ircamAlign : système d'étiquetage et d'alignement

Pierre Lanchantin

8 octobre 2007

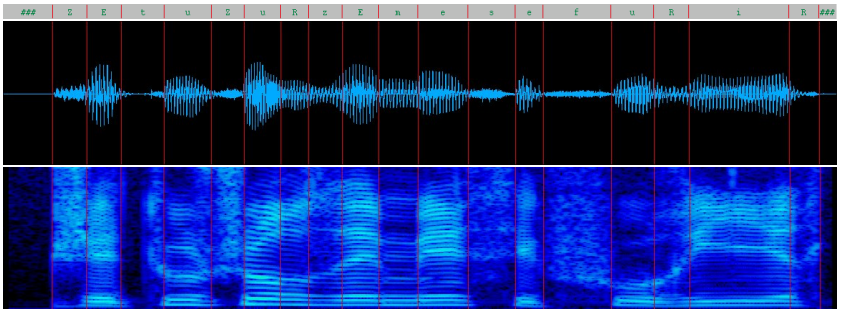


FIG. 1 – Principe de la segmentation d'un signal de parole en phonèmes.

Résumé

Un phonème est défini comme étant la plus petite unité du langage parlé. Le français en comporte 36 dont 16 voyelles et 20 consonnes. Segmenter un signal de parole en phonème consiste dans un premier temps à reconnaître la séquence de phonèmes réalisés en phones lors d'une étape de *reconnaissance*, puis à trouver les marques de début et de fin de phonèmes dans une étape d'*alignement*. De nombreuses applications en découle comme par exemple la synthèse par concaténation d'unités, l'analyse de corpus, les traitements dépendants de la nature des phonèmes ou encore le suivi en temps-réel. Le plan de ce rapport est le suivant. Dans un premier temps, nous décrivons les principes de la segmentation statistique bayésienne non supervisée. Puis nous décrivons la modélisation par chaîne de Markov cachées dans le cas général puis dans le cas plus spécifique de la reconnaissance de parole. Ensuite, nous précisons en détail l'implémentation du système ircamAlign. Finalement, nous donnons quelques exemples d'application.

Table des matières

1 Principes de la segmentation statistique bayésienne

1.1 Modélisation probabiliste

La méthode de segmentation adoptée est un méthode statistique qui nécessite une modélisation probabiliste du problème. Pour cela, sous supposons que le signal de parole est indexé par un ensemble discret fini $\mathcal{N}$ de N *sites*. A chaque site n est associé un vecteur acoustique y_n de coefficient mel cepstraux. L'ensemble de ces vecteurs acoustique $\{y_n\}_{n=1}^{N}$ sera noté y. Nous utiliserons par la suite, pour des raisons de commodité, la notation $y_{m:p}$ pour désigner l'ensemble $\{y_n\}_{n=m}^{p}$. Nous supposons que y est la réalisation d'un processus aléatoire $Y = Y_{1:N}$, chaque Y_n prenant ses valeurs dans l'ensemble $\mathcal{Y}$. De même, pour le signal étiqueté recherché $x = x_{1:N}$, nous supposons que ce dernier est la réalisation d'un processus aléatoire $X = X_{1:N}$, chaque X_n prenant ses valeurs dans une ensemble $\mathcal{X}$ qui dans notre cas est l'ensemble des 36 phonèmes du français plus les 2 phonèmes correspondant aux silences.

$$\begin{array}{ll} X = \{X_n\}_{n\in\mathcal{N}} & : \text{le processus aléatoire caché prenant ses valeurs dans } \mathcal{X}^N \\ Y = \{Y_n\}_{n\in\mathcal{N}} & : \text{le processus aléatoire observable prenant ses valeurs dans } \mathcal{Y}^N \end{array}$$

