



Universidade Federal
do Rio de Janeiro

Escola Politécnica

TRANSCRIÇÃO DE ÁUDIO UTILIZANDO BIBLIOTECAS OFFLINE

Eduardo Naslauský

Projeto de Graduação apresentado ao Curso de Engenharia Eletrônica e de Computação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Flávio Luis de Mello

Rio de Janeiro

Outubro de 2018

TRANSCRIÇÃO DE ÁUDIO UTILIZANDO BIBLIOTECAS OFFLINE

Eduardo Naslauský

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO ELETRÔNICO E DE COMPUTAÇÃO


Autor:


Eduardo Naslauský

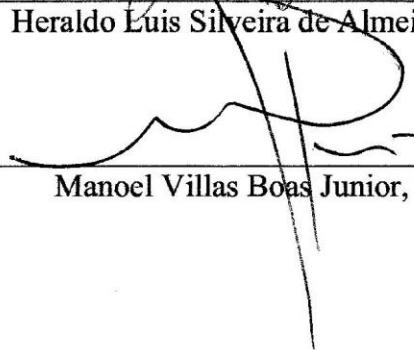
Orientador:


Flávio Luis de Mello, DSc.

Examinador:


Heraldo Luis Silveira de Almeida, DSc.

Examinador:


Manoel Villas Boas Junior, MSc.

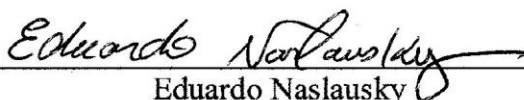
Rio de Janeiro – RJ, Brasil

Outubro de 2018

Declaração de Autoria e de Direitos

Eu, *Eduardo Naslauský* CPF 142.510.247-61, autor da monografia *Transcrição de áudio utilizando bibliotecas offline*, subscrevo para os devidos fins, as seguintes informações:

1. O autor declara que o trabalho apresentado na disciplina de Projeto de Graduação da Escola Politécnica da UFRJ é de sua autoria, sendo original em forma e conteúdo.
2. Excetuam-se do item 1. eventuais transcrições de texto, figuras, tabelas, conceitos e idéias, que identifiquem claramente a fonte original, explicitando as autorizações obtidas dos respectivos proprietários, quando necessárias.
3. O autor permite que a UFRJ, por um prazo indeterminado, efetue em qualquer mídia de divulgação, a publicação do trabalho acadêmico em sua totalidade, ou em parte. Essa autorização não envolve ônus de qualquer natureza à UFRJ, ou aos seus representantes.
4. O autor pode, excepcionalmente, encaminhar à Comissão de Projeto de Graduação, a não divulgação do material, por um prazo máximo de 01 (um) ano, improrrogável, a contar da data de defesa, desde que o pedido seja justificado, e solicitado antecipadamente, por escrito, à Congregação da Escola Politécnica.
5. O autor declara, ainda, ter a capacidade jurídica para a prática do presente ato, assim como ter conhecimento do teor da presente Declaração, estando ciente das sanções e punições legais, no que tange a cópia parcial, ou total, de obra intelectual, o que se configura como violação do direito autoral previsto no Código Penal Brasileiro no art.184 e art.299, bem como na Lei 9.610.
6. O autor é o único responsável pelo conteúdo apresentado nos trabalhos acadêmicos publicados, não cabendo à UFRJ, aos seus representantes, ou ao(s) orientador(es), qualquer responsabilização/ indenização nesse sentido.
7. Por ser verdade, firmo a presente declaração.


Eduardo Naslauský

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Escola Politécnica – Departamento de Eletrônica e de Computação

Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária

Rio de Janeiro – RJ CEP 21949-900

Este exemplar é de propriedade da Universidade Federal do Rio de Janeiro, que poderá incluí-lo em base de dados, armazenar em computador, microfilmar ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es).

Ao meu pai

AGRADECIMENTO

Quando pequeno ouvi dizer que o tempo médio de vida de uma borboleta é de apenas algumas semanas. Seu desenvolvimento passa por muitos estágios antes de culminar na fase adulta. Todos demandam esforço não somente próprio mas principalmente dos progenitores que depositam confiança e esperança em sua prole. Diversos fatores como clima, intempéries ou predadores influenciam o desenrolar da história, seja de forma positiva ou negativa. Apesar de diferentes veredas serem seguidas, o rumo desejado é comum a todos. Ainda, esse caminho é traçado e investido sem a espera de algo em troca. A satisfação de ver o casulo romper-se e assistir ao recém-adulto bater asas é suficiente para justificar qualquer empreitada. A borboleta adulta voa de flor em flor, sem saber para onde os ventos a levarão.

Agradeço então a meu pai que investiu e me confiou seu esforço ininterrupto para essa formação. Levando um dia de cada vez lutamos e enfrentamos as dificuldades, sempre com bom humor e um sorriso no rosto. Participou de toda etapa em minha vida, inclusive levando-me ao ponto de ônibus ou indo buscar-me na escola, sempre narrando ou discutindo um acontecimento. Contendo cada detalhe, suas histórias despertam sempre no ouvinte a vontade de ouvir mais, instigando a curiosidade. Ensinava com exemplos partindo desde “Quanto é um milhão vezes zero?”, que reforçam os conceitos, até “Qual a raiz quadrada de 81?”, que mostram para uma criança que sempre algo além pode ser aprendido e buscado. Em parte, a escolha desse curso deve-se a ele.

Devo agradecimentos também para minha mãe, que busca o melhor para mim. Sempre querendo saber se eu comi bem, fazia comida sem falta e ficava satisfeita ao me ver raspar o prato, satisfação essa que sempre tive prazer de dar e receber em troca seu sorriso. Sua risada e alegria contaminam o ambiente, e mostram que para ela não existe clima ruim.

Por fim, agradeço a todos meus amigos e colegas, que tornaram o dia-a-dia mais tolerável, seja no colégio, na faculdade, ou onde for. Decisões sempre são mais fáceis de serem tomadas quando feitas em grupo. Não vou enunciar nomes aqui por medo de esquecer de alguém. É com alegria que escrevo esse texto e olhos marejados com que o releio. Ser capaz de olhar para trás e lembrar os rostos e acontecimentos é uma dádiva. Obrigado.

RESUMO

Este projeto de graduação representa um estudo de prospecção sobre o conjunto de ferramentas CMUSphinx, sendo sua aplicação o reconhecimento de voz em português com a finalidade de transcrição. A temática é abordada elucidando os aspectos e elementos necessários para seu uso. As funcionalidades gerais do software são discutidas e avaliadas, enquanto realizados os experimentos passo a passo. Os resultados são apresentados e comparados quantitativamente com modelos existentes e justificados baseando-se nas características de cada sistema.

Palavras-Chave: CMUSphinx, PocketSphinx, reconhecimento de voz, modelo acústico.

ABSTRACT

This graduation project represents a prospective study of the CMUSphinx toolkit, with speech recognition in portuguese as its application and having the transcription as objective. These objectives are approached while explaining the concepts and elements required for its use. The general features of the software are discussed and evaluated, while the experiments are made step by step. Results are given and compared quantitatively with existing models and justified based on each system's characteristics.

Key-words: CMUSphinx, PocketSphinx, speech recognition, acoustic model.

SIGLAS

CMUSphinx – *Carnegie Mellon University Sphinx*

COR – Característica de operação do receptor

HMM – *Hidden Markov Model*

LAPS – Laboratório de Processamento de Sinais

SOX – *Sound Exchange*

WER – *Word Error Rate*

Sumário

1	Introdução	1
	1.1 - Tema	1
	1.2 - Delimitação	1
	1.3 - Justificativa	2
	1.4 - Objetivos	2
	1.5 - Metodologia	3
	1.6 - Descrição	3
2	Reconhecimento de voz	4
	2.1 - Introdução	4
	2.2 - Tecnologias envolvidas.	6
	2.3 - Iniciativas para português do Brasil.	8
3	Construção de modelos acústicos para o português do Brasil	11
	3.1 - Instalação das ferramentas Sphinx	11
	3.2 - Instalação das ferramentas utilitárias	16
	3.3 - Datasets utilizados	17
	3.4 - Reconhecimento de voz utilizando um modelo previamente estabelecido	19
	3.5 - Adaptação de um modelo previamente estabelecido com enriquecimento de áudios	23
	3.6 - Reconhecimento de voz com modelo completamente gerado pelo usuário	32
	3.7 - Resultados obtidos	42
4	Conclusões	49
	Bibliografia	50

Lista de Figuras

2.1 – Diagrama de blocos simplificado dos elementos criados e utilizados no processo de reconhecimento de voz	7
3.1 - Disposição dos componentes no diretório do modelo de reconhecimento de voz	22
3.2 – Disposição dos componentes no diretório de adaptação do modelo acústico necessários para o início do processamento	25
3.3 – Disposição dos componentes no diretório após a adaptação do modelo acústico	28
3.4 – Disposição dos componentes no diretório antes da criação do modelo acústico	33

Lista de Tabelas

3.1 – Exemplificação dos arquivos “.fileids” e “.transcription”	24
3.2 – Exemplificação dos arquivos de dicionário e arquivo de fonemas	34
3.3 – Guia para escolha dos parâmetros de treino	36
3.4 – Transcrição da primeira leitura utilizando o modelo original comparada com o texto original	42
3.5 – Transcrição da segunda leitura utilizando o modelo original comparada com o texto original	43
3.6 – Transcrições das duas leituras obtidas utilizando o modelo adaptado	44
3.7 – Resultado dos modelos gerados medido pela taxa de erros de palavras	47
3.8 – Comparação da WER entre o modelo adaptado e a API do Google em dois áudios de nossa base de dados.	48

Capítulo 1

Introdução

1.1 – Tema

O tema deste trabalho é a transcrição de áudio utilizada em ferramentas offline e de baixo consumo computacional, em especial o pacote de ferramentas CMUSphinx. Nesse sentido, o problema a ser resolvido é contornar a falta de modelos de reconhecimento disponibilizados para o português brasileiro. Os poucos modelos que existem foram feitos com pouco material ou não adequam-se para o reconhecimento de uma linguagem coloquial.

1.2 – Delimitação

Este tipo de estudo de caso empregando reconhecimento de voz é útil para qualquer projeto que utilize voz como fonte de dados de entrada. Além disso, a característica de baixo consumo computacional facilita o desenvolvimento para tecnologias móveis e sistemas independentes, como aplicativos para celulares ou dispositivos para pessoas com deficiência física.

Existem diversos bons modelos acústicos para nosso idioma disponíveis, que requerem porém, uma conexão à Internet. São exemplos desse tipo os modelos como o Google Cloud Speech, Microsoft Bing Voice Recognition e o IBM Speech to Text. Além de serem disponibilizados apenas online, não permitem personalização de sotaques e treino para contextos específicos, não adequando-se portanto ao nosso caso. No contexto de reconhecimento de voz offline em português, pouquíssimos modelos completos são encontrados.

Além disso, como o foco deste projeto é a adaptação para o idioma brasileiro, este trabalho visa tal âmbito e um incentivo nacional para ideias e adaptações nessa área, desde o presente momento até serem desenvolvidas e disponibilizadas alternativas melhores.

1.3 – Justificativa

O reconhecimento de áudio é um tópico extremamente complexo e atingir resultados com uma alta acurácia tem sido um desafio por décadas. A voz é um sinal de áudio em que não há uma clara divisão de partes, e ainda diferentes tons de voz, com diferentes formas de fala e sotaques, devem resultar necessariamente em uma mesma sequência de palavras. Todos esses aspectos acrescentam uma barreira que precisa ser ultrapassada para o crescimento de diversas outras áreas da ciência e tecnologia.

A tarefa de uma máquina perceber as características comuns por dentre tantas diferenças e dificuldades tem se mostrado complicada, ainda que o cérebro humano consiga realizar a mesma tarefa sem esforço e com apenas alguns anos de vida.

Com avanços nessa área, a distância entre os usuários e sistemas eletrônicos diminuirá permitindo uma maior automatização de tarefas, uma facilidade e estímulos para a inclusão de novos usuários para diversas tecnologias.

Além disso, este trabalho é particularmente importante porque permite a criação de modelos de reconhecimento de voz específicos para dado contexto. Por exemplo, podem ser construídos modelos voltados para a língua coloquial, ou para áudios de conversas telefônicas, que não seriam englobados ou não teriam uma acurácia tão grande quando reconhecidos por um modelo genérico.

1.4 – Objetivos

O objetivo principal é aprender os passos e demonstrar a possibilidade da criação de um modelo de reconhecimento de voz em português, para permitir, a quem desejar, construir o seu próprio modelo acústico e utilizar-se do resultado para desenvolvimento de novas tecnologias.

Para atingir tal fim, é necessário o aprendizado e o entendimento de como a voz é interpretada em teoria, quais as soluções matemáticas que são utilizadas para sua modelagem e quais são as abordagens utilizadas atualmente nesse ramo. Assim, será possível o estudo e o uso das ferramentas disponíveis para reconhecimento de voz e com isso desenvolver uma prova de conceito da criação de um modelo próprio.

1.5 – Metodologia

Para atingir os objetivos citados foi feito um estudo sobre a plataforma CMUSphinx, juntamente com o aprendizado de sua documentação oficial, que constitui a maior parte da biografia encontrada na pesquisa. Foram realizados experimentos práticos que envolvem a utilização das ferramentas, tais como o reconhecimento de voz utilizando modelos de reconhecimento previamente estabelecidos, adaptações de modelos existentes com gravações próprias e a criação como prova de conceito de um modelo completamente novo.

