. Por exemplo, os próprios classificadores poderiam identificar os parâmetros irrelevantes e desenfatizar suas influências. Na prática, além de reduzir o custo computacional, a redução do número de parâmetros pode levar a uma melhor capacidade de generalização do classificador quando o conjunto de treino é limitado [123]. Dessa forma, o objetivo é reduzir o número de parâmetros usados por cada classificador enquanto se mantém informação suficiente para uma

boa precisão.

Os métodos para redução de dimensionalidade de $\mathcal{X}^L$ podem ser organizados em dois grupos: *seleção* e *extração* de parâmetros [124]. A seleção de parâmetros tenta identificar e reter apenas aqueles parâmetros que contribuem mais significativamente para a tarefa em questão. Esses métodos não modificam os parâmetros mas apenas escolhem um subconjunto entre todos os 2^L subconjuntos possíveis. Os métodos de extração de parâmetros transformam o espaço original $\mathcal{X}^L$ em um espaço com dimensão menor, modificando assim os parâmetros originais. Um exemplo é a análise de componentes principais, ou PCA (de *principal component analysis*) [124], seguida do descarte de componentes não importantes. O presente trabalho adotou a seleção de parâmetros, como discutido a seguir.

5.3.2 Seleção de parâmetros

Métodos para seleção automática de parâmetros podem ser organizados em dois grupos: *filtros* e *wrappers* [125]. O último consiste de métodos que selecionam os parâmetros baseados no próprio algoritmo de aprendizado do classificador $\mathcal{A}$, o qual será usado posteriormente para treinar o classificador (por exemplo, ANN ou SVM). Considerando-se que se tem um total de L parâmetros, na maior parte das vezes é inviável permitir que o *wrapper* explore todas os 2^L possíveis subconjuntos de $\mathcal{X}^L$ (i.e., treinar 2^L SVMs ou ANNs). Portanto, o uso de *wrappers* geralmente é acompanhado da adoção de uma heurística de busca subótima [125].

Métodos do tipo filtro avaliam o valor de um parâmetro x_i usando heurísticas como, e.g., a correlação de x_i com o rótulo y. Portanto, filtros usualmente requerem um custo computacional menor do que *wrappers* e são independentes de $\mathcal{A}$ (i.e., um filtro aplicado a uma ANN não requer o treino de ANNs no estágio de treino). Por outro lado, métodos *wrapper* podem alcançar menor taxa de erro do que filtros [125]. Neste trabalho foram usados apenas filtros.

O método do tipo filtro usado foi o baseado no ganho de informação (*information gain*), também chamado de informação mútua, o qual é um dos mais simples porém populares métodos para seleção de parâmetros [125]. Seu funcionamento é brevemente descrito a seguir.

Uma distribuição $p(y)$ dos rótulos é estimada usando-se o conjunto de treino $\mathcal{T}$, e a variável aleatória Y associada tem entropia

$$H(Y) = -\sum_{y \in \mathcal{Y}} p(y) \log_2 p(y). \tag{5.11}$$

Assume-se que os parâmetros x_i, $i \in \{1, \ldots, L\}$, eram originalmente discretos (não contínuos) ou foram previamente discretizados (neste trabalho foi utilizado o método proposto em [126]).

Após observar x_i, o conjunto de treino pode ser particionado de acordo com seu valor, e distribuições $p(x_i)$ e $p(y|x_i)$ estimadas através da contagem de ocorrências em $\mathcal{T}$. A entropia de Y condicionada na observação da variável aleatória X_i é

$$H(Y|X_i) = -\sum_{x_i \in \mathcal{X}} p(x_i) \sum_{y \in \mathcal{Y}} p(y|x_i) \log_2 p(y|x_i), \tag{5.12}$$

e o ganho de informação $I(X_i; Y)$ é dado por

$$I(X_i; Y) = H(Y) - H(Y|X_i). \tag{5.13}$$

Assim, o método consiste em calcular $I(X_i; Y)$, para $i = 1, \ldots, L$, e daí selecionar os S parâmetros com maior $I(X_i; Y)$.

Tendo apresentado o método proposto, o próximo capítulo aborda os principais resultados obtidos com HMMs e classificadores binários. Serão discutidos experimentos concentrados no reconhecimento de uma sequência de dígitos.

Capítulo 6

Resultados experimentais com HMMs e classificadores

Este capítulo apresenta os resultados para o sistema proposto, baseado em re-escore de *lattices* com classificadores binários. No presente trabalho os experimentos concentraram-se no reconhecimento de uma sequência de dígitos (0 a 9, e "oh") e utilizam para treino e teste a conhecida base de dados TIDIGITS [127]. A Tabela 6.1 apresenta os fones adotados.

Tabela 6.1: Transcrição fonética dos dígitos.

Dígito	Transcrição fonética
one	w ah1 n
two	t uw1
three	th r iy1
four	f ao1 r
five	f ay1 v
six	s ih1 k
seven	s eh1 v ax n
eight	ey1 t
nine	n ay1 n
zero	z iy1 r ow2
oh	ow1

Todas as HMMs possuem três estados emissores que, por consistência com a notação do HTK, são numerados de 2 a 4. Por exemplo, o primeiro estado de $uw1$ é $uw1[2]$ e o último é $uw1[4]$.

6.1 Construção do sistema baseado em HMMs

O *software* HTK foi usado para treinar um sistema ASR para reconhecimento de dígitos baseado em HMMs (*baseline*). Para isso, os arquivos de áudio da base de dados TIDIGITS foram re-amostrados de 20 kHz para 16 kHz (mono, 16 bits). O dicionário fonético adotado foi o mesmo fornecido no pacote TIDIGITS e as transcrições fonéticas podem ser conferidas na Tabela 6.1.

O *front-end* consiste de 12 coeficientes MFCCs, usando C0 como componente de energia e computado a cada 10 ms para um quadro de 20 ms. Em seguida, foram extraídas a primeira e segunda derivadas, compondo um vetor com 39 parâmetros para cada quadro. Finalmente, a técnica *cepstral mean subtraction* foi usada para normalizar os coeficientes MFCCs.

O modelo acústico foi refinado de forma iterativa. A abordagem *flat-start* foi adotada com modelos baseados em monofones e com uma Gaussiana por mistura. Foram utilizados 22 fones (21 monofones e um modelo de silêncio *sp*) com HMMs compostas por 3 estados na estrutura *left-to-right*. O modelo *sp*, com apenas um estado emissor, foi construído através da cópia do estado central do modelo de silêncio.

Depois que os modelos monofones foram criados, a ferramenta HVite (parte do HTK) [66] foi usada para executar um alinhamento forçado da base de treino. Em todo o processo de treino, o algoritmo de Baum-Welch [84] foi usado para re-estimar os modelos.

Na fase de teste (decodificação), a ferramenta HVite foi usada para gerar uma *lattice* contendo múltiplas hipóteses para cada sentença de teste. A gramática usada para guiar o HVite foi criada com a ferramenta HBuild (parte do HTK) [66], sendo composta por *loops* simples, onde cada palavra (dígito) pode ser seguida por qualquer outra. Assim, o sistema não assume conhecimento acerca do número de dígitos em cada sentença e pode haver erros de inserção e deleção.

Como dito, o HVite permite a criação de uma *lattice* para cada sentença de teste. Todavia, a ferramenta HVite original disponibiliza apenas alinhamento no nível de palavra. No intuito de executar o re-escore das *lattices* no nível de fone, foi necessário modificar o código do HVite para gerar alinhamento no nível de fone para cada arco da *lattice*.

6.2 Resultados usando escores como parâmetros

Para se chegar à arquitetura proposta na Seção 5.1, foram feitos diversos testes. Para exemplificar, discute-se na atual seção um estudo preliminar que avaliou a utilidade de se usar

os próprios escores acústicos das HMMs como parâmetros de entrada para os classificadores. Ou seja, nos experimentos descritos a seguir não se usou seleção de parâmetros, mas sim os escores das HMMs como parâmetros de entrada para os classificadores. Um objetivo foi avaliar se, mesmo sem informação proveniente de outros *front-ends*, seria possível melhorar o resultado global usando-se classificadores. Outro ponto estudado foi a flexibilização do projeto do classificador f, que nesta pesquisa, inicialmente, não se restringe a distinguir apenas um par de estados.

O pacote de *software* WEKA [128] foi adotado para treinar os classificadores binários. O algoritmo escolhido para esses primeiros experimentos foi uma rede neural multi-camada, implementado no WEKA com a configuração padrão da classe *MultilayerPerceptron*.

Como primeiro experimento, um único classificador binário f foi implementado para analisar confusão especificamente entre os fones que compõem a classe fonética das vogais anteriores: $\{uw1, ah1, ao1, ow1\}$. Mais especificamente, estudou-se como alterar os escores acústicos de estados do fone $uw1$. Para isso, o conjunto de treino do classificador binário projetado sempre envolve um estado desse fone. Por outro lado, relaxou-se a restrição do classificador binário ser projetado para um par de estados, permitindo-se que a classe positiva ou negativa seja formada por diversas situações, como indicado na Tabela 6.4.

O sistema *baseline* baseado em HMMs foi treinado com 12.549 sentenças extraídas aleatoriamente do corpus TIDIGITS. Já o conjunto de teste foi composto pelas demais 12.547 sentenças do corpus, que não foram vistas na etapa de treino do modelo acústico.

Os quadros usados para treinar o classificador foram extraídos de 10.591 sentenças corretamente reconhecidas pelo sistema *baseline* com a ferramenta HVite, ou seja, situações onde as hipóteses reconhecida $q_\dagger$ e correta q_* são iguais. Por conveniência, a hipótese reconhecida (e também correta) $q_* = q_\dagger$ e as 14 melhores sentenças candidatas $q_1, \ldots, q_{14}$ foram extraídas de cada *lattice*. Em seguida, a hipótese $q_* = q_\dagger$ foi comparada *frame-a-frame* com as demais 14 hipóteses na busca de regiões de confusão. Então, o conjunto de treino da rede neural (isto é, do classificador) foi composto de acordo com a Tabela 6.2.

Por exemplo, as *lattices* geradas pelo sistema *baseline* forneceram 12 quadros onde o estado $uw1[2]$ se confunde com o estado $ow1[2]$ (última linha da tabela). Esses quadros foram incorporados no conjunto de treino com um rótulo "+", indicando que na fase de teste, se o classificador se decidir pela classe "+", os escores de todos os estados de $uw1$ ativos no instante t devem ser aumentados de acordo com a Equação 5.1 assumindo-se que $\alpha = 8$. De outra forma (penúltima linha), tem-se 14 quadros onde o estado $ow1[2]$ se confunde com o estado $uw1[2]$ na situação onde $ow1[2]$ é o estado correto, e daí o rótulo é "-". Assim, quando o classificador se decidir por "-" na fase de teste, os escores de todos os estados de $uw1$ ativos

no instante t não são modificados. O papel do classificador g é realizado por uma heurística simplificada, onde f é disparado se aparecerem simultaneamente ativos na *lattice* qualquer dos pares de estados indicados na Tabela 6.2.

Tabela 6.2: Quantidade de exemplos (quadros) usados no conjunto de treino da rede neural, onde $\Phi = \{ah1[2], ah1[3], ah1[4], ow1[3], ow1[4]\}$.