Nous notons κ la mesure de comptage sur $\mathcal{X}$ et μ une mesure de référence (Lebesgue) sur $\mathcal{Y}$ et nous supposons que la loi du processus couple $Z = (X, Y)$ modélisant l'ensemble des liens statistiques entre les deux processus X et Y admet une densité $p(z)$ relativement à la mesure produit $(\kappa\otimes\mu)^N$. Nous pouvons dès lors exprimer notre connaissance sur X lorsque y est observé à partir de la loi *a posteriori* calculée à partir de

$$p(x|y) = \frac{p(z)}{\sum_{x\in\mathcal{X}^N} p(z)} \tag{1}$$

1.2 Stratégie bayésienne

Supposons que la loi de Z soit parfaitement connue. Le problème de la segmentation se ramène alors au problème probabiliste suivant : comment estimer une réalisation invisible x de X à partir de l'observation y de Y. On adopte une approche bayésienne et on introduit pour cela une stratégie de classification $\mathcal{Y}^N \to \mathcal{X}^N$. Afin de mesurer la gravité de l'erreur commise, nous définissons une fonction de perte $L : \mathcal{Y}^N \times \mathcal{X}^N \to \mathbb{R}^+$ donnant la perte $\lambda_{i,j}$ occasionnée lorsqu'on choisit i au lieu de la vraie valeur j. La stratégie bayésienne, relativement à la fonction de perte L choisie est celle qui minimise l'espérance conditionnelle $\mathbb{E}\{L(\hat{s}(Y), X)|Y = y\}$ pour chaque observation $Y = y$. L'estimateur bayésien $\hat{x}_B$ optimal par rapport à une fonction de perte L est alors donnée par

$$\hat{x}_B(y) = \arg\min_{x'\in\mathcal{X}^N} \sum_{x\in\mathcal{X}^N} \lambda_{x',x} p(x|y) \tag{2}$$

Notre objectif étant de trouver la meilleure configuration de phonèmes conditionnellement aux observations, nous définissons L de manière à pénaliser toutes les configurations estimées différentes de la vraie configuration, nous obtenons ainsi l'estimateur bayésien du maximum *a posteriori* (MAP) que nous utiliserons par la suite :

$$\hat{x}_{MAP}(y) = \arg\max_{x'\in\mathcal{X}^N} p(x'|y) \tag{3}$$

1.3 Estimation des paramètres

Nous avons supposé que la loi de Z était parfaitement connue, or, d'une manière générale, ce n'est pas le cas et on suppose que la densité $p(z)$ est paramétrée. Nous noterons Φ l'espace

des paramètres du modèle $p(z|\phi)$ considéré avec $\phi \in \Phi$. En supposant que le processus Z est stationnaire il est alors possible d'estimer ses paramètres à partir des seules observations y par des approches fondées sur la maximisation de la vraisemblance comme l'algorithme Espérance Maximisation. L'estimateur du maximum de vraisemblance (MV) donne les paramètres du modèle maximisant la vraisemblance $p(y|\phi)$ que nous supposons dérivable par rapport à ϕ. Afin de calculer son maximum, nous devons déterminer les valeurs de ϕ pour lesquelles sa dérivée s'annule. Pour des raisons pratiques, il est plus simple de maximiser la log-vraisemblance, que nous noterons $\mathcal{L}(y;\phi) = \log p(y|\phi)$. L'estimateur du maximum de vraisemblance est alors donné par

$$\hat{\phi}(y) = \arg\max_{\phi'} \mathcal{L}(y;\phi') \tag{4}$$

Pour trouver les valeurs des paramètres maximisant la log-vraisemblance, on utilise alors l'algorithme Espérance-Maximisation (EM), bien adapté aux modèles comprenant des variables cachées. Il s'agit en effet d'un algorithme général d'estimation par maximum de la vraisemblance dans le cas ou les données sont incomplètes. Le déroulement de l'algorithme EM est le suivant : ϕ est d'abord initialisé à une valeur $\phi^{[0]}$ des paramètres. Puis, on itère alternativement entre deux étapes notées E (Espérance) et M (Maximisation).

- Dans l'étape E, nous calculons la fonction $\mathcal{Q}(\phi|\phi^{[q]}) = \mathbb{E}_{\phi^{[q]}}\{\mathcal{L}_c(Z;\phi)|Y=y\}$ dans laquelle $\mathcal{L}_c(z;\phi) = p(z|\phi)$ est la log-vraisemblance complétée.
- Dans l'étape M, les paramètres sont estimés en maximisant la fonction $\mathcal{Q}(\phi|\phi^{[q]})$ par rapport à ϕ : $\phi^{[q+1]} = \arg\max_\phi \mathcal{Q}(\phi|\phi^{[q]})$.

