DANIEL MASSAMI TAKIZAWA
TIAGO DAL PAI FABBRI

SISTEMA AUTOMÁTICO DE RECONHECIMENTO DE VOZ

São Paulo
2009

DANIEL MASSAMI TAKIZAWA
TIAGO DAL PAI FABBRI

SISTEMA AUTOMÁTICO DE RECONHECIMENTO DE VOZ

Trabalho de Conclusão de Curso de graduação em Engenharia

Orientador: Prof. Miguel Arjona Ramírez

São Paulo
2009

RESUMO

Este trabalho apresenta um projeto de um sistema automático de reconhecimento de voz para utilização em computadores baseados no sistema operacional Microsoft Windows. A solução será baseada em Modelos Ocultos de Markov, um método estatístico de caracterização de padrões. Para a implementação será utilizada a biblioteca HTK, que pode ser obtida gratuitamente, e o ambiente de desenvolvimento Microsoft Visual Studio.

Palavras-chave: Reconhecimento de voz. Modelos Ocultos de Markov.

ok

ABSTRACT

This work presents the project of an automatic speech recognition system for computers running the Microsoft Windows operating system. The solution is based on Hidden Markov Models, a statistical method of characterizing patterns. For the realization, the Hidden Markov Model Toolkit and the Microsoft Visual Studio development environment will be used.

Keywords: Speech recognition. Hidden Markov Models.

SUMÁRIO

INTRODUÇÃO

Com a popularização dos sistemas computacionais, torna-se necessária a implementação de interfaces homem-máquina cada vez mais intuitivas e um sistema comandado por voz seria a forma mais natural para esta interface.

Vários métodos já foram desenvolvidos para o problema do reconhecimento de voz e um método bastante utilizado é o método baseado em Modelos Ocultos de Markov, que será o utilizado neste trabalho.

1 PROPOSTA DO PROJETO

1.1 DEFINIÇÃO DO PROBLEMA

O problema que será tratado neste projeto é o de se implementar um sistema de reconhecimento de voz para ser utilizado como interface homem-máquina de programas computacionais em Sistema Operacional Windows[1].

A técnica de reconhecimento de voz que será utilizada é a baseada em Modelos Ocultos de Markov (em inglês *Hidden Markov Models - HMM*) e a implementação será com o *Hidden Markov Model Toolkit (HTK)*, uma ferramenta para implementar e manipular HMMs voltada para o reconhecimento de voz. O HTK é uma ferramenta de uso gratuito para fins acadêmicos e de pesquisa.

1.2 OBJETIVOS

O principal objetivo deste projeto é implementar um sistema de reconhecimento de voz dependente de locutor que será a interface homem-máquina de um programa computacional.

Como o foco do projeto está na implementação do sistema de reconhecimento, a sua demonstração de funcionalidade será na interface de um programa simples: uma calculadora em que todas as entradas de dados e requisições de operações serão através da interface por reconhecimento de voz.

1.3 RESULTADOS A SEREM ALCANÇADOS

O produto final deste projeto será um programa computacional para Sistema Operacional Windows que implementa uma calculadora científica em que a interface homem-máquina será totalmente por comandos de voz.

O sistema de reconhecimento será dependente de locutor e o sistema deverá ser capaz de reconhecer todos os comandos válidos com uma taxa de acerto superior a 80 %.

---

[1] Windows é marca registrada da Microsoft Corporation

1.4 INDICADORES DE DESEMPENHO

A medida de desempenho do sistema de reconhecimento de voz é dada pela taxa de acertos em relação aos comandos dados ao sistema.

Neste projeto serão feitos testes repetitivos de todos os comandos válidos para comprovar que a taxa de acertos do sistema está de acordo com a especificada.

1.5 DESCRIÇÃO DAS PRINCIPAIS ATIVIDADES

As principais atividades na execução deste projeto são: estudo da ferramenta HTK, gravação dos exemplos de treinamento para os HMMs de cada um dos comandos, implementação do sistema de reconhecimento com as ferramentas do HTK e testes de desempenho.

1.6 INFRAESTRUTURA/MEIOS NECESSÁRIOS

Para implementação deste projeto são necessários um computador com o Sistema Operacional Windows, o ambiente de desenvolvimento Microsoft Visual Studio para compilar as ferramentas (programas) do HTK e um ambiente de desenvolvimento para implementar a calculadora e utilizar o sistema de reconhecimento de voz como interface homem-máquina, que para este projeto será o próprio Visual Studio.

Para gravação dos arquivos sonoros dos exemplos de treinamento será utilizado o programa WaveSurfer, onde também serão feitas as transcrições dos arquivos no formato exigido pelo HTK. O WaveSurfer também é um programa gratuito.

1.7 VIABILIDADE DO PROJETO

A ferramenta HTK é de uso gratuito para fins acadêmicos e de pesquisa, portanto para a sua utilização neste projeto não haverá custos. O Visual Studio foi obtido gratuitamente através do programa acadêmico da USP com a Microsoft. Assim, este projeto terá um custo total praticamente nulo.

Tanto o HTK como o Visual Studio são ferramentas de alta complexidade que possuem documentações bastante completas; a documentação do HTK é obtida gratuitamente na Internet e a documentação sobre o Visual Studio está disponível na biblioteca de Engenharia Elétrica da Escola Politécnica.

1.7 CRONOGRAMA DE ATIVIDADES

Semana	Principal atividade relacionada ao projeto
24/8 - 28/8	Estudo do HTK
31/8 - 4/9	Estudo do HTK
7/ - 11/9	Estudo do HTK
28/9 - 2/10	Gravação dos comandos básicos e implementação do sistema
5/10 - 9/10	Implementação e testes do sistema básico
12/10 - 16/10	Implementação e testes do sistema básico
2/11 - 6/11	Estudo das configurações para entrada de áudio em tempo real (*direct audio input*)
9/11 - 13/11	Gravação dos comandos restantes e implementação do sistema de reconhecimento completo
15/11 - 20/11	Implementação e testes do sistema completo
23/11 - 27/11	Testes e elaboração da documentação
30/11 - 4/12	Testes finais
7/12 - 11-12	Apresentação do programa em funcionamento

2 MODELOS OCULTOS DE MARKOV

Os processos do mundo real geralmente produzem saídas observáveis que podem ser classificadas como sinais. Os sinais podem ser de natureza discreta (por exemplo, caracteres de um alfabeto finito) ou contínua (por exemplo, amostras de voz). A fonte do sinal pode ser estacionária (isto é, suas propriedades não variam com o tempo) ou não-estacionário (isto é, suas propriedades variam com o tempo). Os sinais podem ser puros (isto é, originados de uma única fonte) ou podem ser corrompidos por outras fontes (por exemplo, ruído) ou por distorções na transmissão, reverberação, etc.

Um problema de fundamental interesse é a caracterização dos sinais em termos de modelos. As principais razões para a utilização de modelos de sinais são prover uma base para a descrição teórica de um sistema de processamento de sinais (por exemplo, um sistema de redução de ruído) e permitir a obtenção de informações sobre o sinal sem a necessidade de ter a fonte disponível, o que é especialmente importante quando o custo da obtenção de sinais da fonte real é alto.

Existem várias escolhas possíveis para qual modelo de sinal será utilizado. Podemos dividir os modelos sinais em duas grandes classes: a classe dos modelos determinísticos e a classe dos modelos estatísticos. Modelos determinísticos geralmente exploram propriedades específicas conhecidas do sinal, por exemplo, que o sinal é uma senóide, um soma de exponenciais, etc. Nestes casos, a especificação do modelo do sinal é direta; é somente necessário determinar (estimar) os parâmetros do modelo (por exemplo, frequência, amplitude, etc.). A segunda classe de modelos de sinais é a classe de modelos estatísticos, que caracterizam apenas as propriedades estatísticas do sinal. Exemplos de modelos estatísticos são processos Gaussianos, processos de Markov e processos ocultos de Markov.

Para a aplicação em reconhecimento de voz será utilizado um modelo estatístico, os modelos ocultos de Markov.

2.1 PROCESSOS DE MARKOV DE TEMPO DISCRETO

Considere um sistema que pode ser descrito em qualquer instante como estando em um de um conjunto de N estados distintos indexados $\{1,2,...,N\}$ como na

Figura 2.1 (onde $N = 5$ por simplicidade). O sistema muda de estado (possivelmente para o mesmo estado) em intervalos de tempo discretos de acordo com um conjunto de probabilidades associadas a cada estado. Os instantes de tempo em que ocorrem mudanças de estado serão denotados por $t = 1, 2,...$, e o estado no tempo t será denotado q_t.

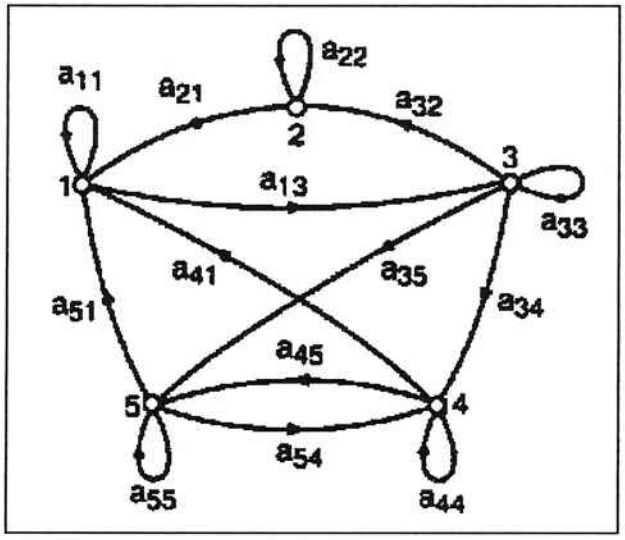

Figura 2.1 - Uma cadeia de Markov com 5 estados e algumas transições de estado selecionadas

Uma descrição completa do sistema acima exigiria, em geral, a especificação do estado atual (no instante t) e os estados precedentes. Para o caso especial de uma cadeia de Markov de tempo discreto e primeira ordem, a dependência probabilística é truncada no estado anterior, isto é,

$$P[q_t = j \mid q_{t-1} = i, q_{t-2} = k, ...] = P[q_t = j \mid q_{t-1} = i] \quad (2.1)$$