Fone/estado da hipótese correta	Fone/estado da hipótese candidata	Classe	Quantidade de quadros
Φ	$uw1[4]$	-	309
$uw1[4]$	Φ	+	522
$ow1[2], ow1[3]$	$uw1[3]$	-	216
$uw1[3]$	$ow1[2], ow1[3]$	+	682
$ow1[2]$	$uw1[2]$	-	14
$uw1[2]$	$ow1[2]$	+	12

Com o classificador binário treinado, foi possível avaliar seu desempenho no processo de re-escore das *lattices*. O conjunto de teste foi composto por 30 sentenças erroneamente reconhecidas e 15 sentenças corretamente reconhecidas pelo sistema *baseline*, sendo que nenhuma delas foi usada no treinamento da rede neural. É importante observar que na fase de teste não é garantido que a hipótese correta esteja presente em todas as *lattices*. Assim, as 45 *lattices* foram analisadas separadamente, comparando suas 15 melhores hipóteses entre si.

Os primeiros resultados obtidos no processo de re-escore das *lattices* são apresentados na Tabela 6.3. Eles indicam que, a partir da informação acústica obtida a cada quadro (*frame-a-frame*), uma melhor separação entre fones competidores foi alcançada e, com isso, a taxa de acerto por sentença foi incrementada.

Tabela 6.3: Desempenho do sistema com um classificador binário.

	Sentenças corretas	Sentenças erradas
Antes do re-escore	15	30
Depois do re-escore	41	4

Em um segundo experimento usando os escores das HMMs como parâmetros de entrada dos classificadores, avaliou-se o sistema implementando-se vinte (20) classificadores binários. Os classificadores foram implementados, novamente, para analisar confusão especificamente entre os fones que compõem a classe fonética das vogais anteriores. Porém, ao contrário do experimento anterior, o conjunto de treino dos classificadores envolveu todas as combinações

possíveis entre os fones da referida classe fonética. Mais uma vez o projeto dos classificadores não ficou restrito a um par de estados.

Para que a comparação com o sistema *baseline* seja justa, foram definidos conjuntos de treino e teste como se segue. Dois modelos acústicos foram construídos com base no corpus TIDIGITS. O primeiro modelo acústico (M1) foi treinado com 17.567 sentenças (70% da base) e testado com 7.529 sentenças (30% da base). O modelo M1 é o sistema *baseline* propriamente dito e seu conjunto de teste é usado para avaliar também o sistema baseado em classificadores. Mas para projeto dos classificadores em si, é necessário conjuntos de treino e teste e os mesmos são obtidos particionando-se o conjunto de treino de M1. Por exemplo, um segundo modelo acústico (M2) foi treinado com 12.296 sentenças (70% do conjunto de treino de M1) e testado com 5.271 sentenças (30% do conjunto de treino de M1).

Os quadros usados para treinar os 20 classificadores binários foram extraídos de 4.991 sentenças do conjunto de teste de M2. Não foram usadas todas as sentenças desse conjunto, pois 280 delas não apresentaram a hipótese correta na sua respectiva *lattice*. Por conveniência, a hipótese correta q_* e as 14 melhores sentenças candidatas $q_1, \ldots, q_{14}$ foram extraídas de cada *lattice*. Então, todos os exemplos de confusão entre fones coletados foram submetidos à classe *MultilayerPerceptron* do WEKA em sua configuração padrão.

Com os classificadores binários treinados, foi possível avaliá-los no processo de re-escore das *lattices*. O conjunto de teste foi composto pelas 7.529 sentenças do conjunto de teste de M1. Primeiramente, cada *lattice* foi analisada separadamente, comparando suas 15 melhores hipóteses entre si. Para cada quadro t, com confusão entre um dos fones alvo e qualquer outro fone, o classificador binário associado foi invocado. Então, dependendo do parâmetro de saída, o escore do fone alvo era incrementado, ou não, de acordo com a Equação 5.1.

Esse método mostrou-se novamente capaz de melhorar a taxa de erro por sentença (SER) do sistema *baseline* através do incremento dos escores acústicos de fones em situação de confusão. Por exemplo, adotando-se $\alpha = 1$ obteve-se um ganho de 3%, conforme os valores apresentados na Tabela 6.4.

Tabela 6.4: Desempenho do sistema com vários classificadores binários.

	Sentenças corretas	Sentenças erradas	SER (%)
Antes do re-escore	6.323	1.206	16,01
Depois do re-escore	6.543	986	13,09

A Figura 6.1 mostra a taxa de erro por sentença a partir da variação da constante empírica α. O melhor resultado nesse caso, onde os parâmetros são os próprios escores das HMMs, foi obtido com $\alpha = 1,7$.

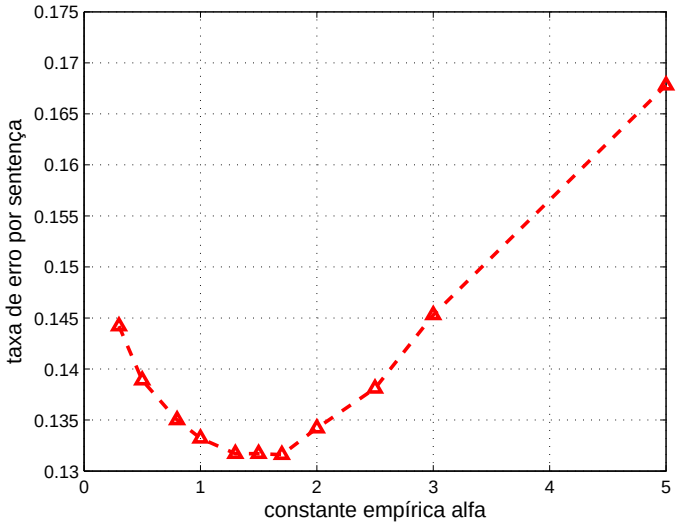


Figura 6.1: Taxa de erro por sentença em função da constante empírica α.

Para verificar o desempenho do sistema perante desbalanceamento, ou seja, quando os classificadores binários são treinados com bem menos exemplos de uma classe do que outra (no caso, menos exemplos da positiva do que negativa), alguns exemplos foram retirados do conjunto de treino. Com isso, foram usados 586 exemplos de classe positiva e 7.312 exemplos de classe negativa para treinar os classificadores. Ainda assim obteve-se ganho após o re-escore, porém com uma taxa de erro por sentença de 14,39% (adotando-se $\alpha = 1$).

Como resultado desses experimentos, concluiu-se que é possível permitir que as classes de f sejam flexíveis como indicado na Tabela 6.2, mas isso torna o projeto do sistema mais complexo e daí se optou por restringir os classificadores a um par de estados. Outra lição é o uso dos escores acústicos, que permite obter melhorias mas não tão significativas quanto às obtidas usando a estratégia de seleção automática de parâmetros. Além do mais, para LVCSR, quando o número de estados é relativamente grande, fica inviável usar os escores de todos os estados das HMMs, e se deveria implementar alguma estratégia de redução desse número.

6.3 Resultados usando parâmetros heterogêneos

Os resultados dessa seção foram obtidos com sistemas que utilizam seleção automática de parâmetros, onde cada classificador f utiliza parâmetros específicos ao respectivo problema.

Os 39 parâmetros do *front-end* MFCC foram complementados como segue. Usando-se a ferramenta HCopy do HTK [66], foram calculados 12 coeficientes PLP e suas primeira e segunda derivadas, o que corresponde a 36 novos parâmetros. Além disso, ainda usando-se o

HCopy, obteve-se a potência de saída de cada filtro do banco de filtros usado para calcular os MFCC. Foram 24 filtros organizados de acordo com a escala mel, dos quais se obteve a potência de saída e suas derivadas, correspondendo a 48 novos parâmetros chamados FBANK. A utilização da saída dos filtros, antes do cálculo da transforma de cosenos (DCT) usada no MFCC, favorece manter a integridade da informação das altas frequências, a qual se conjectura ser útil para modelar / identificar sons fricativos, por exemplo.

Por fim, usou-se um *front-end* chamado "formantes", o qual consiste da probabilidade de vozeamento (ou seja, a probabilidade estimada de ser um som sonoro ou não), frequência fundamental F0 (também chamada de *pitch*, apesar de não serem a mesma coisa) e das primeiras quatro frequências formantes (F1-F4), em um total de 6 parâmetros. Os parâmetros do "formantes" foram obtidos usando-se algoritmos desenvolvidos por David Talkin e outros [129, 130]. Os mesmos faziam parte do popular pacote Waves+, o qual não é mais comercializado. Contudo, eles foram recentemente incorporados ao pacote código-livre Snack.[1] Somando-se os parâmetros MFCC, PLP, FBANK e formantes, tem-se um total de 129 parâmetros a cada quadro.

Usou-se uma janela de 7 quadros para gerar os parâmetros finais. Para um dado quadro $\mathbf{x}_t$, reuniu-se $\mathbf{x}_{t-3}, \mathbf{x}_{t-2}, \ldots, \mathbf{x}_{t+3}$ e foram calculadas as médias e desvios padrões de cada parâmetro usando-se os correspondentes 7 valores nesse intervalo. Além disso, estimou-se a melhor reta no sentido dos mínimos quadrados (*least squares fitting*) que represente os 7 valores de cada parâmetro, e armazenou-se o coeficiente angular dessa reta. Os valores da média, desvio padrão e coeficiente angular são denotados por m, d e c, respectivamente. Assim, o conjunto de parâmetros $\mathcal{X}^L$ do qual cada classificador selecionará um subconjunto, apresenta $3 \times 129 = 387$ parâmetros por quadro.

Dados são requeridos para treinar dois tipos de classificadores, os do tipo g e f. Para isso, é necessário determinar quais os pares de estados que suscitam confusão, definindo dessa forma os classificadores g. Para essa tarefa utilizaremos *lattices* onde ocorreram erro de reconhecimento ($q_* \neq q_\dagger$). Uma vez definidos os classificadores g, é necessário associar para cada um deles um classificador f, cujo conjunto de treino será extraído de todas as *lattices* do conjunto de teste M2. O processo é detalhado nos próximos parágrafos.

Para determinar os pares de estados que suscitam confusão, as *lattices* onde $q_* \neq q_\dagger$ foram analisadas na busca de regiões de confusão $\mathcal{F}_e$. Para cada região, o quadro t' de menor margem foi identificado. A partir do quadro t' é possível selecionar os estados x e y presentes nas hipóteses q_* e $q_\dagger$, respectivamente. Os pares de estados foram selecionados com base em suas margens e são apresentados na Tabela 6.5.

[1] O pacote Snack pode ser obtido em `www.speech.kth.se/snack`. Utilizou-se a versão 2.2.10.

