TR-IT-0144

Preliminary experiment for speaker-independent speech recognition

DOUET Thomas

December 22, 1995

This work focuses on some points of the rapid speaker adaptation. A set of speakers is used to train a Speaker-Independant model, which number of parameters is reduced in order to make it faster. Than, this model is adapted to the current speaker to get a better recognition.

Contents

1 Speech Recognition

1.1 Hidden Markov Model (HMM)

[HMM]

1.1.1 Probability context

Notations:

- T: observation time length
- O: $O_1, O_2, \ldots O_T$ sequence of observations
- N: numbre of states in the model
- L: numbre of symbols
- S=s: set of states
- V=v: set of symbols
- A: transition distribution table $a_{ij} = P(s_{t+1} = j / s_t = i)$
- B: symbols'probability distribution vector $b_j(O_t) = P(O_t / s_t = j)$
- π: initial distribution
- $\lambda = (A, B, \pi)$
- M: maximum number of members per class
- C: maximum number of classes
- P: total number of speakers

Assumption the probability of the output at time t only depends on the state at time t.

Goal $P(O/\lambda)$

Recognition of a word: $arg\max_i P(w_i/O)$ where w_i is a word of the dictionnary.
And

$$P(w_i/O) = \frac{P(O/w_i)P(w_i)}{P(O)}$$

(Bayes)

w_i is modelised by the sequence X=1,2,2,...knowing the model M

$$P(O, X/M) = a_{12}b_2(O_1)a_{22}b_2(O_2)\cdots$$

...but X is unknown.

So the aproximation used is

$$P(O/M) = \sum_X a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(O_t) a_{x(t)x(t+1)}$$

... where x(0) and x(T+1) are fixed.

Or even

$$P(O/M) = \max_X \left\{ a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(O_t) a_{x(t)x(t+1)} \right\}$$

1.1.2 Basic algorithms

Forward-Backward Forward

Let's set $\alpha_t(i) = P(O_1, \cdots O_t, s_t = i/\lambda)$

1. $\forall i \ \alpha_1(i) = \pi_i b_i(O_1)$

$$with \quad \pi_i = \frac{1}{N_I} \ if \ i \varepsilon S_I$$
$$\pi_i = 0 \ otherwise$$

... where S_I is the set of possible beginning states.

2. for t=2 to T and $\forall j \varepsilon S \ \ \alpha_t(j) = [\sum_i \alpha_{t-1}(i) a_{ij}] \, b_j(o_t)$
3.

$$P(O/\lambda) = \sum_{i \varepsilon S_F} \alpha_T(i)$$

... where S_F is the set of finishing states.

Backward

Let's set $\beta_t(i) = P(O_{t+1} \cdots O_T / s_t = i, \lambda)$

1.

$$\beta_T(i) = \frac{1}{N_F} \ \forall i \varepsilon S_F$$
$$= 0 \ otherwise$$

2. for t=T-1 to 1 and $\forall j \ \beta_t(j) = [\sum_i a_{ij} b_i(O_{t+1}) \beta_{t+1}(i)]$
3.

$$P(O/\lambda) = \sum_{i \varepsilon S_I} \pi_i b_i(O_1) \beta_1(i)$$

Viterbi

1. $\forall i\ \delta_1(i) = \pi_i b_i(O_1), \quad \psi_1(i) = 0$
2. for t=2 to T, $\forall j$

$$\begin{cases} \delta_t(j) &= \max_j\{\delta_{t-1}(i)a_{ij}\}b_j(O_t) \\ \psi_t(j) &= arg\max_i\{\delta_{t-1}(i)a_{ij}\} \end{cases}$$

3.

$$P^\star = \max_{s\epsilon S_F}\{\delta_T(s)\}$$

$$s_T^\star = arg\max_{s\epsilon S_F}\{\delta_T(s)\}$$

4. Path: Backtracking
   - $s_t^\star = \psi_{t+1}(s_{t+1}^\star)$
   - $P^\star = \max_{x\epsilon X}\{P(O, x/\lambda)\}$

     ... where X is the set of all the possible sequences x of states.

Baum-Welch

- A-posteriori probability of transition:

$$\begin{aligned} \gamma_t(i,j) &= P(s_t = i, s_{t+1} = j/O, \lambda) \\ &= \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{k\epsilon S_F}\alpha_T(k)} \end{aligned}$$

- ... of being in state i:

$$\begin{aligned} \gamma_t(i) &= P(s_t = i/O, \lambda) \\ &= \frac{\alpha_t(i)\beta_t(i)}{\sum_{k\epsilon S_F}\alpha_T(k)} \\ &= \sum_j \gamma_t(i,j) \quad if\ t < T \end{aligned}$$

re-estimation New parameters $\overline{\lambda}$:

- $$\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1}\gamma_t(i,j)}{\sum_{t=1}^{T-1}\gamma_t(i)}$$

- $$\overline{b}_j(k) = \frac{\sum_{t\epsilon O_t = v_k}\gamma_t(j)}{\sum_{t=1}^{T}\gamma_t(j)}$$

•

$$\overline{\pi}_i = \gamma_1(i)$$

and

$$P(O, s_t = i/\lambda) = \alpha_t(i)\beta_t(i) \;\Rightarrow\; \forall t \; P(O/\lambda) = \sum_i \alpha_t(i)\beta_t(i)$$

1.1.3 Semi-continuous HMM

The symbols v_s of the codebook are represented by a continuous probability fonction, for each state. Then:

$$b_{s_t}(x) = f(x/s_t) = \sum_{j=1}^{L} f(x/v_j, s_t) P(v_j/s_t)$$

... where L=card(V) is the number of symbols,
and f() is the probability density fonction associated with the symbols.

Furthermore, if we assume the independance of the states:

$$b_i(x) = \sum_{j=1}^{L} f(x/v_j) b_i(j)$$

1.1.4 Mixture Density Function

$$\begin{aligned} b_i(x) &= \sum_{k=1}^{M_i} c_{ik} b_{ik}(x) \\ &= \sum_{k=1}^{M_i} c_{ik} \sum_{j=1}^{L} f(x/v_j) b_{ik}(j) \end{aligned}$$

... where M is the maximum number of mixtures, c are the weights of the mixture components ($\forall i \;\; \sum_{k=1}^{M} c_{ik} = 1$) and $b_{ik}()$ is a probability fonction, as presented in 1.1.3.

so

$$\begin{aligned} P(X, S/\lambda) &= \prod_{t=1}^{T} a_{s_{t-1}s_t} b_{s_t}(x_t) \\ &= \sum_{k_1=1}^{M} \cdots \sum_{k_T=1}^{M} \left[\prod_{t=1}^{T} a_{s_{t-1}s_t} b_{s_t k_t}(x_t) c_{s_t k_t} \right] \\ &= \sum_{K \varepsilon \{1 \cdots M\}^T} \prod_{t=1}^{T} a_{s_{t-1}s_t} c_{s_t k_t} b_{s_t k_t}(x_t) \\ &= \left(\prod_{t=1}^{T} a_{s_{t-1}s_t} \right) \sum_{K \varepsilon \{1 \cdots M\}^T} \left(\prod_{t=1}^{T} c_{s_t k_t} b_{s_t k_t}(x_t) \right) \end{aligned}$$

... where K is the T dimensions vecteur of the k_t.

1.1.5 Simplification

A good (as far as the speed of the computation is concerned) way to approximate this probability is to keep only the most significant mixtures:

$$b_i(x) = \sum_{k \epsilon \eta_{ix}} c_{ik} b_{ik}(x)$$

1.2 Hidden Markov Network (HMnet)

[ICASSP92]
The HMnet is produced by the Successive State Spliting (SSS) algorithm.

1.2.1 SSS cell

Each cell is composed of:

- cell number (state number)
- class of possible contexts
- lists of the preceding and following states
- parameters of the output probability
- probability of state transition

1.2.2 SSS algorithm

1. The initial model is a single cell S(0) with a diagonal covariance of two multi-dimension gaussians.
2. The cell S(I) is splited into S'(I) and S(J), where

$$I = arg \max_i \left\{ d_i = \sum_k^K \frac{\sigma_{ik}^2}{\sigma_{Tk}^2} n_i \right\}$$

and

$$\sigma_{ik}^2 = \lambda_{i1}\sigma_{i1k}^2 + \lambda_{i2}\sigma_{i2k}^2 + \lambda_{i1}\lambda_{i2}(\mu_{i1k} - \mu_{i2k})^2$$

with λ_{ix} weight coef, μ_{ixk} k^{th} means, σ_{ixk}^2 k^{th} variances, n_i training data, σ_{Tk} k^{th} variance of the whole data.
NB: if the cell is mono guaussian, σ_{ik}^2 is it's variance.

3. Spliting
   - Spliting on the context:

$$P_C = \max_j \sum_L \max\{p_m(y_{jL}), p_M(y_{jL})\}$$

where j is a context factor, y_{jL} is a part of the data containing the element e_{jL}, L^{th} element of the factor j.
$p_m(y_{jL})$ is the total likelyhood when y_{jL} is on the path of S'(m). idem p_M.
Then e_{jL} is put in the context of the cell which maximises the likelihood.

- Spliting on temporal factor:
  S'(I) and S(J) are put on the same path in an order that maximises the likelihood.

The spliting way that makes the likelihood maximum is used.

4. Retraining to make S'(I) and S(J) with two gaussians.
5. When the final number of cells in reached, reshape each cell into a single gaussian state.