A maior parte das dúvidas e problemas encontrados tiveram suas soluções buscadas no fórum oficial da ferramenta, e, caso não fosse achado algo elucidativo o suficiente, um tópico novo era criado para a colaboração, seja dos criadores da ferramenta ou de outros pesquisadores, no eventual esclarecimento.

Além disso, foram feitas medidas e comparações da taxa de erro no reconhecimento dos diferentes modelos utilizando as métricas mais comuns no meio, para validar os melhores resultados e melhores abordagens para o problema.

1.6 – Descrição

No capítulo 2 é explicitado o conceito do reconhecimento de voz de forma geral. São expostos os conceitos teóricos básicos de reconhecimento de voz, o pacote de ferramentas utilizado, quais de suas funcionalidades ajudam em nosso objetivo e uma discussão geral sobre o estado da arte e as iniciativas de reconhecimento de voz no Brasil. Nele também serão ensinadas as métricas utilizadas para medição da eficiência dos modelos obtidos e como são calculadas.

O capítulo 3 mostra os três experimentos praticados: (1) o reconhecimento de voz dado um modelo acústico pronto; (2) a adaptação de um modelo pronto com a adição de conteúdo novo e; (3) a criação de um modelo completamente novo. Cada passo dado é justificado e explicitado de forma a auxiliar o leitor a repeti-los por conta própria, caso assim deseje. Todos os resultados são expostos e comparados com as opções disponíveis para quem desejar utilizar-se de um modelo para fins específicos.

A conclusão se dá no capítulo 4. Nele o trabalho é terminado e as conclusões acerca dos resultados são tomadas.

Capítulo 2

Reconhecimento de Voz

2.1 – Introdução

O objetivo de um sistema de transcrição de voz é, para dado sinal de voz, seja este oriundo de um arquivo de áudio ou de um microfone, converter em informação textual o que foi dito. As diferenças entre vozes, ambientes acústicos e entonações dificultam tal processo. Além disso, há um aumento da complexidade do problema, pois o mesmo fonema pode comportar-se diferente por dentre múltiplas palavras, dependendo dos fonemas que o envolvem.

As tecnologias com esse objetivo utilizadas atualmente são baseadas em resultados estatísticos, o que significa que um sistema desse tipo nunca obterá uma taxa de erros nula. Elas baseiam-se em dividir o sinal de áudio em diferentes estados que se sucedem com uma dada probabilidade. Pode-se fazer uma comparação de um estado com um fonema, porém como atestado acima o som de um fonema pode variar em relação a sua representação quando sozinho.

Outra abordagem é utilizar técnicas de machine learning e deep learning. Essas técnicas consistem de analisar o sinal de áudio no domínio da frequência, através da transformada de Fourier, e alimentar uma rede pré-treinada cuja saída é a probabilidade de cada som ter sido dito. Analogamente ao método anterior, essa solução é estatística, e nunca apresentará um resultado correto em sua totalidade. Embora apresente resultados melhores, essa abordagem requer um maior custo computacional tanto no momento da decodificação, quanto no momento de treino, além de uma base de dados de gravações muito maior. Por essa razão a maior parte desses sistemas requer uma conexão à Internet, visto que o processamento na nuvem facilita tanto no processamento quanto no treinamento para o aprimoramento da rede.

O modelo utilizado mais comumente é a divisão de fonemas em três partes, duas dessas dependendo dos fonemas anterior e posterior, enquanto a parte interior é regular e

estável, independente de sons adjuntos. Cada estado nesse modelo é chamado então de trifone, tratando assim a diferença entre essas possibilidades.

Para uma melhor eficiência computacional, cada estado é subdividido em muitas partes, pois muitas vezes mesmos trifones podem partilhar de um mesmo começo, por exemplo. Então são descritos a nível computacional detectores dessas subdivisões, que são chamados de sênones.

A modelagem estatística utilizada nesse caso é a proposta pelo modelo oculto de Markov (do inglês, HMM) [1], que também é usada para outros propósitos semelhantes como reconhecimento de gestos ou escrita. Um modelo baseado em HMM requer uma observação prévia do sistema, que associa para cada conjunto de estados, probabilidades de transições entre todos os estados existentes. Múltiplos conjuntos de estados representam o que queremos distinguir, no nosso caso fonemas. Essa observação é chamada de treinamento do modelo.

Com um modelo treinado, pode-se calcular a probabilidade de uma sequência de estados ter sido produzida por cada tipo de fonema, e toma-se o fonema com maior probabilidade como resultado do reconhecimento.

Os resultados podem ser mensurados sobre alguns critérios específicos. O principal é a taxa de erro de palavras (WER, em inglês) [2], que relaciona a transcrição original com o resultado do reconhecimento. É calculada como a razão entre a soma da quantidade de palavras adicionadas, removidas e modificadas, com o número total de palavras original. É medido na maioria das vezes como uma porcentagem.

Além do conceito da WER, existe também a métrica de acurácia [2]. É calculada da mesma forma, porém sem levar em consideração as palavras inseridas. Na maioria dos usos, é uma medida pior do que a WER, pois as inserções costumam representar uma piora no resultado final. Também existe uma proposta de balanceamento da WER [3], que é calculada com a metade do número de inserções e deleções. A justificativa diz que uma deleção e uma inserção constituem uma substituição, e por isso devem ter um peso menor.

Como referência, empresas grandes como Google, IBM e Microsoft alegaram em 2017 valores de WER próximos de 5% em seus sistemas de reconhecimento online. Respectivamente, foram obtidos os valores de 4.9%, 5,5% e 5,1% anunciados todos no ano de 2017 [4, 5, 6]. Estes podem ser considerados valores excepcionais, visto que são obtidos nas tecnologias mais recentes desenvolvidas pelas empresas mais avançadas na área.

O desenvolvedor do software CMUSphinx expõe que um modelo típico treinado com por volta de 10 horas de áudios deve obter algo em torno de 10% de WER, enquanto modelos maiores este valor pode ser maior, alcançando o patamar dos 30% [7]. No modelo pronto em português encontrado na pesquisa deste trabalho foi obtida empiricamente uma taxa de 22%.

Na pesquisa também foi encontrado um artigo também de 2017 que empiricamente compara as taxas entre a API do Google e da Microsoft com o software PocketSphinx [8]. Esse artigo utilizou arquivos de áudio de múltiplas origens distintas e mensurou a WER média. Ele obteve como resultado 9% para a API do Google, 18% para a da Microsoft e 37% para o Sphinx. Estes valores divergem daqueles divulgados por estas entidades [4, 5, 6], o que pode indicar que tais resultados são dependentes do corpus utilizado.

Além disso, outros critérios podem levados em consideração, como a velocidade do reconhecimento, ou a curva da característica de operação do receptor (COR) [2] que é utilizada para ilustrar as chances de falsos positivos ou taxa de acertos e erros. Essas propriedades são importantes em aplicações específicas, e não foram levadas em consideração neste trabalho, que tem seu foco na qualidade de acertos (WER) do reconhecimento.

2.2 – Tecnologias envolvidas

Os softwares utilizados neste trabalho estão inclusos no pacote de ferramentas CMUSphinx [9], desenvolvido e disponibilizado em código aberto pela Universidade Carnegie Mellon. Este pacote fornece o sistema de treinamento e reconhecimento de voz fazendo uso das cadeias ocultas de Markov e com o processamento local, isto é, sem a necessidade de uma conexão à Internet.

Este software foi escrito completamente na linguagem C, com alguns scripts externos em Perl. Isso mantém o processamento estrito ao que é necessário, reduzindo o custo computacional exigido para o reconhecimento, e permitindo assim seu uso em dispositivos com menor poder de processamento ou memória.

As ferramentas utilizadas do CMUSphinx são o PocketSphinx e o SphinxTrain, para reconhecimento e treinamento de modelos respectivamente. Também é utilizado o SphinxBase que é a biblioteca requerida por elas. O PocketSphinx pode ser incluso em aplicativos de celular e qualquer outro dispositivo capaz de rodar código em C, enquanto

o SphinxTrain deve ser utilizado preferencialmente em alguma distribuição Linux, porém tem seu uso permitido em plataformas Windows, com algumas ressalvas.

Um modelo de reconhecimento de voz utilizado por essas ferramentas é constituído por três elementos, como mostrado na figura 2.1. Primeiramente, um modelo acústico, que é criado baseado em áudios e suas transcrições. Esse modelo acústico contém as propriedades sonoras de cada sênone. Ele pode ser ou não referente a um contexto, especializando-se, caso assim seja, no reconhecimento de áudios vindos desse mesmo contexto, como ligações telefônicas, por exemplo. O segundo elemento utilizado é um dicionário fonético. Este dicionário é um arquivo texto que associa cada palavra à sua pronúncia, representada por uma sequência de fonemas. É imprescindível que este dicionário contenha todas as possíveis palavras que possam ser reconhecidas. Pode-se incluir uma mesma palavra mais de uma vez para representar diferentes pronúncias caso essa diferença seja gritante. Isto não deve ser feito para mais do que duas ou três pronúncias, visto que diferenças sutis são distinguidas pelo modelo acústico. Por último, o modelo de linguagem é construído de forma a limitar a busca de palavras pelo reconhecedor. Ele especifica que certas construções de frases são mais comuns do que outras e assim restringe a comparação de palavras substancialmente. Sem esse modelo de linguagem, em teoria, todo o áudio recebido deveria ser comparado com todos os sênone treinados, o que é impraticável.

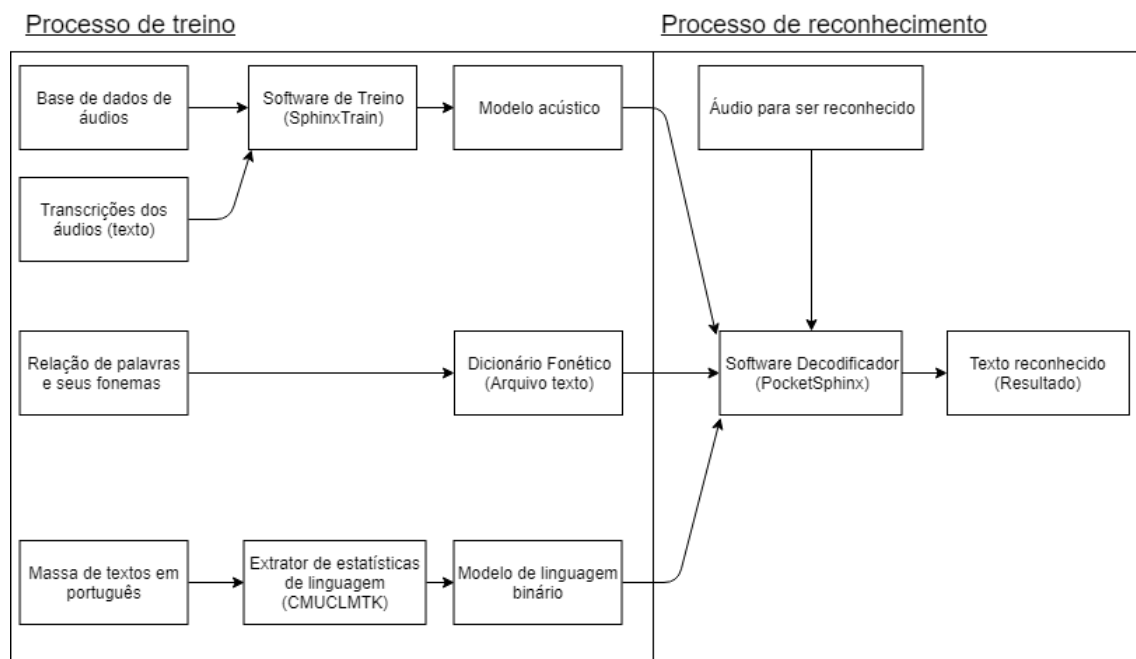


Figura 2.1 – Diagrama de blocos simplificado dos elementos criados e utilizados no processo de reconhecimento de voz (Adaptado de [10]).

A primeira versão do pacote de ferramentas Sphinx foi desenvolvida há mais de vinte anos, e desde então vem sendo melhorada e documentada em seu site oficial. Sua versão mais estável, e mais recente, “5prealpha” foi disponibilizada pela primeira vez em 2016. Repositórios no GitHub do desenvolvedor [11] fornecem o código original e os arquivos necessários para instalação e utilização. Existem alterações (commits) dentre os últimos meses demonstrando que o desenvolvimento ocorre até os dias de hoje.

O desenvolvedor mantém também uma página de tutorial extensa e completa [9], para um primeiro contato, que dita e explica os passos necessários para a utilização das funcionalidades básicas das ferramentas. Neste tutorial, não se cobrem todos os parâmetros e opções implementados no software, porém aborda todos os tópicos obrigatórios, além da resolução de alguns erros comuns encontrados por quem está começando o uso. Para problemas mais complexos, há um fórum online de ajuda [12] onde alguém que encontrou alguma dificuldade pode buscar por tópicos já respondidos ou criar um novo. A comunidade neste fórum é bastante ativa, com mensagens sendo enviadas e respondidas todos os dias, não só por usuários, como por contribuidores que ajudaram a desenvolver o software.