Número	Par	Número	Par	Número	Par	Número	Par
1	z[4] t[4]	61	ay1[2] ow1[3]	121	ey1[3] ax[3]	181	uw1[2] eh1[3]
2	ah1[3] ay1[4]	62	ay1[3] ah1[2]	122	ey1[3] iy1[2]	182	uw1[2] iy1[2]
3	ah1[3] ey1[2]	63	ey1[4] uw1[4]	123	ey1[3] n[4]	183	uw1[2] n[2]
4	ah1[3] uw1[4]	64	f[3] sil[4]	124	ey1[3] r[2]	184	uw1[2] z[3]
5	ah1[4] ay1[4]	65	iy1[3] ey1[3]	125	ah1[3] r[2]	185	uw1[3] ow1[4]
6	ah1[4] ow1[4]	66	ay1[4] ow1[3]	126	ey1[4] v[2]	186	v[2] ah1[3]
7	ao1[4] ay1[3]	67	n[4] uw1[4]	127	f[2] ow1[2]	187	v[2] ay1[2]
8	ao1[4] ow1[4]	68	ow1[2] ow1[4]	128	f[2] s[2]	188	v[2] uw1[3]
9	ay1[2] ey1[2]	69	ow1[3] ow2[2]	129	f[2] th[2]	189	v[4] n[4]
10	ey1[2] t[4]	70	ow1[3] v[2]	130	f[3] th[3]	190	w[2] n[4]
11	ay1[2] ao1[2]	71	ow1[3] w[2]	131	ih1[2] k[4]	191	ax[4] n[4]
12	ey1[3] uw1[3]	72	ow1[4] n[4]	132	ih1[3] s[2]	192	w[2] ow1[2]
13	ey1[4] iy1[4]	73	r[4] ey1[3]	133	ih1[3] sp[2]	193	w[2] sp[2]
14	ey1[4] r[4]	74	s[2] t[4]	134	iy1[3] ax[4]	194	w[3] ao1[3]
15	f[2] t[4]	75	uw1[2] ey1[2]	135	k[3] sil[4]	195	w[3] n[4]
16	f[3] sp[2]	76	v[3] ay1[4]	136	ah1[4] v[3]	196	w[3] ow1[4]
17	f[3] w[2]	77	ay1[4] r[4]	137	n[2] ay1[2]	197	w[4] ao1[4]
18	n[2] ow1[4]	78	v[3] ow1[4]	138	n[4] f[2]	198	w[4] ow1[4]
19	ow1[2] n[2]	79	ah1[3] ay1[3]	139	n[4] ow1[2]	199	z[2] k[3]
20	ow1[2] sil[4]	80	iy1[4] t[2]	140	n[4] t[4]	200	z[2] sil[4]
21	ow1[2] sp[2]	81	n[2] n[4]	141	n[4] w[4]	201	z[2] t[2]
22	ay1[2] ey1[3]	82	n[4] sp[2]	142	ow1[2] ey1[2]	202	ay1[2] ah1[2]
23	ow1[2] t[3]	**83**	**t[2] n[4]**	143	ow1[3] ah1[4]		
24	ow1[3] ao1[4]	**84**	**uw1[2] r[2]**	144	ow1[3] ey1[2]		
25	ow1[3] n[3]	**85**	**v[3] n[2]**	145	ow1[3] ih1[2]		
26	ow1[4] iy1[4]	**86**	**ay1[3] ow1[3]**	146	ow1[3] iy1[3]		
27	ow2[4] ow1[4]	**87**	**f[4] ow1[2]**	147	ao1[2] th[4]		
28	r[3] ey1[2]	88	eh1[3] ey1[2]	148	ow1[3] ow2[4]		
29	r[3] uw1[3]	89	ow1[2] ow1[3]	149	ow1[3] sp[2]		
30	r[3] v[2]	**90**	**ow1[4] r[3]**	150	ow1[4] ay1[4]		
31	sil[2] ih1[4]	**91**	**ay1[2] ow1[2]**	151	ow1[4] w[2]		
32	t[2] s[2]	92	sp[2] t[4]	152	ow2[3] ow1[4]		
33	ay1[2] w[4]	**93**	**t[2] ey1[2]**	153	ow2[3] r[4]		
34	t[2] sil[4]	94	w[2] sil[4]	154	r[2] iy1[3]		
35	t[2] uw1[4]	**95**	**f[3] n[2]**	155	r[2] iy1[4]		
36	t[3] sil[3]	**96**	**ow1[3] uw1[4]**	156	r[2] r[3]		
37	uw1[2] th[4]	**97**	**ow1[3] w[3]**	157	r[3] ey1[3]		
38	uw1[4] z[2]	**98**	**r[4] ow1[4]**	158	ao1[3] ay1[2]		
39	v[3] ow1[3]	99	eh1[3] ih1[2]	159	r[3] ow1[3]		
40	w[3] ow1[2]	**100**	**ey1[3] ih1[2]**	160	r[3] ow2[2]		
41	z[2] t[3]	101	sp[2] t[3]	161	r[4] ao1[3]		
42	z[2] t[4]	102	t[4] sil[4]	162	r[4] ao1[4]		
43	ah1[3] ow1[4]	**103**	**w[2] n[2]**	163	r[4] v[3]		
44	ay1[3] r[2]	104	ow1[4] uw1[4]	164	s[2] sil[3]		
45	ey1[4] ow1[4]	**105**	**ao1[2] ow1[2]**	165	s[3] sil[3]		
46	iy1[3] ey1[2]	**106**	**ow1[3] n[2]**	166	s[3] t[3]		
47	ow1[2] uw1[2]	**107**	**v[4] ow1[4]**	167	s[3] t[4]		
48	ow1[3] n[4]	**108**	**ey1[3] ow1[3]**	168	s[4] iy1[4]		
49	ow1[3] r[2]	109	ow1[3] ow1[4]	169	ao1[3] ow1[4]		
50	ow1[4] t[2]	110	eh1[3] ow1[2]	170	sil[2] t[4]		
51	r[4] r[3]	111	ah1[3] ey1[3]	171	sil[4] sp[2]		
52	t[2] ow1[2]	112	sp[2] t[2]	172	sp[2] ow1[4]		
53	t[2] t[3]	**113**	**n[2] t[4]**	173	sp[2] ow2[3]		
54	t[2] t[4]	**114**	**uw1[3] ow1[3]**	174	t[2] th[2]		
55	ay1[3] v[2]	115	eh1[3] s[4]	175	t[3] k[3]		
56	t[3] sil[4]	116	eh1[4] f[4]	176	t[3] t[4]		
57	uw1[3] ih1[2]	117	eh1[4] r[2]	177	t[4] z[3]		
58	uw1[3] iy1[4]	118	ey1[2] iy1[2]	178	th[3] z[3]		
59	uw1[3] r[4]	119	ey1[2] ow1[4]	179	th[4] iy1[3]		
60	ao1[3] ow1[3]	120	ey1[2] sil[3]	180	ao1[3] r[2]		

Tabela 6.5: Identificação (número) do classificador e o respectivo par de estados. O conjunto $\mathcal{C}^{20}$ dos 20 pares de maior frequência é destacado em negrito.

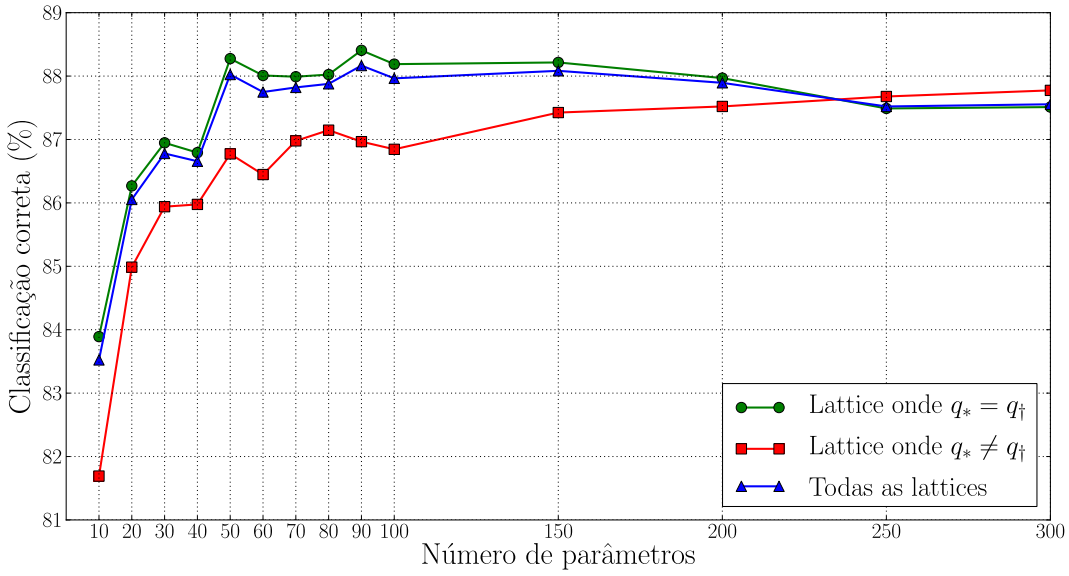
Um classificador f foi criado para cada par de estados (x, y) e o conjunto de classificadores é denotado aqui por $\mathcal{C}$. Para realizar o papel do classificador g, foi adotada a heurística simplificada descrita na Seção 5.1.3. Nota-se que, seguindo essa heurística, a taxa de acertos foi de $A_g = 84,5\%$. Esse valor é considerado relativamente alto, e motivou o foco das pesquisas nos classificadores f em detrimento dos classificadores g (que foram substituídos pela heurística). A explicação para A_g alto nesse caso é que as *lattices* possuem um número pequeno de estados ativos a cada instante t, beneficiando a heurística. Em futuros experimentos com LVCSR, cogita-se que a heurística não funcionará bem e será necessário investir esforços para o correto projeto dos classificadores g.

Para o treino do classificador f foram usadas as regiões $\mathcal{F}_e$ e $\mathcal{F}_i$, sendo as regiões $\mathcal{F}_i$ calculadas apenas para *lattices* corretamente reconhecidas ($q_* = q_\dagger$) e definidas por intervalos de tempo onde a hipótese q_i não possui o mesmo escore que a hipótese correta (q_*). Dessa forma, os quadros t_+ e t_- foram selecionados em cada região $\mathcal{F}_i$, sendo o primeiro o quadro de maior margem da região e o segundo o de menor. Para cada quadro t_+ e t_-, foram coletados os estados z e w presentes nas hipóteses q_* e q_i, respectivamente.

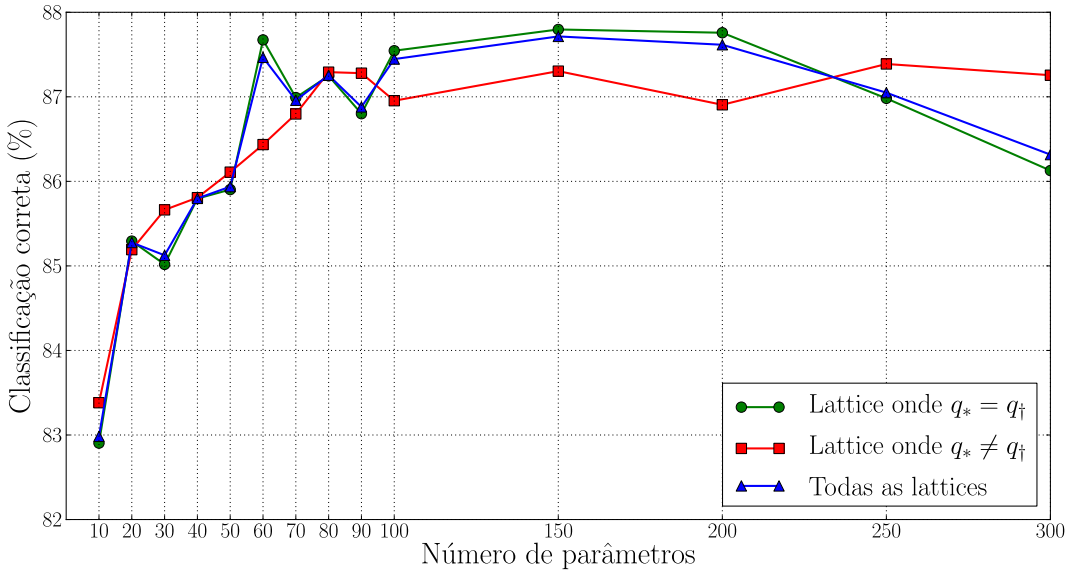
Uma vez definidos os quadros e estados que suscitam confusão, podemos montar o conjunto de treino para os classificadores f. Para isso, os pares (z, w) serão filtrados e somente serão utilizados os presentes na Tabela 6.5, estes serão instâncias de treino para o classificador f, onde a classe será z (visto que esse é o estado presente na hipótese correta). O restante do conjunto de treino será formado pelos pares (x, y), sendo a classe o estado x por novamente este estar na hipótese correta.