1.3 Speaker Independant Modeling and Adaptation

1.3.1 Speaker Independant vs Speaker Dependant Modeling

The advantage of the speaker-dependant model is that it requires only a few training data and achieves a good recognition rate. The advantage of a speaker-independant model is that it can be used for any speaker, what is usely the goal of a voice recognition system.
So, a large part of the present research is related to finding a speaker independant model with good recognition results and needing only a small amount of data. The method discused here is using small speaker dependant HMnets to build a larger one, which aims to be speaker independant. The other goal of this method is to create a HMnet allowing rapid speaker adaptation.

1.3.2 Generation of the HMnet

Assumption: the model structure is independant of the speaker.
Consequently, the model structure is trained with specific data from one speaker, with the Speaker-Independant-SSS algorithm. The speakers' variations are represented through the mixture components.

1.3.3 Modeling

The basic idea is to train many speaker-dependant HMnets by different speakers, all the HMnets having the same structure; then to cluster, for each state, the mixture components of the different speakers, and then compose the mixtures of each cluster, and mix the clusters of each state.

Training of the basic HMnets These HMnets are speaker-dependant nets, built on a single model. They need only a small amount of data per speaker. They only have few parameters, are single multidimension gaussian nets. For each speaker, only the states for which we have data are trained (taken into account).

Clustering For each state, the mixtures of the different speakers are clustered down to a maximum number of clusters (not necessary reached). The clustering is achieved according to the Bhattacharyya distance:
The distance between the two gaussian distributions
$b_1 = \aleph(\mu_1, S_1) \quad b_2 = \aleph(\mu_2, S_2)$ is:

$$\vartheta(b_1, b_2) = \frac{1}{8}(\mu_1 - \mu_2)^t \left(\frac{S_1 + S_2}{2}\right)^{-1} (\mu_1 - \mu_2) + \frac{1}{2} ln \frac{|\frac{S_1+S_2}{2}|}{|S_1|^{\frac{1}{2}}|S_2|^{\frac{1}{2}}}$$

Composition For each cluster, a single gaussian distribution is created by the composition of the gaussian distributions of the speakers of the cluster.
For the cluster l of the state i:

$$\mu_i^l = \sum_{k \varepsilon C_l} \omega_{ik} \mu_{ik}$$

$$S_i^l = \sum_{k \varepsilon C_l} \omega_{ik} S_{ik} + \sum_{k \varepsilon C_l} \omega_{ik} (\mu_{ik} - \mu_i^l)^2$$

$$\omega_{ik} = \frac{n_{ik}}{\sum_{k \varepsilon C_l} n_{ik}}$$

... where ω_{ik} are the weights, and n_{ik} are the amounts of data used to train the state k of the speaker i (the more a mixture is trained, the more it is reliable, and so, the more it is weighted in the sum).

Mixing Finaly, a single gaussian mixture is generated for each state, by mixing the different clusters'mixtures.

$$b_i = \sum_{l=1}^{L} \Omega_{il} \aleph(\mu_i^l, S_i^l)$$

$$\Omega_{il} = \frac{\sum_{k \varepsilon C_l} n_{ik}}{\sum_{k=1}^{N} n_{ik}}$$

1.3.4 Adaptation

The HMnet, result of the generation, is adapted to the speaker(s), the untrained states'mixtures'weights are estimated thanks to the correlation between the mixtures of the generated model. The lowest weighted mixtures are pruned, and the weights are recalculated.

Correlation The correlation is the probability that for one speaker, the mixture of a state belongs to one cluster, knowing that the mixture of another state belongs to a cluster. This probability is computed as the number of speakers for whom this relation occures, divided by the total number of occurences for the state.

Weight Estimation The weights of the untrained states are estimated as the product of the weights of the trained states weighted by the correlations.
ie the probability of occurance of one mixture knowing the trained model is computed as the product of the probabilities of this mixture knowing each trained mixture.
Hyp:

1. $\{Y_1, \ldots, Y_m\}$ untrained independent mixtures.
2. $\{X_1, \ldots, X_n\}$ trained independent, and conditionaly to Y_i $\forall i$ independant mixtures.

$$\begin{aligned}
P(Y_i/X_1, \ldots, X_n) &= \frac{P(Y_i, X_1, \ldots, X_n)}{P(X_1, \ldots, X_n)} \\
&= \frac{P(Y_i, X_1, \ldots, X_n)}{P(Y_i)} * \frac{P(Y_i)}{\prod_{j=1}^{n} P(X_j)} \\
&= P(X_1, \ldots, X_n/Y_i) * \frac{P(Y_i)}{\prod_{j=1}^{n} P(X_j)} \\
&= \left[\prod_{j=1}^{n} \frac{P(X_j/Y_i)}{P(X_j)}\right] P(Y_i) \\
&= \left[\prod_{j=1}^{n} \frac{P(Y_i/X_j)}{P(Y_i)}\right] P(Y_i) \\
&= \left[\prod_{j=1}^{n} P(Y_i/X_j)\right] P(Y_i)^{1-n}
\end{aligned}$$

so the weight of the mixture l of the state i is:

$$\begin{aligned}
\omega_{il} &= \frac{P(Y_i = l/X_1, \ldots, X_n)}{\sum_{l=1}^{L} P(Y_i = l/X_1, \ldots, X_n)} \\
&= \frac{\prod_{j=1}^{n} P(Y_i = l/X_j)}{\sum_{l=1}^{L} \prod_{j=1}^{n} P(Y_i = l/X_j)}
\end{aligned}$$

Pruning The lowest weighted mixtures of each state are pruned. The pruning operation is stoped if the number of remaining mixtures goes under a minimum number, or if the weight of the lowest weighted mixture is over a threshold.

2 Modeling

2.1 General Algorithm

First, the HMnet is created, with the SSS algorithm, adapted to a speaker. We assume that this structure would have been the same whoever the speaker would

have been. The same structure is used to train speaker dependant HMnets for each speaker of the database. Then, for each state, the mixture component of each speaker is extracted. The clustering is done state by state, on the pool of speakers'mixtures.
The number of states of the model HMnet can be chosen, in the rest of this document, if it is not specified, it is fixed to 400. Diagonal covariance mixtures are used, since, for a same amount of data, they achieve better recognition result. The dimension of the mixture is D=34.

2.1.1 Creation of the table of distances

The first operation is to create the distance tables. For each state, for each couple of distributions, the Bhattacharyya distance is calculated as explained in 1.3.3 Clustering. Since the gaussian distributions are diagonal covarianced, the computation is achieved by:

$$\vartheta(b_1, b_2) = \frac{1}{2}\sum_{d=1}^{D}\left[(\mu_1^d - \mu_2^d)^2(s_1^d + s_2^d) + ln\frac{s_1^d + s_2^d}{2} - \frac{1}{2}(lns_1^d + lns_2^d)\right]$$

$$\ldots\ x^d \textit{ being the } d^{th} \textit{ component of the vector } X.$$

2.1.2 Clustering

The clustering is done for each state. Only the trained mixtures are considered (the speakers who had training data for this state). They are clustered in a way that minimises the average distance inside the classes, up to a maximum number of classes.

Initialisation The maximum number of clusters is set to the minimum between the maximum required (option of the program) and the number of speakers for the state.
A stucture is used to represent the classes. Its elements are:

center_no Number of the center of the class, chosen among the members of the class for simplisity. The center is the member which minimises the distance between him and the other members.

count Number of members in the class. This number includes the center.

dist_total Total distance between the center and the other members of the class.

All the mixtures of the state are put in an only first cluster. The other classes are set unaffected by puting their center to the unused number -1. The center of this first class is calculated, and the final conditions are checked (for instance, in case of only one speaker for this state). Then, this class is splited, and the final conditions are checked again. Each time the final conditions are checked, if they are positive, the procedure exits.

Loop If the procedure has not exited during the initialisation part, it begins a loop without exiting conditions (for(;;)). This loop proceeds the following steps:

- Selection of the class to be splited. This class is the class which has the greatest distortion (the distortion being the total distance in the class).
- The selected class is splited:
  - Two members are chosen to be the centers of the new classes. They have to minimise the sum of the total distances of the two created classes. So, all couple of members in the class are tested.
  - The members are affected in the class from which center they are the closest (the memberships are stored in a table).
- The distortion is calculated (average distance: sum of the total distances of all the classes divided by the number of speakers). A loop rearanges the distribution of memberships and centers:
  - The memberships are reestimated: Keeping the centers, each speaker is put in the class to which center it is the closest.
  - The centers are reestimated: Keeping the belongings, for each class, the member who minimises the total distance is chosen as center.
  - The distortion is calculated. If it doesn't change or is greater than last record, the inside loop is exited.
- The final conditions are checked: positive when the number of classes reaches the maximum. If positive, the loop is exited.

Saving The informations of the class parameters and belongings are saved.

2.1.3 Composition

The mixtures of each cluster of each state are composed, not exactly as explained in 1.3.3 but in a more simply way:

$$\mu_i^l = \sum_{k \varepsilon C_l} \omega_{ik} \mu_{ik}$$

$$S_i^l = \sum_{k \varepsilon C_l} \omega_{ik} S_{ik} + \sum_{k \varepsilon C_l} \omega_{ik} \mu_{ik}^2 - (\mu_i^l)^2$$

2.2 Present work

2.2.1 Distortion

Calculation way The distortion was calculated as sum of the distances inside the class (total distance) and the average distortion is the sum of the total distances divided by the number of speakers.