Ayant une nouvelle génération des valeurs des paramètres $\phi^{[q+1]}$, on peut dès lors répéter les étapes E et M. On construit ainsi, de manière itérative, une suite des paramètres $\left(\phi^{[q]}\right)_{q>0}$ telle que $\left(\mathcal{L}(y:\phi^{[q]})\right)_{q\geq 0}$ converge vers un maximum local de la log-vraisemblance $\mathcal{L}(y;\phi)$. En pratique, on arrête l'algorithme lorsque $\mathcal{L}(y;\phi^{[q+1]}) - \mathcal{L}(y;\phi^{[q]}) \leq \varepsilon$ où ε est fixé. L'algorithme EM possède de nombreux avantages parmi lesquels sa stabilité numérique (la log-vraisemblance croît à chaque itération), le fait qu'il soit analytiquement et numériquement simple à implémenter ou encore son coût calculatoire faible à chaque itération. Ces propriétés en ont fait une méthode de référence pour l'estimation des modèles à données manquantes. Cependant, EM possède des limitations comme sa sensibilité à la valeur initiale $\phi^{[0]}$, sa faible vitesse de convergence ou encore des difficultés de calcul possible au niveau des étapes E (calcul analytique de $\mathcal{Q}(\phi|\phi^{[q]})$ impossible) et M.

2 Modélisation HMM

2.1 Loi jointe des processus caché et observé

Dans le cas général, la loi jointe des processus caché (étiquettes des phonèmes) et observé (coefficients MFCC) peut être écrite sous la forme factorisée suivante

$$p(z) = p(z_1)\prod_{n=2}^{N} p(z_n|z_{1:n-1}) \tag{5}$$

Il n'est pas possible de calculer numériquement l'ensemble des noyaux de transitions $p(z_n|z_{1:n-1})$ pour tout $n \in \mathcal{N}$ sans faire d'hypothèse simplificatrice au préalable. Une solution est de supposer que le processus couple Z est une chaîne de Markov cachée. Dans ce modèle, utilisé depuis plus de 30 ans dans le domaine de la reconnaissance de parole, le processus caché X est une chaîne

de Markov (CM) et les observations sont indépendantes conditionnellement à X :

$$\left\{ \begin{array}{l} p(x) = p(x_1) \prod_{n=2}^{N} p(x_n|x_{n-1}) \\ p(y|x) = \prod_{n=1}^{N} p(y_n|x_n) \end{array} \right. \tag{6}$$

Un des inconvénients de cette modélisation est l'hypothèse d'indépendance des observations conditionnellement à X. Cependant, celle-ci peut être contournée en prenant en compte les dérivées premières et secondes des coefficients MFCC. La loi de Z simplifiée (qui est alors également celle une CM) devient $p(z) = p(z_1) \prod_{n=2}^{N} p(z_n|z_{n-1})$ avec $p(z_1) = p(x_1)p(y_1|x_1)$ et $p(z_n|z_{n-1}) = p(x_n|x_{n-1})p(y_n|x_n)$, soit

$$p(z) = p(x_1)p(y_1|x_1) \prod_{n=2}^{N} p(x_n|x_{n-1})p(y_n|x_n) \tag{7}$$

2.2 Passes forward et backward

Un des avantages de cette modélisation est que la loi jointe est décomposable en deux quantités dites *forward* et *backward,* ces deux quantités étant calculables de manière récursive.

$$p(x_n, y) = \underbrace{p(z_n, y_{1:n-1})}_{\alpha(x_n)} \underbrace{p(y_{n+1:N}|z_n)}_{\beta(x_n)} \tag{8}$$

En effet, $\alpha(x_{n+1}) = \sum_{x_n} p(z_{n+1}|z_n)\alpha(x_n)$ et $\beta(x_n) = \sum_{x_{n+1}} p(z_{n+1}|z_n)\beta(x_{n+1})$. On initialise ainsi la récursion forward par $\alpha(x_1) = p(z_1)$ et la récursion backward par $\beta(z_N) = 1$.