Além disso, serão considerados apenas processos em que o lado direito da equação (2.1) é independente do tempo, com um conjunto de probabilidades de transição de estado da forma

$$a_{ij} = P[q_t = j \mid q_{t-1} = i], \quad 1 \leq i,j \leq N \quad (2.2)$$

com as seguintes propriedades:

$$a_{ij} \geq 0 \quad (2.3)$$

$$\sum_{j=1}^{N} a_{ij} = 1 \quad (2.4)$$

O processo estocástico apresentado acima pode ser chamado um processo de Markov observável pois a saída do processo é o conjunto de estados em cada instante de tempo, em que cada estado corresponde a um evento físico (observável). Para fixar as ideias, considere um modelo de Markov com 3 estados para o tempo. Assume-se que uma vez ao dia (por exemplo, ao meio-dia), o tempo é observado da seguinte maneira:

Estado 1: chuva

Estado 2: nublado

Estado 3: ensolarado

Postulamos que o tempo em um dia t está em um dos três estados acima e que a matriz A de probabilidades de transição de estado é:

$$A = \{a_{ij}\} = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$$

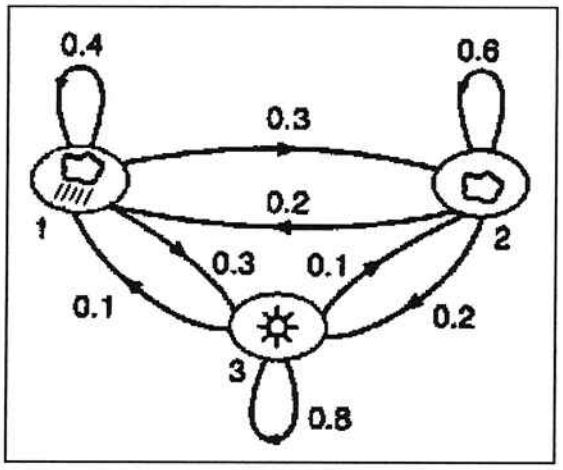

Figura 2.2 - Modelo de Markov para o tempo

Dado que no dia 1 ($t = 1$) o tempo está ensolarado (estado 3), pode-se fazer a questão: Qual a probabilidade (de acordo com o modelo apresentado) de que o tempo para os próximos 7 dias seja "sol-sol-chuva-chuva-sol-nublado-sol"? Mais formalmente, definimos a sequência de observações $O = \{3, 3, 3, 1, 1, 3, 2, 3\}$ correspondendo a $t = 1, 2, \ldots, 8$ e queremos determinar a probabilidade de O dado o modelo, $P(O|\text{Modelo})$, que pode ser calculada da seguinte forma:

$$P(O|\text{Modelo}) = P[3] \cdot P[3|3] \cdot P[3|3] \cdot P[1|3] \cdot P[1|1] \cdot P[3|1] \cdot P[2|3] \cdot P[3|2]$$
$$= \pi_3 \cdot a_{33} \cdot a_{33} \cdot a_{31} \cdot a_{11} \cdot a_{13} \cdot a_{32} \cdot a_{23}$$
$$= 1 \cdot 0.8 \cdot 0.8 \cdot 0.1 \cdot 0.4 \cdot 0.3 \cdot 0.1 \cdot 0.2$$
$$= 1.536 \times 10^{-4}$$

onde a notação

$$\pi_i = P[q_1 = i], \quad 1 \leq i \leq N \quad (2.5)$$

indica as probabilidades para o estado inicial. Como no exemplo o estado inicial era conhecido (estado 3), $\pi_3 = 1$.

Outra questão que pode ser respondida com o modelo é: Dado que o modelo está em um estado conhecido, qual é a probabilidade de ele permanecer neste estado por exatamente d dias? Em outras palavras, queremos saber a probabilidade da sequência de observações ser $O = \{i, i, \ldots, i, j \neq i\}$, correspondendo a $t = 1, 2, \ldots, d, d+1$, dado o modelo,

$$P(O|\text{Modelo}, q_1 = i) = (a_{ii})^{d-1}(1 - a_{ii}) = p_i(d) \quad (2.6)$$

O valor $p_i(d)$ é a função densidade de probabilidade discreta de duração d no estado i, que é uma característica da duração do estado numa cadeia de Markov. Baseando-se em $p_i(d)$, pode-se calcular diretamente o número de observações (duração) em um estado, iniciando naquele estado, como

$$\bar{d}_i = \sum_{d=1}^{\infty} d\, p_i(d) \quad (2.7)$$
$$= \sum_{d=1}^{\infty} d(a_{ii})^{d-1}(1 - a_{ii}) = \frac{1}{1 - a_{ii}} \quad (2.8)$$

Portanto, o número esperado de dias consecutivos de sol, de acordo com o modelo, é 1/0.2 = 5; para nublado, 2.5 e para chuva, 1.67.

2.2 EXTENSÕES PARA MODELOS OCULTOS DE MARKOV

Até agora foram considerados modelos de Markov em que cada estado corresponde a um evento físico (observável). Este modelo é muito restritivo para ser aplicável em muitas situações de interesse. Nesta seção, o conceito de modelos de Markov será estendido para incluir o caso em que a observação é uma função probabilística do estado, isto é, o modelo resultante (chamado modelo oculto de Markov) consiste em um processo estocástico incorporado com um processo estocástico que *não* é observável diretamente (é oculto), mas pode ser observado através de outro conjunto de processos estocásticos que produzem a sequência de observações. Para fixar as ideias, considere o modelo abaixo de experimentos de cara ou coroa.

Considere a seguinte situação. Uma pessoa está numa sala com uma barreira (por exemplo, uma cortina) através da qual ela não pode ver. Do outro lado da barreira está outra pessoa realizando cara ou coroa com uma ou mais moedas; ela não irá dizer exatamente o que está fazendo; apenas irá dizer o resultado do experimento. Assim, uma sequência de experimentos *ocultos* de cara ou coroa é realizada, com a sequência de observações consistindo de uma série de caras ou coroas; por exemplo, uma sequência de observações típica seria

$$\boldsymbol{O} = O_1\, O_2\, O_3 \ldots O_T$$
$$= \mathrm{H\ H\ T\ T\ H\ T\ H} \ldots \mathrm{T}$$

onde H representa cara e T representa coroa.

Dada a situação acima, o problema que nos interessa é como construir um HMM para explicar (modelar) a sequência de caras e coroas. O primeiro problema a ser resolvido é decidir qual o significado dos estados do modelo e então decidir o número de estados do modelo. Uma possível escolha seria assumir que apenas uma moeda viciada (isto é, a probabilidade da ocorrência de cara é diferente da probabilidade da ocorrência de coroa) faz parte do experimento. Neste caso a situação poderia ser modelada com um modelo de dois estados em que cada estado corresponde a um lado da moeda. Este modelo está representado na Figura 2.3(a). Neste caso o modelo de Markov é observável e a especificação completa exigiria apenas decidir o melhor valor de uma das probabilidades (cara, por exemplo).

Uma segunda forma de explicar a sequência observada é dada na Figura 2.3(b). Neste caso há dois estados no modelo e cada estado corresponde a uma

moeda viciada diferente. Cada estado é caracterizado por uma distribuição de probabilidade de cara e coroa e transições entre estados são caracterizadas por uma matriz de transição de estados. O mecanismo físico que determina como as transições de estado ocorrem poderia ser um conjunto de caras ou coroas independentes ou outro evento probabilístico.

Um terceira forma de HMM para explicar o evento é dada na Figura 2.3(c). Este modelo corresponde a usar 3 moedas viciadas e escolher uma entre as três, de acordo com algum evento probabilístico.

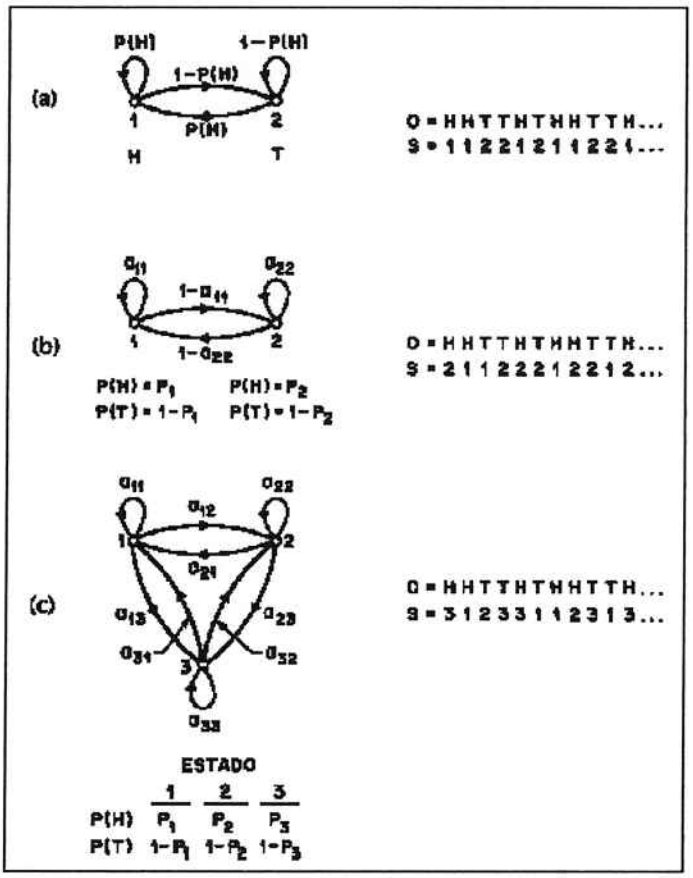

Figura 2.3 - Três modelos de Markov possíveis para o experimento de cara ou coroa. (a) Modelo de 1 moeda. (b) Modelo de 2 moedas. (c) Modelo de 3 moedas.

Dada a escolha entre os três modelos mostrados na Figura 2.3, uma questão natural é qual modelo se aproxima mais das observações verdadeiras. Deve ser claro que o modelo com 1 moeda da Figura 2.3(a) tem apenas 1 parâmetro desconhecido; o modelo com 2 moedas da Figura 2.3(b) tem 4 parâmetros desconhecidos e o modelo com 3 moedas da Figura 2.3(c) tem 9 parâmetros desconhecidos. Portanto, com maiores graus de liberdade, os HMMs maiores

parecem ser mais capazes de modelar uma série de experimentos de cara ou coroa que os modelos menores. Apesar de isto ser teoricamente verdadeiro, há considerações práticas que impõem fortes limitações no tamanho do modelo que será utilizado.