This way of calculating the distortion has the advantage of being strictly decreasing during the spliting operation, since each split, the sum of the total distances of the two new classes is smaller than the total distance of the original class, and the recentering and reaffectation of thememberships can only decrease the total distance. The disadvantage of this calculation is that it will make the biggest classes be splited first, even if the speakers are close to each others. So, I've tried to use, as distortion of each class, the average distance inside the class, and as global distortion, the average distortion of the classes. It didn't show better results on few tests, and has the disadvantage of not being strictly dicreasing, so that the use of a threshold becomes difficult (see 2.2.4). But the fact that sometime the distortion increases is also an indication of the efficiency of the algorithm, and can be used as in 2.2.6 'Add when worse'.

Degree The distortion uses the distance between the speakers as a criterium for spliting and setting the belongings. As explained above, a class with many speakers close to each other is more likely to be splited than a class with fewer members, but far from each other. To this problem can be added the fact that a class with all members at an average distance from each other is more likely to be splited than a class with all its members very close to each other but one, far from the others.
One way to avoid this, is to use a high moment for the clustering of the speakers' distribution. Instead of using the crude distance, the distance at a high power can be used. The informations coming from the distance and the powered distance being different, both are used in a first attempt to find out which gives the best recognition result, the distortion rate being the rate between these two distances.
A new structure is used for the distortion of each class:

mean the mean distance is the total distance divided by the number of members in the class (*dist_total/count* of the class structure).

fifth the average powered distance, its name comes from its original degree (5).

rate this is the distortion rate calculated during the selection of the class to be splited (which will become this class). It's the distortion rate that would make *mean* and *fifth* at the same weight in the decision of spliting.

lmr the local mean rate is the average *rate* calculated on the existing *rates* .

Most of the parts of the clustering algorithm have to be changed to use this new feature, and one problem is to handle the difference of scale created by the use of different degrees. Following are the changed on the procedure of selection of the class to be splited.

2.2.2 Optimisation

As explained in 2.1.2 the belongings and centers are successively reestimated to minimise the distortion. In fact, this happens quite few times, and is not always good for the global distortion. In order to check the efficiency of this action, I tried to test without it (test *29*). The result confirms that it's better with.

2.2.3 Selection

Distortion This procedure selects the best class to split, according to the distortion. It returns the number of this class, if possible, and -1 otherwise.

- Initialisation of the loop: the first class which has more than one member is selected as the class to be splited. If there is no such class, the procedure exits returning the value -1.
- Beginning of the loop on the classes. The class with only one member are not further considered, and the distortion *rate* is set to 0.
  - The differences of distortion between the last registered 'split class' (class to be splited) and the current class are decomposed into exponential and fractional parts. If an exponential part is greater than the last recorded, the corresponding fractional part is put to 10.
    Let's set:

$$\begin{aligned} S &= (1 - Distortion\_rate) * frac\_mean \\ &\quad + Distortion\_rate * frac\_fifth \end{aligned}$$

    ... where frac_mean is the fractional part of the difference of the means.
  - The *rate* is calculated so that $S = 0$. It is added to a sum and a counter is incremented.
  - If $S > 0$ (*ie* the distortion is greater) the current class is recorded as new 'split class'.
- The *rate* is the sum divided by the counter, *ie* the average value for the classes (only if the value is meaningfull: between 0 and 1).
- The *lmr* is calculated, and the distortion rate is updated by:

$$\begin{aligned} Distortion\_rate &= Change\_rate * cdist[n].lmr \\ &\quad + (1 - Change\_rate) * Distortion\_rate\_mem \end{aligned}$$

  ... where *Change_rate* is a user parameter, *cdist* is the distortion structure, n is the indice of the class to be created and *Distortion_rate_mem* is the distortion rate at the beginning (user parameter).
- The 'split class' number is returned.

If we assume that $Distortion\_rate = 1$ (for simplicity, and because it gave a good result) and if me recall the equation of 1.1.4; the sum is done on the set $K \varepsilon \{1 \cdots M\}^T$. So,

$$\begin{aligned} K &= arg \min_{\wp(\{1 \cdots M\}^T)} \left\{ \frac{1}{P} \sum_{l=1}^{C} \sum_{k \varepsilon C_l} \vartheta(\aleph_k, \aleph_{c_l})^\gamma \right\} \\ &= arg \min_{\{\delta_1 \cdots \delta_C\}} \left\{ \frac{1}{P} \sum_{l=1}^{C} \sum_{k=1}^{P} \delta_l(k) \vartheta(\aleph_k, \aleph_{c_l})^\gamma \right\} \end{aligned}$$

... where $\wp$ is the set of the elements of $\{1 \cdots M\}^T$, c_l is the center of the cluster C_l, γ is the degree, δ is defined by:

$$\begin{aligned} with \quad \delta_l(k) &= 1 \; if \; k \varepsilon C_l \\ \delta_l(k) &= 0 \; otherwise \end{aligned}$$

Then, the equation of the 1.1.4 section can be written as follows:

$$\begin{aligned} P(X, S/\lambda) &= \left(\prod_{t=1}^{T} a_{s_{t-1}s_t}\right) \sum_{l_1=1}^{C_{s_1}} \sum_{k_1 \varepsilon C_{s_1}} \cdots \sum_{l_T=1}^{C_{s_T}} \sum_{k_T \varepsilon C_{s_T}} \left(\prod_{t=1}^{T} c_{k_t} b_{k_t}(x_t)\right) \\ &= \left(\prod_{t=1}^{T} a_{s_{t-1}s_t}\right) \sum_{l_1=1}^{C_{s_1}} \sum_{k_1=1}^{P} \cdots \sum_{l_T=1}^{C_{s_T}} \sum_{k_T=1}^{P} \left(\prod_{t=1}^{T} \delta_{l_t}(k_t) c_{k_t} b_{k_t}(x_t)\right) \end{aligned}$$

The procedure of clustering consists in finding the good functions δ and the good numbers of clusters C_s in order to minimise the distortion.

Distribution A new parameter is added to the distortion structure: the state distribution variance for each class:

$$\sigma_i = \frac{1}{N_i - 1} \sum_{j \varepsilon C_i} (\vartheta(c_i, j) - m_i)^2$$

... where $m_i = \frac{1}{N_i - 1} \sum_{j \varepsilon C_i} \vartheta(c_i, j)$, m_i is the average distance to the center, N_i is the number of members of the class i, c_i is the center of the class i, C_i is the set of the members of the class i.

This variance of the distribution of the members inside the classes is used instead of the distortion in the selection procedure (of the 'split class') *ie* the class with the greatest variance is chosen to be splited. The goal of spliting on the distribution variance instead of the distortion is the avoid the spliting of the homogene distributions.

Results

Degree The degree that gives the best recognition result seems to depend on the rest of the program (all the more as the number of mixture components depends on the degree when thresholds are used: see tests *69*, *70* and *73*). The tests *14* and *16* to *21* (the number of mixtures is constant) show an optimal degree of 8. But the variations of the recognition rates seem to indicate that the tests are not much relevant.

Selection The distortion and change rates give oposit results. The best results are achieved with a change rate of 0 (not used) as can be seen in the tests *14* and *15*. The inversion of the use of the change rate has also been tested (avoid the use of the two distortion components (*mean* and *fifth*) at the same time) but

gives a bad result (test *32*).
The distortion rate shows that it's better to use a high degree component as can be seen through the tests *8* to *14*, *63* & *66* & *73*, *70* & *79*. The test *81*, compared with *79*, confirms this. Only the tests *76* and *77* give different results.

Alternance The *mean* and *fifth* components are used in distinct parts: in the test *81*, the *mean* is used in all the program but the spliting which only uses the *fifth* . The test *82* computes the oposit. Both tests give worth results than when using only the *fifth* . The test *84* uses both components for the program but the selection, and only the *mean* for the selection, and it gives a far worse recognition rate. This tends to show that the *fifth* is the best component to use and that it's better to use the same criterium of distortion in the whole program.

Variance The tests *89*, *92*, *107*, *108*, *116* and *118* use the variance for the selection of the split class. The two first ones give better result than the spliting on the distortion, but the other ones, as compared with the tests *111*, *112*, *114*, *117* and *132* give results about the same.

2.2.4 Distortion threshold

The distortion threshold is used in the final check procedure to stop the clustering if the distortion is low enough.

Implementation The distortion has two components: the *mean* and the *fifth*. In order to limit the number of parameters, only one threshold is used. So, the two components have to be brought to a same scale. Furthermore, the initial distortion depends greatly on the state, since some states have a lot of speakers and some other ones have very few. Consequently, I used:

$$\begin{aligned}\chi \;=\; & -\log_{10}(vmean) * (1 - Distortion\_rate) \\ & -\log_{10}(vfifth) * Distortion\_rate/Degree\end{aligned}$$

...where *vmean* is the present *mean* divided by the initial one (calculated during the first step of the clustering).

Note that the initial distortion components can be nul if there are only two speakers for the state.
The spliting is stoped when the distortion indicator χ becomes greater than the threshold.

Results As the tests from *38* to *54* show it, the best recognition results are achieved when the threshold is not used. But it can be noticed also that, after a decrease (for small thresholds) the recognition rate increases, sometime up to a better value than the initial one (without threshold): tests *38* to *43*, and *49* to *54*.

