Referências Bibliográficas

[1] Rabiner, L., and B. H. Juang. Fundamentals of speech recognition, Prentice Hall, 1993.

[2] Lara, M. J. P., L. V. Grande, y J. A. S. Sánches, Teoría y aplicaciones de reconocimiento automático del habla, publicação – Comunicaciones de Telefónica, Investigacion y Desarrolo, n.º 3, Junho 1991.

[3] Nicola, J. e U. Infante, Gramática contemporânea da língua portuguesa, Scipione, 1998.

[4] Bispo, S. C., e A. Alcaim, Fundamentos de reconhecimento de voz, Publicação – CETUC-DID-01/95, Setembro de 1995.

[5] Bispo, S. C. "Reconhecimento de voz contínua para o português utilizando modelos de Markov escondidos," Tese de doutorado, Dezembro de 1997.

[6] Lee, K. F., "Context dependent phonetic hidden Markov models for speaker independent continuous speech recognition," IEEE Trans. Acoust., Speech, Signal Process., vol.38, n.º 4, April, 1990.

[7] Solewicz, J. A., "Síntese de voz a partir de texto para o português do Brasil," Dissertação de mestrado, Agosto de 1993.

[8] Bahl, L. R., et al. "Acoustic Markov models used in the Tangora speech recognition system," ICASSP, New York, April, 1988.

[9] Chow, I.. Y., R. Schwartz, et al. "The role of word-dependent coarticulatory effects in a phoneme based speech recognition system," ICASSP, Tokyo, April, 1986.

[10] Papoulis, A., Probability, Random Variables, and Stochastic Processes, McGraw Hill, 1991.

[11] Rabiner, L. R., "A tutorial on hidden Markov models and selected applications in speech recognitions," Proceedings of the IEEE, vol.77, n.º2, pp. 257-286, February, 1989.

[12] Gales, M. J. F., Model-based techniques for noise robust speech recognition, PH. D. Thesis, Setember, 1995.

[13] Baum, L. E. and G. R. Sell, "Growth functions for transformations on mainfolds," Pac. J. Math., vol. 27, n.º 2, pp. 211-227, 1968.

[14] Peinado, A. M., V. Sánchez, J. L. Pérez-Córdoba, e A. Torre, "HMM –based channel error mitigation and its application to distributed speech recognition," Speech Communication, vol.41, pp. 549-561, March, 2003.

[15] Kim, H. K., S. H. Choi, e H. S., Lee, "On approximating line spectral frequencies to LPC cepstral coefficients," IEEE Trans. Speech and Audio Processing, vol. 8, pp. 195-199, March, 2000.

[16] Kim, H. K., and R. V. Cox, "A Bitstream-Based Front-End for Wireless Speech recognition on IS-136 Communications System," IEEE Trans. Speech and Audio Processing, vol. 9, pp. 558-568, July, 2001.

[17] Santos, D. A. O., Reconhecimento de voz em presença de ruído, Dissertação de mestrado, julho de 2001.

[18] Ghitza, O., "Auditory Models and Human Performance in Tasks Related to Speech Coding and Speech Recognition," IEEE Trans. Speech and Audio Processing, vol. 2, pp. 115-132, January, 1994.

[19] Gajic, B., and K. K. Paliwal, "Robust Speech Recognition using Features based on Zero Crossings with Peak Amplitude," ICASSP, Hong Kong, April, 2003.

[20] Kabal, P., "ITU-T G723.1 Speech Coder: A Matlab Implementation," Department of Electrical & Computer Engineering, McGill University, November, 2003.

[21] Mitra, S. K., Digital Signal Processing: A Computer-Based Approach, McGraw-Hill International Editions, 1998.

[22] Sotomayor, C. A. M., Realce de Voz Aplicado à Verificação Automática de Locutor, Dissertação de mestrado, Instituto Militar de Engenharia, 2003.

[23] Deller JR., J. R., J. H. L. Hansen and J. G. Proakis, Discrete-Time Processing of Speech Signals, IEEE Press., New York, 2000.