Para cada classificador f, a tarefa é encontrar de forma automática, usando-se o filtro baseado no ganho de informação, os S parâmetros que mais influenciam na distinção do par de estados (x, y). Nota-se que poderia ser usado um valor de S diferente para cada classificador, porém isso tornaria o projeto mais complicado. Também para diminuir o tempo de processamento, usou-se inicialmente apenas os 20 pares (x, y) mais frequentes da Tabela 6.5, os quais formam o conjunto denotado como $\mathcal{C}^{20}$.

A Figura 6.2 apresenta os resultados encontrados usando-se os classificadores ANN e SVM, variando-se o número S de parâmetros selecionados para cada classificador binário. Essas figuras distinguem os casos onde a hipótese reconhecida $q_\dagger$ é igual à correta q_* ou não. O conjunto de teste foi composto pelas 7.529 sentenças do conjunto de teste de M1. Na Figura 6.2(a) pode-se observar que, ao se usar classificadores ANN, um razoável valor para o número S de parâmetros é 50. Já quando se usa classificadores SVM, pode-se observar na Figura 6.2(b), que um valor razoável para S é 150.



(a) ANN.



(b) SVM.

Figura 6.2: Taxa de acerto dos classificadores $\mathcal{C}^{20}$ por número de parâmetros selecionados.

Deve-se atentar que tanto a ANN quanto SVM possuem diversos parâmetros que precisam ser sintonizados para a aplicação em questão. Para realizar essa tarefa, conhecida como seleção do modelo (*model selection*), foram escolhidos alguns classificadores específicos e feita a seleção de forma automática, usando-se um conjunto de validação. Assim, encontrou-se para a SVM o modelo de kernel linear, com fator de complexidade $C = 1$ e os demais parâmetros

sendo os padrões do WEKA. Para a rede neural perceptron multi-camada, encontrou-se que o número de neurônios na camada oculta foi $H = 25$ e os outros parâmetros do modelo podem ser os padrões do WEKA. Ilustra-se a necessidade de uma etapa de seleção de modelo através da sintonia do fator de complexidade C da SVM. Assumindo-se $S = 50$, a Figura 6.3 mostra como o desempenho da SVM varia com C para os 20 classificadores de $\mathcal{C}^{20}$.

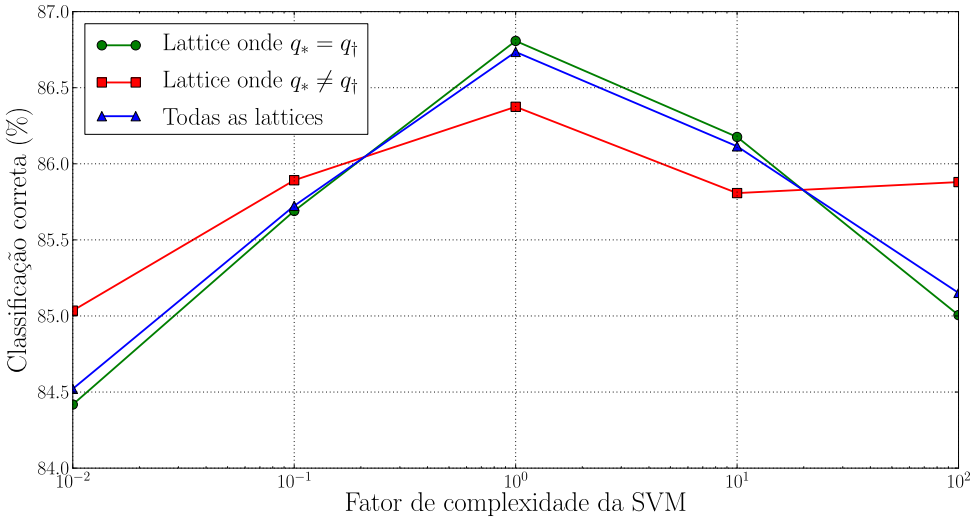
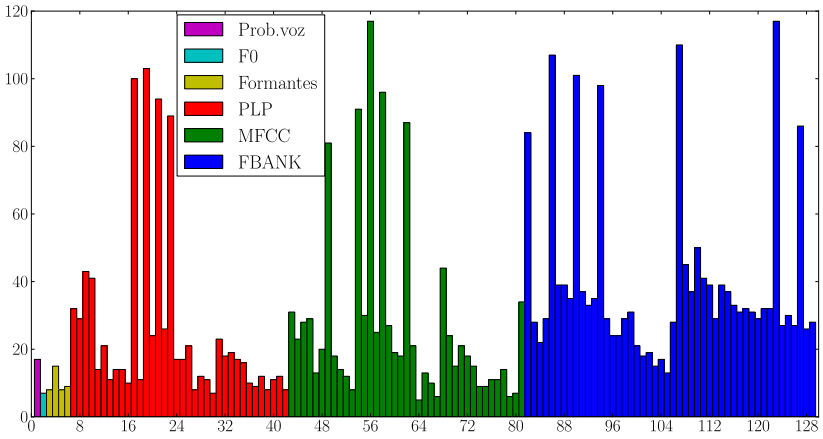


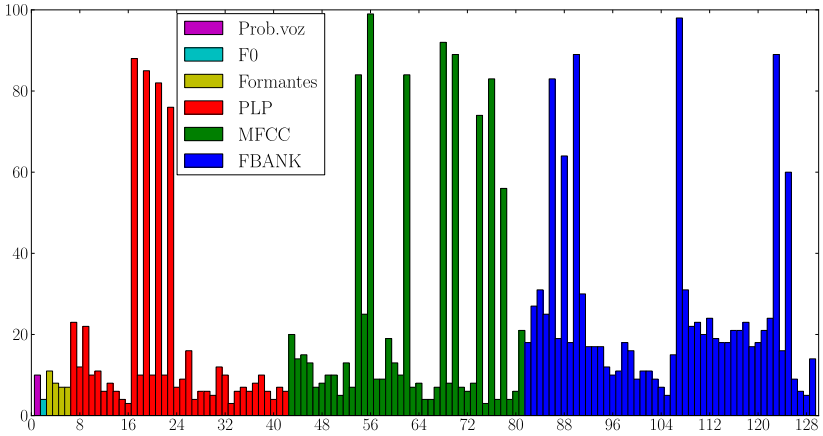
Figura 6.3: Taxa de acerto dos classificadores $\mathcal{C}^{20}$ para valores do fator de complexidade C da SVM utilizando $S = 50$.

É interessante investigar quais os parâmetros são mais selecionados. Assumindo-se $S = 50$, a Figura 6.4 apresenta o número de vezes que cada parâmetro foi selecionado para os 202 classificadores em $\mathcal{C}$. Para interpretar essa figura, deve-se observar que os parâmetros foram numerados de acordo com a seguinte ordem: probabilidade de vozeamento (prob.voz), F0, F1 a F4, os 12 parâmetros "estáticos" PLP, suas primeira e segunda derivadas, os 13 parâmetros estáticos MFCC (iniciando em $C0$ que depende diretamente da energia do sinal até o de mais alta ordem $C12$), suas primeira e segunda derivadas, os 24 valores de saída dos filtros (FBANK) e sua primeira derivada. A Figura 6.4 separa ainda os parâmetros correspondente a média (m), desvio padrão (d) e coeficiente angular (c), em três grupos.

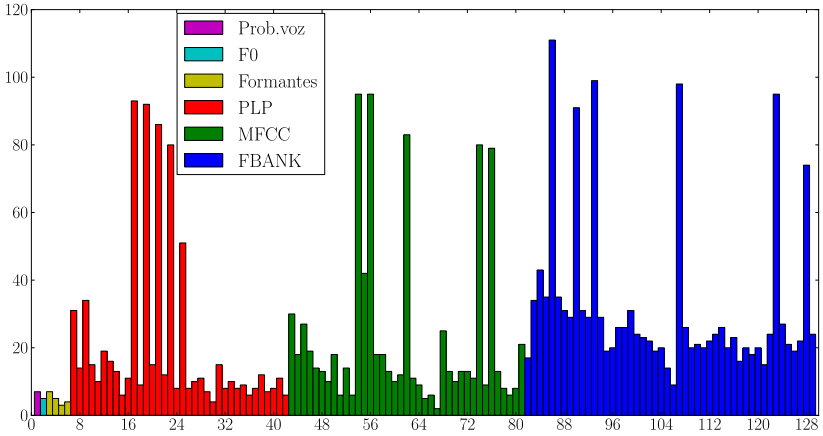
Pode-se observar na Figura 6.4 que os parâmetros prob.voz e F0 são relativamente pouco usados quando comparados aos demais. Contudo, para alguns classificadores esses parâmetros são considerados importantes pelo método de seleção de parâmetros. Por exemplo, para o classificador de número 178 ($th[3]$ e $z[3]$), que busca distinguir um som não-vozeado de um vozeado, o parâmetro considerado mais importante dentre os 50 selecionados para aquele classificador foi o prob.voz_m, ou seja, a média da probabilidade de vozeamento.



(a) Média da janela.



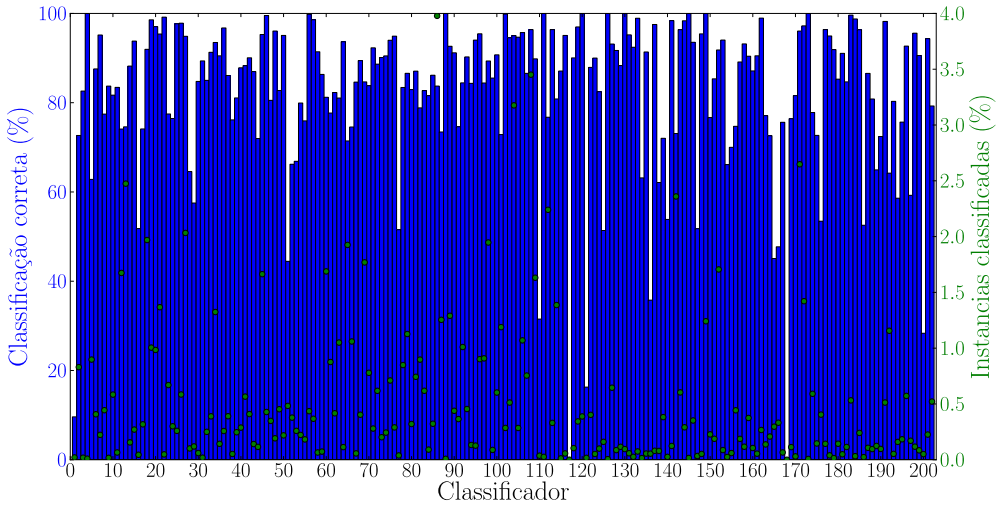
(b) Desvio padrão da janela.