This may indicate that the states which go easely over the threshold should be kept splited, probably because they are easy to split what means that the speakers are naturaly building classes. But the increase in the recognition result for high thresholds shows that it's good to stop spliting when the state has a distribution of speakers difficult to set into classes.

2.2.5 State distribution

The use of a higher degree shows good results of recognition. The effect of the use of a high moment for the spliting in the distribution of the speakers is to increase the variance of this distribution.

Parameter The parameter used is:

$$\Delta = \frac{1}{N} \sum_{i=1}^{N} |n_i - n|$$

with

$$n = \frac{1}{N} \sum_{i=1}^{N} n_i$$

... where N is the number of classes and n_i is the number is members if the class i.

The variance or the standard deviation could also be used as well.

Algorithm The spliting is stoped if Δ decreases after having increased. Some precautions have to be taken into account in order to avoid stoping at the beginning of the spliting.

Results The comparison of the tests *39* to *43*, with the tests *44* to *48* shows that it increases the recognition rate. The decrease of Δ probably happens because the classes are homogene and so the spliting creates classes with aproximativly the same number of members (because it minimises the total distance). So, as shown with the distortion threshold, it's better to stop the spliting when the distribution of the members in the classes is homogene.

2.2.6 Addition

The algorithm used for spliting the classes doesn't guaranty that an optimal repartition of the members will be reached. First because we don't exactly know what the meaning of optimal is, and secondly because the method may not lead to a global minimum of distortion. I tried to find a way to control the spliting by being able to go back. But keeping the history of the classes would require a lot of memory, and would only give the ability to go back on the same way. So I tried some different ways of adding classes, in different situations. My goal was to reach a distribution of the memberships that would minimise the distortion, and in many cases, I reached

it. But must of the distributions, even with a smaller distortion, achieved worse recognition results. So the basic conclusion of this part is that the distortion may not be the only parameter to take into account...

Addition only First, I tried to do only additions, what is to say, I took the opposit foot to the spliting.
The initial distribution is set to one class per speaker. Then, the classes are added down to the maximum number of classes. The addition method is to add the two classes which centers are the closest.
The result is bad as can be seen with the test *31*.

Add when worse This method deals with adding the two closest classes each time the split has produced a increase in the distortion. For that purpose, the distortion used is the alternative presented in the 2.2.1 Calculation way (the global distortion is the average of the class distortions). The distance between two classes is computed as the average distance between all the couples of members of the two classes:

$$D(i,j) = \frac{1}{n_i n_j} \sum_{m_i \varepsilon C_i} \sum_{m_j \varepsilon C_j} \vartheta(m_i, m_j)$$

...where C_i is the class i and m_i is a member is this class.

The test is *22*, and can be compared with *19*: a little bit worse.

Regular additions Here, additions are done the same way as for last example, but with the regularity of one per two splits. The result is even worse (test *23*). This shows that it's better to use the addition procedure only when the spliting method is not able to cluster efficiently (the efficiency being rather difficult to define as it depends on what is reliable in the distribution of speakers).

Addition with threshold The addition procedure for this and the following examples begins by choosing a class, and then by erasing it. The class chosen is the one which minimises the average distance of its members to the closest centers (but the center of the class itself).

$$J = arg \min_j \left\{ \frac{1}{n_j} \sum_{m_{ij} \varepsilon C_j} [c_{ij} = \min_{c \varepsilon \Theta_j} \vartheta(m_{ij}, c)] \right\}$$

...where n_j is the number of members of the class j, C_j is the class j and m_{ij} is a member of this class, Θ_j is the pool of the centers exepted the center of the class j.

The addition is done by puting each member m_{iJ} in the class c_{iJ}.
In this first example, additions are proceded not more than two spliting loops after each other, to avoid an infinit loop between the additions and splits, and only if the variation of the distortion is small:

$$(mean - p\_mean)/p\_mean + (fifth - p\_fifth)/p\_fifth < 0.5$$

... where mean is the *mean* component of the present distortion, and p_mean is the *mean* of the distortion calculated during the previous loop.

For most of the states, the additions will occure at the end of the spliting algorithm, when the number of classes is getting close to the maximum. So it usely adds classes that have been splited before, and does not help getting a better distribution. The result is bad, as can be seen in test *24*.

Jump In order to avoid the problem of adding classes that have been splited just before, the number of successive additions (one each spliting loop) is set to three (it was two in the last example) but when the added classes are the last splited ones, the addition is counted twice (as if it had been two succesive additions: $1 + jump$; with $jump = 1$). So, when the algorithm tends to split and add the same classes, the additions are stoped. This gives a slightly better result (test *25*).

Smoothing The problem of doing only additions is that it does not allow really to go back in the spliting, and try to find the 'best' distribution. It only manipulates the classes, and usely, it erases a class without reshuffling the whole distribution.
In this part, a 'smoothing' method is used to mix the classes on their edges. Some speakers are set in the class from which center they are the closest (but the center of the class they belong to) relatively to the distance to the center of their class. Three kinds of smoothing are used:

1. Smoothing 1 is changing the belonging only of the speaker who minimises the relative distance to another center.

$$K = arg \min_{c_k \varepsilon \Theta_j} \left\{ \frac{\vartheta(m_{ij}, c_k) - \vartheta(m_{ij}, c_j)}{\vartheta(m_{ij}, c_j)} \right\}$$

2. Smoothing 2 is doing the same but for the best member of each class.

3. Smoothing 3 is doing it for all the speakers (but the centers of the classes).

After each smoothing, the centers are recalculated, and then the belongings.
Here, the maximum number of succesive additions is set to 2, and $jump = 2$. Smoothing 1 is achieved every three loops, smoothing 2 every four loops, and smoothing 3 every addition. The recognition rate is better than the previous tests of additions (test *26* and *28*).

Deletion The smoothing 3 has a bad side effect: the creation of single member classes; when all the members of the class are changed to the closest classes, but no speaker is affected to this class. The problem is that this center won't move during the recenter procedure since no member belongs to its class, and then, with the checking of the memberships, all the former members of this class will come back to it, annulating the effect of the smoothing. So that this center can follow the rest of the class and be affected to another class, all the single member classes are deleted

after each smoothing 3. The speaker is put in the class from which center it is the closest. This method gives a recognition rate better than previous test (see test *27*). This algorithm has a strong effect on the distribution of the class number of members, since it cuts all the lowest classes, and usely such classes are numerous. Usely, after a short but strong increase, the Δ parameter introduced in 2.2.5 decreases. An attempt to modifiy the distribution according to the variation of Δ is the test *30*: each time the variation of Δ is under -40%, the single member classes are deleted. The result is a little bit worse than for the test *17*.

2.2.7 Double distance

Principe All the previous tests have been achieved using the Bhattacharyya distance as explained in 2.1.1. This distance is synthetising all the informations of 34 dimensions in one distance. But, the clustering is trying to build classes of speakers who have about the same speaking caracteristics. The information inside the 34 dimensions could be roughly used by doing a clustering according to all the dimensions, using a 34 dimensional distance. All the information would be available for the clustering, but the clustering algorithm would become rather complexe, all the more as it would require some knowledge about where the meaningful information is.
I splited the distance into a double dimension distance, in order to have a double way of spliting. The double distance is simply computed as Bhattacharyya distance on the first 17 dimensions, and on the others. The distance is composed by two halfs of the same kind:

- 1 : log power of wave form.
- 16 : capstrum coefficient.
- 1 : delta log power.
- 16 : delta-capstrum.

Tests First, each component of the distance was tested separatly: test *56* for the first and *57* for the second. Of course, the recognition results are worse than with the complete distance, but just a little bit worse for the first part (non delta). So I tried to use a *rate* between the two components (0.5 being an equal use of the two distances, which is equivalent to the normal distance) and set the *rate* to 1-*rate* every split loop. Tests for *rates* of 0.3 (*58*) 0.4 (*60*) and 0.45 (*61*) show a best result for 0.4, only 0.5% worse than with the full distance. The conclusion of this is that only a small part of what is used in the distance is really data, and the rest may unfortunately be noise.

2.2.8 New Composition

As can be seen with the comparison of 1.3.3 and 2.1.3, the implementation used a more simple calculation way, which is equivalent.

Basic implementation The implementation of 1.3.3 has been tried for the tests *97* which can be compared with the test *70* and gave about the same result. The tests *117*, *114* and *112*, as compared with *127* and *123*, show a result about the same. This shows that the implementation doesn't influence the result.

Quadratic weights The test *121* used for the variance:

$$S_i^l = \sum_{k \epsilon C_l} \omega_{ik}^2 S_{ik} + \sum_{k \epsilon C_l} \omega_{ik}^2 (\mu_{ik} - \mu_i^l)^2$$

The recognition rate is really bad, the problem being that the weights are not normalised anymore. So I used:

$$S_i^l = \sum_{k \epsilon C_l} \frac{n_{ik}^2}{\sum_{j \epsilon C_l} n_{ij}^2} S_{ik} + \sum_{k \epsilon C_l} \frac{n_{ik}^2}{\sum_{j \epsilon C_l} n_{ij}^2} (\mu_{ik} - \mu_i^l)^2$$

... where the n_{ik} are defined in 1.3.3.

As can be seen through the tests *125*, *134*, *131* and *128*, the result is as good as the old version.