[24] Santos, D. A. O., Reconhecimento de voz em presença de ruído, Dissertação de mestrado, Pontifícia Universidade Católica do Rio de Janeiro, 2001.

[25] Rabiner, L., and M. Samur, Technical report, The Bell System Technical Journal, February, 1974.

[26] Lima, C. B., Sistemas de verificação de locutor independente do texto baseados em GMM e AR-vetorial utilizando PCA, Dissertação de mestrado, Instituto Militar de Engenharia, 2001.

[27] Kim, D.-S., S.-Y. Lee, e R. M. Kil, "Auditory Processing of Speech Signals for Robust Speech Recognition in Real-World Noisy Environments," IEEE Trans. Speech and Audio Processing, vol. 7, pp. 55- 69, January, 1999.

[28] Stevens, S. S., and J. Volkman, "The realtion of pitch of frequency: A revised scale", Am. J. Psychol, vol.53, pp.329-353, 1940.

[29] Ohshima, Y., "Environmental Robustness in Speech Recognition using Physiologically-Motivated Signal Processing," PH. D. Thesis, Carnegie Mellon University, Pittsburgh, Pennsylvanya, December, 1993.

[30] Oppenheim, A. V., e D. H., Johnson, "Discrete Representation of signals," Proc. IEEE, vol.60, pp.681- 691, June, 1972.

[31] Wölfel, M., J., McDonough, e A., Waibel, "Minimum Variance Distortionless Response on a Warped Frequency Scale," Eurospeech, Geneva, 2003.

[32] Kleijn, W. B., e K. K. Paliwal, "Speech Coding and Syntheis", Amsterdam, The Netherlands: Elsevier, 1995.

[33] Choi, H. S., H. K., Kim, e H. S., Lee, "Speech recognition using quantized LSP parameters and their transformations in digital communication", Speech Communication, vol.30, pp. 223- 233, 2000.

[34] Gurgen, F. S., S., Sagayama, e S., Furui, "Line spectrum frequency-based distance measures for speech recognition," ICSLP, Kobe, Japan, November, 1990.

[35] Milner, B., S. Shemnani, P. Walther and T. Twell, "Robust Distributed Speech Recognition across IP Networks," Proc. IEE Colloquium ISDS, pp.6/1-6/6, 1999.

[36] Milner, B., and S. Shemnani, "Robust Speech Recognition over IP Networks," ICASSP, Istanbul, June, 2000.

[37] Coqueiro, V. M. e A. Alcaim, "Efeito da Natureza e Quantidade de Atributos em Reconhecimento Automático de Voz Independente do Locutor," Congresso Brasileiro de Automática, Natal, Setembro de 2002.

[38] Lilly, B. T. and K. K. Paliwal, "Effect os Speech Coders on Speech Recognition Performance," ICSLP, Philadelphia, October, 1996.

[39] Milner, B., "A Comparison of Front-End Configurations for Robust Speech Recognition", ICASSP, Orlando, May, 2002.

[40] Hanson, B. A., and T. H. Applebaum, "Robust speaker-independent word features using static, dynamic and acceleration features," ICASSP, Albuquerque, April, 1990.

[41] Milner, B. P., "Inclusion of temporal information into features for speech recognition," ICSLP, Philadelphia, October, 1996.

[42] Young, S., G. Evermann, T. Hain, D. Kershaw, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev and P. Woodland, The HTK Book (for HTK Version 3.2.1), December 2002.

[43] Gallardo-Antolín, A., F. Díaz-de-Maria and F. Valverde-Albacete, "Avoiding Distortion due Speech Coding and Transmission Errors in GSM ASR Tasks," ICASSP, Phoenix, March, 1999.

[44] Hwang, T.-H. and H.-C. Wang, "Adaptation Scheme for Hidden Markov Models in Noisy Speech Recognition," Electronics Letters, vol.33, nº.4, pp.257-258, February, 1997.

[45] Choi, S. H., H. K. Kim, H. S. Lee and R. M. Gray, "Speech Recognition method using quantised LSP parameters in CELP-Type Coders," Electronics Letters, vol.34, nº.2, pp.156-157, January, 1998.