$$\left\{ \begin{array}{l} \alpha(x_1) = p(z_1) \\ \alpha(x_{n+1}) = \sum_{x_n} p(z_{n+1}|z_n)\alpha(x_n) \end{array} \right. et \left\{ \begin{array}{l} \beta(x_N) = 1 \\ \beta(x_n) = \sum_{x_{n+1}} p(z_{n+1}|z_n)\beta(x_{n+1}) \end{array} \right. \tag{9}$$

On peut alors en déduire les probabilités marginales *a posteriori* pour tout $n \in \mathcal{N}$ de même que les probabilité jointe des états x_{n-1} et x_n *a posteriori* pour tout $1 \leq n < N$:

$$\left\{ \begin{array}{l} \gamma_n(x_n) = p(x_n|y) \propto \alpha(x_n)\beta(x_n), \forall n \in \mathcal{N} \\ \psi_n(x_{n:n+1}) = p(x_{n:n+1}|y) \propto \alpha(x_n)p(z_{n+1}|z_n)\beta(x_{n+1}), \forall n \in \mathcal{N} \setminus N \end{array} \right. \tag{10}$$

2.3 Algorithme de Baum-Welch

On peut dès lors estimer les paramètres ϕ. Pour cela, on suppose que Z est stationnaire et on maximise par rapport à ϕ la fonction $\mathcal{Q}(\phi|\phi^{[q]}) = \mathbb{E}_{\phi^{[q]}}\{\mathcal{L}_c(Z;\phi)|Y = y\}$ dans laquelle $\mathcal{L}_c(Z;\phi) = \log p(z|\phi)$. On obtient ainsi les formules de réestimations EM des différents paramètres utilisés, ce qui nécessite les valeurs des probabilités *a posteriori* déterminées par les passes forward-backward. Cet algorithme qui n'est autre que l'algorithme EM appliqué aux HMM, est appelé algorithme de Baum-Welch dans la littérature. Dans le cas où la loi d'observation est donnée par une gaussienne multivariée, les différentes formules de réestimation sont les suivantes

$$\left\{ \begin{array}{l} p^{[q+1]}(x_n = i, x_{n+1} = j) = \sum_{n=1}^{N-1} \psi_n^{[q]}(i,j)/(N-1) \\ m_i^{[q+1]} = \sum_{n=1}^{N} \gamma_n^{[q]}(i) y_n / \left(\sum_{n=1}^{N} \gamma_n^{[q]}(i) \right) \\ \Gamma_i^{[q+1]} = \sum_{n=1}^{N} \left(y_n - m_i^{[q+1]} \right) \left(y_n - m_i^{[q+1]} \right)^t \gamma_n^{[q]}(i) / \left(\sum_{n=1}^{N} \gamma_n^{[q]}(i) \right) \end{array} \right. \tag{11}$$

2.4 Algorithme de Viterbi

Pour le calcul du MAP, on utilise l'algorithme de Viterbi qui consiste à remplacer les sommations par des maximisations dans la passe avant. Ainsi, on défini la probabilité $\delta_n(x_n)$ correspondant au chemin optimal pour atteindre l'état x_n à l'instant n avec les observations associées $\delta_n(x_n) = \max_{x_{1:n-1}} p(y_{1:n-1}, x_{1:n-1}, x_n)$ Autrement dit, on cherche le MAP sur la sous-chaîne s'arrêtant à l'instant n et avec X_n fixé. On a alors la récurrence suivante $\delta_{n+1}(x_{n+1}) = \max_{x_n} p(z_{n+1}|z_n)\delta_n(x_n)$. L'argument réalisant ce maximum est $\xi_{n+1}(x_n) = \arg\max_{x_n} p(z_{n+1}|z_n)\delta_n(x_n)$. La dernière étape consiste ainsi à effectuer une passe arrière, le chemin optimal étant donné par $\hat{x}_n = \xi_{n+1}(\hat{x}_{n+1}),\ \forall n \in \mathcal{N} \setminus N$.