2.2.1 Elementos de um HMM

O exemplo acima nos deu um ideia simples sobre o que é um HMM e como pode ser aplicado para situações simples. Agora, os elementos de um HMM serão definidos formalmente e o método de geração de sequências de observações será explicado.

Os parâmetros que caracterizam um HMM para observações discretas são:

1) N, o número de estados no modelo. Mesmo que os estados sejam ocultos, para muitas aplicações práticas em geral há um significado associado aos estados ou conjuntos de estados do modelo. Geralmente os estados estão interconectados de uma maneira que cada estado pode ser alcançado a partir de qualquer estado (modelo ergódico).
2) M, o número de observações distintas por estado – isto é, o tamanho do alfabeto discreto. Os símbolos observáveis correspondem à saída física do sistema modelado. A notação para símbolos individuais é $V = \{v_1, v_2, \ldots, v_M\}$
3) A função massa de probabilidade de transição do estado q_i é

$$a_{ij} = P[q_{t+1} = j \mid q_t = i], \quad 1 \leq j \leq N \quad (2.9)$$

que é a i-ésima linha da matriz A_{NxN}.

Para o caso especial em que cada estado pode ser alcançado a partir de todos os outros estados em um único passo, $a_{ij} > 0$ para todo i,j.

4) A massa de probabilidade de observação de símbolos, $B = \{b_j(k)\}$ em que

$$b_j(k) = P[o_t = v_k \mid q_t = j], \quad 1 \leq k \leq M, \quad (2.10),$$

onde o_t denota a observação no instante t, define a distribuição de símbolos para o estado j, $j = 1, 2, \ldots, N$.

5) A massa de probabilidade de estado inicial $\pi = \{\pi_i\}$ em que

$$\pi_i = P[q_1 = i], \quad 1 \leq i \leq N \quad (2.11).$$

Dados valores apropriados de *N, M, A, B* e π, o HMM pode ser usado como um gerador para dar a sequência de observações

$$\boldsymbol{O} = O_1 O_2 \ldots O_T \quad (2.12)$$

(onde cada observação O_t é um dos símbolos de *V* e *T* é o número de observações na sequência) da seguinte forma:

1) Escolhe-se um estado inicial $q_1 = i$ de acordo com a distribuição de estado inicial π.
2) Define-se t = 1.
3) Escolhe-se $O_t = v_k$ de acordo com a distribuição de probabilidade de observação de símbolos para o estado *i*, isto é, $b_i(k)$.
4) Muda-se para um novo estado $q_{t+1} = j$ de acordo com a distribuição de probabilidades de transição de estado para o estado *i*, isto é, a_{ij}.
5) Define-se $t = t + 1$; retorna ao passo 3 se $t < T$; caso contrário, o procedimento é terminado.

Pela discussão acima, nota-se que a especificação completa de um HMM requer a especificação de dois parâmetros de modelo, *N* e *M*, especificação dos símbolos de observação e a especificação de três conjuntos de probabilidades: A, B e π. Por conveniência, utilizaremos a notação compacta

$$\lambda = (A, B, \pi) \quad (2.13)$$

para indicar o conjunto completo de parâmetros do modelo.

2.2.2 Os três problemas básicos para HMMs

Dada a forma de HMM da seção anterior, há três problemas básicos que devem ser resolvidos para que o modelo seja útil em aplicações o mundo real. Estes problemas são os seguintes:

Problema 1: Dada a sequência de observações $O = O_1 O_2 ... O_T$ e um modelo $\lambda = (A, B, \pi)$, como calcular de forma eficiente $P(O|\lambda)$, a probabilidade da sequência de observações, dado o modelo?

Problema 2: Dada a sequência de observações $O = O_1 O_2 ... O_T$ e o modelo λ, como escolher uma sequência de estados $Q = q_1 q_2 ... q_T$ correspondente que é ótima em algum sentido (isto é, melhor "explica" as observações)?

Problema 3: Como ajustar os parâmetros de modelo $\lambda = (A, B, \pi)$ para maximizar $P(O|\lambda)$?

O Problema 1 é o problema de avaliação, ou seja, dados um modelo e uma sequência de observações, como calcular a probabilidade de que a sequência tenha sido produzida pelo modelo. o problema também pode ser visto como um modo de se avaliar o quão bem um modelo combina com uma observação dada. Este último ponto de vista é extremamente útil em reconhecimento de voz. Por exemplo, se considerarmos o caso em que estamos escolhendo entre vários modelos, a solução para o Problema 1 nos permite escolher o modelo que melhor combine com as observações.

O Problema 2 é quando tentamos "descobrir" a parte oculta do modelo, isto é, encontrar a sequência de estados "correta". Deve ser claro que para todos os modelos, exceto os degenerados, não há uma sequência de estados "correta" para ser encontrada. Portanto, para aplicações práticas, usualmente utilizamos um critério ótimo para resolver este problema.

O Problema 3 é quando tentamos otimizar os parâmetros do modelo para melhor descrever como um dada observação é gerada. A sequência de observações usada para ajustar os parâmetros do modelo é chamada sequência de treinamento já que é utilizada para "treinar" o HMM. Este é um problema crucial para a aplicação em reconhecimento de voz, já que permite adaptar de forma ótima os parâmetros do modelo aos dados de treinamento, isto é, criar melhores modelos para o fenômeno real.

Para fixar as ideias, considere o exemplo de um sistema simples de reconhecimento de palavras isoladas. Para cada palavra de um vocabulário com W palavras, nós queremos definir um HMM de N estados. O sinal de voz de uma dada palavra é representado como um sequência temporal de vetores espectrais

codificados. Assumimos que a codificação é feita usando-se um entre M vetores espectrais de um conjunto de códigos; portanto, cada observação é o índice do vetor espectral mais próximo (em algum sentido espectral) ao sinal de voz. Assim, para cada palavra do vocabulário, temos uma sequência de treinamento consistindo de um número de repetições de sequências de índices da palavra (enunciadas por um ou mais locutores). A primeira tarefa é construir modelos individuais das palavras. Esta tarefa é feita usando a solução para o Problema 3 para estimar os parâmetros ótimos do modelo de cada palavra. Para entender o significado físico dos estados do modelo, usamos a solução para o Problema 2 para segmentar cada sequência de treinamento em estados, e então estudar as propriedades dos vetores espectrais que levam às observações ocorrendo em cada estado. o objetivo aqui é refinar o modelo (por exemplo, aumentar o número de estados) a fim de melhorar sua capacidade de modelar as sequências de palavras faladas. Finalmente, uma vez que o conjunto de W HMMs foi definido, otimizado e estudado, o reconhecimento de uma palavra desconhecida é realizado usando a solução para o Problema 1 para avaliar cada modelo de palavra em frente à sequência de observação de teste e selecionar o modelo cuja avaliação obteve o maior resultado.

2.2.3 Tipos de HMMs

Até agora, apenas foi considerado o caso especial de HMMs ergódicos ou totalmente conectados em que cada estado pode ser alcançado (em um único passo) a partir de qualquer estado do modelo (estritamente falando, um modelo ergódico tem a propriedade de que cada estado pode ser alcançado por qualquer outro estado em um número finito de passos). Como mostrado na Figura 2.4(a), um modelo de 4 estados, este modelo tem a propriedade de que todos os coeficientes a_{ij} são positivos.

Para algumas aplicações outros tipos de HMMs exploram melhor as propriedades do sinal sendo modelado que o modelo ergódico. Um desses modelos é apresentado na Figura 2.4(b). Este modelo é chamado de modelo esquerda-direita ou modelo de Bakis porque a sequência de estados associada ao modelo tem a propriedade de que com o passar do tempo o índice do estado aumenta (ou permanece o mesmo), isto é, os estados evoluem da esquerda para direita. Claramente o modelo esquerda-direita tem a propriedade de poder modelar sinais

cujas propriedades mudam com o tempo como, por exemplo, voz.

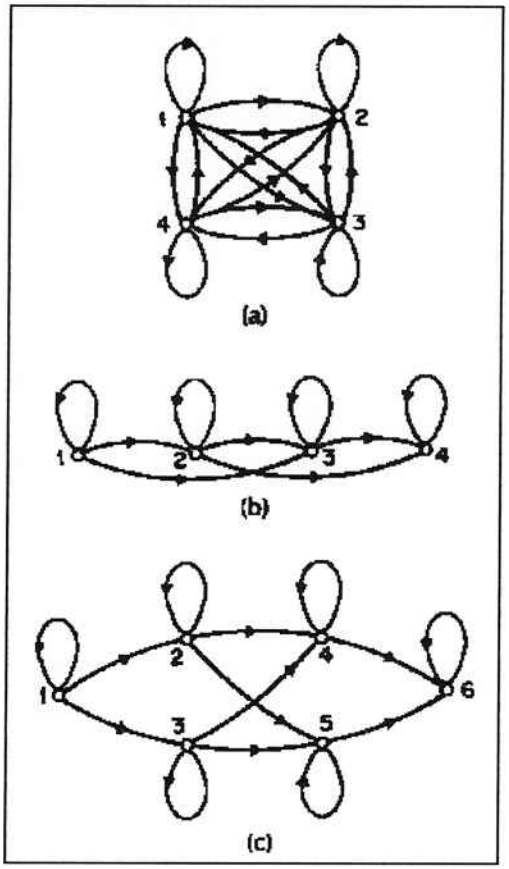

Figura 2.4 - Ilustração de 3 tipos distintos de HMMs. (a) Modelo ergódico com 4 estados. (b) Modelo esquerda-direita com 4 estados. (c) Modelo esquerda-direita com caminhos paralelos com 6 estados.

A propriedade fundamental de todos os modelos esquerda-direita é que os coeficientes de transição de estado são da forma

$$a_{ij} = 0, \quad j < i \quad (2.14)$$

isto é, não são permitidas transições de estado para estados com índice menor que o do estado atual. Além disso, a probabilidade de estado inicial tem a propriedade

$$\pi_i = \begin{cases} 0, & i \neq 1 \\ 1, & i = 1 \end{cases} \quad (2.15)$$

já que a sequência de estados deve iniciar no estado 1 (e terminar no estado N).