Embora tais softwares não tenham o mesmo alcance de utilização que outras ferramentas de voz mais comumente utilizadas, como o do Google Cloud Speech, sua característica de utilização off-line e baixo custo computacional atrai desenvolvedores ainda atualmente, e nesta categoria o PocketSphinx é o mais utilizado e documentado dentre os pesquisados. Aplicativos que o utilizam estão disponíveis na PlayStore, sendo facilmente encontrados e demonstrando que é uma tecnologia ainda usufruída por muitos.

O modelo de reconhecimento para o inglês já vem incluso na instalação padrão do PocketSphinx. Modelos prontos para idiomas adicionais podem ser baixados no site oficial, e são disponibilizados nas seguintes linguagens: Alemão, Grego, Inglês Indiano, Francês, Holandês, Espanhol, Italiano, Mandarim, Hindi, Cazaque e Russo. O português não está contemplado com um modelo oficial, o que desencoraja a criação de projetos nesse meio no nosso idioma.

2.3 – Iniciativas para o português do Brasil

Para um dado idioma, é proveitoso ter à disposição múltiplos modelos de reconhecimento. Desta forma pode-se adequá-los para o uso em diferentes contextos e

aplicações. Modelos podem ser focados em diferentes sotaques e diferentes níveis de coloquialismo, ou até mesmo diferentes características acústicas do ambiente de gravação, por exemplo.

No site do desenvolvedor do CMUSphinx, diversos modelos para a língua inglesa são disponibilizados e alguns para outras linguagens, como francês, chinês e italiano. Para o português nenhum modelo oficial é disponibilizado.

O único modelo disponibilizado para treino e para uso foi um produzido pelo Grupo FalaBrasil [13], que é um grupo reunido pelo Laboratório de Processamento de Sinais (LaPS) da Universidade Federal do Pará (UFPA). Ele tem como objetivo justamente a criação e fornecimento de modelos e recursos para o reconhecimento da voz em nosso idioma.

Na página deste grupo podemos ver que não existem atualizações e a última publicação data de 2011. São encontrados nesta página programas, modelos e base de dados relacionados ao reconhecimento de voz. Esses, por sua vez, são as versões iniciais e nenhum acréscimo notável foi feito desde então. É concluído então que este projeto encontra-se com seu desenvolvimento estacionado e não existe nenhuma sinalização de avanço.

Em seu site é disponibilizado um modelo chamado “constituição”, com todos os áudios e transcrições originais. Este modelo foi treinado em gravações de apenas um leitor ditando textos de nossa constituição. Trata-se, portanto, de um linguajar e pronúncia mais formais, não adequando-se para muitos casos específicos. É notável a sua qualidade de gravação e perfeição das transcrições, sendo assim um modelo bem robusto e conciso para adaptação ou até mesmo uso direto em muitas situações.

Além disso, existe um site internacional, chamado VoxForge [14], que tem como proposta coletar e reunir gravações de áudios transcritos em várias linguagens. Esse site conta com a participação coletiva de seus usuários, que gravam por seus próprios meios áudios contendo as frases requisitadas. Esse site foi idealizado e produzido pela própria universidade Carnegie Mellon, desenvolvedora do Sphinx, e dá suporte para submissões em dezenove linguagens, incluindo o português. O idioma que conta com o maior número de submissões é o inglês, com quase sete mil, enquanto o português vem em oitavo, com quatrocentas e vinte e três, de diversos usuários distintos. Infelizmente, nenhuma colaboração data de além de meados de 2017, o que indica que não há novidades no último ano.

Para o português, o VoxForge contém pouco mais de quatro horas de gravações, o que não é muito. Ainda, como cada usuário realiza suas gravações, a diferença de volume e qualidade por dentro os áudios é enorme, sem existir a garantia das transcrições estarem em conformidade com a fala.

O nosso idioma conta com um número grande de dialetos, sotaques e expressões regionais. O modo de fala varia imensamente por dentro os territórios do país, desde pronúncias diferentes para as palavras, como expressões e entonações próprias. Idealmente, com uma base de dados grande o suficiente para cada especificação, pode-se treinar modelos acústicos para todas, permitindo projetos direcionados para um espaço ou determinado objetivo.

Mesmo modelos disponibilizados online como o do Google Cloud Speech são voltados para aplicações específicas, como digitação de mensagens ou comandos, onde as possibilidades de transcrição são restritas. Ainda, o reconhecimento de voz de usuários com sotaques mais carregados possui uma taxa de erros consideravelmente maior, excluindo alguns indivíduos dos benefícios da tecnologia envolvida. Por esse motivo, é importante em uma iniciativa, como este Projeto de Graduação, treinar seus próprios modelos, permitindo assim a especificação e personalização dependendo da nossa necessidade. Pode-se pensar em um modelo de português “genérico”, que agrupe todas especificações e reconheça quaisquer variantes existentes. Essa é uma ideia até certo ponto plausível porém com muitos problemas. Nenhum modelo é verdadeiramente genérico, e seus arquivos de áudios irão retratar com pesos diferentes cada enfoque. Muitas variações de uma mesma palavra dificultam a percepção das semelhanças acústicas, e os tempos de treino e decodificação iriam aumentar substancialmente.

Embora modelos até certo ponto genéricos com uma base de dados grande ainda possa atingir resultados aceitáveis, eles nunca alcançarão a mesma precisão de um modelo específico, e são voltados para idiomas com menos variações existentes. No site oficial, por exemplo, notamos que existe outro modelo para Inglês indiano, deixando clara a interpretação do mesmo como a necessidade de dois modelos acústicos distintos.

Percebe-se então a necessidade urgente de modelos em português, ou então de uma base de dados de áudios transcritos para treinamento. Pode-se tentar obter esse material por outras fontes, como vídeos de sermões ou leituras de livros, mas a base de dados obtida por este tipo de áudio não iria refletir a fala rotineira e cotidiana.

Capítulo 3

Construções de Modelos Acústicos para o Português do Brasil

3.1 – Instalação das ferramentas Sphinx

O pacote de ferramentas CMUSphinx fornece instrumentos para a utilização, manuseio, adaptação e criação de modelos de reconhecimento de voz. A distribuição principal inclui os pacotes SphinxBase, PocketSphinx e o SphinxTrain. O PocketSphinx foca no reconhecimento em si e sua implementação nos diversos sistemas, enquanto o SphinxTrain é voltado para o treinamento dos modelos necessários. O SphinxBase é a biblioteca utilizada por ambos e é necessária para qualquer objetivo.

Além disso, existem ferramentas adicionais, de mesma autoria das demais, chamadas “cmuclmtk” e “g2p-seq2seq”, que realizam a construção do modelo de linguagem e do dicionário fonético, respectivamente. Sendo componentes importantes em um reconhecimento de voz, são disponibilizados separadamente pois existem outros programas alternativos que realizam as mesmas tarefas que estes. Contudo, para este trabalho foram escolhidos estes pois tratam-se do mesmo desenvolvedor do Sphinx e em muitos casos demonstram uma eficácia melhor.

Antes de seu uso, entretanto, é necessária a instalação individual das partes que serão necessárias. Para a instalação das ferramentas principais, o site oficial do desenvolvedor fornece para download arquivos comprimidos, para plataformas Windows e Linux. Os experimentos realizados neste trabalho se deram todos em uma distribuição Linux, como recomendado pelo desenvolvedor. Para a instalação em um sistema Windows, o programa Microsoft Visual Studio é necessário para compilar o código fonte e passos extras citados no site oficial são necessários.

Faça o download dos arquivos referentes ao SphinxBase, PocketSphinx e SphinxTrain na página SourceForge do desenvolvedor [15] ou em seu repositório no GitHub [11]. Após baixados estes arquivos, deve ser verificado se todos condizem com a

mesma versão, no caso deste trabalho, trata-se da mais recente: “5prealpha”. Feita essa verificação, todos eles devem ser descompactados, e ainda, no mesmo diretório devem estar contidas as subpastas “sphinxbase-5prealpha”, “pocketsphinx-5prealpha” e “sphinxtrain-5prealpha”.

Em cada uma dessas três pastas estão contidos os arquivos de instalação das respectivas ferramentas. Pela linha de comando, é necessário acessar cada uma delas e executar os comandos descritos a seguir. O primeiro comando só deve ser executado se os arquivos de instalação foram obtidos diretamente do repositório do desenvolvedor:

```
$ ./autogen.sh #Apenas caso obtido do repositório oficial.  
$ ./configure  
$ make  
$ sudo make install
```

É importante lembrar que o último comando requer permissão de superusuário e possivelmente os outros também, dependendo da configuração do sistema em que as ferramentas estão sendo instaladas.

Além disso, é necessário incluir as pastas com os arquivos binários na variável “PATH” de forma que o sistema saiba onde encontrá-los quando forem requisitados. Para isso, os seguintes comandos precisam ser executados:

```
$ export LD_LIBRARY_PATH=/usr/local/lib  
$ export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
```

Realizados os procedimentos, as ferramentas devem estar instaladas e funcionando corretamente. Para verificar, podemos testar utilizando o reconhecimento de voz padrão em inglês, com o áudio de um microfone conectado ao computador, diretamente pela linha de comando:


```
$ pocketsphinx_continuous -inmic yes
```

Quando executado, este comando capta o áudio do microfone e transcreve sequencialmente para a saída do terminal, usando o modelo incluso em inglês. Outra forma de testar a instalação é verificar se o sistema encontra as bibliotecas necessárias usando o comando:

```
$ pkg-config --cflags --libs pocketsphinx sphinxbase
```

Sendo a saída deste comando caminhos para os executáveis, ela deverá ser da seguinte forma:

```
-I/usr/local/include -I/usr/local/include/sphinxbase -  
I/usr/local/include/pocketsphinx -L/usr/local/lib -  
lpocketsphinx -lsphinxbase -lsphinxad
```

Para a ferramenta de criação de um dicionário fonético, chamada de g2p-seq2seq, passos semelhantes devem ser seguidos. Para isso, é necessário o Python previamente instalado. Os arquivos de instalação encontram-se em um arquivo compactado que é disponibilizado junto com os mencionados acima. Podem também, como neste caso, ser obtidos diretamente do repositório do desenvolvedor. As linhas de comandos para obtenção e instalação que devem ser utilizados são as seguintes:

```
$ git clone https://github.com/cmusphinx/g2p-seq2seq  
$ cd g2p-seq2seq/  
$ sudo pip install six --ignore-installed six  
$ pip install tensorflow  
$ pip install --upgrade tensorflow
```

```
$ pip install tensor2tensor==1.6.6
$ sudo python setup.py install
```

Esses comandos fazem uma cópia local do repositório oficial, e instalam as dependências nas versões adequadas antes da instalação da ferramenta propriamente dita. Essa ferramenta funciona sobre a linguagem Python e se utiliza de redes neurais e por isso requer os pacotes “tensorflow” e “tensor2tensor”. É aconselhável a instalação desses em um ambiente virtual Python, pois não trata-se das últimas versões de alguns pacotes.

Esta ferramenta conta com um script de teste de instalação próprio, que pode ser invocado logo após os anteriores utilizando o comando a seguir, e deverá ter como resultado algumas mensagens para cada teste feito. Após essas mensagens, caso nenhum erro seja encontrado, no terminal será impressa a palavra “OK”, indicando o sucesso na instalação.

```
$ python setup.py test
```

Com relação à ferramenta para criação de um modelo de linguagem, intitulada cmuclmtk, devem ser realizados passos análogos aos das demais ferramentas. Seu download pode ser feito analogamente pela página SourceForge do desenvolvedor [15]. Primeiramente, encontra-se o arquivo compactado localizado junto aos demais, no site mencionado, e seu download deve ser feito. Após descompactado em algum diretório arbitrário, os seguintes comandos devem ser executados para a compilação e geração dos executáveis necessários:

```
$ ./configure
$ sudo make install
```

Novamente, o último comando requer privilégios de superusuário, e o primeiro também dependendo das configurações do sistema alvo da instalação. Após esses comandos, serão gerados os programas necessários no diretório relativo interno:

```
cmuclmtk-0.7/src/programs/
```

Por último, o reconhecimento do pocketSphinx é projetado para utilização na linguagem C, e para seu uso ser feito em outra linguagem é necessário a instalação de um invólucro (do inglês, wrapper) para tal. Neste trabalho, foi utilizada a linguagem Python, juntamente com o pacote “SpeechRecognition”, que facilita o reconhecimento de arquivos áudio, sem a preocupação de inicialização de variáveis reconhecedoras, divisões e ponteiros para diferentes segmentos do áudio, etc. Felizmente, esse pacote aceita como API o pacote pocketsphinx para python, que também requer instalação.