3 HMM en reconnaissance de parole

Nous avons présenté le modèle de chaîne de Markov cachée dans la section précédente dans le cas général. Nous précisons à présent comment ce dernier est utilisé dans le cas de la reconnaissance de parole. La loi jointe des processus caché et observé peut-être décomposée classiquement sous la forme d'un produit d'une densité *a priori* et d'une loi d'observation :

$$p(z) = \underbrace{p(x)}_{\text{modèle de langage}} \times \underbrace{p(y|x)}_{\text{modèle acoustique}} \tag{12}$$

Nous explicitons par la suite le modèle de langage et le modèle acoustique.

3.1 Modèle de langage

Dans la modélisation HMM, notre connaissance *a priori* est modélisée par une chaîne de Markov dont les états sont les phonèmes de la langue considérée. En fonction de notre connaissance *a priori*, la chaîne deMarkov utilisée aura une topologie différente.

3.1.1 Texte inconnu

Dans le cas où le texte correspondant au signal de parole n'est pas connu, nous disposons tout de même d'une connaissance *a priori*, a savoir la langue utilisée. Il est possible d'estimer les dépendances entre phonèmes à partir d'un ensemble de textes phonétisés. On en déduit une chaîne de Markov sur l'ensemble des phonèmes de la langue considérée. Cette topologie qui sera utilisée en *reconnaissance* est représentée sur la Figure 2a.

3.1.2 Texte connu

Dans le cas ou l'on connaît le texte correspondant au signal de parole à estimer, on utilise une topologie différente, à savoir une chaîne de Markov gauche-droite dont les transitions sont égales à 1. Etant donné que la suite des phonèmes est connue, il ne s'agit plus que de trouver la position des marques de début et de fin de phonèmes. On parlera alors d'*alignement*. Il est possible de définir des modèles intermédiaire lorsque le texte est connu mais que nous souhaitons prendre en compte des variantes de prononciations ou lorsque le texte est approximativement connu. Pour cela, on part du graphe gauche droite défini précédemment auquel on ajoute des états lorsque des pronnonciations différentes peuvent être envisagées. Cette topologie de la chaîne est représentée sur la Figure 2b.

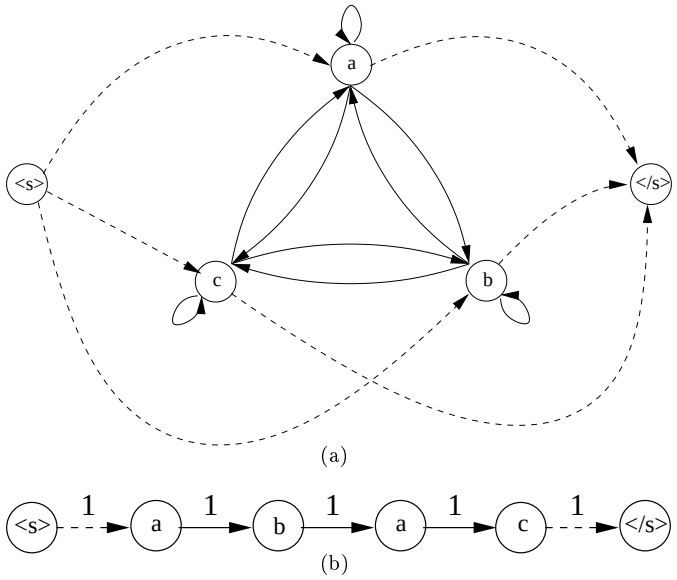


FIG. 2 – Topologie de la chaîne de Markov cachée en fonction de la connaissance *a priori*. a) texte inconnu : modèle de langage (*reconnaissance*), b) texte connu, chaîne de Markov gauche-droite (*alignement*).

3.2 Modèle acoustique

Au niveau du modèle acoustique, nous souhaitons modéliser les différentes variabilités des réalisations de phonèmes.

3.2.1 Mélange gaussien

La première variabilité modélisée est la variabilité intra-locuteur : un locuteur peut prononcer un même phonème de manières différentes et inter-locuteur : différents locuteurs prononcent un même phonème de manières différentes. Pour prendre en compte cette variabilité, on utilise un modèle de mélange gaussien.