Geralmente, em modelos esquerda-direita, restrições adicionais são impostas aos coeficientes de transição para garantir que mudanças grandes de estado não ocorram; assim, uma restrição da forma

$$a_{ij} = 0, \quad j > i + \Delta \quad (2.16)$$

é geralmente usada. Em particular, no exemplo da Figura 2.4(b), o valor de Δ é 2, isto é, não são permitidos saltos de mais que 2 estados. Deve estar claro que para o último estado em um modelo esquerda-direita os coeficientes de transição de estado são

$$a_{NN} = 1 \quad (2.17a)$$
$$a_{Ni} = 0, \quad i < N \quad (2.17b)$$

Há ainda outros tipos de HMMs. Por exemplo, a Figura 2.4(c) mostra uma combinação paralela de dois modelos esquerda-direita.

2.3 APLICAÇÃO EM RECONHECIMENTO DE VOZ

Sistemas de reconhecimento de voz geralmente assumem que o sinal de voz é um realização de uma mensagem codificada como uma sequência de um ou mais símbolos, como ilustrado na figura 2.5.

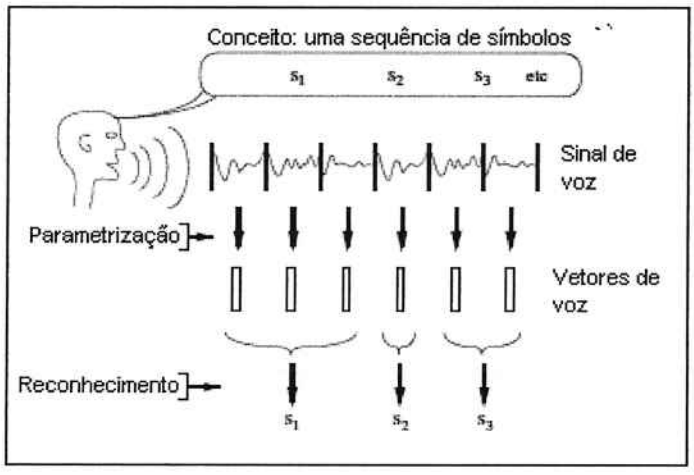

Figura 2.5 - Codificação e decodificação da mensagem

Para efetuar a operação de reconhecer um símbolo dada uma locução, o sinal de voz é primeiramente convertido em uma sequência de vetores de parâmetros igualmente espaçados. Assume-se que esta sequência de vetores de parâmetros formam uma representação exata do sinal de voz, dado que na duração de um único vetor (tipicamente 10 ms) o sinal de voz pode ser considerado estacionário. Apesar de isto não ser estritamente verdadeiro, é uma aproximação razoável. Representações paramétricas típicas são coeficientes de predição linear e suas formas derivadas.

A tarefa para o sistema de reconhecimento é efetuar um mapeamento entre sequências de vetores de voz e suas correspondentes sequências de símbolos. Dois problemas fazem esta tarefa muito difícil. Primeiro, o mapeamento de símbolos para voz não é unívoco pois símbolos diferentes podem gerar sinais de voz semelhantes. Segundo, as fronteiras entre símbolos não podem ser explicitamente identificadas a partir do sinal de voz. Portanto, não é possível tratar o sinal de voz como uma sequência concatenada de parâmetros estáticos.

O problema do não conhecimento das fronteiras entre palavras pode ser resolvido restringindo-se a tarefa ao reconhecimento de palavras isoladas. Como mostrado na Figura 2.6, isto implica que o sinal de voz corresponde a um único símbolo (por exemplo, uma palavra) escolhida em um vocabulário fixo. Apesar deste problema simples ser de certa forma artificial, ele tem um grande número de aplicações. Portanto, ele serve como uma base para introduzir as ideias básicas de sistemas de reconhecimento de voz baseados em HMMs. Assim, o reconhecimento de palavras isoladas será tratado a seguir.

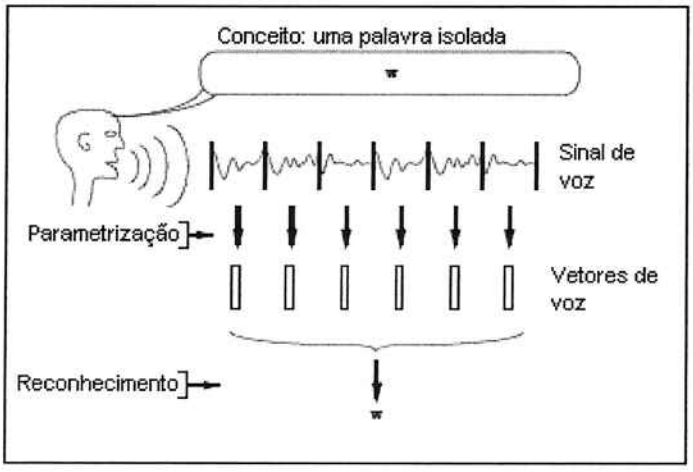

Figura 2.6 - Reconhecimento de palavras isoladas

2.3.1 Reconhecimento de palavras isoladas

Seja cada palavra falada representada por uma sequência de vetores de voz ou observações $\boldsymbol{O}$, definida como em (2.12) onde O_t representa o vetor de voz observado no instante t. O problema do reconhecimento de palavras isoladas pode ser então definido como o de se calcular

$$\arg\max_i \{P(w_i|\boldsymbol{O})\} \quad (2.18)$$

onde w_i é a i-ésima palavra do vocabulário. Esta probabilidade não é diretamente calculável; contudo, se um modelo paramétrico de produção de palavras como um modelo de Markov é assumido, o problema é substituído pelo problema mais simples de estimar os parâmetros do modelo oculto de Markov.

No reconhecimento de voz baseado em HMM, é assumido que a sequência de vetores de voz observados correspondendo a cada palavra é gerada por um modelo oculto de Markov como mostrado na Figura 2.7.

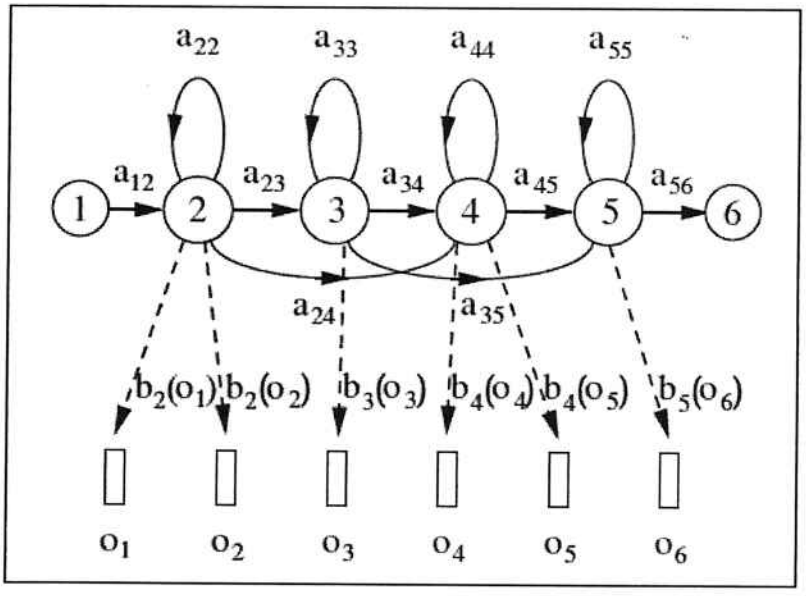

Figura 2.7 - Um HMM do tipo esquerda-direita com 6 estados

Dado um conjunto de modelos M_i, (2.18) é resolvida calculando-se $P(\boldsymbol{O}|M_i)$ para todos i e encontrando o maior valor. Um resumo do método de reconhecimento de palavras isoladas está na Figura 2.8, onde está um exemplo com um vocabulário de 3 palavras.

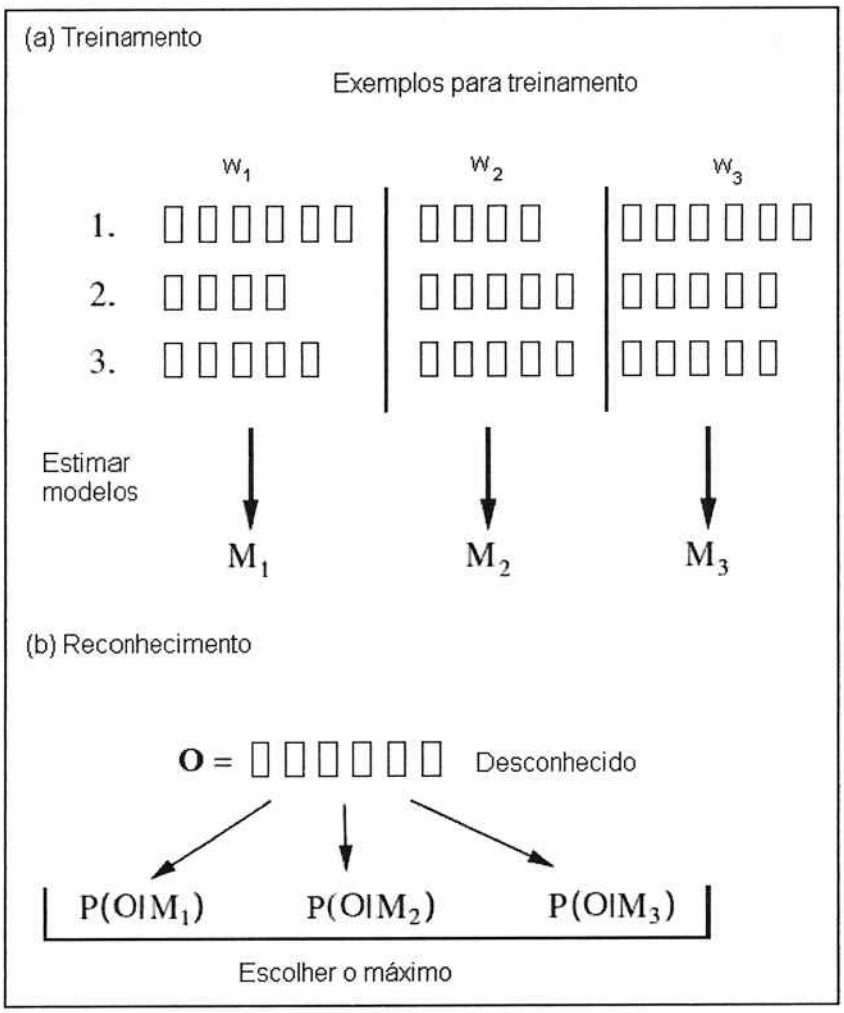

Figura 2.8 - Reconhecimento de palavras isoladas com HMM

3 O HTK

3.1 VISÃO GERAL DA FERRAMENTA

O HTK é um conjunto de ferramentas (*toolkit*) para implementação de HMMs com o intuito principal de utilização para reconhecimento de voz. A ferramenta é de uso gratuito para fins acadêmicos e de pesquisa. Há dois estágios principais de processamento, como pode ser visto na Figura 3.1. Primeiro, as ferramentas de treinamento do HTK são usadas para estimar os parâmetros de um conjunto de HMMs utilizando conjuntos de treinamento (sinais de voz) e suas transcrições associadas. Segundo, entradas desconhecidas são transcritas usando as ferramentas de reconhecimento do HTK.