Os seguintes comandos tratam-se da instalação dos pacotes citados, verificando as dependências necessárias, removendo quaisquer versões antigas e desatualizadas, e encarregando-se do funcionamento do pocketSphinx para a linguagem Python:

```
$ pip install SpeechRecognition  
$ sudo apt-get install python python-all-dev python-pip  
build-essential swig git libpulse-dev  
$ sudo apt-get install libasound2 libasound2-dev  
$ pip uninstall pocketsphinx  
$ pip install -upgrade pocketsphinx
```

Para testar essa funcionalidade, pode-se listar os pacotes instalados e verificar se encontram-se em sua última versão o pacote SpeechRecognition (no momento deste trabalho, 3.8.1) e o pacote pocketsphinx (no momento deste trabalho, 0.1.15). Este teste pode ser realizado fazendo:

```
$ pip list | grep -e SpeechRecognition -e pocketsphinx
```

Comando este que exibirá como saída uma tabela, relacionando os pacotes instalados com suas respectivas versões:

<i>pocketsphinx</i>	<i>0.1.15</i>
<i>SpeechRecognition</i>	<i>3.8.1</i>

3.2 – Instalação das ferramentas utilitárias

A maior parte dos experimentos realizados neste trabalho envolvem arquivos de áudio de alguma forma. As ferramentas que manipulam esse arquivo muitas vezes requerem que os áudios estejam em condições específicas para funcionar corretamente. No caso específico deste trabalho, para o treinamento dos modelos acústicos é necessário que os arquivos de áudio estejam no formato “.wav”, normalizados, com apenas um canal de gravação e frequência de amostragem de 16kHz. Os silêncios inicial e final no áudio não podem ultrapassar 200ms. Por essa razão foram utilizadas as ferramentas auxiliares SoX (Sound eXchange) [16] e Audacity[17], ambas de código aberto e uso gratuito.

O SoX é uma ferramenta de linha de comando que se autodenomina “o canivete suíço dos programas de processamento de som”. Ela disponibiliza opções para diversos tipos de operações e aceita as mais variadas extensões de áudio que existem. Sua instalação requer apenas a execução de uma linha de comando:

```
$ sudo apt-get install sox
```

Sua instalação pode ser testada digitando os dois comandos mais básicos da ferramenta, e verificando se o texto de ajuda é exibido:

```
$ sox  
$ soxi
```

O Audacity é uma aplicação para área de trabalho que fornece através de uma interface de usuário a manipulação, visualização, reprodução e edição de arquivos de áudio. Sua instalação pode ser feita realizando o download do instalador no site oficial do desenvolvedor e seguindo os passos descritos, ou então usando o seguinte comando no terminal:

```
$ sudo apt-get install audacity
```

O sucesso e integridade de sua instalação podem ser verificados abrindo o programa em si, que deve estar disponível junto às outras aplicações presentes no sistema.

3.3 – Datasets utilizados

São escassos os conjuntos de dados de voz para Português do Brasil. Sendo assim, qualquer massa de áudio ou texto é sempre bem vinda em experimentos envolvendo reconhecimento de voz. Neste trabalho foram empregados componentes de três origens distintas.

Primeiramente, foi utilizado o material fornecido pelo Grupo FalaBrasil em seu site [13]. Na seção de downloads deste site, é encontrado um modelo de reconhecimento de voz já treinado, projetado para a plataforma Sphinx, chamado de “constituição”. O modelo acústico, o dicionário fonético e o modelo de linguagem, foram desenvolvidos com gravações em um ambiente controlado de um leitor pronunciando trechos de nossa Constituição. Além disso são também disponibilizados todos os arquivos que foram utilizados para a criação de tal modelo acústico.

Esses arquivos contam com oito horas e cinquenta e oito minutos de gravações, distribuídas em 1.255 arquivos de áudio. Todos os arquivos de áudio possuem sua respectiva transcrição em um arquivo de texto homônimo. O dicionário fonético incluso possui por volta de 65 mil linhas, que representam aproximadamente o mesmo número de vocábulos. O modelo de linguagem, por sua vez foi composto por “frases dos corpora CETENFolha, Spoltech, OGI-22, Westpoint, LapsStory e LapsNews, totalizando 1,6 milhões de frases” [18], montando um arquivo de 54 MB.

A segunda fonte de dados foi o site VoxForge [14], este de mesma autoria do Sphinx. Ele conta com a participação coletiva de leitores que enviam pela Internet gravações baseadas em pequenas frases sugeridas pelo próprio site. Por conta de sua proposta de ambiente de gravação mais informal, fica evidente a variação no volume e qualidade das gravações. As transcrições de cada áudio estão inclusas, porém não há a garantia de que o que foi dito condiz com a transcrição referente.

Para o português do Brasil, o VoxForge conta com 423 submissões de usuários até o momento deste trabalho. A duração de todos os áudios reunidos é de quatro horas e dezesseis minutos, distribuídos em duzentas e dezoito frases diferentes. No total, o material conta com 4610 arquivos de áudio. Não é possível saber a quantidade exata de leitores distintos, visto que existem submissões anônimas, mas estima-se baseado nos nomes de usuários não ocultos que esse número seja no mínimo igual a trinta. Para o download, apenas uma submissão é permitida por vez, e portanto sua automatização foi realizada por meio de shell scripts. Também foi automatizada a distribuição pelos diretórios corretos e aglutinação dos arquivos de transcrições, visto que para qualquer experimento realizado as transcrições necessitam estar em um único arquivo.

O último conjunto de dados utilizado foi uma massa de arquivos de áudio relacionados com processos criminais transitados em julgado. A qualidade do áudio é boa, porém devido à natureza das gravações de interceptações telefônicas por muitas vezes ruídos de fundo são encontrados e, além disso, grandes pausas iniciais estão presentes, necessitando assim de um processamento prévio dos arquivos. Por último, nenhuma transcrição estava presente obrigando um exercício extra de transcrição manual para o treinamento de modelos utilizando este conjunto.

Essa massa de áudios conta com 702 arquivos que juntos agregam 13 horas e vinte e cinco minutos de gravações. Como mencionado anteriormente, a maioria desses arquivos contava com grandes pausas dispostas no início de cada gravação, além de existirem arquivos de duração muito curta ou nula. Desta forma, após o processamento e

corte das pausas, e remoção dos arquivos com duração menor que dois segundos, o total de horas de gravação passou a ser de 12 horas e vinte e dois minutos, distribuídos entre 517 arquivos de áudio.

Com esses conjuntos de dados é possível gerar combinações e adaptá-los de forma a obter diferentes resultados. Existem múltiplos parâmetros a serem otimizados para cada combinação de dados e requerem muitas experimentações e treinamentos diferentes para seus valores serem determinados. Este trabalho visa demonstrar os procedimentos para geração de diferentes modelos, utilizando algumas dessas combinações como exemplo.

3.4 – Reconhecimento de voz utilizando um modelo previamente estabelecido

O pacote “SpeechRecognition” para Python é compatível com as versões 2.6, 2.7 e 3.3+ da linguagem. Ele facilita a implementação do reconhecimento de voz para os desenvolvedores, de forma que sua utilização seja simples e agregando os benefícios de diversas API’s de reconhecimento, dentre elas a do Sphinx. Este experimento foca em transcrever o sinal de voz oriundo de um arquivo de extensão “.wav” local fazendo uso de um modelo de reconhecimento previamente estabelecido, explicando e comentando o que cada linha de comando faz.

Para tal, primeiramente deve-se abrir o interpretador Python que esteja com os pacotes devidamente instalados:

```
$ python
```

A primeira linha a ser digitada é a de importação do pacote SpeechRecognition. Note que o pacote pocketsphinx não é necessário de ser importado. Isto é, o SpeechRecognition acessa seus arquivos e modelos automaticamente.

```
import speech_recognition as sr
```

O componente mais fundamental e importante dessa importação é a classe `Recognizer`. É por meio de instâncias desta classe que as configurações e funcionalidades são ajustadas:

```
r = sr.Recognizer()
```

O primeiro passo necessário utilizando tal classe é informar a ela de onde ela irá obter o áudio. Isto é feito através do método `record`, tendo como entrada a saída do método `AudioFile`, visto que desejamos obter o áudio de um arquivo no sistema:

```
with sr.AudioFile('audio_reconhecido.wav') as source:  
    audio = r.record(source)
```

Com isso, o reconhecedor estará programado para receber como entrada o sinal de áudio do arquivo citado. É importante reparar que o nome do arquivo específico está descrito na linha, neste exemplo “audio_reconhecido.wav”. Este nome deve ser o mesmo nome de um arquivo de áudio contido no diretório de onde o interpretador Python foi iniciado.

Com essa configuração acertada, o processo de reconhecimento está pronto para começar. Para isso existe um método para cada API de reconhecimento que o pacote suporta. É incluso no nome do método a API que ele utiliza, como por exemplo o `recognize_google()`, ou, no nosso caso, `recognize_sphinx()`. Sua forma mais básica requer apenas como argumento a fonte de áudio a ser reconhecida:

```
r.recognize_sphinx(audio)
```

A saída desse comando é o resultado do reconhecimento do áudio no idioma padrão inglês americano. Para o reconhecimento utilizando outros idiomas, o parâmetro

“language” pode ser passado como argumento. Quando utilizado em os outras API’s, esse argumento precisa seguir a norma ISO-639[19], que estabelece e padroniza códigos de letras para representar os idiomas, porém, como o reconhecimento utilizando a API do Sphinx é local e offline, isso não é necessário. Neste caso o parâmetro “language” representa o nome da pasta em que contém o modelo pronto, dentro do diretório do pacote pocketsphinx, ou ainda, uma tupla indicando o caminho absoluto para os elementos que constituem um modelo completo.

```
r.recognize_sphinx(audio, language='en-US')
```

```
r.recognize_sphinx(audio, language='pt-BR')
```

```
r.recognize_sphinx(audio, language=( '/home/user/direto  
rio-modelo-acustico',           '/home/user/modelo-de-  
linguagem.lm.bin',             '/home/user/dicionario-  
fonetico.dict'))
```

No primeiro caso, a adição do parâmetro “language” não deve alterar o resultado, visto que o idioma padrão e único modelo incluso na instalação inicial é o inglês. Na segunda linha, ele deverá exibir uma resposta em português, caso exista a pasta de nome “pt-BR” instalada corretamente. No terceiro caso, a tupla indica os caminhos absolutos dos componentes desejados. O primeiro elemento da tupla é o caminho para o diretório em que está contido o modelo acústico. O segundo, por sua vez, é o caminho para o modelo de linguagem. Por último, o terceiro elemento da tupla é o caminho para o dicionário fonético.

Para a instalação de um modelo arbitrário na pasta do pacote pocketsphinx, a pasta do modelo precisa estar disposta da seguinte maneira. Os arquivos de modelo de linguagem e dicionário fonético devem estar nomeados como “language-model.lm.bin” e “pronunciation-dictionary.dict” respectivamente. A subpasta do modelo acústico deve chamar-se “acoustic-model”. Com esses três elementos corretos, a pasta que os contém deve ser nomeada arbitrariamente, no exemplo “pt-BR”.

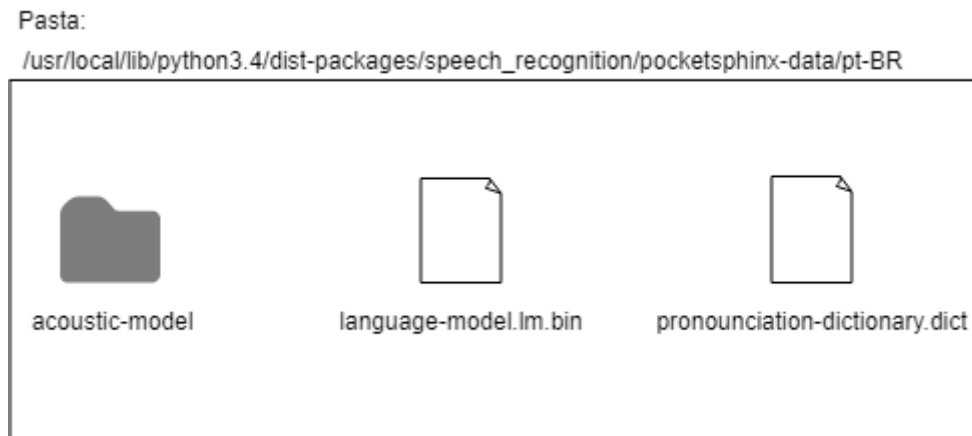


Figura 3.1 – Disposição dos componentes no diretório do modelo de reconhecimento de voz.

Para descobrir o local em que essa pasta deve ser movida, o usuário pode digitar o seguinte comando no terminal:

```
$ python -c "import speech_recognition as sr, os.path as p; print(p.dirname(sr.__file__))"
```

Que deverá ter como saída um caminho absoluto do tipo:

```
/usr/local/lib/python3.4/dist-packages/speech_recognition
```

Dentro dessa pasta, existe uma subpasta denominada “pocketsphinx-data”, que é o local para onde o modelo de reconhecimento de voz deve ser copiado. É importante verificar as permissões das pastas e arquivos copiados para permitir ao Python a correta execução do reconhecimento. Assim, eventualmente pode ser necessário corrigir os privilégios de acesso ao conteúdo desta pasta fazendo:

```
$ sudo chmod -R 755 *
```

Concluindo, este foi o exemplo mais elementar de reconhecimento de voz. A partir dele, pode-se utilizar da lógica da linguagem de programação para incrementar suas funcionalidades, como a leitura de múltiplos arquivos de áudio em um mesmo comando, ou ainda requisitar ao usuário a entrada do arquivo para ser reconhecido ou modelo a ser usado. Cabe ao desenvolvedor que irá fazer uso do reconhecimento adequar-se em seu contexto e elaborar a melhor forma para o fazer.

3.5 – Adaptação de um modelo previamente estabelecido com enriquecimento de áudios

Uma possibilidade para se obter um modelo de reconhecimento de voz melhor adequado sem a necessidade de um conjunto de áudios grande é adaptar um modelo já existente. Adicionando áudios ao seu treinamento, pode-se direcionar um modelo qualquer para objetivos específicos. Desta forma, pode-se por exemplo adaptar um modelo genérico da língua com um conjunto menor de áudios referentes a algum sotaque ou ambientação específica.