[46] ITU-T Recommendation G.723.1, Dual Rate Speech Coder for Multimedia Communications Transmitting at 5.3 and 6.3 kbit/s, March, 1996.

Apêndice

O conjunto de ferramentas HTK (HMM Toolkit) [42] pode ser dividido basicamente em quatro grupos:

1. Preparação dos Dados
2. Treino
3. Teste
4. Análise dos Resultados

A Fig. A.1 procura evidenciar tal divisão e explicita o conjunto de programas que compõe o HTK. Ressalta-se, contudo, que a utilização de todos eles não é obrigatória, pois depende do grau de complexidade do sistema reconhecedor que se deseja implementar. Isto é ratificado a seguir, quando são apresentadas as etapas necessárias à obtenção das taxas de acerto do sistema reconhecedor simples – reconhecimento de palavras isoladas independentes do locutor.

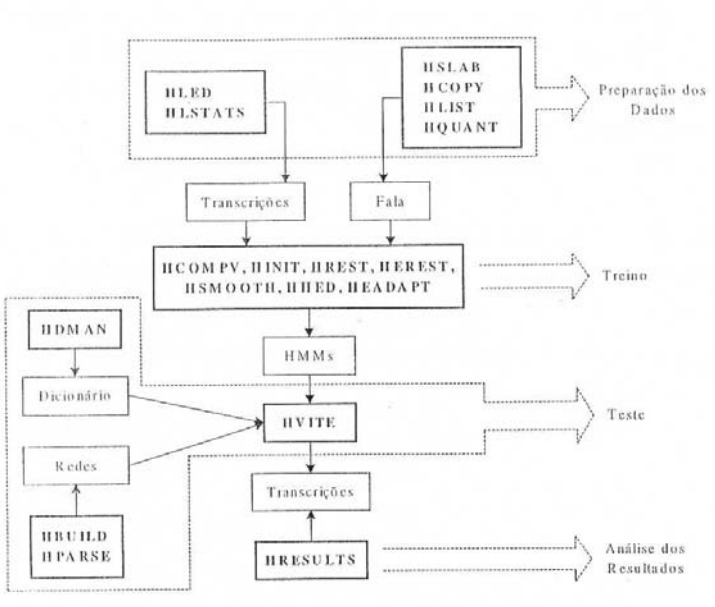

Figura A.1 – Conjunto de ferramentas HTK

A.1.
Preparação dos Dados

Cumpre salientar que, embora o HTK disponha de uma ferramenta para gravação (HSLAB), optou-se por utilizar as locuções gravadas a partir do programa CreativeWaveStudio®.

A detecção dos pontos terminais (endpoints) também foi efetuada fora do ambiente HTK, por um programa implementado em MATLAB®, conforme algoritmo apresentado no Capítulo 4, gerando arquivos com extensão "wav".

Para a extração dos atributos, última etapa da preparação dos dados, não foi utilizado o HTK, pois os atributos que se deseja utilizar, o HTK não os implementava, por se tratarem de atributos extraídos de forma muito peculiar, como apresentado no Capítulo 4.

Como pode-se observar, nenhum componente do grupo preparação dos dados do HTK foi utilizado nesta dissertação. Porém como foi utilizado o HTK para efetuar todas as demais operações de reconhecimento, foi necessário implementar uma interface entre o mesmo e o MATLAB®, o que cabe ser descrito em preparação dos dados.

Esta interface será feita gerando-se os arquivos de atributos no formato do HTK a cada locução a ser reconhecida.