Simplification As the best result is achieved with a simple composition, I tried even much more simple, in the test *129*:

$$S_i^l = \sum_{k \epsilon C_l} \omega_{ik} S_{ik}$$

and the result is just a little bit worse.

Variance of the sum As shown in 4.1.3, the variance of the sum can be used as variance of the gaussian. I tried this, with an error of implementation:

$$V = \sum_{i=1}^{C} \omega_i^2 S_i + \sum_{i=1}^{C} \sum_{j=1 \ j \neq i}^{C} \frac{\omega_i \omega_j}{\sqrt{2\pi}} \left(-\mu_i \mu_j + \frac{\mu_i S_j + \mu_j S_i}{2(S_i + S_j)} e^{-\frac{S_i S_j}{S_i + S_j} (\mu_i - \mu_j)^2} \right)$$

The test *122* shows a very bad recognition rate. As previously shown, quadratic weights can be used, tested in *126* but this doesn't improve much the recognition rate. One of the problem may be that the second part summing on both i and j has a quadratic number of members as compared to the old version or the first part. So I tried:

$$\begin{aligned} V \ = \ & \sum_{i=1}^{C} \frac{n_i^2}{\sum_{j=1}^{C} n_j^2} S_i \\ & + ln \left| \sum_{i=1}^{C} \sum_{j=1 \ j \neq i}^{C} \frac{n_i n_j}{\sum_{k=1}^{C} \sum_{l=1}^{C} n_k n_l} \cdot \frac{1}{\sqrt{2\pi}} \cdot \tilde{A}(i,j) \right| \end{aligned}$$

... where

$$\tilde{A}(i,j) = \left(-\mu_i \mu_j + \frac{\mu_i S_j + \mu_j S_i}{2(S_i + S_j)} e^{-\frac{S_i S_j}{S_i + S_j}(\mu_i - \mu_j)^2} \right)$$

but the result is not much better (test *130*) because the logarithm as a bad effect every time the sum is nul, which happens much often (untrained states). To avoid this, as we know the number of members, we can simply scale the second part by dividing it by the number of members:

$$\begin{aligned} V &= \sum_{i=1}^{C} \frac{n_i^2}{\sum_{j=1}^{C} n_j^2} S_i \\ &\quad + \frac{1}{C} \sum_{i=1}^{C} \sum_{j=1 \; j \neq i}^{C} \frac{n_i n_j}{\sum_{k=1}^{C} \sum_{l=1}^{C} n_k n_l} |\tilde{A}(i,j)| \end{aligned}$$

... where

$$\tilde{A}(i,j) = -\mu_i \mu_j + \frac{\mu_i S_j + \mu_j S_i}{\sqrt{2\pi}(S_i + S_j)} e^{-\frac{S_i S_j}{S_i + S_j}(\mu_i - \mu_j)^2}$$

The implementation error being corrected. The result is much better, but still a little bit worse than the old implementation (test *133*).
Other implementations have been tried with some small variations:

$$\begin{aligned} V &= \sum_{i=1}^{C} \frac{n_i^2}{\sum_{j=1}^{C} n_j^2} S_i \\ &\quad + \frac{1}{C} \left| \sum_{i=1}^{C} \sum_{j=1 \; j \neq i}^{C} \frac{n_i n_j}{\sum_{k=1}^{C} \sum_{l=1}^{C} n_k n_l} \tilde{A}(i,j) \right| \end{aligned}$$

... in the test *135* which doesn't give a better result.

$$\begin{aligned} V &= \sum_{i=1}^{C} \frac{n_i^2}{\sum_{j=1}^{C} n_j^2} S_i \\ &\quad + \frac{1}{C} \sum_{i=1}^{C} \sum_{j=1 \; j \neq i}^{C} \frac{n_i n_j}{\sum_{k=1}^{C} \sum_{l=1}^{C} n_k n_l} \tilde{A}(i,j)^2 \end{aligned}$$

... in the test *138* which gives a recognition rate a little better.

$$\begin{aligned} V &= \sum_{i=1}^{C} \frac{n_i^2}{\sum_{j=1}^{C} n_j^2} S_i \\ &\quad + \frac{1}{C} \sqrt{\sum_{i=1}^{C} \sum_{j=1 \; j \neq i}^{C} \frac{n_i n_j}{\sum_{k=1}^{C} \sum_{l=1}^{C} n_k n_l} \tilde{A}(i,j)^2} \end{aligned}$$

...in the tests *145*, *150* and *152* which show a result quite equivalent to the original implementation.

$$V = \sum_{i=1}^{C} \frac{n_i^2}{\sum_{j=1}^{C} n_j^2} S_i + \sqrt{\frac{1}{C} \sum_{i=1}^{C} \sum_{j=1 \; j \neq i}^{C} \frac{n_i n_j}{\sum_{k=1}^{C} \sum_{l=1}^{C} n_k n_l} \tilde{A}(i,j)^2}$$

...in the test *147* which shows a recognition rate a little worse than the original implementation.

2.2.9 Distortion as a product

Reason The distortion was used as a sum of the distances between the members in the class. The distance used is the Bhattacharyya distance (cf 1.3.3). This distance is a kind of distance between the means increased by the variances. This can be understood by the fact that the average distance between two random guaussian variables will be the distance between the means of the distributions. But, the greater the variances are, the more likely the random variables may occure to be further from each other.
The random variable being the probability of the utterance to be one of this speaker (for this state) the distance is also the distance between the speakers. The distance between two speakers can be seen as the inverse of the probability of the utterance to be one of the first speaker, knowing that it's an utterance of the second speaker (for instance, it can be the probability of the value of the random variable of the second speaker to occure as a value of the random variable of the first speaker):

$$P(\aleph_i/\aleph_j) = \frac{1}{\vartheta(\aleph_i;\aleph_j)}$$

Implementation For the selection of the class to be splited, only one component is used, since the degree wouldn't bring anything:

$$\prod_{k \varepsilon C_l} \vartheta(\aleph_k;\aleph_{c_l})$$

...where k is an element of the class l and c_l is the center of the class l.

For the general distortion,

$$\sqrt[P]{\prod_{k=1}^{P} \vartheta(\aleph_k;\aleph_{c_k})}$$

...where c_k is the center of the class to which k belongs.

Result The results of the tests *109*, *115*, *113* and *110* can be compared with the results of the tests *111*, *117*, *114* and *112*. The recognition rate is about the same.

3 Adaptation

3.1 General algorithm

The adaptation aims to adapt the network to the current speaker, by adapting the distribution and also by making the recognition faster.

3.1.1 Adaptation

The mixtures of the network are trained and adapted to the speaker. The Forward-Backward algorithm is used, with the training data of the current speaker. The counts of the training data per mixture are established and saved. This counts correspond to the sum of the probabilities of each mixture of each state during each recognition (training) phase:

$$c(i,j) = \sum_{\sigma \varepsilon S} p((s = i, m = j) \varepsilon X_\sigma / M, S)$$

... where s is a cell (state) m is a mixture, σ is a sample and S is the set of training data, X is a sequence (path) for σ, and M is the model.

3.1.2 Deletion

The parts of the HMnet corresponding to the silence phonemes and the rest of the HMnet are splited. The following algorithms apply to the non-silence part.

3.1.3 Trained weights

Normalisation The weights of each output are normalised if the total count is not too small (untrained mixture) *ie* the sum of the counts of the mixtures are compared with a threshold (user defined 'minimum counter') and if it's over the threshold, each mixture's weight is set to the count divided by the sum. Then the weight is updated by the maximum between this fraction and a minimum weight:

$$w_{ij} = \max\left(\frac{c(i,j)}{\sum_j c(i,j)}, min\_weight\right)$$

... where $c(i,j)$ is the count of the j^{th} mixture of the i^{th} output, w_{ij} is its weight, and min_weight is the minimum weight, defined by the user.

So, the probability that the mixture j of the state i appears in an utterance of the current speaker, knowing that the state i appears in the utterance, is:

$$\begin{aligned} P((s = i, m = j)/s = i) &= \frac{c(i,j)}{\sum_{k=1}^{C} c(i,k)} \\ &= w_{ij} \end{aligned}$$

and if it happens to be very small (under the threshold) the probability is set to a very small value ϵ for computation reasons. This small value doesn't change the normalisation according to the normal precision used in the calculation.

Zero Only the outputs which have a count over the threshold are taken into acount. The other are considered untrained, and their weight is set to zero.

- While the number of mixtures is over the maximum number, the lowest weighted mixture is set to 0 (it's weight). The weights of the other mixtures of the output are recalculated (normalisation).
- While the number of mixtures is over the minimum number and the count of the output is over the minimum counter, the least weighted mixture is chosen. If the weight of this mixture is greater than a ratio (user defined) divided by the number of remaining mixtures of the output, the loop is broken, otherwise, the mixture is set to zero.

3.1.4 Untrained weights

Setting First the outputs are checked to find which are trained (count greater than the minimum count) and which are not. Then all the correlations are increased by 1.
The calculation of the untrained mixtures' weights begins by choosing an untrained output that has more than one mixture: i.

$$w_{ij} = \prod_{k \epsilon S_{trained}} \frac{\sum_{l \epsilon \Lambda_k} w_{kl} * \zeta_{ijkl}}{\sum_{l \epsilon \Lambda_k} \zeta_{ijkl}}$$

... where w_{ij} is the weight of the j^{th} mixture of the i^{th} output, $S_{trained}$ is the set of trained output, Λ_k is the set of mixture of the k^{th} output, and ζ is the correlation matrice.