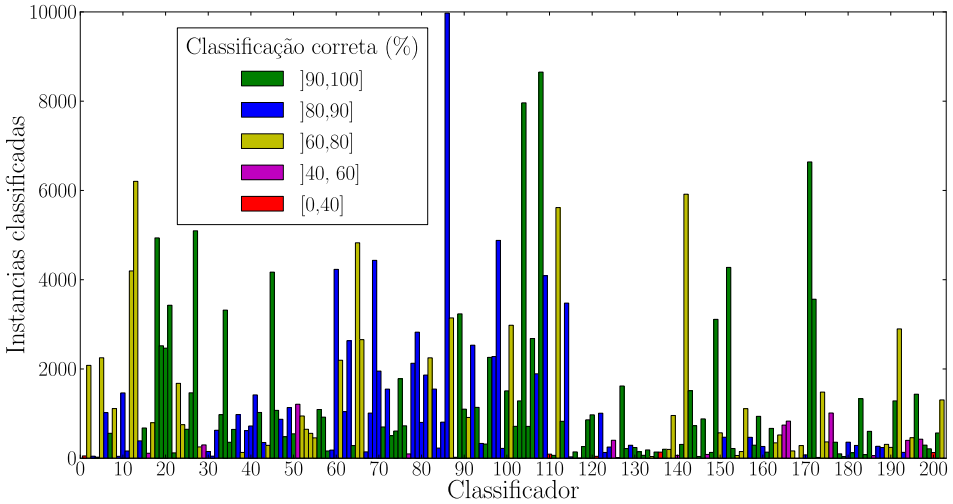
(c) Coeficiente angular da reta que melhor se ajusta aos pontos da janela.

Figura 6.4: Histograma para a quantidade de vezes que o parâmetro foi selecionado usando 50 parâmetros por classificador nos classificadores $\mathcal{C}$.

São necessários mais experimentos para se avaliar de maneira específica se a inclusão de um parâmetro como F0 (pouco "popular" mas intuitivamente importante para distinção de alguns sons) tem ou não impacto nos classificadores que o usam e depois no sistema como um todo, mas o estudo em si do uso de parâmetros heterogêneos é bastante instigante. Por exemplo, o *front-end* MFCC prioriza resolução em baixas frequências em detrimento das altas. Mas para o classificador 83 ($t[2]$ e $n[4]$), que busca distinguir uma plosiva de uma nasal, o parâmetro considerado mais importante foi o FBANK22_m, ou seja, a potência do antepenúltimo filtro.



(a) Taxa de acertos.



(b) Quantidade de instâncias classificadas.

Figura 6.5: Desempenho de cada um dos 202 classificadores para os pares da Tabela 6.5, usando ANN com 25 neurônios na camada oculta e $S = 50$.

A Figura 6.5 indica, para cada um dos 202 classificadores para os pares da Tabela 6.5, o desempenho de cada classificador e o número de exemplos de teste para o qual o mesmo foi invocado. Por exemplo, na Figura 6.5(b) percebe-se que o classificador 86 foi o mais usado e gerou resultado para aproximadamente 10 mil quadros. Sua taxa de acertos (ou seja, sua contribuição para o A_f global) ficou na faixa de 80 a 90%. Observando-se com atenção a Figura 6.5(a), percebe-se que o classificador 108 atinge taxa de acerto superior a 90% e corresponde a mais do que 3% dos quadros de teste, o que corresponde a mais de 8 mil quadros, como pode ser visto em (b). Note que os pontos verdes cujos valores são lidos na ordenada do lado direito da Figura 6.5(a) somam 100%. De forma geral, nota-se que muitos dos classificadores mais utilizados apresentam uma boa taxa de acertos.

Um dos fatores importantes a serem estudados é o valor de α na Equação 5.1. A Figura 6.6 apresenta a comparação entre o desempenho dos classificadores ANN utilizando 50 parâmetros e SVM com 150 com a variação de α. Como explicado, usou-se o mesmo α para todos os classificadores. Observa-se que para ambos classificadores valores razoáveis podem ser adotados em torno de $\alpha = 10$. Dessa figura observa-se também a ligeira superioridade das ANNs em relação às SVMs no tocante à taxa de erro SER. Por outro lado, o custo computacional de treino das SVMs (nesse caso de kernel linear) é significativamente menor do que o das ANNs e o custo computacional da etapa de teste também é menor para as SVMs pois o kernel linear permite que cada SVM seja transformada em um perceptron [97].

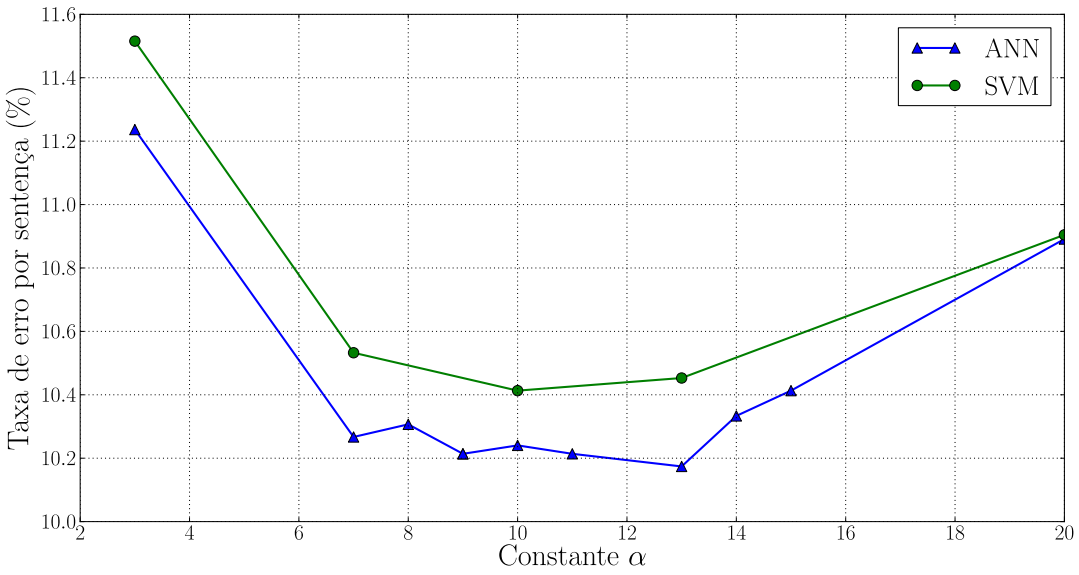


Figura 6.6: Taxa de erro por sentença em função da constante α usando 50 parâmetros por ANN e 150 parâmetros por SVM nos classificadores $\mathcal{C}$.

Por fim, adotou-se $\alpha = 13$ e novas ANNs foram construídas variando-se o número de neurônios na camada oculta. O resultado, tanto em termos de A_f (ordenada à esquerda) quanto SER (ordenada à direita), é mostrado na Figura 6.7. Observa-se que as 202 ANNs com apenas 10 neurônios na camada escondida permitiram obter um erro por sentença SER=10,08%, o que é um bom resultado em comparação ao *baseline* baseado apenas em HMMs, o qual conduziu a uma SER=16,01%. Em termos de WER, o desempenho dos sistemas foi 2,28% e 2,95%, respectivamente.

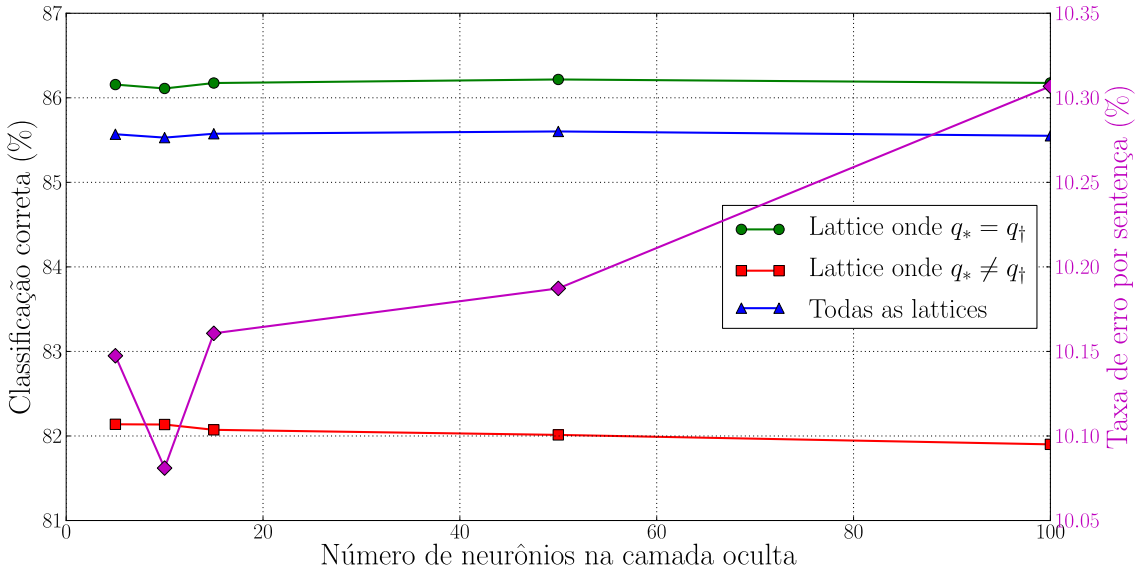


Figura 6.7: Desempenho dos classificadores $\mathcal{C}$ usando ANN em função do número de neurônios na camada oculta e $S = 50$.

Capítulo 7

Conclusões

Como contribuições deste trabalho destacam-se a melhoria das regras para PB, em especial as para conversão G2P (grafema-fone), apresentada na Seção 3.1, e para silabação (Seção 3.2). Outra contribuição foi a utilização de técnicas para adaptação ao locutor na minimização dos efeitos deletérios do descasamento acústico entre corpora. Os resultados dessa abordagem foram apresentados na Seção 4.2. O Capítulo 5 apresenta uma estratégia inovadora para melhorar os resultados provenientes de um sistema baseado em HMMs convencional.

Como exposto, uma das grandes dificuldades em se construir sistemas LVCSR e TTS, que dependem da língua, é a escassez de dados para treino e teste, além de procedimentos para lidar com os mesmos (*scripts*). Pode-se afirmar que o compartilhamento de bases de dados entre pesquisadores norte-americanos e europeus contribuiu, e muito, para o progresso alcançado por essas regiões na área de processamento de voz nas últimas décadas [12]. Assim, esta tese buscou apresentar uma contribuição no sentido de mudar a realidade de que não existem bases de dados de uso comum. Todos os módulos desenvolvidos encontram-se disponíveis no repositório do Projeto FalaBrasil [1]. Os recursos e ferramentas específicas para PB disponibilizados são:

- Um dicionário fonético com 65.532 palavras;
- Um conversor grafema-fone com determinação de vogal tônica (arquivo executável);
- Um separador silábico com determinação de sílaba tônica;
- Um corpus de texto de jornais com aproximadamente 672 mil sentenças formatadas;
- Dois corpora de áudio multi-locutor que correspondem juntos a 17,2 horas de áudio. Somente parte desses corpora encontra-se publicamente disponível e corresponde a 9 horas e 54 minutos de áudio;

- *Scripts* para implementação de modelos acústicos e de linguagem;
- Um modelo acústico independente de locutor no formato do HTK e um modelo de linguagem trigrama no formato ARPA;
- Uma API código-aberto para controlar o decodificador Julius. A API possui seu próprio conversor de gramática SAPI XML para o formato suportado pelo Julius; e
- Recursos necessários à construção de um sistema TTS completo para PB usando a plataforma MARY.