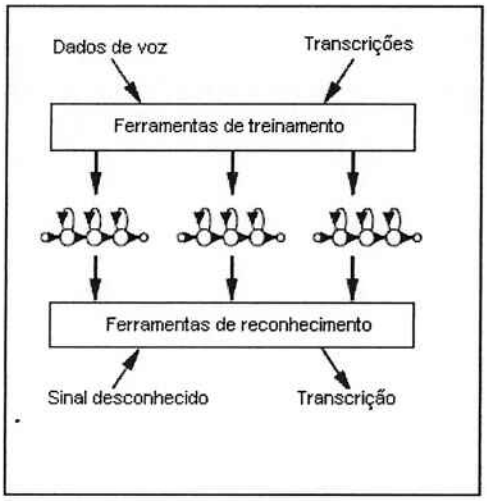

Figura 3.1 - Estágios de processamento do HTK

3.1.1 Arquitetura do HTK

A maior parte da funcionalidade do HTK está em módulos de biblioteca. Estes módulos garantem que as interfaces de cada ferramenta sejam concordantes entre si. Também proporcionam um recurso central para funções comumente utilizadas. A Figura 3.2 ilustra a arquitetura do HTK e mostra suas interfaces de entrada e saída.

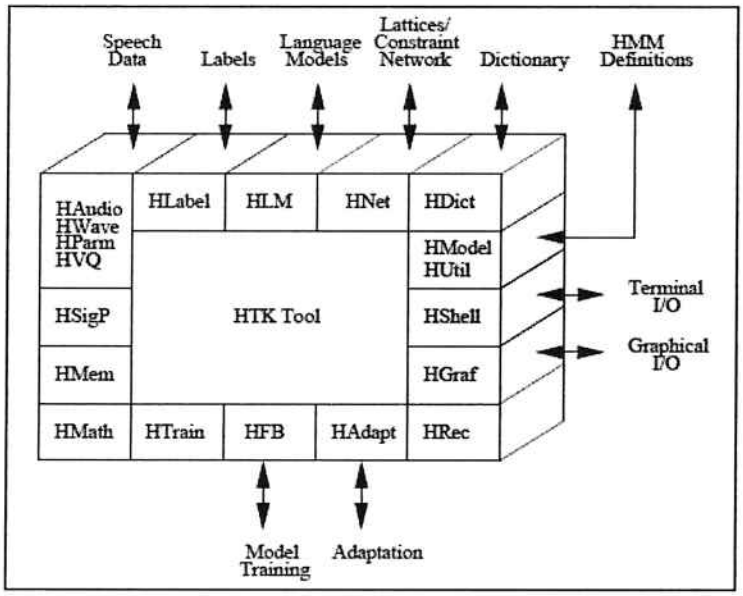

Figura 3.2 - Arquitetura do HTK

A entrada e saída e interação com o sistema operacional são controladas pelo módulo HShell e o gerenciamento de memória é controlado pelo módulo HMem. Suporte a funções matemáticas é provido pelo módulo HMath e as operações de

processamento de sinal necessárias para análise de voz estão em HSigP. Cada tipo de arquivo requerido pelo HTK tem um módulo de interface dedicado. HLabel provê a interface para arquivos de *label*, HLM para arquivos de modelos de linguagem, HNet para redes e treliças (*lattices*), HDict para dicionários e HModel para definições de HMMs.

3.2 O CONJUNTO DE FERRAMENTAS

Nesta seção são apresentados os principais grupos de ferramentais do HTK e suas funcionalidades.

3.2.1 Ferramentas de preparação de dados

Para construir um conjunto de HMMs, um conjunto de sinais de voz e suas transcrições são necessários. Antes que os dados possam ser usados para treinamento, eles devem ser convertidos para uma forma paramétrica adequada. Em geral são utilizados coeficientes de predição linear (*LPC - Linear Prediction Coefficients*) e MFCC (*Mel Frequency Cepstral Coefficients*). Estas tarefas são implementadas pelo conjunto de ferramentas de preparação de dados.

3.2.2 Ferramentas de treinamento

Após a etapa de preparação de dados é definida a topologia requerida para cada HMM por uma definição de protótipo. As definições de HMMs podem ser armazenadas como arquivos de texto para que possam ser facilmente editadas.
A sequência de treinamento é descrita a seguir. Primeiramente, um conjunto inicial de modelos deve ser criado. Uma vez que o conjunto inicial de modelos tenha sido criado, a ferramenta HERest executa o treinamento sobre todo o conjunto de testes utilizando o algoritmo de Baum-Welch. Quando todo o conjunto de treinamento foi processado, os parâmetros do HMM são recalculados.

A filosofia da construção de sistema no HTK é que HMMs devem ser refinados incrementalmente.

3.2.3 Ferramentas de reconhecimento

O HTK dispõe de uma ferramenta de reconhecimento chamada HVite que permite o reconhecimento utilizando modelos de linguagem e treliça (*lattice*) utilizando o algoritmo de Viterbi. O HVite recebe como entrada uma rede descrevendo as sequências de palavras permitidas, um dicionário definindo cada palavra e um conjunto de HMMs. Opera convertendo uma rede de palavras em uma rede de sons e então associa cada exemplo de som a um HMM apropriado.

3.2.4 Ferramenta de análise

Uma vez que o sistema de reconhecimento foi construído, é necessário avaliar sua performance. Isto é usualmente feito usando o sistema para transcrever algumas sentenças de treinamento pré-gravadas e comparar a saída do sistema com as transcrições de referência. Esta comparação é feita por uma ferramenta chamada HResults.

3.3 PASSOS BÁSICOS PARA CONSTRUIR O SISTEMA

A construção do sistema de reconhecimento de voz utilizando o HTK pode ser resumida nos seguintes passos:

1) Definição da gramática, onde são definidas as palavras e expressões, ou seja, a arquitetura básica do sistema de reconhecimento;
2) Definição do dicionário, onde as palavras são associadas aos modelos;
3) Gravação dos dados para treinamento;
4) Criação dos arquivos de transcrição, que definem a estrutura dos dados, por exemplo, definindo os trechos de silêncio inicial, voz e silêncio final;
5) Parametrização dos dados usando a ferramenta HCopy;
6) Criação dos HMMs;
7) Inicialização dos modelos com a ferramenta HInit;
8) Refinamento dos modelos com a ferramenta HRest;
9) Reconhecimento usando a ferramenta HVite.

4 IMPLEMENTANDO O SISTEMA E A CALCULADORA

4.1 DEFINIÇÃO DA GRAMÁTICA

A gramática é definida em um arquivo texto de acordo com algumas regras de sintaxe. O arquivo que foi utilizado neste projeto está apresentado abaixo.

```
$DIGITO = UM | DOIS | TRES | QUATRO | CINCO |·SEIS | SETE | OITO |
NOVE | ZERO | PONTO;
$OPER = MAIS | MENOS | VEZES | DIV | IGUAL | FECHAR;
( { SIL_INI } [ $DIGITO | $OPER ] { SIL_FIM } )
```

Neste arquivo definiu-se a arquitetura do sistema de reconhecimento de palavras isoladas.

O símbolo `$` é utilizado para definir variáveis e a barra vertical denota alternativas. Assim, no arquivo acima definem-se duas variáveis: `DIGITO`, que define os dígitos décimas mais o ponto decimal e `OPER`, que define as quatro operações mais o comando de resultado e o comando de fechar o programa.

As chaves `{ }` indicam nenhuma ou várias repetições da palavra e os colchetes `[ ]` indicam nenhuma ou apenas uma ocorrência da palavra. Assim, o sistema definido acima reconhece uma palavra do tipo `DIGITO` ou `OPER` entre sequências de silêncios ou apenas silêncios.

4.2 DEFINIÇÃO DO DICIONÁRIO

O dicionário é um arquivo texto em que cada palavra é associada a um modelo. O dicionário do sistema de reconhecimento projetado está apresentado abaixo.

```
UM [1] um
DOIS [2] dois
TRES [3] tres
QUATRO [4] quatro
CINCO [5] cinco
SEIS [6] seis
```

```
SETE [7] sete
OITO [8] oito
NOVE [9] nove
ZERO [0] zero
MAIS [+] mais
MENOS [-] menos
VEZES [*] vezes
DIV [/] div
PONTO [.] ponto
IGUAL [=] igual
FECHAR [f] fechar
SIL_INI [] sil
SIL_FIM [] sil
```

Na primeira coluna estão os nomes que foram definidos na gramática. A última coluna associa cada nome a um dos modelos (HMMs) do sistema. A coluna do meio é um item opcional, são os símbolos que serão utilizados na saída do sistema de reconhecimento.

4.3 GRAVAÇÃO DOS ARQUIVOS PARA TREINAMENTO

Foram gravadas cinco repetições de cada comando. A gravação foi feita com o programa WaveSurfer, em formato wave e taxa de amostragem de 16 kHz com canal único.