3.2.2 Sous-états

D'autre part, les caractéristiques spectrales évoluent au cours du temps. Ainsi, les caractéristiques spectrales sont différentes en début de phonème, en milieu de phonème et en fin de phonème. La solution consiste a considérer non pas un état par phonème mais des sous-états ayant chacun leur loi d'observation associée. Ainsi, à chaque phonème est associé une HMM gauche-droite dite modèle de Bakis généralement à 3 états. On aura donc 36+2 HMMs à estimer. On parlera dans ce cas de modélisation par monophones.

3.2.3 Triphones

D'autre part, les caractéristiques spectrales en début de phonème et en fin de phonème seront différentes en fonction du contexte (phonèmes voisins). Une solution consiste à considérer une HMM pour chaque chaque voisinage gauche-droit. Ces différents HMMs seront appelés modèles

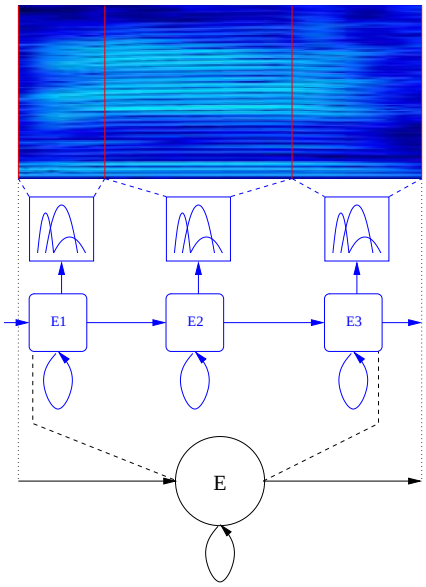


FIG. 3 – Modélisation des différentes variabilités au sein d'un même phonème

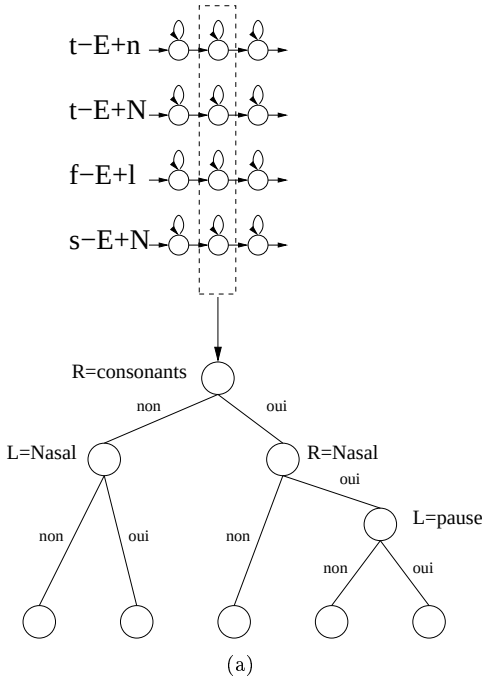
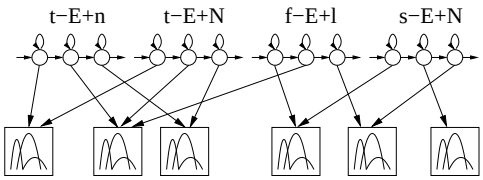


FIG. 4 – Méthodes de réduction du nombre de paramètres dans le cas de la modélisation par triphones par arbre de décision.



(a)

FIG. 5 – Méthodes de réduction du nombre de paramètres dans le cas de la modélisation par triphones par tying.

de *triphones*. Un des problèmes provient du nombre considérable des HMMs à estimer. Ainsi, pour le cas du français qui comprend 36 phonèmes + 2 modèles de silences, le nombre de triphones est de 38^3 soit 54872 modèles à estimer. Afin de palier à ce problème, on utilise une technique dite de tying qui consiste à associer aux sous-états ayant des caractéristiques spectrales semblable la même loi d'observation. Pour choisir les sous états ayant les mêmes caractéristiques spectrale, on procède à un clustering de chaque sous état par arbre de décision, chaque noeud correspondant à une question sur les caractéristiques phonétiques des contextes gauche et droit.