Os próximos parágrafos discutem possíveis trabalhos futuros, ao mesmo tempo que ilustram brevemente o estágio atual das pesquisas.

O dicionário fonético construído não foca um dialeto específico da língua portuguesa. Como trabalho futuro, pretende-se implementar uma nova versão do conversor G2P, adaptando as regras fonológicas ao dialeto falado na cidade de Belém do Pará. O objetivo é avaliar o desempenho do sistema LVCSR, com essa nova versão do dicionário fonético, com um grupo de locutores nativos da região de Belém do Pará.

Experimentos preliminares mostraram que o desempenho do sistema LVCSR piorou quando as sentenças do corpus LapsNews foram adicionadas ao conjunto de treino do modelo de linguagem. A explicação deve-se ao fato do LapsNews ainda ser pequeno, considerando o número de sentenças, em comparação ao CETENFolha. Por exemplo, adicionando as sentenças do LapsNews ao conjunto de treino com 1.534.980 sentenças (veja Figura 4.2), a perplexidade do modelo caiu de 143 para 142, ou seja, o LapsNews pouco contribuiu para melhorar a qualidade do modelo. Com isso, pretende-se dar continuidade à coleta automática de texto e aumentar o número de jornais (*sites*) explorados. A mesma ideia estende-se aos corpora de áudio, ou seja, almeja-se aumentar a quantidade de horas gravadas e o número de locutores.

De fato, após tornar os recursos disponíveis, o interesse por eles aumentou consideravelmente, devido em parte à consistência e usabilidade da API proposta. A API abstrai do programador detalhes de baixo nível relacionados à operação do *engine* e promove a integração do decodificador Julius com linguagens de programação populares. Entretanto, a interface de programação desenvolvida não segue uma especificação consagrada como a SAPI ou JSAPI. Assim, seria interessante implementar uma camada extra, entre a API e a aplicação, para padronizar o sistema de acordo com alguma especificação consagrada como SAPI ou JSAPI.

Embora o decodificador Julius tenha apresentado piores resultados em todos os testes, alguns estudos (p.e. [131]) comprovam que ele pode eventualmente superar o HDecode. Esta pesquisa ainda não foi capaz de fazer o Julius atingir o mesmo nível de desempenho do HDecode com os dados disponíveis para PB. Apesar disso, o Julius tem uma licença flexível que

permite seu uso comercial. Outro ponto que foi levado em consideração, é que o Julius foi construído especificamente para aplicações em tempo-real, onde ele próprio realiza a extração de parâmetros (semelhante ao processo realizado pela ferramenta HCopy do HTK [66]) e suporta entrada de áudio através do microfone. Por sua vez, o HDecode não possui nenhum suporte a tempo-real e a entrada de áudio é realizada apenas por arquivos padrões obtidos com a ferramenta HCopy.

Em sua primeira parte, este trabalho contribui com um sistema LVCSR no estado de arte para PB. A disponibilidade de tais recursos já promove ações de sucesso no desenvolvimento de aplicativos habilitados para PB. Por exemplo, o projeto SpeechOO [132], que é um módulo de reconhecimento de voz para o pacote OpenOffice.org. Um outro exemplo do relativo alto impacto alcançado por esta pesquisa, principalmente dentro da comunidade código-livre, é a quantidade de mensagens postadas no respectivo fórum de discussão (veja Figura 7.1), que, atualmente, conta com 115 membros.

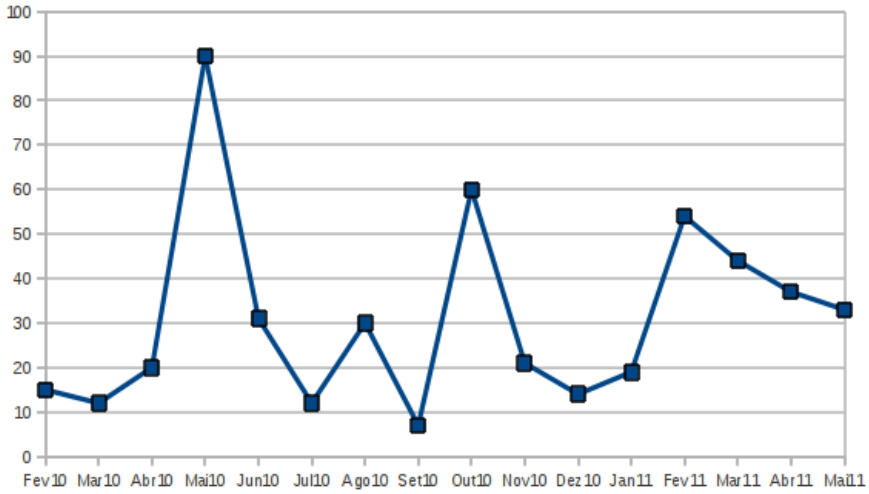


Figura 7.1: Evolução das mensagens postadas no fórum de discussão [1].

Para melhorar a qualidade da voz gerada pelo sistema TTS, pretende-se adicionar novos componentes ao módulo *front-end*, como um etiquetador morfológico, por exemplo. Neste trabalho, pouco esforço foi dispensado ao módulo *back-end*. Logo, almeja-se gerar, futuramente, voz sintetizada diretamente com o *software* HTS, e não mais usar as ferramentas da plataforma MARY para treinar os modelos estatísticos, pois as mesmas oferecem menor grau de liberdade quando comparadas ao HTS.

Com relação ao método proposto baseado em classificadores discriminativos, os resultados preliminares foram promissores, mas certamente a ideia requer um número bem maior de experimentos e adequações. Por exemplo, o conjunto de detectores de confusão pode ser melhorado através da introdução de recursos adicionais de fonética. Um objetivo futuro deste

estudo é aplicar a técnica proposta à tarefa de reconhecimento para grandes vocabulários.

É importante observar que o desenvolvimento de recursos para *baselines* de LVCSR e TTS é quase unanimamente considerado um passo essencial para pesquisas na área. Contudo, tal tarefa consome considerável energia e tem pouco apelo no âmbito da academia. Buscou-se neste trabalho atingir um balanço entre ações exploratórias de avanço do estado da arte (segunda parte do trabalho) e as necessárias para consolidar as pesquisas na área de processamento de voz realizadas pelo Laboratório de Processamento de Sinais da Universidade Federal do Pará (primeira parte). Espera-se assim que se tenha provido uma contribuição efetiva para a área de processamento da fala, em especial no que se refere ao PB.

Deve-se registrar que há dois questionamentos naturais perante os experimentos realizados com classificadores binários. O primeiro é inferir o ganho em termos de redução da SER ou WER caso o sistema de *baseline* para reconhecimento de dígitos fosse o melhor possível. Por exemplo, a equipe do Professor Chin-Hui Lee desenvolveu sistemas que atingem 99,43% de SER para o TIDIGITS [100]. Qual o impacto dos classificadores binários para esses sistemas de alto desempenho é uma questão importante.

Outro questionamento é como o método proposto se comporta em tarefas de LVCSR em termos de precisão e custo computacional. Essa é uma linha de pesquisa para trabalhos futuros, utilizando-se o sistema desenvolvido para PB na Universidade Federal do Pará.

Apesar do presente trabalho não ter respondido essas duas importantes questões, acredita-se que o mesmo levantou problemas interessantes e criou as condições necessárias para as promissoras investigações no assunto.

Apêndice A

Alfabeto fonético

```
<allophones name="sampa" xml:lang="pt-BR" features="vlng vheight vfront vrnd ctype cplace
    cvox">

 <silence ph="_"/>
 <!-- Oral vowels -->
 <vowel ph="a" vlng="s" vheight="3" vfront="1" vrnd="-"/>
 <vowel ph="E" vlng="s" vheight="2" vfront="1" vrnd="-"/>
 <vowel ph="e" vlng="s" vheight="2" vfront="1" vrnd="-"/>
 <vowel ph="i" vlng="s" vheight="1" vfront="1" vrnd="-"/>
 <vowel ph="O" vlng="s" vheight="2" vfront="3" vrnd="+"/>
 <vowel ph="o" vlng="s" vheight="2" vfront="3" vrnd="+"/>
 <vowel ph="u" vlng="s" vheight="1" vfront="3" vrnd="+"/>
 <!-- Nasal vowels -->
 <vowel ph="a~" vlng="l" vheight="3" vfront="2" vrnd="-"/>
 <vowel ph="e~" vlng="l" vheight="2" vfront="2" vrnd="-"/>
 <vowel ph="i~" vlng="l" vheight="1" vfront="2" vrnd="-"/>
 <vowel ph="o~" vlng="l" vheight="2" vfront="3" vrnd="+"/>
 <vowel ph="u~" vlng="l" vheight="1" vfront="3" vrnd="-"/>
 <!-- Semi-vowels -->
 <vowel ph="w" vlng="d" vheight="2" vfront="2" vrnd="0" ctype="v"/>
 <vowel ph="j" vlng="d" vheight="2" vfront="2" vrnd="0" ctype="p"/>
 <vowel ph="w~" vlng="d" vheight="2" vfront="2" vrnd="0" ctype="v"/>
 <vowel ph="j~" vlng="d" vheight="2" vfront="2" vrnd="0" ctype="p"/>
 <!-- Unvoiced fricatives -->
 <consonant ph="f" ctype="f" cplace="b" cvox="-"/>
 <consonant ph="s" ctype="f" cplace="a" cvox="-"/>
 <consonant ph="S" ctype="f" cplace="p" cvox="-"/>
 <!-- Voiced fricatives -->
 <consonant ph="z" ctype="f" cplace="a" cvox="+"/>
 <consonant ph="v" ctype="f" cplace="b" cvox="+"/>
 <consonant ph="Z" ctype="f" cplace="p" cvox="+"/>
 <!-- Affricatives -->
```

```
<consonant ph="tS" ctype="a" cplace="p" cvox="-" />
<consonant ph="dZ" ctype="a" cplace="p" cvox="+" />
<!-- Plosives -->
<consonant ph="b" ctype="s" cplace="l" cvox="+" />
<consonant ph="d" ctype="s" cplace="l" cvox="+" />
<consonant ph="t" ctype="s" cplace="d" cvox="-" />
<consonant ph="k" ctype="s" cplace="v" cvox="-" />
<consonant ph="g" ctype="s" cplace="v" cvox="+" />
<consonant ph="p" ctype="s" cplace="l" cvox="-" />
<!-- Liquids -->
<consonant ph="l" ctype="l" cplace="a" cvox="+" />
<consonant ph="L" ctype="l" cplace="p" cvox="+" />
<consonant ph="R" ctype="l" cplace="a" cvox="+" />
<consonant ph="X" ctype="l" cplace="a" cvox="+" />
<consonant ph="r" ctype="l" cplace="a" cvox="+" />
<!-- Nasal consoants -->
<consonant ph="m" ctype="n" cplace="l" cvox="+" />
<consonant ph="n" ctype="n" cplace="a" cvox="+" />
<consonant ph="J" ctype="n" cplace="p" cvox="+" />
</allophones>
```

Referências Bibliográficas

[1] www.laps.ufpa.br/falabrasil, Visited in June, 2010.