Para esse experimento, foram gravados pelo autor deste trabalho doze arquivos de áudio referentes ao poema VII do livro “O Guardador de Rebanhos” de Fernando Pessoa [20]. Para cada um dos 10 versos presentes no poema, foi gravado um áudio isolado de sua leitura. Além disso, para fins de testes foram gravados dois áudios contendo a leitura completa, o primeiro de forma mais calma e pausada e o segundo de maneira mais acelerada.

Todos os áudios foram dispostos no mesmo diretório. As especificações necessárias para tais áudios são ditas pelo desenvolvedor do Sphinx, sendo elas a frequência de amostragem ser igual a 16kHz, canal único, normalizado no formato “.wav”. Para isso, foi utilizado o sox com o seguinte comando:

```
$ for i in $(ls *.wav)
do sox $i --norm -r 16000 fp/$i
```

done

Após esse comando ser executado, todos os áudios estarão adequados para o uso encontrando-se no diretório representado no exemplo por “fp”. Esse diretório deve ter o nome do modelo adaptado a ser criado. No diretório ainda, devem ser postos os componentes do modelo de reconhecimento original, que deseja-se adaptar.

Prosseguindo, é necessário criar um arquivo de extensão “.fileids” com o nome do modelo, no exemplo “fp.fileids”, para listar o nome dos arquivos de áudio utilizados no treinamento. Esses nomes precisam estar dispostos um em cada linha, sem a extensão. Foram utilizados neste trabalho os dez áudios da leitura do poema, enquanto os outros que restaram foram utilizados para teste da eficiência.

Em par com esse arquivo, outro de mesmo nome deve ser criado com a extensão “.transcription”. Este por sua vez deve conter as transcrições dos arquivos de áudio contidos no arquivo “.fileids”, linha por linha. Isto é, a transcrição do áudio no arquivo de transcrições deve estar na linha de mesmo número da linha que contém o nome deste áudio, no arquivo “.fileids”. Todas as transcrições devem começar e terminar com o fonema de silêncio, comumente denotado por “<s>” e “</s>”, contido no dicionário fonético. Ainda, no final de cada linha deve constar entre parênteses o nome do arquivo ao qual se refere.

Tabela 3.1 – Exemplificação dos arquivos “.fileids” e “.transcription”.

Exemplo do arquivo “fp.fileids”	Exemplo do arquivo “fp.transcription”
verso1	<s> da minha [...] universo </s> (verso1)
verso2	<s> por isso [...] qualquer </s> (verso2)
verso3	<s> porque eu [...] que vejo </s> (verso3)
verso4	<s> e nao [...] altura </s> (verso4)
verso5	<s> nas [...] pequena </s> (verso5)
verso6	<s> que [...] outeiro </s> (verso6)
verso7	<s> na cidade [...] chave </s> (verso7)
verso8	<s> escondem [...] o ceu </s> (verso8)
verso9	<s> tornam [...] podem dar </s> (verso9)
verso10	<s> e tornam [...] ver </s> (verso10)

Assim, o diretório atual “fp” estará pronto para o começo da adaptação. Ilustrado pela figura 3.2, ele deve ser composto pelo conjunto dos áudios utilizados para o enriquecimento, o arquivo que os lista, de nome “fp.fileids”, e o arquivo contendo as transcrições, chamado de “fp.transcription”.

Pasta da adaptação: fp

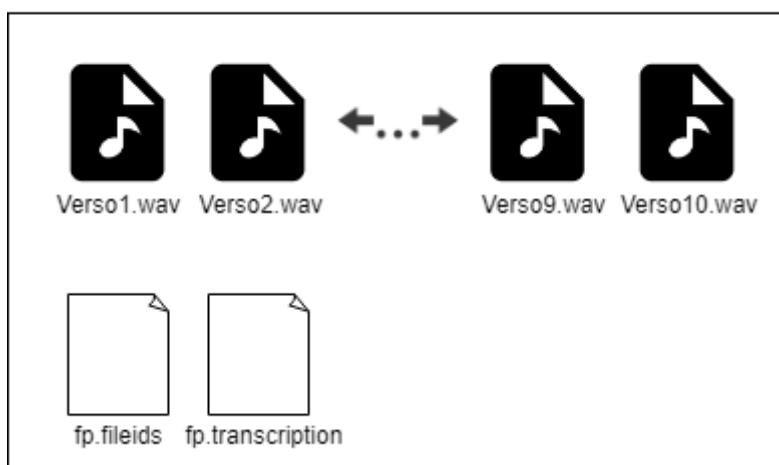


Figura 3.2 – Disposição dos componentes no diretório de adaptação do modelo acústico necessários para o início do processamento.

Com esses arquivos dispostos corretamente pode-se realizar o primeiro comando para treinamento. Este comando irá obter características acústicas dos arquivos “.wav” gerando um arquivo de extensão “.mfc” para cada. Essa obtenção deverá ser feita utilizando os mesmos parâmetros e configurações que o modelo acústico original usou. Por essa razão, como argumento esse comando recebe o arquivo “feat.params” contido no modelo acústico original.

```
$ sphinx_fe -argfile pt-BR/acoustic-model/feat.params  
-samprate 16000 -c fp.fileids -di . -do . -ei wav -eo mfc -  
mswav yes
```

Sendo assim, o primeiro argumento indica o caminho para o arquivo mencionado. Já o segundo parâmetro “fp.fileids” indica ao programa o caminho do arquivo “.fileids”, para assim tomar conhecimento de todos os arquivos de áudio.

O modelo original deve conter o arquivo “mdef” no formato texto. Caso ele esteja em formato binário, que não foi o caso em nosso experimento, o sphinxTrain possui uma ferramenta para conversão:

```
$ pocketsphinx_mdef_convert -text pt-BR/mdef pt-BR/mdef.txt
```

Para o próximo passo são necessários os executáveis “bw”, “map_adapt” e “mk_s2sendump”, que estão presentes na pasta de instalação do sphinxTrain. Portanto deve-se copiá-los para o diretório local:

```
$cp /usr/local/libexec/sphinxtrain/bw .  
$cp /usr/local/libexec/sphinxtrain/map_adapt .  
$cp /usr/local/libexec/sphinxtrain/mk_s2sendump .
```

Assim, pode-se coletar estatísticas dos arquivos “.mfc” gerados utilizando o seguinte comando:

```
$. /bw -hmmdir ./pt-BR/acoustic-model -moddefn ./pt-BR/acoustic-model/mdef -ts2cbfn .cont. -feat ls_c_d_dd -cmn current -agc none -dictfn ./pt-BR/pronunciation-dictionary.dict -ctlfn fp.fileids -lsnfn fp.transcription -accumdir .
```

Neste comando, são passados como argumentos o caminho para a pasta do modelo acústico original e para o arquivo “mdef” respectivo. Além disso são passados também os caminhos para o dicionário fonético utilizado e os arquivos “.fileids” e “.transcription” das gravações que estão sendo adicionadas ao modelo acústico. O parâmetro “-ts2cbfn”

representa o tipo de modelo que se deseja treinar, no caso representando um modelo contínuo pelo argumento “.cont.”. Outras opções como “.semi.” ou “.ptm.” representam outros tipos de modelo que não são adequados para transcrições de ditado.

Por último, podem ser necessários ajustes adicionais, a depender do modelo original utilizado. Caso este contenha o arquivo “feature_transform”, é necessário adicionar o parâmetro “-lda feature_transform” para o comando acima. Além disso, caso o arquivo “noisedict” esteja ausente no modelo original, deve-se copiar o arquivo “fillerdict” com este nome, pois são equivalentes.

Este comando gerou arquivos “mixw_counts”, “tmat_counts” e “gauden_counts”. Esses arquivos representam quantidades estatísticas obtidas dos áudios e são essenciais para o funcionamento interno da adaptação.

Como último passo da adaptação, no diretório local deve-se copiar duas vezes a pasta contendo o modelo acústico, com nomes distintos, para representar a origem e destino da adaptação. Por exemplo no caso em questão, utiliza-se o nome original “acoustic-model” e o nome “acoustic-model-fp” para o destino da adaptação. O comando da adaptação segue da seguinte forma:

```
$. /map_adapt -moddefn acoustic-model/mdef -  
ts2cbfn .ptm. -meanfn acoustic-model/means -varfn acoustic-  
model/variances -mixwfn acoustic-model/mixture_weights -  
tmatfn acoustic-model/transition_matrices -accumdir . -  
mapmeanfn acoustic-model-fp/means -mapvarfn acoustic-model-  
fp/variances -mapmixwfn acoustic-model-fp/mixture_weights -  
maptmatfn acoustic-model-fp/transition_matrices
```

Após esse comando, todos os arquivos adaptados encontram-se na pasta destino escolhida. O restante dos arquivos é reutilizado do modelo original, sem modificações. Portanto para utilizar este modelo completo, deve-se copiar para a pasta destino os arquivos originais “feat.params”, “mdef” e “noisedict”. Essa pasta deve-se chamar “acoustic-model” e estar presente juntamente com o dicionário fonético e modelo de linguagem originais. Estando assim o modelo acústico pronto para uso.

O diretório da adaptação “fp” em seu estado final é composto por arquivos e dois subdiretórios, o primeiro contendo o modelo acústico original, e o segundo contendo o modelo acústico adaptado pronto. Dentre os arquivos, inclui todos os áudios “.wav” e seus respectivos pares de extensão “.mfc”. Contém o par de arquivos que descrevem os áudios “.fileids” e “.transcription”, os executáveis utilizados “bw”, “map_adapt” e “mk_s2sendump”, e os arquivos de informações estatísticas “mixw_counts”, “tmat_counts” e “gauden_counts”. Essa organização é ilustrada pela figura 3.3.

Pasta da adaptação: fp

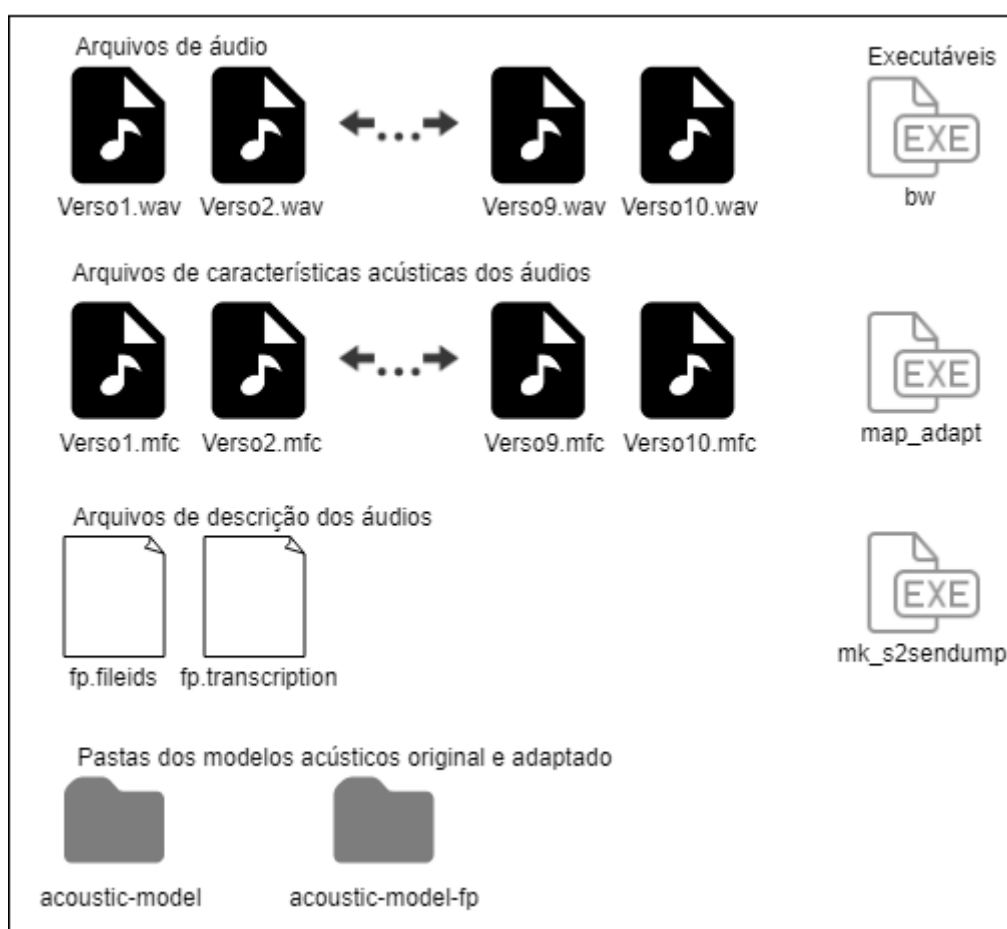


Figura 3.3 – Disposição dos componentes no diretório após a adaptação do modelo acústico.

Um modelo de reconhecimento completo contém além do modelo acústico, um dicionário fonético e um modelo de linguagem. O modelo de linguagem está vinculado diretamente ao idioma em si, e não varia conforme ambientação das gravações ou sotaques. Portanto, este pode e deve ser aproveitado de um modelo adaptado, dado que foi criado representando bem o idioma utilizado.