3.3 Estimation des paramètres

En fonction de la modélisation choisie (triphones ou monophones), il est nécessaire d'estimer l'ensemble des modèles. Avant de décrire la procédure d'estimation, nous décrivons les corpus utilisés à l'Ircam.

3.3.1 Corpus disponibles à l'ircam

La description détaillée de ces corpus est disponible sur le wiki sur le lien

`http://wiki.rd/index.php?title=Corpus`

Nous disposons entre autre d'un corpus lié à la conversion de voix utilisé par F. Villavicencio constitué de phrases phonétiquement équilibrées prononcées par différents locuteurs. Un corpus utilisé par S.Farner lié à l'étude des transformation de voix, un corpus de voix expressives lié à l'étude de G.Beller, un corpus de Cocteau utilisé à des fins de test et enfin un corpus prononcé par X.rodet pour la synthèse de parole par concaténation d'unité et un corpus multilocuteur BREF80 utilisé pour l'estimation des paramètres des modèles multilocuteurs utilisés pour la segmentation.

3.3.2 Embedded training

Deux jeux de modèles ont été estimés sur ces deux dernier corpus. La méthode d'apprentissage utilisé est l'*embedded training* prenant en entrée l'ensemble des signaux audio du corpus d'apprentissage et les phonétisation des textes correspondant. La phonétisation est effectué par le phonétiseur Liaphon qui a été modifié afin de prendre en compte les variantes de prononciation. De manière basique, l'embedded training est un algorithme EM qui consiste à augmenter itérativement la vraisemblance. Il s'agit un algorithme itératif relativement lourd en temps de calcul. Pour cela les calculs sont répartis sur l'ensemble des machines de l'équipe AS en fonction de la charge de chaque processeur. On se base sur le fait que les calculs des quantités nécessaires à l'estimation des paramètres peuvent être fait séparément.

3.3.3 Optimisation par A.C.Morris

Concernant les modèles monolocuteurs, ceux-ci ont été optimisé par A.C.Morris realtivement à l'indice match Accuracy. Il a testé différentes topologies de HMMs, les modèles triphones et monophones. Il est arrivé à la conclusion que la topologie la plus adaptée au corpus X.Rodet est un modèle de triphones à 7 états par phonème et 5 gaussiennes. Cette topologie donne de bons résultats avec 93.20 de phonèmes bien reconnu (sans le texte). Nous avons représenté la matrice de confusion pour mettre en évidence nombre relativement peu important d'erreur. Ce type de matrice permet d'identifier les erreurs les plus communes.

4 ircamAlign : implémentation

Venons à présent à la description d'ircamAlign. pour résumer ce que nous avons dit précédemment, ircamAlign est un système de segmentation de signaux de parole au niveau phonème fondé sur la modélisation HMM. 2 jeux de HMMs sont disponibles : un jeu de modèle monolocuteur appris sur le corpus X.Rodet et un jeu de HMMs multilocuteurs appris sur le corpus BREF80. Au niveau du décodage, nous avons deux possibilités : si on ne dipose pas du texte, on utilise alors un modèle de langage au niveau phonème et si l'on dispose du texte, un graphe phonétique est déduit du texte, prenant en compte les variantes de prononciations et permettant également les sauts et les répétitions de mots, erreurs souvent commises par les locuteurs. Les résultats sont évidemment bien meilleurs en utilisant le texte. La ligne de commande sous linux est la suivante. D'autres options sont disponibles, notamment concernant la construction du graphe phonétique. Un manuel est disponible en tapant man ircamAlign dans une console.

L'implémentation a été faite en langage PERL pour des raisons de commodités. On détaille ici l'implémentation en décrivant chacune des fonctions contenue dans les packages PERL

4.1 Prétraitements

4.1.1 Texte

Dans le cas ou l'on dispose du texte, celui est d'abord filtré puis phonétisé. A partir de la phonétisation avec prononciation multiple, on construit un graphe phonétique. Dans le cas contraire, on utilise un modèle de grammaire appris sur des textes en français. L'ensemble des traitements suivant concernant le texte sont effectués par la fonction `Language::phonetize` du package `Language`.