If we set

$$\frac{\zeta_{ijkl}}{\sum_{l \epsilon \Lambda_k} \zeta_{ijkl}} = P((\tilde{s} = i, \tilde{m} = j)/(s = k, m = l), \tilde{s} = i)$$

then

$$w_{ij} = \prod_{k \epsilon S_{trained}} \left(\sum_{l \epsilon \Lambda_k} P((k,l)/s = k) P((\tilde{s} = i, \tilde{m} = j)/(k,l), \tilde{s} = i) \right)$$

... where $\tilde{s}$ and $\tilde{m}$ are the assumed state and mixture.

This new weights are normalised by:

$$w_{ij} \leftarrow \max \left(\frac{w_{ij}}{\sum_j w_{ij}}, min\_weight \right)$$

Then, the counters of the trained states are put to 0, and the counters of the untrained states are put to the minimum count.

Zero The same zero procedure as in 3.1.3 is used. As the counters have been inverted during the setting (see above) the procedure acts on the 'untrained' states this time.

3.1.5 Prune

While there are more mixtures than the minimum number:

- The least weighted mixture is chosen and if its weight is over the threshold (user defined) the loop is broken.
- Otherwise, the chosen mixture is deleted by reordering the mixtures without it.

The maximum number of mixtures is calculated.
So:

$$P(X, S/\lambda) = \sum_{l_1=1}^{C_{s_1}} \cdots \sum_{l_T=1}^{C_{s_T}} \prod_{t=1}^{T} I(c(s_t, l_t) - Cst)\aleph(\mu_{s_t l_t}; S_{s_t l_t})$$

...where I is the fonction defined by:

$$\begin{aligned} I(x) &= 1 \; if \; x > 0 \\ &= 0 \; otherwise \end{aligned}$$

and Cst is the threshold depending on the number of deleted mixtures.

3.2 Present work

3.2.1 Product

The weight reestimation, as explained in 3.1.4 takes the weighted (by the correlations) sum of the weights as new weight. I also tried to use the weighted product:

$$w_{ij} = \prod_{k \epsilon S_{trained}} \frac{\prod_{l \epsilon \Lambda_k} w_{kl} * \zeta_{ijkl}}{\left(\sum_{l \epsilon \Lambda_k} \zeta_{ijkl}\right)^{card(\Lambda_k)}}$$

This has been implemented in test *141* and gives exactely the same result as with the sum (test *137*).

3.2.2 Composition

If only a small amount of data is used during the adaptation, and it is likely to occure if we use a recognition system on a brand new speaker, the mixtures with small weights may not be the less important, and pruning them is maybe not the best solution. For accoustic environment reasons and also physic condition of the speaker reasons, a set of close mixtures may not be used, althrough they would match the speaker's utterances in other conditions. In order to keep these mixtures

without reducing the network's performances, I tried to compose the less weighted components instead of pruning them.
The composition can generate a mistaking in the path during the recognition since it keeps mixtures which may not correspond to the speaker for one state, but may interfer with mixtures of the speaker for other states.

Implementation The composition is the same as the one explained in 2.1.3 and only between two mixtures: the two less weighted mixtures. The composed mixture is kept and has a weight equal to the sum of the weights of the two mixtures.

result As can by seen with the test *154* as compared to the test *146* the result is not worse. Unfortunately, this test is on data of the same kind as the training data, and the HMnet has already been much trained, so the effect of the composition can not really be seen here.

3.2.3 Exponent

As seen in 3.1.4

$$w_{ij} = \prod_{k \epsilon S_{trained}} \left(\sum_{l \epsilon \Lambda_k} P((k,l)/s=k) P((\tilde{s}=i, \tilde{m}=j)/(k,l), \tilde{s}=i) \right)$$

so

$$w_{ij} = \prod_{k \epsilon S_{trained}} \frac{1}{P(s=k)} \sum_{l \epsilon \Lambda_k} \frac{P(s=k,m=l) P(\tilde{s}=i, \tilde{m}=j, s=k, m=l)}{P(s=k, m=l, \tilde{s}=i)}$$

if we assume that *(k,l)* is independant from *(i,j)*, which means that the supposed mixture is independant from the real one (but not of the sample),

$$\begin{aligned} w_{ij} &= \prod_{k \epsilon S_{trained}} \frac{P(\tilde{s}=i, \tilde{m}=j)}{P(s=k) P(\tilde{s}=i)} \sum_{l \epsilon \Lambda_k} P(s=k, m=l) \\ &= \prod_{k \epsilon S_{trained}} P(\tilde{s}=i, \tilde{m}=j / \tilde{s}=i) \\ &= P(\tilde{s}=i, \tilde{m}=j / \tilde{s}=i)^{card(S_{trained})} \end{aligned}$$

This has been implemented in the test *155* and achieves about the same result as without the degree.

4 Annexe

4.1 Composition

4.1.1 Goal

For each cluster, we have many mixtures, corresponding to the different speakers which are close to each other, and we'd like to compose theses mixtures in order to get

a unique mixture. What's more, this composition is a weighted composition, since we'd like to make the more reliable mixtures more important in the composition. As had been seen in 2.2.3:

$$\begin{aligned} P(X,S/\lambda) &= \left(\prod_{t=1}^{T} a_{s_{t-1}s_t}\right) \sum_{l_1=1}^{C_{s_1}} \sum_{k_1=1}^{P} \cdots \sum_{l_T=1}^{C_{s_T}} \sum_{k_T=1}^{P} \left(\prod_{t=1}^{T} \delta_{l_t}(k_t) c_{k_t} b_{k_t}(x_t)\right) \\ &= \left(\prod_{t=1}^{T} a_{s_{t-1}s_t}\right) \sum_{l_1=1}^{C_{s_1}} \cdots \sum_{l_T=1}^{C_{s_T}} \prod_{t=1}^{T} \sum_{k_1=1}^{P} \cdots \sum_{k_T=1}^{P} \delta_{l_t}(k_t) c_{k_t} b_{k_t}(x_t) \end{aligned}$$

The $b_{k_t}(x_t)$ are gaussian distributions: $\aleph_{k_t}(x_t)$.
So $\forall t \varepsilon \{1, \ldots T\}$

$$\begin{aligned} \Sigma &= \sum_{k_t=1}^{P} \delta_{l_t}(k_t) c_{k_t} b_{k_t}(x_t) \\ &= \sum_{k_t=1}^{P} \delta_{l_t}(k_t) c_{k_t} \aleph(\mu_{k_t}; S_{k_t})(x_t) \end{aligned}$$

4.1.2 Two gaussians

We want to approximate this sum of gaussian probabilities, by a gaussian probability since it allows to do the same tests and further treatments to the network.
The mean of the sum is:

$$\begin{aligned} M &= E[\aleph_1 + \aleph_2] \\ &= \mu_1 + \mu_2 \end{aligned}$$

The variance is:

$$\begin{aligned} V &= E[(\aleph_1 + \aleph_2 - \mu_1 - \mu_2)^2] \\ &= E[(\aleph_1 - \mu_1)^2 + (\aleph_2 - \mu_2)^2 + 2(\aleph_1 - \mu_1)(\aleph_2 - \mu_2)] \\ &= S_1 + S_2 + \frac{1}{\pi S_1 S_2} E\left[e^{-\frac{1}{2S_1S_2}[(S_1+S_2)x^2 - 2x(\mu_1 S_2 + \mu_2 S_1) + \mu_1^2 S_2 + \mu_2^2 S_1]}\right] - 2\mu_1\mu_2 \\ &= S_1 + S_2 - 2\mu_1\mu_2 + \sqrt{\frac{2}{\pi}} e^A B \end{aligned}$$

... where,

$$\begin{aligned} A &= -\mu_1^2 S_2 - \mu_2^2 S_1 + \frac{\mu_1^2 S_2^2 + \mu_2^2 S_1^2 + 2\mu_1\mu_2 S_1 S_2}{S_1 + S_2} \\ &= -\frac{S_1 S_2}{S_1 + S_2}(\mu_1 - \mu_2)^2 \end{aligned}$$

...and ...

$$\begin{aligned} B &= E\left[\frac{1}{\sqrt{2\pi}S_1S_2}e^{-\frac{1}{2S_1S_2}(\sqrt{S_1+S_2}x-\frac{\mu_1S_2+\mu_2S_1}{\sqrt{S_1+S_2}})^2}\right] \\ &= \frac{1}{\sqrt{2\pi}S_1S_2}\int_{-\infty}^{+\infty} e^{-\frac{1}{2S_1S_2}(\sqrt{S_1+S_2}x-\frac{\mu_1S_2+\mu_2S_1}{\sqrt{S_1+S_2}})^2} \\ &= \frac{1}{\sqrt{S_1+S_2}}E\left[\aleph(\frac{\mu_1S_2+\mu_2S_1}{\sqrt{S_1+S_2}};S_1S_2)\right] \\ &= \frac{\mu_1S_2+\mu_2S_1}{S_1+S_2} \end{aligned}$$

$$\ldots \text{using } y=\sqrt{S_1+S_2}x.$$

so

$$V = S_1+S_2-2\mu_1\mu_2+\frac{\mu_1S_2+\mu_2S_1}{S_1+S_2}\sqrt{\frac{2}{\pi}}e^{-\frac{S_1S_2}{S_1+S_2}(\mu_1-\mu_2)^2}$$