O dicionário fonético reaproveitado pode não possuir a pronúncia de todas as palavras que deseja-se reconhecer. Embora seja possível adicionar manualmente as pronúncias de algumas palavras, esta tarefa torna-se impraticável quando se tem uma lista maior de palavras. Para isso, pode-se utilizar a ferramenta “g2p-seq2seq” com o objetivo de expandir um dicionário para conter a pronúncia de novas palavras.

Este software de mesma autoria do Sphinx baseia-se em redes neurais para alcançar um patamar de acurácia maior do que os similares. A utilização deste software se dá em duas etapas. Primeiro, treina-se uma rede neural alimentando-a com um dicionário existente. Em um segundo estágio pós-treino, esta rede estará apta para exibir a melhor estimativa de pronúncia para uma dada palavra pedida pelo usuário.

Para o treino dois diretórios devem ser criados. Um deles irá abrigar o dicionário original e eventuais subdivisões que serão feitas nele. O outro representa a saída com a rede treinada, que pode ser copiada e utilizada livremente uma vez terminado o processo. Dispostos corretamente, pode-se iniciar o treino com o seguinte comando:

```
$g2p-seq2seq                                --train  
pasta_do_dicionario/pronunciation-dictionary.dict    --  
model_dir pasta_modelo_g2p
```

Onde o primeiro parâmetro é o caminho para o dicionário original, e o segundo é o diretório que deverá conter o resultado do treino. Este comando por si só é suficiente e completamente funcional para o treinamento, porém outros argumentos podem ser passados para especificar e aprimorar o resultado. São eles:

```
--max_epochs 0  
--size 256  
--num_layers 3  
--filter_size 512  
--num_heads 4  
--valid  
--test
```

```
--min_length_bucket 6  
--max_length 30  
--length_bucket_step 1.5
```

O valor ao lado de cada argumento representa seu valor padrão, isto é, qual valor eles tomariam caso não fossem passados para o software. O primeiro deles, “max_epochs” representa o número de iterações (épocas) no treinamento. Sendo 0, o software irá continuar treinando a rede neural até não haver melhora perceptível.

Os parâmetros “size”, “num_layers”, “filter_size” e “num_heads” dizem respeito a configuração da rede neural em si. Eles ditam respectivamente o tamanho de uma camada, o número de camadas utilizadas, o tamanho da camada de filtro e o número de divisões no sistema “multi-atenção”.

Os parâmetros “valid” e “test” são utilizados para escolher manualmente as subdivisões do dicionário fornecido. Isto é, qual parte das palavras é utilizada para o treino em si e qual é usada para mensurar a qualidade e retorno das camadas. Caso não sejam passados, o próprio dicionário passado é dividido automaticamente e o programa se encarrega dessas escolhas.

Por último, o resultado de uma rede neural com esse objetivo pode diferenciar muito dependendo do tamanho das palavras envolvidas. Ou seja, palavras grandes costumam ter pronúncias diferenciadas quando comparadas com palavras pequenas de mesmo conjunto de letras. Sendo assim, são divididas para o treinamento as palavras baseadas em seu tamanho, especificando pelos parâmetros “min_length_bucket”, “max_length_bucket” e “length_bucket_step” os tamanhos da divisão. O primeiro parâmetro diz ao programa qual o tamanho máximo de uma palavra para ela se enquadrar na menor divisão, e o segundo analogamente diz qual o tamanho mínimo de uma palavra para ela se encaixar na maior divisão. O terceiro parâmetro especifica o passo da divisão, e conseqüentemente quantas serão feitas.

Após o treinamento estar concluído, a última mensagem no terminal representará o resultado final, informando a taxa de perda e o número de passos dados:

```
INFO:tensorflow:loss = 0.028207932, step = 14001;
```

Com a rede neural pronta, pode-se recuperar pronúncias pelo modo interativo, em que as palavras são digitadas e a pronúncia é exibida individualmente, ou fornecendo ao programa um arquivo texto com uma palavra por linha. Em ambos os casos, a pasta contendo a rede neural é passada como argumento:

```
$g2p-seq2seq          --interactive          --model_dir
pasta_modelo_g2p
>cimo
    s i i m u
>outeiro
    o w t e j r u
```

Vemos que as pronúncias são adequadas, quando comparadas com os fonemas utilizados no dicionário original. Para passar um arquivo texto, o seguinte comando é utilizado:

```
$g2p-seq2seq --decode lista_de_palavras --model_dir
pasta_modelo_g2p --output dicionario_saida.dict
```

Com isso, todas as palavras dentro do arquivo indicado por “lista_de_palavras” irão compor um dicionário novo “dicionario_saida.dict”, que poderá ser acrescentado ao original concatenando-o formando assim o dicionário adaptado.

3.6 – Reconhecimento de voz com modelo completamente gerado pelo usuário

Há casos em que a adaptação de um modelo pronto não é suficiente. Situações em que existe uma diferença muito grande entre o tipo de áudio em que o modelo original se baseou e o tipo de áudio que deseja-se reconhecer não resultam em uma acurácia boa. Nesses casos, o ideal a se fazer é a criação de um modelo acústico próprio, utilizando os áudios adequados de treino.

Essa criação requer recursos que nem sempre estão disponíveis. São necessários uma base de dados consideravelmente grande para um treinamento de um modelo contínuo, um processo de anotação extenso, um tempo de preparo alto para otimizar os parâmetros e configurações utilizados e conhecimento da estrutura da linguagem utilizada. Todos esses elementos são indispensáveis para a criação de um modelo de boa qualidade.

Para um modelo de ditado o desenvolvedor do Sphinx recomenda um mínimo de dez horas de áudios para criação de um modelo acústico voltado para reconhecimento de um único locutor. Para uma maior eficiência com múltiplos locutores, o valor sugerido aumenta para cinquenta horas, diversificadas por duzentos locutores distintos. São valores sugeridos, porém necessários. Não é possível gerar um modelo operacional com uma base de dados significativamente menor do que o citado.

O processo de treino requer escolhas de parâmetros e argumentos que ditam características internas do software. Essas escolhas embora sejam feitas com base em grandezas sugeridas, precisam ser ajustadas e tem seu valor definitivo acertado ao longo de tentativas e erros, comparando os resultados e medindo sua eficiência. Isso demanda um tempo de trabalho, que para modelos genéricos grandes, pode alcançar um patamar de quase um mês.

O primeiro passo para a criação é organizar e dispor os arquivos corretamente (ver Figura 3.4). Um diretório para o treinamento deve ser criado, com o nome do modelo, e dentro dele deverão constar as sub-pastas “etc” e “wav”. A subpasta “wav” é onde deve-se pôr os arquivos de áudio, enquanto a subpasta “etc” contém os arquivos auxiliares. De forma análoga ao processo de adaptação, deve-se gerar o par de arquivos de extensão “.fileids” e “.transcription”, contendo respectivamente o caminho relativo dos áudios e suas transcrições.

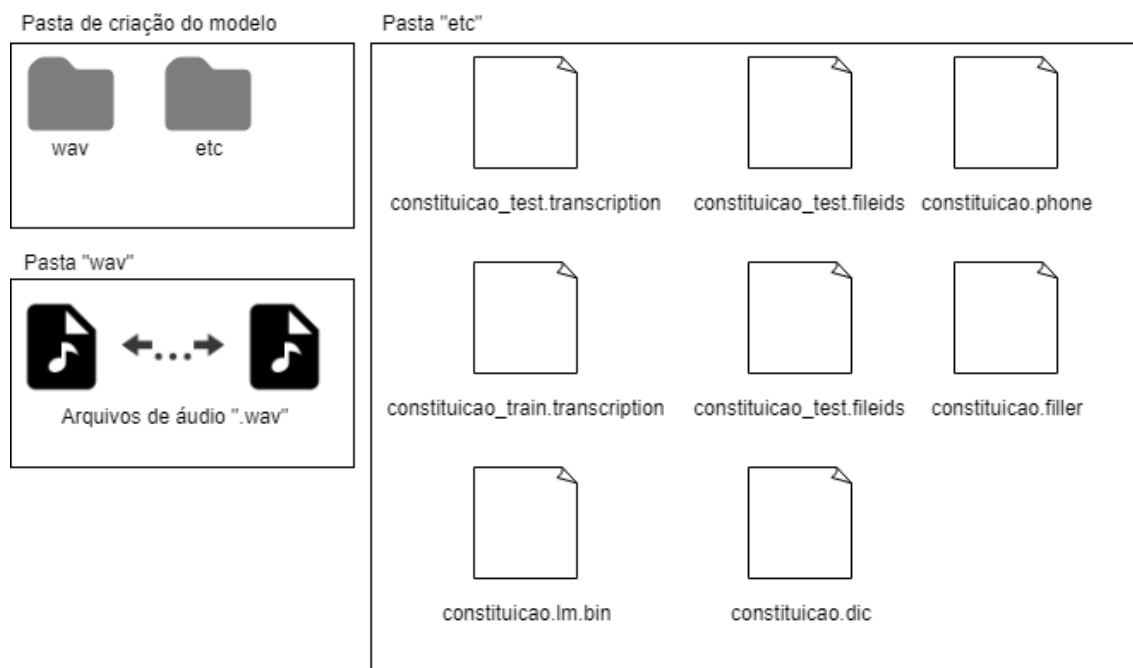


Figura 3.4 – Disposição dos componentes no diretório antes da criação do modelo acústico.

Diferente do processo de adaptação, a base de áudios precisa ser dividida em duas. Uma para o treinamento em si, e outra para fins de teste. Recomenda-se que dez por cento do conjunto de dados seja reservado para testes, embora sugira-se que esse número não ultrapasse quatro horas de gravações. Desta forma, os arquivos “.fileids” e “.transcription” devem ser divididos em dois, adicionando-se o sufixo “_test” e “_train” a seus nomes.

Também deve-se ter o modelo de linguagem utilizado, de extensão “.lm.bin”, necessário apenas para o processo de teste, e os dicionários utilizados. Um dicionário fonético padrão, de extensão “.dic”, com a representação da pronúncia de todas as palavras reconhecidas, e um dicionário de extensão “.filler” contendo a representação dos sons que não representam fala, como ruídos ou silêncios. Por último, deve-se ter um arquivo de extensão “.phone” contendo a lista dos fonemas utilizados no dicionário fonético, um por linha.

Esses últimos arquivos de dicionário devem ser vistos como um conjunto único (ver tabela 3.2). Toda a informação contida nos arquivos de áudio e que deseja-se reconhecer deve estar listada neles. Por isso, no dicionário fonético deve constar todos os vocábulos presentes, enquanto o arquivo “.filler” deve listar os sons presentes que não seriam desejados em um reconhecimento. Por último, o arquivo “.phones” é uma lista dos

fonemas utilizados, isto é, todos os sons contidos nas pronúncias dos vocábulos e também os sons indesejados.

Tabela 3.2 – Exemplificação dos arquivos de dicionário e arquivo de fonemas.

Arquivo constituicao.dic	constituicao.filler	constituicao.phone
[...]	<s> SIL	SIL
abacate a b a k a ts i	</s> SIL	a
abacaxi a b a k a sm i	<sil> SIL	b
abacaxis a b a k a sm i j s		k
abadia a b a dz i a		ts
abaeté a b a j t em		sm
[...]		[...]

Com os arquivos dispostos corretamente pode-se iniciar o processo de treinamento. O primeiro passo é executar, no diretório raiz do modelo, o seguinte comando para inicializar os arquivos e pastas necessários. Este comando recebe como argumento o nome do modelo. No caso deste experimento, será reconstruído o modelo “constituicao” original e por essa razão esse foi o nome escolhido.

```
$ sphinxtrain -t constituicao setup
```

Este comando irá gerar dentro do diretório “etc” os arquivos de configurações “feat.params” e “sphinx_train.cfg”. Esse arquivo de extensão “.cfg” deve ser modificado para adequar-se ao tipo de treinamento desejado. Muitas configurações relacionadas ao funcionamento interno estão disponíveis, porém as mais importantes são as seguintes:

```
$CFG_WAVFILES_DIR = "$CFG_BASE_DIR/wav"
```

Essa linha dita aonde são encontrados os arquivos de áudio, sendo a macro “CFG_BASE_DIR” o diretório raiz do modelo. É relativo a este caminho que as linhas

do arquivo “.fileids” serão postas. No sentido prático, o caminho definido configuração será concatenada com cada linha do arquivo “.fileids”.

```
$CFG_WAVFILE_EXTENSION = 'wav';  
$CFG_WAVFILE_TYPE = 'mswav'; # one of nist, mswav, raw
```

Essa configuração rege a extensão do arquivo de áudio. O padrão é wav, permitindo outros padrões e codificações de serem utilizadas.

```
$CFG_DICTIONARY = "$CFG_LIST_DIR/$CFG_DB_NAME.dic";  
$CFG_RAWPHONEFILE =  
"$CFG_LIST_DIR/$CFG_DB_NAME.phone";  
$CFG_FILLERDICT = "$CFG_LIST_DIR/$CFG_DB_NAME.filler";  
$CFG_LISTOFFILES =  
"$CFG_LIST_DIR/${CFG_DB_NAME}_train.fileids";  
$CFG_TRANSCRIPTFILE =  
"$CFG_LIST_DIR/${CFG_DB_NAME}_train.transcription";  
$CFG_FEATPARAMS = "$CFG_LIST_DIR/feat.params";
```

Essas seis linhas especificam o nome dos arquivos citados. O procedimento padrão é eles possuírem o mesmo nome do modelo, com as extensões especificando do que se trata. Caso este seja o caso, as configurações referentes não precisam ser mudadas. Caso desejado, pode-se arbitrar os nomes de todos os arquivos.

```
$CFG_WAVFILE_SRATE = 16000.0;
```

Essa configuração dita ao programa qual a frequência de amostragem utilizada. Só existem duas opções aceitas nesse caso. A padrão é 16kHz, e é utilizada na maior parte das aplicações. Pode-se utilizar também 8kHz para aplicações em telefones celulares.

```
$CFG_HMM_TYPE = '.cont.'; # Sphinx 4, PocketSphinx
#$CFG_HMM_TYPE = '.semi.'; # PocketSphinx
#$CFG_HMM_TYPE = '.ptm.'; # PocketSphinx (larger data
sets)
```

Acima, define-se qual o tipo de modelo a ser treinado deixando sem a marcação de comentário a linha desejada. No nosso caso trata-se do modelo contínuo, representado pela primeira linha de valor “.cont.”.

```
$CFG_FINAL_NUM_DENSITIES = 16;
$CFG_N_TIED_STATES = 1000;
```

Essas configurações tratam do procedimento de treinamento em si. O primeiro valor, “CFG_FINAL_NUM_DENSITIES” pode valer qualquer potência de 2, representando o número de gaussianas utilizadas. Isso depende do tamanho do dicionário e da quantidade de horas de áudio. O segundo valor é a quantidade de sênones utilizados no treinamento. Quanto maior o número de sênones, mais preciso será o reconhecimento, ao passo de que um número muito grande acarretará em um treinamento menos genérico, como um caso de “overfitting”. Não existe resposta certa para esses números e eles devem ser aprimorados gerando-se múltiplos modelos e verificando a melhor acurácia. O desenvolvedor como guia fornece uma tabela para servir de base:

Tabela 3.3 – Guia para escolha dos parâmetros de treino [7].