Os arquivos de atributos a serem utilizados pelo HTK tem o formato que consiste em uma seqüência de atributos precedida por um cabeçalho. Cada atributo é um vetor de 4 bytes em formato *float point.*

O cabeçalho do arquivo de atributos do HTK, tem 12 bytes de comprimento e contém as seguintes informações:

- **nSamples** (4-Bytes no formato *integer*) – armazena a informação de quantas amostras de vetores de atributos existem neste arquivo de reconhecimento;
- **sampPeriod** (4-Bytes no formato *integer*) – armazena o período entre amostras em unidades de 100 ns – ex: para um período de 10 ms este parâmetro deverá assumir o valor de 100000 (um dos valores usado nesta dissertação);

- **sampSize** (2-Bytes no formato *integer*) – comprimento do vetor de atributos em bytes – ex: para um vetor de 20 atributos (10 componentes e 10 derivadas) deverá assumir o valor de 20x4 = 80;
- **parmKind** (2-Bytes no formato *integer*) – indica o tipo de atributo que estará sendo armazenado neste arquivo. Recomenda-se consultar o manual do HTK [42] para obter os valores de preenchimento – ex: para um atributo definido pelo usuário em conjunto com suas respectivas derivadas de primeira ordem, consultando o manual, obtém-se o valor 9 pelo atributo definido pelo usuário e 256 pela primeira derivada do atributo, sendo assim o valor do parmKind será 256+9 = 265.

Não foi colocado exemplo para o parâmetro **nSamples** do cabeçalho pois o mesmo variará a cada nova locução.

Em seguida ao cabeçalho seguem os atributos obtidos, no caso, pelo MATLAB®, sendo que os mesmos devem ser colocados sempre em seqüência, a saber: um vetor de atributos, a sua respectivas componentes de energia, derivada à primeira e a derivada à segunda, omitindo os que não forem utilizados (ex: no caso dessa dissertação estarão presentes apenas o vetor de atributos e a primeira derivada).

Um possível código de MATLAB® que implementa a escrita deste arquivo de atributos de reconhecimento do MATLAB® é apresentado na função, onde o mesmo é capaz de armazenar vetores de atributos e suas derivadas de primeira ordem:

```
function gera_arq_par_htk(arquivo1,periodo,tipo_parametro,parm1,parm2)
%gera arquivos de parametros do htk
[linhas1,colunas1]=size(parm1);
[linhas2,colunas2]=size(parm2);
a(1,1)=linhas1; % número de amostras
a(2,1)=periodo;
b(1,1)=4*(colunas1+colunas2);
b(2,1)=tipo_parametro;
aux=zeros(linhas1,(colunas1+colunas2));
aux(:,1:colunas1)=parm1;
aux(:,(colunas1+1):(colunas1+colunas2))=parm2;
[linhas3,colunas3]=size(aux);
c=zeros((linhas3*colunas3),1);
u=0;
```

```
for i=1:linhas3;
   for j=1:colunas3;
      u=u+1;
      c(u,1)=aux(i,j);
   end;
end;
fidesc=fopen(arquivo1,'ab');
fwrite(fidesc,a,'2*int32');
fwrite(fidesc,b,'2*int16');
fwrite(fidesc,c,'float32');
fclose(fidesc);
```

Onde os dados a serem fornecidos a esta função são:

- **arquivo1** – nome com que se deseja salvar o arquivo, o mesmo tem que terminar em ".mfc" (tem que ser colocado o caminho completo na estrutura de diretórios do computador onde se quer salvar o arquivo) ;
- **periodo** – mesmo valor da variável **sampPeriod** do cabeçalho do HTK definida anteriormente;
- **tipo_parametro** – mesmo valor da variável **parmKind** do cabeçalho do HTK definida anteriormente;
- **parm1** – matriz contendo os valores dos atributos, onde cada linha contém um vetor de atributos;
- **parm2** - matriz contendo os valores das derivadas à primeira dos atributos, onde cada linha contém um vetor de derivada dos atributos da matriz **parm1**.

Com os dados preparados, passa-se a etapa de treinamento das HMMs, o que será tratado na Seção A.2.