4.1.3 Multiple gaussian

The previous equation can be easly extended to a P gaussian weighted sum:

$$\begin{aligned} M &= E[\sum_{i=1}^{P}\omega_i\aleph(\mu_i;S_i)] \\ &= \sum_{i=1}^{P}\omega_i\mu_i \end{aligned}$$

and,

$$\begin{aligned} V &= E\left[\sum_{i=1}^{P}(\omega_i(\aleph(\mu_i;S_i)-\mu_i))^2\right] \\ &= E\left[\sum_{i=1}^{P}\sum_{j=1}^{P}\omega_i\omega_j(\aleph(\mu_i;S_i)-\mu_i)(\aleph(\mu_j;S_j)-\mu_j)\right] \\ &= \sum_{i=1}^{P}\omega_i^2S_i+\sum_{i=1}^{P}\sum_{j=1\ j\neq i}^{P}\omega_i\omega_j\left(-\mu_i\mu_j+\frac{\mu_iS_j+\mu_jS_i}{\sqrt{2\pi}(S_i+S_j)}e^{-\frac{S_iS_j}{S_i+S_j}(\mu_i-\mu_j)^2}\right) \end{aligned}$$

4.1.4 Quadratic error

The problem in this method is that the sum of gaussian distributions is usely not a gaussian distribution. But if we want to use a gaussian distibution, we can try to find the closest one, by minimising the quadratic error. So, we try to approximate

$$\sum_{i=1}^{P}\omega_i\aleph(\mu_i;S_i)$$

by

$$\aleph(\mu; S)$$

$$R = E\left[(\aleph(\mu; S) - \sum_{i=1}^{P} \omega_i \aleph(\mu_i; S_i))^2\right]$$

using the fact that $\sum_{i=1}^{P} \omega_i = 1$ (the weights are normalised) we get

$$\begin{aligned}
R &= E\left[(\sum_{i=1}^{P} \omega_i(\aleph(\mu; S) - \aleph(\mu_i; S_i)))^2\right] \\
&= E\left[\sum_{i=1}^{P}\sum_{j=1}^{P} \omega_i\omega_j(\aleph(\mu; S) - \aleph(\mu_i; S_i))(\aleph(\mu; S) - \aleph(\mu_j; S_j))\right] \\
&= \sum_{i=1}^{P}\sum_{j=1}^{P} \omega_i\omega_j E\left[\aleph(\mu; S)^2 - \aleph(\mu; S)(\aleph(\mu_i; S_i) + \aleph(\mu_j; S_j)) + \aleph(\mu_i; S_i)\aleph(\mu_j; S_j)\right] \\
&= \sum_{i=1}^{P}\sum_{j=1}^{P} \omega_i\omega_j(f(\mu; S) - g_i(\mu; S) - g_j(\mu; S) + Cst)
\end{aligned}$$

where:

$$\begin{aligned}
f(\mu; S) &= \frac{1}{2\sqrt{2\pi}S} E\left[\frac{1}{\sqrt{2\pi}\frac{S}{2}} e^{-\frac{1}{2\frac{S}{2}}(x-\mu)^2}\right] \\
&= \frac{\mu}{2\sqrt{2\pi}S}
\end{aligned}$$

and

$$g_i(\mu; S) = \frac{\mu S_i + \mu_i S}{\sqrt{2\pi}(S_i + S)} e^{-\frac{S_i S}{S_i + S}(\mu_i - \mu)^2}$$

The optimal $(\mu; S)$ can be found by:

$$\begin{pmatrix} \frac{\partial R}{\partial \mu} \\ \frac{\partial R}{\partial S} \end{pmatrix} = 0$$

using a gradiant method to compute it (as an aproximation).

4.2 Tests

4.2.1 Nomenclatura

All the tests have been done on a 400 states HMnet (*RES_TEST400, res400, r4*).

- *mix, mi* : Maximum number of mixtures for each state.
- *dr* : Distortion rate (rate between the *fifth* and the *mean* components).

- *cr* : Change rate (rate between the change rate at the beginning and the one calculated during the spliting).
- *de* : Degree used for the *fifth* component.
- *thx.xxx* : Threshold for the distortion.
- *th* : Stop the spliting for variance of the distribution of membership purpose.
- *xxxx* : Total number of mixtures.
- *p* : Pruned.
- *myais, m* : Test on the current speaker for the adaptation.
- *bis, b* : Test that has already been done on another machine, or at another time, in other comditions...
- *#, , @* : The tests are computed on different computers (atrh15, 25, 30, and the others on 23).
- *pr* : Use of the product for the weight estimation.
- *wox.xxx* : The *Zero* procedure is not used in the weight settings (trained and untrained). The number is the pruning coeficient.
- *wox.xxx* : The *Zero* procedure is used in the weight settings (trained and untrained). The number is the pruning coeficient.
- *co* : Instead of being pruned, the less weighted mixtures are composed.
- *ce* : Use of the coeficient for the estimation of the weights of the untrained states.

4.2.2 List of the tests

0	*RES_TEST*400.05.*myais*	63.582224%
1	*RES_TEST*400.05.*p.myais*	64.979386%
2	*RES_TEST*400.10.*myais*	64.567107%
3	*RES_TEST*400.10.*p.myais*	67.109483%

4	*res*400.05	67.333465%
5	*res*400.10	67.414074%
6	*res*400.15	66.845015%
7	*res*400.05.*p*	64.121468%

8	*res*400.05.01.05	65.333888%
9	*res*400.05.05.05	65.859971%
10	*res*400.05.09.05	66.330405%

... where the three numbers correspond to *mi_dr_cr* .

11	*r4_mi10_dr0.0*	66.070690%
12	*res400_mix10_dr0.1_cr0.0_de5*	66.098607%
13	*res400_mix10_dr0.5_cr0.0_de5*	66.123568%
14	*res400_mix10_dr0.9_cr0.0_de5*	66.447962%

(cr0.0)

15	*res400_mix10_dr0.9_cr0.9_de5*	66.152899%
16	*r4_mi10_dr0.9_cr0.0_de7*	66.384485%
17	*r4_mi10_dr0.9_cr0.0_de8_3517*	67.002979%
18	*r4_mi10_dr0.9_cr0.0_de9_3517*	66.793946%
19	*r4_mix10_dr0.9_cr0.0_de10_3517*	66.668463%
20	*r4_mi10_dr0.9_cr0.0_de15_3517*	66.559235%
21	*r4_mix10_dr0.9_cr0.0_de20_3517*	66.578662%
22	*r4_mi10_dr0.9_cr0.0_de10_add*	66.578284%
23	*r4_mi10_dr0.9_cr0.0_de10_add0*	65.829154%
24	*r4_mi10_dr0.2_cr0.1_de8_add1_0.5*	65.103728%
25	*r4_mi10_dr0.9_cr0.0_de8_add2_3*	65.543243%
26	*r4_mi10_dr0.9_cr0.0_de9_add2_s*	66.356848%
27	*r4_mi10_dr0.9_cr0.0_de9_add2_sd*	66.646918%
28	*r4_mix10_dr1.0_de8_add2_s*	66.367650%
29	*r4_mi10_dr0.9_cr0.0_de10_sdw*	66.250198%
30	*r4_mi10_dr0.9_de8_sm*	66.588930%
31	*s1_r4_mi10_dr0.9_cr0.0_de10*	63.097083%
32	*r4_mi10_dr0.9_cr0.9_de10_inv*	65.804846%
33	*r4_mi10_dr0.0_cr0.9_de8*	66.393957%
34	*r4_mi10_dr0.9_th0.3*	65.912357%
35	*r4_mi10_dr0.9_th0.4*	66.879780%
36	*r4_mi10_dr0.9_th0.5_sm.add.s3*	66.092006%

New version of setting centers and belongings.

37	*r4_mi10_dr0.9_de8_04.10.95*	66.373653%
38	*r4_mi11_dr0.9_de8_th0.3_3547*	66.163365%
39	*r4_mi12_dr0.9_de8_th0.2525_3517*	66.088352%
40	*r4_mi13_dr0.9_de8_th0.23602_3517*	65.580444%
41	*r4_mi14_dr0.9_de8_th0.228_3517*	65.817028%
42	*r4_mi15_dr0.9_de8_th0.2235_3517*	65.929358%
43	*r4_mi16_dr0.9_de8_th0.22035_3517*	65.931970%

th0.0 means that it is not used.

44	$r4\_mi10\_dr0.9\_de8\_th0.0\_th\_3389$	66.723766%
45	$r4\_mi11\_dr0.9\_de8\_th0.3547\_th\_3517$	56.646464%*
46	$r4\_mi12\_dr0.9\_de8\_th0.2831\_th\_3517$	66.604741%
47	$r4\_mi13\_dr0.9\_de8\_th0.2587\_th\_3517$	66.274240%
48	$r4\_mi14\_dr0.9\_de8\_th0.24406\_th\_3517$	66.262743%*
87	$r4\_mi15\_dr0.9\_de8\_th0.23601\_th\_3536$	65.758869%
49	$r4\_mi5\_dr0.9\_de8\_th0.0\_th\_1872$	65.911284%*
50	$r4\_mi6\_dr0.9\_de8\_th0.1625\_th\_1872$	65.767119%*
51	$r4\_mi7\_dr0.9\_de8\_th0.14545\_th\_1872$	65.651652%*

debug

52	$r4\_mi8\_dr0.9\_de8\_th0.139\_th\_1872$	65.331449%#
53	$r4\_mi9\_dr0.9\_de8\_th0.136\_th\_1872$	65.258028%*
54	$r4\_mi10\_dr0.9\_de8\_th0.13533\_th\_1872$	66.067122%#

th0.0 is used in the following tests.
dd is the distance rate between the two components of the distance.