4.4 CRIAÇÃO DOS ARQUIVOS DE TRANSCRIÇÃO

Os arquivos de transcrição definem a estrutura dos arquivos de treinamento, ou seja, definem os trechos de silêncio e o trecho de voz. As transcrições também foram feitas no programa WaveSurfer no formato exigido pelo HTK. Na Figura 4.1 está apresentada a tela do programa durante uma gravação e transcrição.

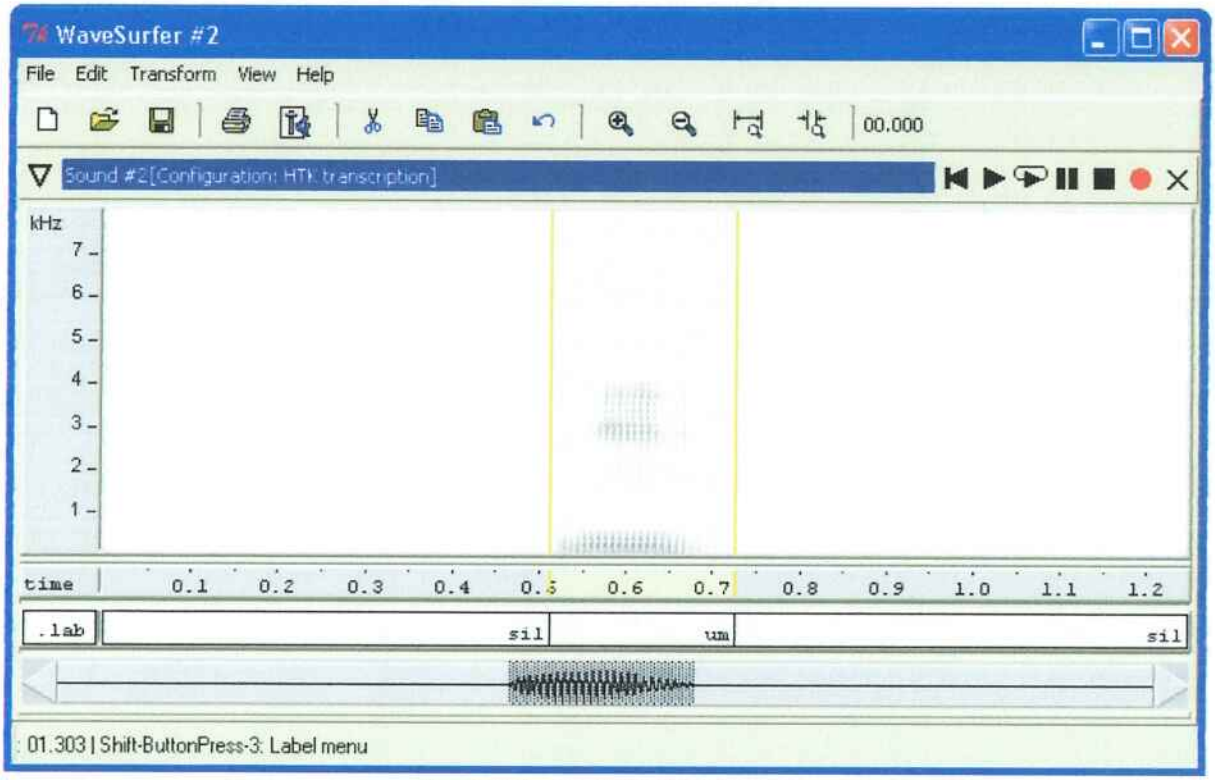

Figura 4.1 - Programa WaveSurfer

Neste exemplo foi feita a gravação da palavra "um". Notar os trechos da gravação: silêncio inicial, voz e silêncio final. A transcrição é gravada num arquivo texto com o formato lab, que é aceito pelo HTK. Um exemplo de arquivo de transcrição está apresentado abaixo.

```
0 5428903 sil
5428903 7863215 um
7863215 12480680 sil
```

O arquivo mostra o inicio e o fim de cada trecho da gravação em unidades de 100 ns. No exemplo acima, o trecho com a palavra "um" está entre 0.54 e 0.79 segundos e a duração total da gravação é de 1.25 s.

4.5 PARAMETRIZAÇÃO DOS DADOS

Com as gravações e suas respectivas transcrições, os dados são parametrizados através da ferramenta HCopy. Neste projeto a parametrização será por MFCC. A configuração da parametrização é definida em um arquivo. As

configurações utilizadas neste projeto estão apresentadas abaixo.

```
SOURCEFORMAT = WAVE
TARGETKIND = MFCC_0_D_A
WINDOWSIZE = 250000.0
TARGETRATE = 100000.0
NUMCEPS = 12
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
```

O arquivo acima define que:

- as gravações estão em formato wave (`SOURCEFORMAT = WAVE`);
- a parametrização é com MFCC (`TARGETKIND = MFCC`);
- os 12 primeiros coeficientes (c_1,...,c_{12}) serão calculados (`NUMCEPS = 12`);
- será calculado o coeficiente c_0, que é proporcional à energia total na janela de análise (`_0`);
- serão calculados os coeficientes Delta, uma estimativa da primeira derivada dos coeficientes MFCC, (`_D`) ;
- serão calculados os coeficientes de aceleração, uma estimativa da segunda derivada dos coeficientes MFCC, (`_A`) ;
- a janela de análise é de 250000 * 100 ns = 25 ms, (`WINDOWSIZE = 250000.0`);
- o período de análise é de 100000 * 100 ns = 10 ms, (`TARGETRATE = 100000.0`);
- uso de janela de Hamming, (`USEHAMMING = T`);
- coeficiente de pré-enfase igual a 0.97, (`PREEMCOEF = 0.97`);
- número de canais de *filterbank* igual a 26, (`NUMCHANS = 26`);
- comprimento de *liftering* (filtragem) igual a 22, (`CEPLIFTER = 22`).

A utilização dos coeficientes Delta e de aceleração melhora a performance do sistema. Ao todo são calculados 39 coeficientes. Todos os valores de configurações acima foram obtidos de sugestões para construção de um sistema de reconhecimento de palavras isoladas no manual do HTK e que em nosso projeto resultaram num sistema que atende às especificações.

A parametrização dos arquivos é efetuada com o comando

```
HCopy -A -D -C analysis.conf -S targetlist.txt
```

a opção `-A` imprime os argumentos de entrada (ou seja, repete o comando) e a opção `-D` imprime os parâmetros de configuração. O arquivo de configuração é definido em `-C analysis.conf` e a lista de entradas e saídas é definida em `-S targetlist.txt`, com um trecho mostrado abaixo.

```
data/train/sig/um01.wav data/train/mfcc/um01.mfcc
data/train/sig/um02.wav data/train/mfcc/um02.mfcc
data/train/sig/um03.wav data/train/mfcc/um03.mfcc
data/train/sig/um04.wav data/train/mfcc/um04.mfcc
data/train/sig/um05.wav data/train/mfcc/um05.mfcc
```

4.6 CRIAÇÃO DOS HMMs

A criação dos HMMs é feita através de um arquivo texto onde é definida a topologia de cada HMM. Neste arquivo são definidos o número de estados, as funções de observação dos estados emissores e a matriz de transição de estados.

O arquivo que definiu a topologia para o HMM da palavra "um" está mostrado abaixo. A topologia para todos HMMs foi a mesma, com exceção da HMM para o silêncio, que foi definida com 6 estados.

```
~o <VecSize> 39 <MFCC_0_D_A>
~h "um"
<BeginHMM>
<NumStates> 12
<State> 2
<Mean> 39
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<Variance> 39
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
<State> 3
<Mean> 39
```

```
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<Variance> 39
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0

...

<State> 11
<Mean> 39
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<Variance> 39
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
<TransP> 12
0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.4 0.3 0.3 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.4 0.3 0.3 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.4 0.3 0.3 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.4 0.3 0.3 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.4 0.3 0.3 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.4 0.3 0.3 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.4 0.3 0.3 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.4 0.3 0.3 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.4 0.3 0.3
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<EndHMM>
```

O arquivo acima define um HMM do tipo esquerda-direita da seguinte forma:

- `~o <VecSize> 39 <MFCC_0_D_A>` define o tamanho e o tipo de coeficientes que serão utilizados no treinamento do modelo;
- `~h "um"` define o nome do modelo;
- `<NumStates> 12` define o número de estados do modelo sendo os estados 1 e

12 não emissores;

- <State> 2 define a função de observação para cada estado emissor (estados 2 a 11),
  <Mean> 39 define o vetor media da função de observação com 39 elementos, é inicializado arbitrariamente com 0.0,
  <Variance> 39 define o vetor de variância da função de observação com 39 elementos, é inicializado arbitrariamente com 1.0;
- `<TransP> 12` define a matriz de transição de estados, se uma transição não é permitida, a inicialização deve ser feita com 0.0, caso contrario utiliza-se um valor arbitrário desde que a soma em cada linha seja 1.0.

4.7 INICIALIZAÇÃO DOS MODELOS

A partir dos arquivos gerados na etapa anterior, a ferramenta HInit faz a inicialização dos modelos com os dados de treinamento para garantir uma convergência rápida na próxima etapa.

A inicialização de um modelo é feita com o comando

```
HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_um
-l um -L data/train/lab um
```

que deve ser repetido para cada modelo, onde:

- a opção `-T 1` faz que o progresso da ferramenta seja impresso;
- o arquivo `trainlist.txt` contém a lista completa dos arquivos de parametrização;
- `model/hmm0` é o diretório onde o modelo inicializado será salvo;
- `model/proto/hmm_um` é o caminho do arquivo que contém a topologia do HMM;
- `-l um` indica quais trechos dos arquivos de gravação serão utilizados no treinamento;
- `data/train/lab` é o diretório onde estão os arquivos de transcrição;
- `um` é o nome do modelo a ser inicializado.

Para a etapa de refinamento, é interessante que exista um arquivo que contenha valores mínimos para os vetores de variância das funções de observação

dos estados a fim de evitar erros de computação. Este arquivo, chamado `vFloors`, é gerado com o comando

```
HCompv -A -D -T 1 -S trainlist.txt -M model/hmm0flat -H
model/proto/hmm_um -f 0.01 um
```

este comando é executado apenas uma vez com qualquer um dos arquivos criados no item 4.6, onde

- `-M model/hmm0flat` é o diretório onde o arquivo `vFloors` será salvo;
- `-f 0.01` é um fator multiplicativo para os valores em `vFloors`.