A.2. Treino

Esta etapa é responsável por gerar os modelos, utilizando as ferramentas **HINIT** e **HREST**, no caso de reconhecimento de palavras isoladas independentes do locutor, e envolve cinco passos:

1º Passo: Criação de um arquivo texto simples contendo o protótipo do modelo a ser gerado. Ele tem que ser salvo sem extensão e deve apresentar o seguinte conteúdo (o que está em itálico deve ser substituído por um valor válido):

~o <VecSize> *Número de coeficientes ex:20 <Tipo de coeficiente - ex: USER_D>*

~h "*nome dado ao presente arquivo (arquivo protótipo) ex: prot_5estados_3gaus_USER_D*"

<BeginHMM>

<NumStates> *número de estados (NS) ex:7*

<State> 2 <NumMixes> *número de misturas neste estado (MI) ex:3*

<Mixture> 1 *Probabilidade desta mistura (P), tem que totalizar 1 somando todas as misturas do estado ex: 0.3*

<Mean> *dimensão do vetor média (M) ex:20*

0.0 0.0 ... 0.0

<Variance> *considerando-se uma matriz de covariâncias diagonal, corresponde a ordem desta (C) ex:20*

1.0 1.0 ... 1.0

⋮

<Mixture> *MI P*

<Mean> *M*

0.0 0.0 ... 0.0

<Variance> *C*

1.0 1.0 ... 1.0

<State> 3<NumMixes> *MI*

<Mixture> 1 *P*

<Mean> *M*

0.0 0.0 ... 0.0

<Variance> *considerando-se uma matriz de covariâncias diagonal, corresponde a ordem desta (C) ex:20*

1.0 1.0 ... 1.0

⋮

<Mixture> *MI P*

<Mean> *M*

0.0 0.0 ... 0.0

<Variance> *C*

1.0 1.0 ... 1.0

⋮

<State> *NS-1*<NumMixes> *MI*

<Mixture> 1 *P*

<Mean> *M*

0.0 0.0 ... 0.0

<Variance> *considerando-se uma matriz de covariâncias diagonal, corresponde a ordem desta (C) ex:20*

1.0 1.0 ... 1.0

⋮

<Mixture> *MI P*

<Mean> *M*

0.0 0.0 ... 0.0

<Variance> *C*

1.0 1.0 ... 1.0

<TransP> ordem da matriz de transição de probabilidades (coincide com o número de estados)

0.0 1.0 0.0 0.0 ... 0.0 (a primeira linha só pode ter o segundo termo em 1.0 os demais tem que ser 0.0)

0.0 0.0 ... 0.0 (esta linhas tem que somar 1)

⋮

0.0 0.0 ... 0.0 (esta linhas tem que somar 1)

0.0 0.0 ... 0.0 (esta linha tem que ser sempre toda zerada)

<EndHMM>

A seguir apresenta-se, como exemplo, um protótipo criado com o nome "prot_5estados_3gaus_USER_D" para a definição de uma HMM esquerda-direita, com cinco estados, três gaussianas associadas a cada um deles e matrizes covariâncias diagonais. São considerados como atributos: 10 atributos definidos pelo usuário e 10 derivadas dos mesmos.

~o <VecSize> 20 <USER_D>

~h "prot_5estados_3gaus_USER_D"

<BeginHMM>

<NumStates> 5

<State> 2 <NumMixes> 3

<Mixture> 1 0.3

<Mean> 20

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> 20

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

<Mixture> 2 0.3

<Mean> 20

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> 20

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

<Mixture> 3 0.4

<Mean> 20

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> 20

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 <NumMixes> 3

<Mixture> 1 0.3

<Mean> 20

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> 20

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

<Mixture> 2 0.3

<Mean> 20

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> 20

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

<Mixture> 3 0.4

<Mean> 20

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> 20

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> 4 <NumMixes> 3

<Mixture> 1 0.3

<Mean> 20

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> 20

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

<Mixture> 2 0.3

<Mean> 20

```
    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> 20
    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
  <Mixture> 3 0.4
   <Mean> 20
    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> 20
    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> 5
  0.0 1.0 0.0 0.0 0.0
  0.0 0.6 0.4 0.0 0.0
  0.0 0.0 0.6 0.4 0.0
  0.0 0.0 0.0 0.6 0.4
  0.0 0.0 0.0 0.0 0.0
<EndHMM>
```

Diante do exposto, algumas observações devem ser feitas:

- O HTK considera densidades de probabilidade associadas aos estados internos. Por este motivo, a definição do protótipo inicia no estado 2 e termina no estado *NS*-1.
- A forma mais rápida e prática é definir todos os estados internos da mesma maneira:
  - ⇒ Médias nulas e
  - ⇒ Matrizes de covariância diagonais, com variâncias unitárias.
- A última linha de TransP tem que ser toda nula.