[2] C.-H. Lee, "From knowledge-ignorant to knowledge-rich modeling: A new speech research paradigm for next-generation automatic speech recognition," *Proc. ICSLP*, 2004.

[3] L. Rabiner and B. Juang, *Fundamentals of Speech Recognition*. Englewood Cliffs, N.J.: PTR Prentice Hall, 1993.

[4] X. Huang, A. Acero, and H. Hon, *Spoken Language Processing*. Prentice-Hall, 2001.

[5] T. Dutoit, *An Introduction to Text-To-Speech Synthesis*. Kluwer, 2001.

[6] P. Taylor, *Text-To-Speech Synthesis*. Cambridge University Press, 2009.

[7] J. Allen, M. S. Hunnicutt, D. Klatt, R. Armstrong, and D. Pisoni, *From Text to Speech: The MITalk System*. Cambridge University Press, 1987.

[8] J.-C. Junqua and J.-P. Haton, *Robustness in Automatic Speech Recognition*. Kluwer, 1996.

[9] J. Odell and K. Mukerjee, "Architecture, user interface, and enabling technology in Windows Vista's speech systems," *IEEE Transactions on Computers*, vol. 56, no. 9, pp. 1156–1168, 2007.

[10] www.google.com/chrome, Visited in June, 2010.

[11] M. Schramm, L. Freitas, A. Zanuz, and D. Barone, "A Brazilian Portuguese language corpus development," *International Conference on Spoken Language Processing*, vol. 2, pp. 579–582, 2000.

[12] R. Teruszkin and F. Vianna, "Implementation of a large vocabulary continuous speech recognition system for Brazilian Portuguese," *Journal of Communication and Information Systems*, vol. 21, pp. 204–218, 2006.

[13] C. A. Ynoguti and F. Violaro, "A Brazilian Portuguese speech database," *XXVI Simpósio Brasileiro de Telecomunicações*, 2008.

[14] J. Neto, H. Meinedo, M. Viveiros, R. Cassaca, C. Martins, and D. Caseiro, "Broadcast news subtitling system in Portuguese," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2008.

[15] J. Neto, C. Martins, H. Meinedo, and L. Almeida, "The design of a large vocabulary speech corpus for Portuguese," *Proc. European Conference on Speech Technology*, 1997.

[16] D. Paul and J. Baker, "The design for the Wall Street Journal-based CSR corpus," *Proc. International Conference on Spoken Language Processing*, 1992.

[17] I. T. M. Ribeiro, I. Duarte, and G. Matos, "Corpus de diálogo CORAL," *III Encontro para o Processamento Computacional da Língua Portuguesa Escrita e Falada*, 1998.

[18] I. Trancoso, R. Martins, H. Moniz, A. Silva, and M. Ribeiro, "The LECTRA corpus: Classroom lecture transcriptions in European Portuguese," *Language Resources and Evaluation Conference*, 2008.

[19] V. Valtchev, J. J. Odell, P. C. Woodland, and S. J. Young, "MMIE training of large vocabulary recognition systems," *Speech Communication*, vol. 22, no. 4, pp. 303–14, 1997.

[20] http://intervox.nce.ufrj.br/dosvox/, Visited in March, 2010.

[21] P. Vandewalle, J. Kovacevic, and M. Vetterli, "Reproducible research in signal processing - what, why, and how," *IEEE Signal Processing Magazine*, vol. 26, pp. 37–47, 2009.

[22] http://www.microsoft.com/portugal/mldc/default.mspx, Visited in June, 2010.

[23] "http://www.nuance.com," Visited in March, 2011.

[24] T. Dutoit, V. Pagel, N. Pierret, F. Bataille, and O. V. der Vrecken, "The MBROLA project: Towards a set of high quality speech synthesizers free of use for non commercial purposes," in *Proc. ICSLP '96*, vol. 3, Philadelphia, PA, October 1996, pp. 1393–1396.

[25] A. Lee, *The Julius Book*. 0.0.2 ed. - rev 4.1.2, 2009.

[26] W. Walker, P. Lamere, P. Kwok, B. Raj, R. Singh, E. Gouvea, P. Wolf, and J. Woelfel, "Sphinx-4: A flexible open source framework for speech recognition," Sun Microsystems Inc, Tech. Rep., 2004.

[27] D. Huggins-Daines, *PocketSphinx API Documentation*. Version 0.6, 2010.

[28] www.microsoft.com/speech/, Visited in November, 2010.

[29] N. Neto, E. Sousa, V. Macedo, A. Adami, and A. Klautau, "Desenvolvimento de software livre usando reconhecimento e síntese de voz: O estado da arte para o Português Brasileiro," *6th Fórum Internacional Software Livre*, 2005.

[30] S. Santos and A. Alcaim, "Um sistema de reconhecimento de voz contínua dependente da tarefa em língua portuguesa," *Revista da Sociedade Brasileira de Telecomunicações*, vol. 17, no. 2, pp. 135–147, 2002.

[31] R. Fagundes and I. Sanches, "Uma nova abordagem fonético-fonológica em sistemas de reconhecimento de fala espontânea," *Revista da Sociedade Brasileira de Telecomunicações*, vol. 95, pp. 225–239, 2003.

[32] E. Silva, L. Baptista, H. Fernandes, and A. Klautau, "Desenvolvimento de um sistema de reconhecimento automático de voz contínua com grande vocabulário para o Português Brasileiro," *XXV Congresso da Sociedade Brasileira de Computação*, 2005.

[33] A. Abad, I. Trancoso, N. Neto, and M. Ribeiro, "Porting an European Portuguese broadcast news recognition system to Brazilian Portuguese," *In Interspeech, Brighton, UK*, 2009.

[34] L. Gomes, E. Nagle, and J. Chiquito, "Text-to-speech conversion system for Brazilian Portuguese using a formant-based synthesis technique," *SBT/IEEE International Telecommunications Symposium*, pp. 219–224, 1998.

[35] P. Barbosa, F. Violaro, E. Albano, F. Simões, P. Aquino, S. Madureira, and E. Francozo, "Aiuruetê: a high-quality concatenative text-to-speech system for brazilian portuguese with demisyllabic analysis-based units and hierarchical model of rhythm production," in *Proceedings of the Eurospeech'99, Budapest, Hungary*, 1999, pp. 2059–2062.

[36] I. Seara, M. Nicodem, R. Seara, and R. S. Junior, "Classificação sintagmática focalizando a síntese de fala: Regras para o Português Brasileiro," *Simpósio Brasileiro de Telecomunicações*, pp. 1–6, 2007.

[37] R. Maia, H. Zen, K. Tokuda, T. Kitamura, and F. Resende, "An HMM-based Brazilian Portuguese speech synthetiser and its characteristics," *Journal of Communication and Information Systems*, vol. 21, pp. 58–71, 2006.

[38] T. Yoshimura, K. Tokuda, T. Masuko, T. Kobayashi, and T. Kitamura, "Simultaneous modeling of spectrum, pitch and duration in HMM-based speech synthesis," *Proc. of EUROSPEECH, 5*, pp. 2347–2350, 1999.

[39] D. Silva, "Algoritmos de processamento da linguagem natural para sistemas de conversão texto-fala em Português," Ph.D. dissertation, Facultade de Filoloxía da Universidade da Coruña, 2008.

[40] P. Woodland and S. Young, "The HTK tied-state continuous speech recognizer," *In: Proc. Eurospeech'93, Berlim*, 1993.

[41] P. Batista, P. Silva, N. Neto, and A. Klautau, "A non-visual web-browsing system using speech recognition for Brazilian Portuguese," *The International Conference on Computational Processing of Portuguese - Demos Session*, 2010.

[42] A. M. da Cunha and L. Velho, "Métodos probabilísticos para reconhecimento de voz," Laboratório VISGRAF - Instituto de Matemática Pura e Aplicada, Tech. Rep., 2003.

[43] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Trans. on ASSP*, vol. 26, no. 1, pp. 43–49, 1978.

[44] F. Jelinek, *Statistical methods for speech recognition.* MIT Press, 1997.

[45] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–86, Feb. 1989.

[46] H. Juang and R. Rabiner, "Hidden Markov models for speech recognition," *Technometrics*, vol. 33, no. 3, pp. 251–272, 1991.

[47] P. Woodland and D. Povey, "Large scale discriminative training of hidden Markov models for speech recognition," *Computer Speech and Language*, vol. 16, pp. 25–47, 2002.

[48] A. Lee, T. Kawahara, and K. Shikano, "Gaussian mixture selection using context-independent HMM," *In Proceedings IEEE-ICASSP*, 2001.

[49] C.-C. Cheng, "Online learning of large margin hidden Markov models for automatic speech recognition," Ph.D. dissertation, UCSD, 2011.

[50] M. Cohen, H. Franco, N. Morgan, D. Rumelhart, and V. Abrash, "Hybrid neural network/hidden markov model continuous speech recognition," *Proceedings of the International Conference on Spoken Language Processing*, 1992.

[51] H. Schwenk, "Using boosting to improve a hybrid HMM/neural network speech recognizer," in *ICASSP*, 1999, pp. 1009–12.

[52] J. Picone, "Signal modeling techniques in speech recognition," *Proceedings of the IEEE*, vol. 81, no. 9, pp. 1215–47, Sep. 1993.

[53] S. Davis and P. Merlmestein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Trans. on ASSP*, vol. 28, pp. 357–366, Aug. 1980.

[54] N. Deshmukh, A. Ganapathiraju, and J. Picone, "Hierarchical search for large-vocabulary conversational speech recognition," *IEEE Signal Processing Magazine*, pp. 84–107, 1999.

[55] N. Jevtić, A. Klautau, and A. Orlitsky, "Estimated rank pruning and Java-based speech recognition," in *Automatic Speech Recognition and Understanding Workshop*, 2001.

[56] P. Ladefoged, *A Course in Phonetics*, 4th ed. Harcourt Brace, 2001.

[57] L. Bahl, P. Souza, P. Gopalakrishnan, D. Nahamoo, and M. Picheny, "Context dependent modeling of phones in continuous speech using decision trees," in *DARPA Speech and Natural Language Processing Workshop*, 1994, pp. 264–270.

[58] G. Antoniol, R. Fiutem, R. Flor, and G. Lazzari, "Radiological reporting based on voice recognition," *Human-computer interaction. Lecture Notes in Computer Science*, vol. 753, pp. 242–253, 1993.

[59] C. H. Lee and J. L. Gauvain, "Speaker adaptation based on MAP estimation of HMM parameters," *IEEE ICASSP*, pp. 558–561, 1993.

[60] L. C. Sousa, "Adaptação de locutor em sistemas de reconhecimento de fala contínua empregando eigenvoices," Master's thesis, Universidade Estadual de Campinas, 2004.

[61] S. G. Ralf and R. Kompe, "A combined MAP + MLLR approach for speaker adaptation," *Proc Sony Res Forum*, vol. 9, pp. 9–14, 2000.

[62] K. Tokuda, H. Zen, and A. Black, "An HMM-based speech synthesis system applied to English," *Proceedings of IEEE Workshop on Speech Synthesis*, pp. 227–230, 2002.