4.8 REFINAMENTO DOS MODELOS

Após a inicialização dos modelos é feito o refinamento dos mesmos com a ferramenta HRest. Para o sistema implementado foram feitas três etapas de refinamento para cada modelo.

O refinamento de um modelo é feito com o comando

```
HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_um
-H model/hmm0flat/vFloors -l um -L data/train/lab um
```

onde:

- `model/hmm1` é o diretório onde será salvo o modelo refinado;
- `model/hmm0/hmm_um` é o caminho do arquivo que contém o modelo a ser refinado;
- `model/hmm0flat/vFloors` é o caminho do arquivo `vFloors` gerado no item 4.7.

4.9 DEFINIÇÃO DA ARQUITETURA DO SISTEMA

A partir do arquivo de gramática definido no item 4.1, a arquitetura do sistema é gravada em um arquivo com formato slf através do comando

```
HParse -A -D -T 1 gram.txt net.slf
```

onde:

- `gram.txt` é o arquivo gerado no item 4.1;
- `net.slf` é o arquivo que contém a arquitetura do sistema de reconhecimento.

4.10 TREINAMENTO

Todos os comandos dos itens 4.7 ao 4.9 foram sintetizados em um único programa, cujo código fonte (arquivo `train.c`) está apresentado abaixo.

```
#include <stdio.h>
#include <stdlib.h>

void hinit(){
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_sil -l sil -L
data/train/lab sil");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_um -l um -L
data/train/lab um");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_dois -l dois -L
data/train/lab dois");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_tres -l tres -L
data/train/lab tres");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_quatro -l
quatro -L data/train/lab quatro");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_cinco -l cinco
-L data/train/lab cinco");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_seis -l seis -L
data/train/lab seis");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_sete -l sete -L
data/train/lab sete");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_oito -l oito -L
data/train/lab oito");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_nove -l nove -L
data/train/lab nove");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_zero -l zero -L
data/train/lab zero");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_mais -l mais -L
data/train/lab mais");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_menos -l menos
-L data/train/lab menos");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_vezes -l vezes
-L data/train/lab vezes");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_div -l div -L
data/train/lab div");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_ponto -l ponto
-L data/train/lab ponto");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_igual -l igual
```

```
-L data/train/lab igual");
    system("HInit -A -D -T 1 -S trainlist.txt -M model/hmm0 -H model/proto/hmm_fechar -l
fechar -L data/train/lab fechar");
}

void hrest(){
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_sil -H
model/hmm0flat/vFloors -l sil -L data/train/lab sil");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_sil -H
model/hmm0flat/vFloors -l sil -L data/train/lab sil");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_sil -H
model/hmm0flat/vFloors -l sil -L data/train/lab sil");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_um -H
model/hmm0flat/vFloors -l um -L data/train/lab um");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_um -H
model/hmm0flat/vFloors -l um -L data/train/lab um");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_um -H
model/hmm0flat/vFloors -l um -L data/train/lab um");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_dois -H
model/hmm0flat/vFloors -l dois -L data/train/lab dois");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_dois -H
model/hmm0flat/vFloors -l dois -L data/train/lab dois");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_dois -H
model/hmm0flat/vFloors -l dois -L data/train/lab dois");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_tres -H
model/hmm0flat/vFloors -l tres -L data/train/lab tres");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_tres -H
model/hmm0flat/vFloors -l tres -L data/train/lab tres");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_tres -H
model/hmm0flat/vFloors -l tres -L data/train/lab tres");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_quatro -H
model/hmm0flat/vFloors -l quatro -L data/train/lab quatro");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_quatro -H
model/hmm0flat/vFloors -l quatro -L data/train/lab quatro");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_quatro -H
model/hmm0flat/vFloors -l quatro -L data/train/lab quatro");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_cinco -H
model/hmm0flat/vFloors -l cinco -L data/train/lab cinco");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_cinco -H
model/hmm0flat/vFloors -l cinco -L data/train/lab cinco");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_cinco -H
model/hmm0flat/vFloors -l cinco -L data/train/lab cinco");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_seis -H
model/hmm0flat/vFloors -l seis -L data/train/lab seis");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_seis -H
model/hmm0flat/vFloors -l seis -L data/train/lab seis");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_seis -H
model/hmm0flat/vFloors -l seis -L data/train/lab seis");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_sete -H
model/hmm0flat/vFloors -l sete -L data/train/lab sete");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_sete -H
```

```
model/hmm0flat/vFloors -l sete -L data/train/lab sete");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_sete -H
model/hmm0flat/vFloors -l sete -L data/train/lab sete");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_oito -H
model/hmm0flat/vFloors -l oito -L data/train/lab oito");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_oito -H
model/hmm0flat/vFloors -l oito -L data/train/lab oito");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_oito -H
model/hmm0flat/vFloors -l oito -L data/train/lab oito");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_nove -H
model/hmm0flat/vFloors -l nove -L data/train/lab nove");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_nove -H
model/hmm0flat/vFloors -l nove -L data/train/lab nove");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_nove -H
model/hmm0flat/vFloors -l nove -L data/train/lab nove");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_zero -H
model/hmm0flat/vFloors -l zero -L data/train/lab zero");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_zero -H
model/hmm0flat/vFloors -l zero -L data/train/lab zero");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_zero -H
model/hmm0flat/vFloors -l zero -L data/train/lab zero");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_mais -H
model/hmm0flat/vFloors -l mais -L data/train/lab mais");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_mais -H
model/hmm0flat/vFloors -l mais -L data/train/lab mais");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_mais -H
model/hmm0flat/vFloors -l mais -L data/train/lab mais");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_menos -H
model/hmm0flat/vFloors -l menos -L data/train/lab menos");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_menos -H
model/hmm0flat/vFloors -l menos -L data/train/lab menos");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_menos -H
model/hmm0flat/vFloors -l menos -L data/train/lab menos");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_vezes -H
model/hmm0flat/vFloors -l vezes -L data/train/lab vezes");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_vezes -H
model/hmm0flat/vFloors -l vezes -L data/train/lab vezes");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_vezes -H
model/hmm0flat/vFloors -l vezes -L data/train/lab vezes");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_div -H
model/hmm0flat/vFloors -l div -L data/train/lab div");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_div -H
model/hmm0flat/vFloors -l div -L data/train/lab div");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_div -H
model/hmm0flat/vFloors -l div -L data/train/lab div");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_ponto -H
model/hmm0flat/vFloors -l ponto -L data/train/lab ponto");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_ponto -H
model/hmm0flat/vFloors -l ponto -L data/train/lab ponto");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_ponto -H
model/hmm0flat/vFloors -l ponto -L data/train/lab ponto");
```

```
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_fechar -H
model/hmm0flat/vFloors -l fechar -L data/train/lab fechar");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_fechar -H
model/hmm0flat/vFloors -l fechar -L data/train/lab fechar");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_fechar -H
model/hmm0flat/vFloors -l fechar -L data/train/lab fechar");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm1 -H model/hmm0/hmm_igual -H
model/hmm0flat/vFloors -l igual -L data/train/lab igual");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm2 -H model/hmm1/hmm_igual -H
model/hmm0flat/vFloors -l igual -L data/train/lab igual");
    system("HRest -A -D -T 1 -S trainlist.txt -M model/hmm3 -H model/hmm2/hmm_igual -H
model/hmm0flat/vFloors -l igual -L data/train/lab igual");
}

int main(){
    system("HCopy -A -D -C analysis.conf -S targetlist.txt");
    hinit();
    system("HCompv -A -D -T 1 -S trainlist.txt -M model/hmm0flat -H model/proto/hmm_um -f 0.01
um");
    hrest();
    system("HParse -A -D -T 1 gram.txt net.slf");

    system("PAUSE");
    return 0;
}
```

4.11 A CALCULADORA

O código fonte (arquivo `calc.c`) do programa que implementa a calculadora está apresentado abaixo.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){
 FILE *rec;
 char op[2][21], oper;
 int i , j, f;
 char c;
 double res;

 printf("Calculadora via Comandos de Voz\n");
 printf("Por Daniel Massami Takizawa e Tiago Dal Pai Fabbri\n");
 printf("Utiliza a biblioteca HTK - http://htk.eng.cam.ac.uk/\n\n");
 printf("Comandos disponiveis: 0-9 , . , + , - , * , / , = , fechar\n\n");
```

```
 i = j = f = 0;
 op[0][0] = op[1][0] = oper = '\0';
 while(f == 0){
  system("HVite -e -o WTS -C directin.conf -H model/hmm3/hmm_zero -H
model/hmm3/hmm_um \
  -H model/hmm3/hmm_dois -H model/hmm3/hmm_tres -H model/hmm3/hmm_quatro -H
model/hmm3/hmm_cinco \
  -H model/hmm3/hmm_seis -H model/hmm3/hmm_sete -H model/hmm3/hmm_oito -H
model/hmm3/hmm_nove \
  -H model/hmm3/hmm_mais -H model/hmm3/hmm_menos -H model/hmm3/hmm_vezes -H
model/hmm3/hmm_div \
  -H model/hmm3/hmm_ponto -H model/hmm3/hmm_fechar -H model/hmm3/hmm_igual -H
model/hmm3/hmm_sil \
  -w net.slf dict.txt hmmlist.txt");
  rec = fopen("0001.rec", "r");
  if(rec != NULL){
   fscanf(rec, "%c", &c);
   if(!feof(rec)){
    printf("%c", c);
    if(c == '+' || c == '-' || c == '*' || c == '/'){
     oper = c;
     i = 1;
     j = 0;
    }
    if(c == 'f'){
     f = 1;
     printf("\n\n");
    }
    if(c == '='){
     switch(oper){
      case('+'): res = atof(&op[0][0]) + atof(&op[1][0]); break;
      case('-'): res = atof(&op[0][0]) - atof(&op[1][0]); break;
      case('*'): res = atof(&op[0][0]) * atof(&op[1][0]); break;
      case('/'): res = atof(&op[0][0]) / atof(&op[1][0]); break;
      default: ;
     }
     printf("\n%s %c %s = %g\n\n", &op[0][0], oper, &op[1][0], res);
     op[0][0] = op[1][0] = oper = '\0';
     i = j = 0;
    }
    if(c >= '0' && c <= '9' || c == '.'){
     op[i][j] = c;
     j++;
     op[i][j] = '\0';
    }
   }
```

```
    if(c != '=' && c != 'f')
     printf("\n%s %c %s\n", &op[0][0], oper, &op[1][0]);
    fclose(rec);
   }
  }

 system("PAUSE");
 return 0;
}
```

O reconhecimento de voz ocorre na chamada à ferramenta HVite, onde:

- a opção `-e` gera um arquivo de saída com o resultado do reconhecimento;
- a opção `-o WTS` define que apenas os símbolos definidos no dicionário serão apresentados no arquivo de saída;
- `directin.conf` é um arquivo que define os parâmetros da entrada de áudio em tempo real (*direct audio input*);
- `-H model/hmm3/hmm_zero` até `-H model/hmm3/hmm_sil` definem os modelos que serão utilizados;
- `net.slf` é o arquivo gerado com o comando HParse;
- `dict.txt` é o dicionário;
- `hmmlist.txt` é um arquivo com todos os nomes dos modelos.

O arquivo `directin.conf` utilizado está apresentado abaixo.