2º Passo: Criação de um arquivo de texto simples contendo os nomes dos arquivos de atributos das locuções a serem usadas na confecção de cada um dos modelos das unidades adotadas – tem que ser colocado o caminho completo na estrutura de diretórios do computador onde estão salvos os arquivos – (este passo não é obrigatório, mas é conveniente porque evita que o usuário escreva na linha de comando, um a um, os nomes dos arquivos). É denominado arquivo de *script* e deve ser salvo com a extensão ".scp" . Para gerar o modelo do dígito cinco com 70 % da base de locuções, por exemplo, o arquivo de script poderia ser denominado de "D5_35_loc.scp" e deveria apresentar o seguinte conteúdo:

D5R1LF01.mfc

⋮

D5R3LM35.mfc

3º Passo: Criação de um arquivo texto simples que conterá a configuração a ser utilizada pelo HTK no treinamento. Ele tem que ser salvo sem extensão e deve apresentar o seguinte conteúdo:

#Coding parameters (comentário de início de arquivo)
NATURALREADORDER=T
NATURALWRITEORDER=T

Estes dois parâmetros forçam que o HTK leia e escreva, na ordem natural, os *bytes* nos arquivos do computador, permitindo o mesmo trabalhar com os arquivos de atributos gerados pelo MATLAB®.

4º Passo: Inicialização de cada um dos modelos.

Hinit -C *nome do arquivo de configuração* –o *nome com o qual o modelo será salvo* –S *arquivo de script Protótipo* (tem que ser colocado o caminho completo na estrutura de diretórios do computador onde está salvo ou onde será salvo o arquivo)

Ex:

hinit -C config -o Cinco -S D5_35_loc.scp prot_5estados_3gaus_USER_D

5º Passo: Refinamento de cada modelo inicializado no item anterior.

Hrest -C *nome do arquivo de configuração* –S *arquivo de script Nome do modelo já inicializado* (tem que ser colocado o caminho completo na estrutura de diretórios do computador onde está salvo ou onde será salvo o arquivo)

Ex:

hrest -C config -S D5_35_loc.scp Cinco

A.3. Teste

Nesta etapa é feita a comparação das locuções de teste com os modelos criados anteriormente. Para tanto, utiliza-se a ferramenta **HVITE** e envolve cinco passos:

1º Passo: Criação do dicionário, o que é feito a partir de um arquivo de texto simples que deve ser salvo sem extensão, apresentando, por exemplo, o seguinte conteúdo para o reconhecimento de dígitos isolados independentes do locutor, em que a unidade fonética é a palavra:

Cinco Cinco
Dois Dois
Meia Meia
Nove Nove
Oito Oito
Quatro Quatro
Seis Seis
Sete Sete
Tres Tres
Um Um
Zero Zero

Cumpre salientar que este arquivo impõe algumas restrições:

⇒ Tem que estar organizado em ordem alfabética;

⇒ Deve ser acrescida uma linha em branco após a última linha de dados. No exemplo, acima, portanto, o arquivo apresenta 12 linhas;

⇒ As grafias das palavras do dicionário e dos nomes dos modelos gerados devem ser iguais, respeitando inclusive a condição de maiúsculas e minúsculas.

2º Passo: Criação da rede, que nada mais é do que uma representação para o HTK das seqüências possíveis – é necessário criar a rede para o reconhecimento

de palavras isoladas, mas sua utilização faz mais sentido no reconhecimento de palavras conectadas. É composta por duas fases:

a) Criação de um arquivo de texto simples que deve ser salvo sem extensão, apresentando como conteúdo as seqüências possíveis.