Vocabulário usado	Horas de áudio	Sênones	Gaussianas	Objetivo do modelo
20	5	200	8	Dígitos

100	20	2000	8	Comandos simples
5000	30	4000	16	Ditado básico
20000	80	4000	32	Ditado complexo
60000	200	6000	16	Transmissão de noticiário
60000	2000	12000	64	Conversas telefônicas altamente complexas

```

$CFG_QUEUE_TYPE = "Queue";
# How many parts to run Forward-Backward estimation in
$CFG_NPART = 1;
$DEC_CFG_NPART = 1; # Define how many pieces to split
decode in

```

Estas linhas configuram a possibilidade do processamento paralelo em múltiplas threads. É ideal para processadores com mais de um núcleo, comum nos dias de hoje. Para configurar desta maneira, a primeira macro deverá receber o valor “Queue::POSIX”, enquanto a segunda especificará o número de threads utilizado no treinamento. A última macro também determina o número de threads, porém utilizados no reconhecimento de teste que ocorre ao final do treinamento.

```

$DEC_CFG_DICTIONARY =
"$CFG_BASE_DIR/etc/$CFG_DB_NAME.dic";
$DEC_CFG_FILLERDICT =
"$CFG_BASE_DIR/etc/$CFG_DB_NAME.filler";
$DEC_CFG_LISTOFFILES =
"$CFG_BASE_DIR/etc/${CFG_DB_NAME}_test.fileids";

```

```

$DEC_CFG_TRANSCRIPTFILE =
"$CFG_BASE_DIR/etc/${CFG_DB_NAME}_test.transcription";
$DEC_CFG_RESULT_DIR      = "$CFG_BASE_DIR/result";
$DEC_CFG_LANGUAGEMODEL   =
"$CFG_BASE_DIR/etc/${CFG_DB_NAME}.lm.DMP";

```

Analogamente ao processo de treino, essas configurações determinam o nome dos arquivos de teste. Com os arquivos possuindo o mesmo nome do modelo, nenhuma alteração deve ser feita neste aspecto. A única alteração que deve ser feita é a extensão do modelo de linguagem para o formato binário, mais moderno e eficaz, “.lm.bin”. A extensão antiga não é mais utilizada.

Feitas todas essas configurações, pode-se executar o processo que inicia o treino. Ele deve ser executado no diretório raiz do modelo:

```
$sphinxtrain run
```

A partir desse momento o processamento começa e os scripts internos funcionam alertando quaisquer erros e sucessos. No final, o processo de decodificação se inicia automaticamente, tentando reconhecer o conjunto de testes utilizando o modelo treinado. Caso desejado, esse processo de teste pode ser feito manualmente utilizando o seguinte comando:

```
$sphinxtrain -s decode run
```

Todo o processo de treino pode demorar de uma hora até um mês, dependendo do tamanho da base de dados e das configurações escolhidas. O modelo acústico pronto pode ser encontrado dentro da pasta relativa “model_parameters/constituicao_4000”, este número representando o número de sênones utilizado.

Além do modelo acústico, pode-se gerar os outros elementos de um modelo de reconhecimento. A criação de um dicionário fonético sem uma base inicial é bem difícil. Algumas alternativas existem, como scripts para busca em páginas que fornecem a pronúncia de palavras e traduzem em fonemas definidos pelo usuário. Outra opção é criar regras específicas de identificação de fonemas em palavras. Por exemplo, em nosso idioma, a letra acentuada “É” sempre irá gerar o mesmo fonema. De qualquer forma, isso não é funcional para um dicionário completo, e deve ser feito apenas para os primeiros milhares de verbetes, permitindo assim sua expansão utilizando o método apresentado anteriormente neste trabalho.

A criação de um modelo de linguagem pode ser feita de múltiplas maneiras. Primeiramente, pode-se criar arquivos com listas de palavras-chave. Desta forma, arbitramos a probabilidade de certas expressões e combinações serem ditas, privilegiando combinações de palavras mais comuns. Isso é útil para modelos de reconhecimento de simples comandos e impraticável de ser feito manualmente para todas as palavras de um idioma. Um exemplo deste tipo de modelo de linguagem segue:

```
oh mighty computer /1e-40/  
hello world /1e-30/  
other phrase /1e-20/
```

Outra possibilidade aceita pelo Sphinx é a utilização do Java Speech Grammar Format, abreviado por JSGF [21]. Este formato possibilita a criação de um arquivo de gramática, onde regras são geradas e escritas manualmente para dado idioma. Por exemplo, após um “Bom dia” comumente um vocativo ou nome se sucede. Novamente, tal formato de modelo de linguagem é impraticável para o reconhecimento de ditado. Um exemplo de regra gramatical dado acima segue:

```
#JSGF V1.0;  
grammar hello;  
public <saudacao> = (bom dia|olá) (jose|maria|joão);
```

A terceira forma de geração é uma solução estatística, onde se fornece ao software uma massa de texto grande o suficiente para que probabilidades e correlações entre palavras diferentes possam ser consideradas corretas. O software por sua vez extrai estatísticas dessa massa de textos e gera o modelo de linguagem em formato binário. Essa é a opção recomendada para modelos de reconhecimento de ditado. Além de não necessitar de decisões arbitrárias, devido ao seu processo de geração, possibilita toda combinação de palavras presentes no texto, e não apenas as que ocorreram. Isto é, existindo “vinte e um” e “trinta e dois” no texto fornecido, o software contabiliza também a possibilidade da combinação “vinte e dois”.

Para a criação desse modelo de linguagem a massa de textos deve ser previamente preparada. Todas as palavras devem ser limpas, sem abreviações, com números escritos por extenso. O desenvolvedor sugere para a obtenção desses dados a utilização de legendas de filmes ou transcrições de transmissões televisivas, por exemplo. Para começar, precisamos copiar os executáveis necessários gerados pela instalação do “cmuclmtk”.

```
$cp cmuclmtk-0.7/src/programs/text2wfreq .  
$cp cmuclmtk-0.7/src/programs/wfreq2vocab .  
$cp cmuclmtk-0.7/src/programs/text2idngram .  
$cp cmuclmtk-0.7/src/programs/idngram2lm .
```

Em seguida, o seguinte comando é utilizado, criando uma lista de palavras representando o vocabulário presente na massa de textos. Após esse comando, é recomendado remover manualmente palavras indesejadas, como números ou siglas que por ventura estavam presentes.

```
$/text2wfreq < texto.xml | ./wfreq2vocab >texto.vocab
```

Em seguida, basta rodar os seguintes comandos para finalização do processo:

```
$. /text2idngram -vocab texto.vocab -idngram  
texto.idngram < texto.xml  
$. /idngram2lm -vocab_type 0 -idngram texto.idngram -  
vocab texto.vocab -arpa texto.lm
```

O modelo de linguagem já está pronto porém é fornecido em um formato que não é mais utilizado, sendo necessária sua conversão para o formato binário. Para isso, utiliza-se o comando:

```
$sphinx_lm_convert -i texto.lm -o texto.lm.bin
```

O modelo de linguagem “texto.lm.bin” está terminado e pronto para uso.

3.7 – Resultados obtidos

O modelo de reconhecimento pronto do grupo FalaBrasil, denominado “Constituição”, apresenta inicialmente uma taxa de erros baixa (WER=22,2%), como mostra a saída final do processo de treino:

```
MODULE: DECODE Decoding using models previously trained
Aligning results to find error rate
SENTENCE ERROR: 100.0% (155/155) WORD ERROR RATE: 22.2%
(1938/8737)
```

Obtém-se uma taxa de erro alta quando avaliadas as sentenças como um todo, isto é, nenhum arquivo de áudio foi transcrito inteiramente correto. Esta circunstância é comum em modelos de ditado, visto que essa métrica é mais utilizada para modelos de comandos simples, cujos áudios transcritos são compostos de poucas palavras. Além disso, é importante observar que a taxa de erros de 22% é um valor bom, porém vale lembrar que essa acurácia se baseia em transcrições de um conjunto de testes construído nos mesmos moldes do conjunto em que se foi treinado.

Para o experimento de adaptação, foram gravados dois áudios da leitura de um poema de Fernando Pessoa. Quando o modelo original foi testado com esses áudios, ficou evidente que a transcrição resultante estava longe do esperado, como demonstrado pela WER igual a 89,32% em ambos os casos:

Tabela 3.4 – Transcrição da primeira leitura utilizando o modelo original comparada com o texto original.

Texto Original (Lido)	Transcrição da leitura pausada
Da minha aldeia vejo quanto da terra se pode ver do Universo...	damião ver vejo curta terras e oito vive vez
Por isso a minha aldeia é tão grande como outra terra qualquer,	foi dessa em vez da um crítico multa terra com o quer
Porque eu sou do tamanho do que vejo	porque os outra em o que vejo

<p>E não do tamanho da minha altura...</p> <p>Nas cidades a vida é mais pequena Que aqui na minha casa no cimo deste outeiro.</p> <p>Na cidade as grandes casas fecham a vista à chave, Escondem o horizonte, empurram o nosso olhar para longe de todo o céu, Tornam-nos pequenos porque nos tiram o que os nossos olhos nos podem dar, E tornam-nos pobres porque a nossa única riqueza é ver.</p>	<p>e não tá nem dá me algo</p> <p>na cidade a viver nas kennedy e que ninguém casa do sino vinte ufrir na cidade hagi vinte que as graves pra achar de onde rua visão em ti e impor novas olhar para longe do céu por gíbi que age quer ágil que nós podem dar o ibama no de nove de por que a nação em que ele quesito rever</p>
--	---

O valor de erro obtido foi muito superior ao esperado, impossibilitando qualquer tipo de compreensão para quem lê. No segundo áudio, onde foi feita uma leitura mais acelerada, o resultado é semelhante:

Tabela 3.5 – Transcrição da segunda leitura utilizando o modelo original comparada com o texto original.

Texto original (Lido)	Transcrição da leitura acelerada
<p>Da minha aldeia vejo quanto da terra se pode ver do Universo...</p> <p>Por isso a minha aldeia é tão grande como outra terra qualquer, Porque eu sou do tamanho do que vejo E não do tamanho da minha altura...</p> <p>Nas cidades a vida é mais pequena Que aqui na minha casa no cimo deste outeiro.</p> <p>Na cidade as grandes casas fecham a vista à chave,</p>	<p>da minha mulher vejo conta terras e oito veto</p> <p>viver leves caminhão ver um crítico multa ver com o quer porque o fruto da mãe que vende na nota muita mil dos</p> <p>nas cidades em vídeo nas que em casa massimo vígio veio na cidade gigante que as de medição indy urnas ligar para o registro do céu por gíbi que há de chico ricos ali de maringá</p>

Escondem o horizonte, empurram o nosso olhar para longe de todo o céu, Tornam-nos pequenos porque nos tiram o que os nossos olhos nos podem dar, E tornam-nos pobres porque a nossa única riqueza é ver.	ituano di pode por canais liga e quinze via
---	--

Neste caso o resultado é semelhante, não se percebe muitas semelhanças com o texto original. Aqui é calculada uma WER de 88,34%, valor próximo do anterior e ainda muito alto. Quando testado com o modelo adaptado com os versos do próprio poema, o resultado é diferente, onde pode-se praticamente reconhecer o texto original quando se lê. O resultado da transcrição dos dois áudios utilizando o modelo adaptado segue:

Tabela 3.6 – Transcrições das duas leituras obtidas utilizando o modelo adaptado.