[63] K. Tokuda, *An HMM-Based Approach to Flexible Speech Synthesis*, ser. Lecture Notes in Computer Science. Springer Berlin, November 2006, vol. 4274/2006.

[64] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[65] "P.800 - ITU: Methods for subjective determination of transmission quality," Visited in June, 2011. [Online]. Available: www.itu.int/rec/T-REC-P.800-199608-I/en

[66] S. Young, D. Ollason, V. Valtchev, and P. Woodland, *The HTK Book*. Cambridge University Engineering Department, version 3.4, 2006.

[67] A. Stolcke, "SRILM - an extensible language modeling toolkit," *International Conference on Spoken Language Processing*, 2002.

[68] M. Schroder, "Emotional speech synthesis: A review," *In The Proceedings of Eurospeech*, vol. 1, pp. 561–564, 2001.

[69] D. Caseiro, I. Trancoso, L. Oliveira, and C. Viana, "Grapheme-to-phone using finite-state transducers," in *In IEEE Workshop on Speech Synthesis*, 2002.

[70] I. Trancoso, M. Viana, and F. Silva, "On the pronunciation of common lexica and proper names in European Portuguese," in *In: Proc. 2nd Onomastica*, 1994.

[71] E. Franzen, D. Augusto, and C.Barone, "Automatic discovery of Brazilian Portuguese letter to phoneme conversion rules through genetic programming," *International Conference on Computational Processing of Portuguese Language*, 2003.

[72] A. Teixeira, C. Oliveira, and L. Moutinho, "On the use of machine learning and syllable information in European Portuguese grapheme-phone conversion," *Computational Processing of the Portuguese Language, Lecture Notes in Computer Science*, vol. 3960, pp. 212–215, 2006.

[73] I. Seara et al, "Geração automática de variantes de léxicos do português brasileiro para sistemas de reconhecimento de fala," in *XX Simpósio Brasileiro de Telecomunicações*, 2003, pp. v.1. p.1–6.

[74] D. Silva, A. de Lima, R. Maia, D. Braga, J. F. de Moraes, J. A. de Moraes, and F. R. Jr, "A rule-based grapheme-phone converter and stress determination for Brazilian Portuguese natural language processing," *VI International Telecommunications Symposium*, 2006.

[75] C. Hosn, L. Baptista, T. Imbiriba, and A. Klautau, "New resources for Brazilian Portuguese: Results for grapheme-to-phoneme and phone classification," *In VI International Telecommunications Symposium*, 2006.

[76] www.phon.ucl.ac.uk/home/sampa/, Visited in June, 2011.

[77] A. Faria, "Applied phonetics: Portuguese text-to-speech," University of California, Tech. Rep., 2003.

[78] acdc.linguateca.pt/cetenfolha/, Visited in June, 2010.

[79] D. Silva, D. Braga, and F. Resende, "Separação das sílabas e determinação da tonicidade no Português Brasileiro," *XXVI Simpósio Brasileiro de Telecomunicações*, 2008.

[80] R. J. Cirigliano, C. Monteiro, F. L. de F. Barbosa, F. G. V. R. Jr, L. Couto, and J. Moraes, "Um conjunto de 1000 frases foneticamente balanceadas para o Português Brasileiro obtido utilizando a abordagem de algoritmos genéticos," *XXII Simpósio Brasileiro de Telecomunicações*, 2005.

[81] N. Neto, P. Silva, A. Klautau, and A. Adami, "Spoltech and OGI-22 baseline systems for speech recognition in Brazilian Portuguese," *International Conference on Computational Processing of Portuguese Language - PROPOR*, 2008.

[82] P. Silva, N. Neto, A. Klautau, A. Adami, and I. Trancoso, "Speech recognition for Brazilian Portuguese using the Spoltech and OGI-22 corpora," *XXVI Simpósio Brasileiro de Telecomunicações*, 2008.

[83] F. Weimar, D. Barone, and A. Adami, "A baseline system for continuous speech recognition of Brazilian Portuguese using the West Point Brazilian Portuguese speech corpus," *International Conference on Computational Processing of Portuguese Language*, 2010.

[84] L. R. Welch, "Hidden Markov models and the Baum-Welch algorithm," *IEEE Information Theory Society Newsletter*, vol. 53, pp. 10–12, 2003.

[85] E. Silva, M. Pantoja, J. Celidônio, and A. Klautau, "Modelos de linguagem n-grama para reconhecimento de voz com grande vocabulário," in *III Workshop em Tecnologia da Informação e da Linguagem Humana*, 2004.

[86] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech and Language*, vol. 13, pp. 359–394, 1999.

[87] R. Kneser and H. Ney, "Improved backing-off for M-gram language modeling," *IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 1, pp. 181–184, 1995.

[88] msdn.microsoft.com/en-us/ms723632(VS.85).aspx, Visited in June, 2010.

[89] A. Kornai, *Extended Finite State Models of Language*. Cambridge University Press, 1999.

[90] T. Lander, R. Cole, B. Oshika, and M. Noel, "The OGI 22 language telephone speech corpus," *In Proc. Eurospeech, Madrid*, 1995.

[91] C. Bishop, *Neural networks for pattern recognition*. Oxford Univerisy Press, 1995.

[92] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning*. Springer Verlag, 2001.

[93] R. Duda, P. Hart, and D. Stork, *Pattern classification*. Wiley, 2001.

[94] R. Leonard, "A database for speaker independent digit recognition," *Proc. ICASSP*, 1984.

[95] A. Klautau, N. Jevtić, and A. Orlitsky, "Speech recognition based on discriminative classifiers," in *SBT, Rio de Janeiro, Brazil*, 2003.

[96] G. D. Forney, "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 3, pp. 268–278, 1973.

[97] A. Smola, P. Bartlett, B. Scholkopf, and D. Schuurmans, *Advances in Large Margin Classifiers.* MIT Press, 2000.

[98] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. [Online]. Available: citeseer.nj.nec.com/cortes95supportvector.html

[99] X. Li and H. Jiang, "Solving large-margin hidden Markov model estimation via semidefinite programming," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 15, pp. 2383–92, Nov 2007.

[100] J. Li, M. Yuan, and C.-H. Lee, "Approximate test risk bound minimization through soft margin estimation," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 15, pp. 2393–2404, Nov. 2007.

[101] N. Bitar and C. Espy-Wilson, "Knowledge-based parameters for HMM speech recognition," *Proceedings of ICASSP*, pp. 29–32, 1996.

[102] E. Eide, "Distinctive features for use in an automatic speech recognition system," *Proceedings of the EuroSpeech*, pp. 1613–1616, 2001.

[103] S. Siniscalchi, J. Li, and C.-H. Lee, "A study on lattice rescoring with knowledge scores for automatic speech recognition," *Proc. Interspeech*, 2006.

[104] C. Ma, Y. Tsao, and C.-H. Lee, "A study on detection based automatic speech recognition," *Proc. Interspeech*, 2006.

[105] K. Kirchhoff, G. Finkb, and G. Sagerer, "Combining acoustic and articulatory feature information for robust speech recognition," *Speech Communication*, vol. 37, pp. 303–319, 2002.

[106] S. Siniscalchi, T. Svendsen, and C.-H. Lee, "A phonetic feature based lattice rescoring approach to LVCSR," *Proceedings of ICASSP*, pp. 3865–3868, 2009.

[107] B. Launay, O. Siohan, A. Surendran, and C.-H. Lee, "Towards knowledge-based features for HMM based large vocabulary automatic speech recognition," *Proceedings of ICASSP*, pp. 817 – 820, 2002.

[108] T. Hori, C. Hori, Y. Minami, and A. Nakamura, "Efficient WFST-based one-pass decoding with on-the-fly hypothesis rescoring in extremely large vocabulary continuous speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, pp. 1352–1365, 2007.

[109] J. Zeng, J. Duana, and C. Wua, "A new distance measure for hidden Markov models," *Expert Systems with Applications*, vol. 37, pp. 1550–1555, 2010.

[110] I. Nabney, *Netlab algorithms for pattern recognition.* Springer, 2002.

[111] H. Bourlard and N. Morgan, *Connectionist speech recognition.* Kluwer, 1994.

[112] N. Morgan and H. A. Bourlard, "Neural networks for statistical recognition of continuous speech," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 742–72, 1995.

[113] F. Beaufays, H. Bourlard, H. Franco, and N. Morgan, "Neural networks in automatic speech recognition," *in The Handbook of Brain Theory and Neural Networks*, 2000.

[114] B. Scholkopf and A. Smola, *Learning with kernels.* MIT Press, 2002.

[115] A. Tikhonov and V. Arsenin, *Solutions of Ill-Posed Problems.* Winston, 1977.

[116] R. Rifkin, "Everything old is new again: A fresh look at historical approaches in machine learning," Ph.D. dissertation, MIT, 2002.

[117] G. Kimeldorf and G. Wahba, "Some results on Tchebychean spline functions," *J. Math. Anal. Applic.*, vol. 33, pp. 82–95, 1971.

[118] A. Halberstadt, "Heterogeneous acoustic measurements and multiple classifiers for speech recognition," Ph.D. dissertation, MIT, 1998.

[119] H. Hermansky, "Perceptual linear predictive (PLP) analysis of speech," *Journal of the Acoustical Society of America*, vol. 87, no. 4, pp. 1738–52, Apr. 1990.

[120] P. Baggenstoss, "A modified Baum-Welch algorithm for hidden Markov models with multiple observation spaces," *IEEE Trans. on Speech and Audio Proc.*, vol. 9, no. 4, pp. 411–16, 2001.

[121] H. Bourlard, S. Dupont, and C. Ris, "Multi-stream speech recognition," *CC-AI, The Journal for the Integrated Study of Artificial Intelligence, Cognitive Science and Applied Epistemology*, vol. 15, no. 3, pp. 215–34, 1998.

[122] H. Christensen, B. Lindberg, and O. Andersen, "Employing heterogeneous information in a multi-stream framework," in *ICASSP*, vol. 3, 2000, pp. 1571–4.

[123] A. Ng, "On feature selection: Learning with exponentially many irrelevant features as training examples," in *International Conf. on Machine Learning*, 1998, pp. 404–412.

[124] A. Webb, *Statistical Pattern Recognition.* Oxford University Press Inc., 1999.

[125] M. Hall, "Correlation-based feature selection for machine learning," Ph.D. dissertation, University of Waikato, 1999.

[126] U. Fayyad and K. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *13th International Joint Conf. on Artificial Intelligence*, 1993, pp. 1022–1027.

[127] "http://www.ldc.upenn.edu," Visited in March, 2011.

[128] http://www.cs.waikato.ac.nz/ml/weka/, Visited in November, 2010.

[129] D. Talkin, "Speech formant trajectory estimation using dynamic programming with modulated transition costs," AT & T Bell Laboratories 11222-870720-07TM, Tech. Rep., 1987.

[130] Talkin, *A Robust Algorithm for Pitch Tracking RAPT.* Elsevier Science B.V., 1995.

[131] X. Yao, P. Bhutada, K. Georgila, K. Sagae, R. Artstein, and D. Traum, "Practical evaluation of speech recognizers for virtual human dialogue systems," *7th International Language Resources and Evaluation*, 2010.

[132] W. D. Colen and P. Batista, "Veja mamãe, sem as mãos! SpeechOO, uma extensão de ditado para o BrOffice.org," *11th Fórum Internacional Software Livre*, 2010.