Como exemplo, a seguir, apresenta-se o arquivo "Digitos", e novamente recomenda-se atenção a grafia:

```
(
 Zero | Um | Dois | Tres | Quatro | Cinco | Seis | Sete | Oito | Nove | Meia
)
```

b) Criação propriamente dita da rede, através do programa **HPARSE**.

Hparse -C *nome do arquivo de configuração nome do arquivo criado no item a) nome que será dado à rede* (tem que ser colocado o caminho completo na estrutura de diretórios do computador onde está salvo ou onde será salvo o arquivo)

Ex:

hparse -C config digitos rede

3º Passo: Criação de uma lista com os nomes dos modelos gerados. É um arquivo de texto simples salvo sem extensão. O arquivo "Lista" abaixo é apresentado como exemplo:

```
Cinco
Dois
Meia
Nove
Oito
Quatro
Seis
Sete
Tres
Um
Zero
```

4º Passo: Criação de um arquivo de *script* contendo os nomes dos arquivos de atributos das locuções a serem reconhecidas – tem que ser colocado o caminho completo na estrutura de diretórios do computador onde estão salvos os arquivos.

Ex:

D0R1LF36.mfc

⋮

DMR3LM50.mfc

5º Passo: Reconhecimento propriamente dito.

Hvite -C *nome do arquivo de configuração* -w *nome da rede criada* –S *nome do arquivo de script criado no item anterior* –i *nome do arquivo que armazenará os resultados nome do dicionário lista com os nomes dos modelos* (tem que ser colocado o caminho completo na estrutura de diretórios do computador onde está salvo ou onde será salvo o arquivo)

Ex:

hvite -C config -w rede -S teste.scp -i resultados.mlf dicionario lista

A.4. Análise dos resultados

É responsável por apresentar na tela as taxas de acertos no reconhecimento, utiliza a ferramenta Hresults e envolve dois passos:

1º Passo: Criação de um arquivo de texto simples, contendo os nomes dos arquivos testados (trocando a extensão de MFC para LAB) e os modelos a que cada um deve corresponder para que seja considerado acerto no reconhecimento – tem que ser colocado o caminho completo na estrutura de diretórios do computador onde estão salvos os arquivos. Como exemplo, apresenta-se o conteúdo do arquivo “relatorio.mlf”:

#!MLF!#

"D0R1LF36.lab”

Zero

⋮

“DMR3LM50.mfc”

Meia

2º Passo: Apresentação das taxas.

Hresults -C *nome do arquivo de configuração –I arquivo de referências criado no passo anterior lista com o nome dos modelos Arquivo onde os resultados foram salvos* (tem que ser colocado o caminho completo na estrutura de diretórios do computador onde está salvo ou onde será salvo o arquivo)

Ex:

hresults -C config -I relatorio.mlf lista resultados.mlf

O relatório será impresso na tela na seguinte forma:

========== HTK Results Analysis ==========

Date: Data e hora em que foi solicitada a exibição do relatório

Ref : Nome do arquivo de referências

Rec : Nome do arquivo onde foram salvos os resultados

---------- Overall Results ----------

SENT: %Correct=*Porcentagem de acertos* [H=*número de acertos*, S=*número de substituições*, N=*número total de locuções testadas*]

WORD: %Corr= *Porcentagem de acertos*, Acc= *Porcentagem de acertos* [H= *número de substituições*, D= *número de deleções*, S= *número de substituições*, I= *número de inserções*, N= *número total de locuções testadas*]

Como exemplo apresenta-se a seguir um relatório obtido a partir de um teste realizado com 990 locuções, que produziu uma taxa de acertos no reconhecimento de 81,52%.

========== HTK Results Analysis ==========

Date: Sat Feb 12 00:11:58 2005

Ref : relatorio.mlf

```
  Rec : resultados.mlf
---------- Overall Results ----------
SENT: %Correct=81.52 [H=807, S=183, N=990]
WORD: %Corr=81.52, Acc=81.52 [H=807, D=0, S=183, I=0, N=990]
==========================================================
```