55	$r4\_mi11\_dr0.9\_de8\_thadd3\_3678$	65.083226%
56	$r4\_mi10\_dr0.9\_de8\_dd0.0\_th$	65.941939%
57	$r4\_mi10\_dr0.9\_de8\_dd1.0\_th\_3517$	61.801889%*
58	$r4\_mi10\_dr0.9\_de8\_dd0.3\_th\_3517$	66.437180%
59	$r4\_mi10\_dr0.9\_de8\_dd0.3\_thc\_3134$	65.710306%*
60	$r4\_mi10\_dr0.9\_de8\_dd0.4\_th\_3299$	66.510795%
61	$r4\_mi11\_dr0.9\_de8\_dd0.45\_th\_3560$	66.467058%
62	$r4\_mi11\_dr0.9\_de8\_dd0.45\_thd\_5839$	65.710306%*

Tests 64, 68 and 71 used a bad list for adaptation.

63	$r4\_mi10\_dr0.9\_de11\_th0.0\_th\_3399$	66.925151%
64	$r4\_mi10\_dr0.9\_de11\_th\_3399\_p\_3312$	66.046115%*
65	$r4\_mi10\_dr0.8\_de12\_th0.0\_th\_3399$	66.673827%*
66	$r4\_mi10\_dr0.5\_de11\_th0.0\_th\_3390$	66.602610%*
67	$r4\_mi11\_dr1.0\_de13\_th\_3682$	66.758899%*
68	$r4\_mi11\_dr1.0\_de13\_th\_3682\_p\_3314$	66.182076%*
86	$r4\_mi10\_dr1.0\_de5\_th\_3400$	66.240831%
69	$r4\_mi10\_dr1.0\_de9\_th\_3404$	66.909524%*
90	$r4\_mi10\_dr1.0\_de10\_th\_3399\_0$	66.792979%
70	$r4\_mi10\_dr1.0\_de10\_th\_3399$	66.792979%*
71	$r4\_mi10\_dr1.0\_de10\_th\_3399\_p\_3312$	66.132605%*
72	$r4\_mi10\_dr1.0\_de10\_th\_3399\_p\_2042$	63.204754%
73	$r4\_mi10\_dr1.0\_de11\_th\_3597$	66.925151%#
96	$r4\_mi10\_dr1.0\_de20\_th\_3407$	66.607942%

74	*r4_mi9_dr1.0_de9_th_3283*	66.357236%#
75	*r4_mi9_dr1.0_de10_th_3115*	66.201761%*

The tests 83, 93, 88, 78 and 80 were not completed because of a bug in the recognition procedure.

76	*r4_mi8_dr0.0_de10_th_2819*	65.630345%*
83	*r4_mi8_dr0.0_de10_th_2819_p_1831_mya*	68.328333%*
91	*r4_mi8_dr0.3_de10_th_2823*	65.800309%*
93	*r4_mi8_dr0.3_de10_th_2823_p_1843_mya*	67.653674%*
85	*r4_mi8_dr0.5_de10_th_2824*	65.428486%*
88	*r4_mi8_dr0.5_de10_th_2824_p_1802_mya*	67.241377%*
94	*r4_mi8_dr0.7_de10_th_2825*	65.400269%*
77	*r4_mi8_dr1.0_de10_th_2824*	65.613686%*
78	*r4_mi8_dr1.0_de10_th_2824_p_1772_mya*	66.454965%*
95	*r4_mi10_dr0.0_de10_th_3586*	66.688156%#
79	*r4_mi10_dr0.0_de10_th_3388*	65.624267%
80	*r4_mi10_dr0.0_de10_th_3388_p_2088_my*	68.515742%
81	*r4_mi10_dr0.0_de10_ths_3393*	66.783781%
82	*r4_mi10_dr1.0_de10_ths0_3396*	66.571711%
84	*r4_mi8_dr0.5_de10_ths0_2823*	65.509596%*
89	*r4_mi8_dr1.0_de10_thv_2826*	65.987464%*
92	*r4_mi8_dr1.0_de10_v_2877*	65.894361%*
97	*r4_mi10_dr1.0_de10_thn_3399*	66.822004%#

New test version. A bug occured in the program.

98	*r4_mi10_dr1.0_de10_thnp_3517*	64.949392%#
99	*r4_mi10_dr1.0_de10_thnv_3517*	64.949392%#
100	*r4_mi10_dr0.0_de10_thnv_3517*	64.949392%
101	*r4_mi10_dr1.0_de10_thnp_3517_bis*	64.949392%#
102	*r4_mi10_dr0.5_de10_thn_3517*	64.949392%#
103	*r4_mi8_dr1.0_de10_thnv_2877*	64.383387%#
104	*r4_mi8_dr1.0_de10_thnp_2877*	64.383387%#
105	*mi8_dr0.3_de10_th_2823_mi5_1644_m_b*	66.347468%*
106	*r4_mi10_dr0.5_de10_thn_3517_bis*	64.949392%

debug, new test procedure, new tables.

107	r4_mi10_dr1.0_de10_thnv_3473	64.006835%*
116	r4_mi10_dr0.5_de10_thnv_3282	64.067224%*
108	r4_mi10_dr0.0_de10_thnv_3298	64.610183%
118	r4_mi10_dr1.0_de10_nv_3506	64.391512%
109	r4_mi10_dr1.0_de10_thnp_3294	64.953024%#
115	r4_mi10_dr0.8_de10_thnp_3528	63.758108%*
113	r4_mi10_dr0.5_de10_thnp_3521	63.677615%*
110	r4_mi10_dr0.0_de10_thnp_3304	64.691566%
119	r4_mi10_dr1.0_de10_np_3506	65.019750%#
111	r4_mi10_dr1.0_de10_thn_3514	64.881671%*
117	r4_mi10_dr0.8_de10_thn_3502	64.810552%*
114	r4_mi10_dr0.5_de10_thn_3306	63.276830%
112	r4_mi10_dr0.0_de10_thn_3273	63.969881%
132	r4_mi10_dr1.0_de10_n_3506	64.642974%
120	r4_mi10_dr1.0_de10_th_3514	64.881671%*
124	r4_mi10_dr1.0_de10_th_3514_bis	64.881671%*
127	r4_mi10_dr0.5_de10_th_3500	65.010248%*
123	r4_mi10_dr0.0_de10_th_3495	64.344641%*
121	r4_mi10_dr1.0_de10_thn1_3312	37.705567%
122	r4_mi10_dr1.0_de10_thn0_3312	20.182004%#
125	r4_mi10_dr1.0_de10_thn2_3312	64.636428%
134	r4_mi10_dr0.5_de10_thn2_3500	64.876940%*
128	r4_mi10_dr0.0_de10_thn2_3273	64.323654%#
131	r4_mi10_dr1.0_de10_n2_3700	64.580324%*
126	r4_mi10_dr1.0_de10_thn3_3312	21.108129%#
129	r4_mi10_dr1.0_de10_thn4_3514	63.697929%*
130	r4_mi10_dr1.0_de10_thn5_3312	22.591623%
133	r4_mi10_dr1.0_de10_thn6_3312	62.483481%#
135	r4_mi10_dr1.0_de10_thn7_3312	62.156524%#
138	r4_mi10_dr1.0_de10_thn8_3312	62.387838%@
150	r4_mi10_dr0.0_de10_thn9_3273	63.554518%#
152	r4_mi10_dr0.5_de10_thn9_3306	63.485627%#
145	r4_mi10_dr1.0_de10_thn9_3514	64.089156%*
147	r4_mi10_dr1.0_de10_thn10_3312	63.226641%#
136	r4_mi8_dr1.0_de10_3002	64.057879%*
143	r4_mi8_dr1.0_de10_3002_p_2333_m	64.975077%*
157	r4_wi0.01_ce_3002_p12_1948_m	65.043044%*

All the following test use _*mi*10_*dr*1.0_*de*10_*n*_.

137	*r4_mi10_dr1.0_de10_n_3506_p_2578_m*	66.719526%
140	*r4_wo0.0_3506_p0_3416_m*	66.696876%
141	*r4_pr_3506_p1_xxxx_m*	66.719526%
142	*r4_wo0.0001_3506_p2_2866_m*	66.696876%
143	*r4_wo0.001_3506_p3_2622_m*	66.719526%
144	*r4_wo0.01_3506_p4_2073_m*	66.855460%
156	*r4_wi0.01_co_3506_p11_2161_m*	66.674221%
151	*r4_wo0.05_3506_p7_1408_m*	67.059356%
153	*r4_wi0.08_3506_p8_1153_m*	66.968733%
146	*r4_wo0.1_3506_p5_1019_m*	67.217940%
154	*r4_wi0.1_co_3506_p9_1266_m*	66.946083%
155	*r4_wi0.1_ce_3506_p10_1019_m*	67.217940%
149	*r4_wo0.5_3506_p6_410_m*	65.382874%

References

[HMM] Huang Ariki Jack. *HMM for speech recognition* . Edinburgh University Press, 1990.

[ICASSP92] *ICASSP 92 Conference Proceedings* . pp 573-576.