Transcrição da leitura pausada	Transcrição da leitura acelerada
da minha aldeia vejo quanto da terra se pode ver do universo por isso a minha aldeia tão grande como outra terra qualquer porque os show do tamanho que vejo e não do tamanho da minha altura nas cidades a vida mas pequena que que na minha casa nos cinco vejo teve na cidade às grandes casas fecham a vista a chave com vigor nizan em pouco ou nosso lhe ar para longe do que se ao tornam nos pequenos porque nos tiram que nossos olhos luz podem dar e tornam nos pobres porque a nossa unica riqueza rever	da minha aldeia vejo quanto da terra se pode ver do universo por isso a minha aldeia do grande como outra ter qualquer porque o show do tamanho que vejo não do tamanho da mil por nas cidades ave demais pequena quer que na minha casa nesse cotejo ver na cidades grandes casas fecham a vista chaves com aquele distante urnas olhar para o rio do céu torno dos pequenos porque nos tiram que nossos olhos podem dar e torno dos pobres porque a nossa única riqueza ver

No caso da leitura pausada, a taxa de erros foi mais de três vezes menor, com um valor de 28.15%. Analogamente, na segunda leitura, o valor foi de 40,77%, ainda menor que a metade do valor encontrado sem adaptação.

Isso demonstra que o processo de adaptação é muito eficaz quando se deseja especificar um objetivo para um modelo. Neste caso por exemplo, reconhecimento da própria voz ou de poemas deste estilo. Todo o processo é de fácil realização e não demanda esforço e nem um material muito extenso, visto que neste caso foram utilizados áudios com menos de 2 minutos de duração total.

Para o experimento de criação do modelo acústico completamente novo, foi criado um modelo acústico baseado na união das bases de dados disponíveis. Como o conjunto de áudios de interceptações telefônicas não continha as transcrições, foi necessário o fazer manualmente. Por tratar-se de um trabalho extenso, apenas uma fração de 42 minutos deste foi transcrita. Para efeito de registro, vale chamar a atenção que cada 1 minuto de áudio transcrito exigiu 7 minutos de trabalho por um agente humano. O primeiro conjunto formado baseou-se na junção dos arquivos de áudio do modelo “Constituição”, com os obtidos no VoxForge e esta fração dos áudios de interceptações telefônicas.

Para a escolha dos parâmetros, foram criados modelos com 16 e 32 gaussianas, utilizando-se dos valores de 1000, 2000 e 4000 sênones, resultando em um total de 6 modelos fechados. Essa escolha de parâmetros foi baseada na tabela 3.3 de sugestões para construção de modelos acústicos. Em adição a esses modelos criados, foi também feita a adaptação do modelo Constituição com o conjunto de interceptações telefônicas. Como o processo de adaptação utiliza as mesmas configurações do modelo original, não é permitida a variação dos parâmetros.

Em relação ao conjunto em que esses modelos serão testados, foram transcritos como referência mais 5 minutos de áudios de interceptações telefônicas. Essa transcrição é utilizada pelo decodificador para comparar com o resultado obtido pelos modelos e mensurar sua acurácia.

O pacote de ferramentas utilizado na criação conta também com ferramentas para medir a taxa de erros de modelos. Para tal, deve-se rodar um comando de preparação que inclui como argumentos arquivos “.fileids” e “.transcriptions” do conjunto de testes, além do modelo acústico em questão. O comando utilizado segue abaixo:

```
$pocketsphinx_batch -adcin yes -cepdire wav -cepext .wav
-ctl etc/modelo_test.fileids -lm etc/language-model.lm.bin
-dict etc/modelo.dic -hmm pasta_do_modelo_testado -hyp
hyp/arquivo_saida.hyp
```

Neste comando o argumento “-cepdire” indica ao programa o caminho para os arquivos de áudio. O arquivo de saída de extensão “.hyp” é o que vai ser utilizado para a comparação de strings com o resultado. Em seguida, precisamos utilizar o script em Perl “word_align.pl” contido na pasta de instalação do pacote SphinxTrain. Portanto, copie-se o mesmo para o diretório local e em seguida realiza-se a execução utilizando como argumentos o arquivo de transcrições e o modelo acústico a ser treinado:

```
$cp
/usr/local/lib/sphinxtrain/scripts/decode/word_align.pl .
$perl word_align.pl etc/modelo_test.transcription
arquivo_saida.hyp
```

Esse comando tem como saída a comparação do resultado obtido com o esperado para cada arquivo de áudio, e no final a medida total dos coeficientes desejados:

```
TOTAL Words: 518 Correct: 119 Errors: 399
TOTAL Percent correct = 22.90% Error = 77.10% Accuracy
= 22.90%
TOTAL Insertions: 0 Deletions: 137 Substitutions: 262
```

Com os modelos treinados foram encontrados os resultados apresentados na Tabela 3.7. Nas configurações referentes ao modelo adaptado não há taxa de erro nos valores de 2000 e 4000 sênones porque o processo de adaptação não permite modificação

nos parâmetros, isto é, deve-se adaptar utilizando os mesmos valores utilizados pelo modelo original, que no caso foi de 1000 sênones, com 16 gaussianas.

Tabela 3.7 – Resultado dos modelos gerados medido pela taxa de erros de palavras, WER.

	1000 sênones	2000 sênones	4000 sênones
Modelo total; 32 gaussianas	WER=95.10%	WER=90.50%	WER=92.30%
Modelo total; 16 gaussianas	WER=88.20%	WER=85.10%	WER=90.80%
Modelo adaptado; 16 gaussianas	WER=77.10%	(Não aplicável)	(Não aplicável)

Todos os valores de WER obtidos foram altos, isto é, foram encontrados resultados ruins, principalmente quando comparados por exemplo ao obtido com a criação do Constituição, que teve como valor de WER 22.2%. Isto deve-se provavelmente ao fato de que o conjunto de testes utilizado era composto apenas por áudios de interceptações telefônicas, enquanto que no conjunto de treinamento esses áudios de interceptações somavam apenas 42 minutos de um total de 13 horas treinadas. Sendo assim, dentre os áudios para treinamento apenas pouco mais de 5% era oriundo da mesma origem que o conjunto de testes.

Neste sentido, pode-se perceber que o melhor modelo acústico realizado foi o referente à adaptação do modelo Constituição, o que demonstra que essa é a melhor escolha quando não se tem um conjunto de treino suficientemente grande. Ao olharmos estritamente os resultados de criação, podemos perceber também que os melhores parâmetros escolhidos foram 16 gaussianas com 2000 sênones.

O processo de treino do modelo acústico gerado neste trabalho durou entre 2 horas e 2 horas e meia, dependendo da escolha de parâmetros. Valores menores para os parâmetros acarretaram em um treino ligeiramente mais rápido, porém essa variação não foi maior do que os 30 minutos citados, isto é, aproximadamente 20% da duração.

A maior dificuldade encontrada para a criação de um modelo é a obtenção e preparação dos dados. É necessária uma quantidade alta de horas de áudios, que para serem transcritos manualmente, requerem no mínimo um tempo sete vezes maior. Sobre

a utilização das ferramentas nenhum problema foi encontrado e o refinamento dos parâmetros foi feito sem dificuldades.

Para fins de comparação com API's online foram escolhidos dois áudios curtos em que nosso modelo treinado retornava uma transcrição boa e outra ruim. Esses áudios constavam na base de dados de interceptações telefônicas e não foram utilizados no processo de construção do modelo acústico. Utilizando então a API de reconhecimento de voz do Google foram comparadas as performances da transcrição, com o resultado que segue na tabela 3.8:

Tabela 3.8 – Comparação da WER entre o modelo adaptado e a API do Google em dois áudios de nossa base de dados.

	API Google	Modelo adaptado da Constituição
Áudio 1	WER=26.2%	WER=26.2%
Áudio 2	WER=68,4%	WER=92,1%

É importante destacar que os valores encontrados para a WER no primeiro áudio foram iguais por uma mera coincidência de números. Ambos os áudios escolhidos não eram longos, possuindo como duração um valor próximo de vinte segundos. Consequentemente, como o número total de palavras ditas é o denominador no cálculo da WER, existe um conjunto pequeno de valores finais possíveis, tornando coincidências como essa mais comuns.

Podemos perceber então que em certos áudios de nossa base de dados a taxa de erros de nosso modelo assemelha-se com a do Google. Embora em outros nossa taxa de erros possa ser considerada altíssima, a API do Google também demonstrou um resultado ruim.

Capítulo 4

Conclusões

Neste trabalho todos os objetivos iniciais foram cumpridos. Foi aprendido e demonstrado o funcionamento das ferramentas e tecnologias em questão, enaltecendo os detalhes e procedimentos nas decisões tomadas. Qualquer tentativa de recriação desses experimentos deverá ocorrer sem problemas e com plena capacidade de melhorias.

Tira-se como conclusão que o uso das ferramentas citadas está bem documentado oficialmente e não cria problemas para quem desejar utilizá-las. Embora o software ainda esteja em desenvolvimento e algumas partes dos processos percorridos pudessem ser automatizadas, qualquer erro encontrado é sempre bem claro e explicita bem qual o problema encontrado, para facilitar resolução do usuário.

Como trabalho futuro pode-se pensar em criar uma base de áudios transcritos maior para melhoria dos resultados encontrados. Em adição a isso, é possível a união de outras tecnologias para melhoria ou facilitação dos fluxos envolvidos, como por exemplo o uso de corretores ortográficos ou scripts para automatização de múltiplos treinos. Outra alternativa é alterar a abordagem de transcrição, experimentando técnicas de machine learning. Qualquer que seja a forma escolhida para a melhoria dos resultados, é necessário obter um conjunto maior de áudios e transcrições para nosso idioma, tendo sido este o maior empecilho encontrado na realização deste trabalho.

Bibliografia

- [1] RABINER, Lawrence R. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, Proceedings of the IEEE, v. 77, n. 2, February 1989.
- [2] MELLON, C. Basic concepts of speech recognition.
<https://cmusphinx.github.io/wiki/tutorialconcepts>, (Acesso em 27 outubro de 2018).
- [3] HUNT, Melvyn J. *Figures of Merit for Assessing Connected Word Recognisers*, Speech Communication, v. 9, pp. 239-336, August 1990).
- [4] IBM. Reaching new records in speech recognition.
<https://www.ibm.com/blogs/watson/2017/03/reaching-new-records-in-speech-recognition/>, (Acesso em 27 outubro de 2018).
- [5] VENTUREBEAT. Google's speech recognition technology now has a 4.9% word error rate. <https://venturebeat.com/2017/05/17/googles-speech-recognition-technology-now-has-a-4-9-word-error-rate/>, (Acesso em 27 outubro de 2018).
- [6] MICROSOFT. Microsoft researchers achieve new conversational speech recognition milestone. <https://www.microsoft.com/en-us/research/blog/microsoft-researchers-achieve-new-conversational-speech-recognition-milestone/>, (Acesso em 27 outubro de 2018).
- [7] MELLON, C. Tutorial para treinamento de um modelo acústico.
<https://cmusphinx.github.io/wiki/tutorialam>, (Acesso em 27 outubro de 2018).
- [8] KĚPUSKA, V. *Comparing Speech Recognition Systems (Microsoft API, Google API And CMU Sphinx)*, Int. Journal of Engineering Research and Application, v. 7, pp.20-24, March 2017.
- [9] MELLON, C. Open Source Speech Recognition Toolkit.
<https://cmusphinx.github.io>, (Acesso em 27 outubro de 2018).
- [10] CARDOSO, Bruno Lima. *Sistema De Reconhecimento De Fala Baseado Em PocketSphinx Para Acessibilidade Em Android*, M. Sc. Dissertation, Universidade Federal do Rio de Janeiro, Outubro de 2015.
- [11] MELLON, C. Repositórios do CMUSphinx. <https://github.com/cmusphinx>, (Acesso em 27 outubro de 2018).
- [12] MELLON, C. Fórum de ajuda para as ferramentas do pacote CMUSPHINX.
<https://sourceforge.net/p/cmusphinx/discussion/help>, (Acesso em 27 outubro de 2018).

- [13] UFPA, Laboratório de Processamento De Sinais. Sobre o Grupo FalaBrasil. <http://labvis.ufpa.br/falabrasil>, (Acesso em 27 outubro de 2018).
- [14] MELLON, C. VoxForge: Free speech corpus and acoustic model repository for open source speech recognition engines. <http://www.voxforge.org>, (Acesso em 27 outubro de 2018).
- [15] MELLON, C. Repositório de arquivos SourceForge do CMUSPHINX. <https://sourceforge.net/projects/cmusphinx/files/>, (Acesso em 27 outubro de 2018).
- [16] BAGWELL, C. HomePage do SoX - Sound eXchange. <http://sox.sourceforge.net/>, (Acesso em 27 outubro de 2018).
- [17] MAZZONI, D. HomePage do Audacity. <https://www.audacityteam.org/>, (Acesso em 27 outubro de 2018).
- [18] UFPA, Laboratório de Processamento De Sinais. Página de downloads do Grupo FalaBrasil. <http://labvis.ufpa.br/falabrasil/downloads/>, (Acesso em 27 outubro de 2018).
- [19] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. Codes for the representation of names of languages, ISO 639-4:2010, 2010.
- [20] PESSOA, F. Poemas de Alberto Caeiro. 10 ed, Lisboa: Ática, 1993.
- [21] INTEL. Documentação acerca do Java Speech Grammar Format. https://software.intel.com/sites/landingpage/realSense/camera-sdk/v1.1/documentation/html/doc_speech_specifying_commands_using_java.html, (Acesso em 27 outubro de 2018).