```
SOURCERATE = 625.0
SOURCEKIND = HAUDIO
SOURCEFORMAT = WAVE
USESILDET = T
MEASURESIL = F
OUTSILWARN = T
ENORMALISE = F
SILENERGY = 14.333
SPEECHTHRESH = 37.001
TARGETKIND = MFCC_0_D_A
WINDOWSIZE = 250000.0
TARGETRATE = 100000.0
NUMCEPS = 12
USEHAMMING = T
```

```
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
```

onde:

- SOURCERATE = 625.0 define o período de amostragem para o sinal de entrada em unidades de 100 ns, o que define a frequência de amostragem em 16 kHz;
- SOURCEKIND = HAUDIO define que a entrada será em tempo real;
- SOURCEFORMAT = WAVE define o formato de gravação do áudio;
- USESILDET = T define que será utilizado o detector de silêncio para marcar o inicio e o fim da gravação;
- MEASURESIL = F define se serão feitas as medidas das variáveis SILENERGY e SPEECHTRESH, que serão explicadas abaixo;
- ENORMALISE = F define que não será feita normalização da energia do sinal, condição necessária quando a entrada de áudio é em tempo real;
- SILENERGY = 14.333 representa o nível médio de energia do silêncio e pode ser medido automaticamente com a configuração MEASURESIL = T;
- SPEECHTHRESH = 37.001 representa o nível mínimo para interpretar o sinal como voz, pode ser medido automaticamente com a configuração MEASURESIL = T;
- as configurações seguintes são as mesmas utilizadas na parametrização dos dados.

A Figura 4.2 apresenta uma execução do programa no modo de medida das variáveis SILENERGY e SPEECHTRESH (com a configuração MEASURESIL = T) e, respectivamente, os valores medidos.

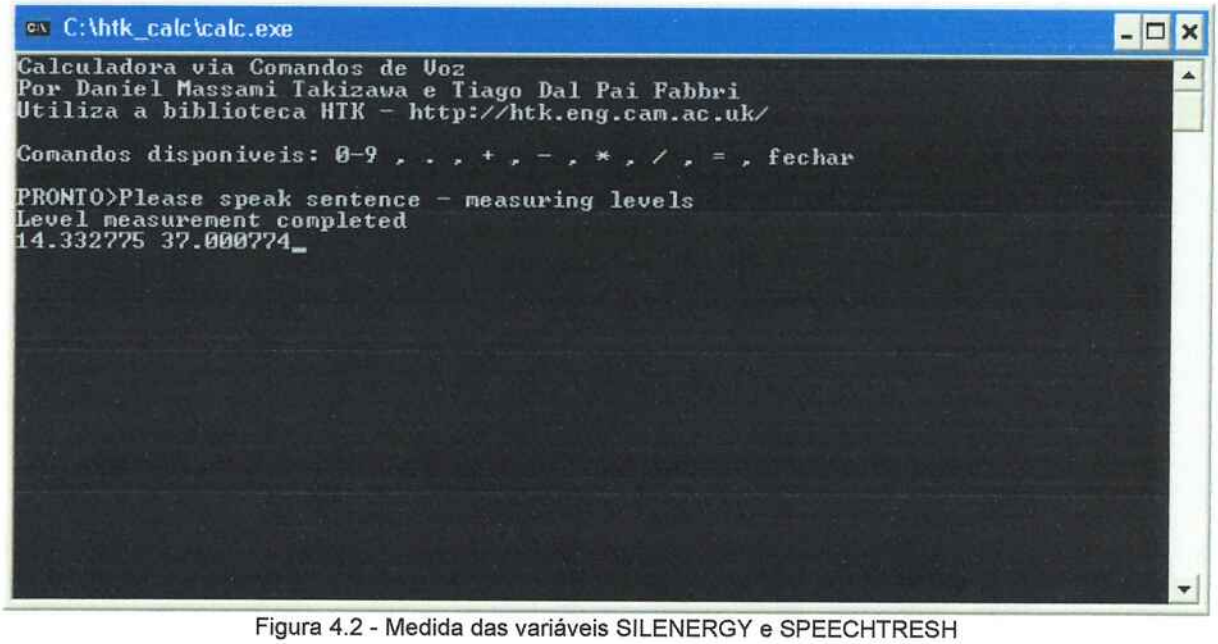

Figura 4.2 - Medida das variáveis SILENERGY e SPEECHTRESH

Os valores devem ser copiados manualmente no arquivo de configuração `directin.conf` e a configuração `MEASURESIL = F` deve ser utilizada para que o programa seja executado em modo calculadora, o que está apresentado na Figura 4.3.

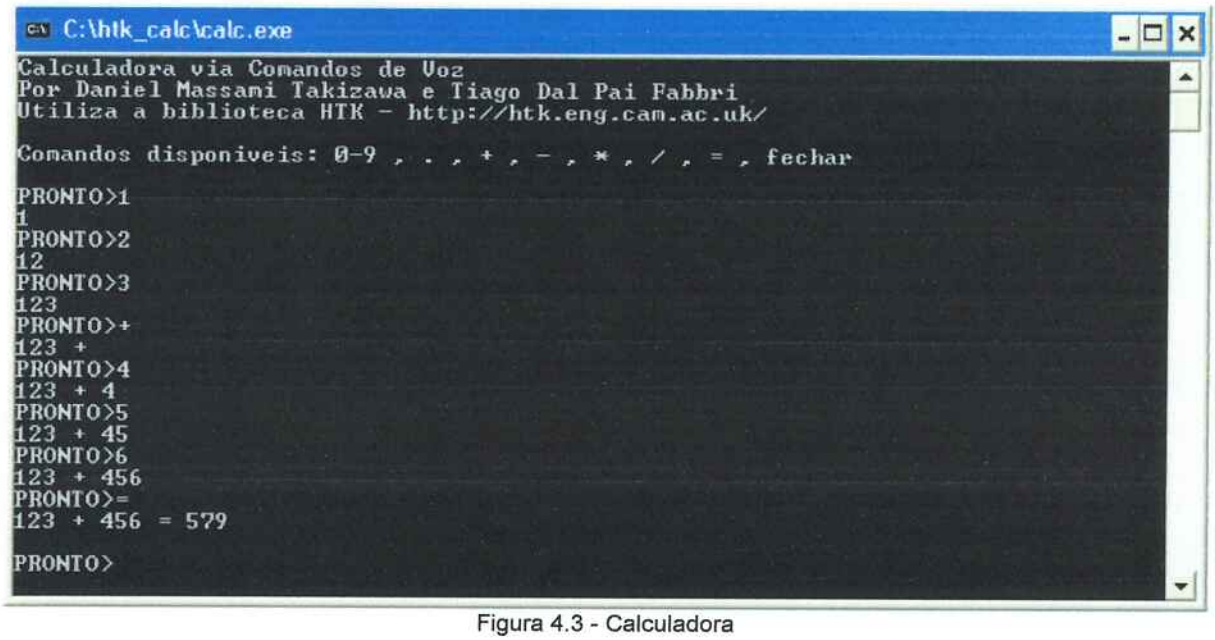

Figura 4.3 - Calculadora

4.12 TESTES

Foi feito um teste com 100 comandos dados para a calculadora e o reconhecimento foi correto para 92 % dos casos.

O teste acima foi realizado num ambiente "silencioso" e sem eco.

Num ambiente com eco e com muitos ruídos, a taxa de acertos do sistema de reconhecimento é reduzida drasticamente.

5 CONCLUSÕES

O intuito deste trabalho foi apresentar o projeto de um sistema automático de reconhecimento de voz. A escolha do método baseado em modelos ocultos de Markov para implementação de sistemas de reconhecimento de voz é justificada pois este é um dos métodos mais utilizados e portanto possui uma documentação a respeito bastante extensa.

As ferramentas HTK facilitam e otimizam a implementação de modelos ocultos de Markov para utilização em sistemas automáticos de reconhecimento de voz. Seu estudo em conjunto com a teoria de HMMs permite a implementação de sistemas de reconhecimento e voz de grande acurácia.

REFERÊNCIAS

RABINER, L. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. **Proceedings of the IEEE**, v. 77, n. 2, p. 257-286, 1989.

RABINER, L.; JUANG, B. H. **Fundamentals of Speech Recognition**. Upper Saddle River: Prentice-Hall, 1993. 507 p.

YOUNG, S. et al. **The HTK Book (for HTK Version 3.4)**. [S.l.]: Cambridge University Engineering Department, 2009. 384 p. Disponível em: <http://htk.eng.cam.ac.uk/prot-docs/htk_book.shtml>. Acesso em 5 mai. 2009.

Softwares utilizados:

TMH KTH. Wavesurfer, 2005, version 1.8.5
Dispnível em: <http://www.speech.kth.se/wavesurfer/> Acesso em 20/06/2009

ActiveState. Active Perl for Windows, version 5.10
Dispnível em: <http://www.activestate.com/activeperl/> Acesso em 05/05/2009