Filtrage On filtre les caractères suivants pouvant être mal interprétés par liaphon : ; ; ! ?"_+(){}[]/ on également les tirets en début de phrase, ainsi que dans certaines configuration posant des problèmes. On filtre également les espaces en début de phrase. Ces opérations de filtrage sont effectuées par la fonction `Language::filtre`.

Phonétisation via Liaphon Une fois le texte filtré, on procède à la phonétisation et à la construction du graphe phonétique en utilisant le phonétiseur liaphon. La construction du graphe phonétique est effectuée par la fonction `Language::buildGraph`. Deux types de phonétisation peuvent être calculés :

- Une phonétisation sur l'ensemble de la phrase prenant en compte la structure de celle-ci ainsi que les liaisons entre mots : `lia_text2phon` ;
- Un phonétisation par mot donnant les différentes variantes de prononciation : `lia_text2phon_lattice`.

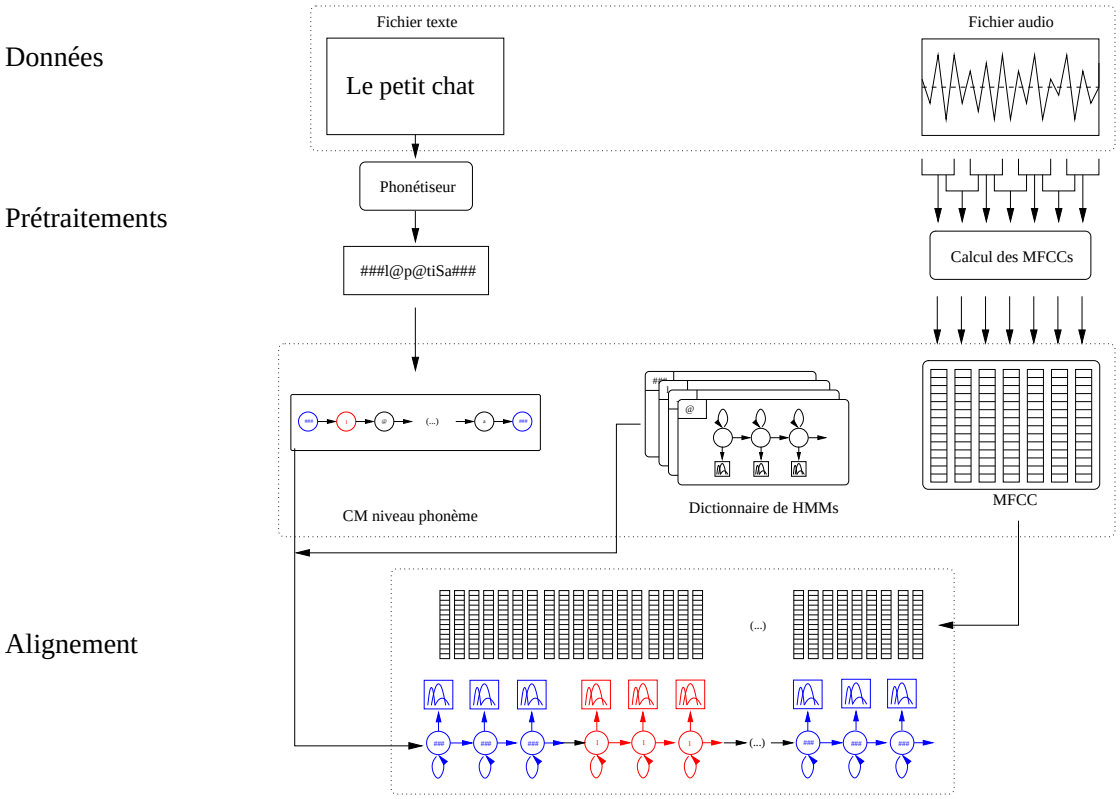


FIG. 6 – Résumé des traitements effectués pour aligner segmenter une signal de parole connaissant le texte prononcé.

Exemple "le petit chat" :

- Sortie `lia_text2phon` :

```
<s> ## [ZTRM->EXCEPTION]
le llee [DETMS]
petit ppeettii [AMS]
chat chaa [NMS]
</s> ## [ZTRM->EXCEPTION]
<FIN> ???? []
```

- Sortie `lia_text2phon_lattice` :

```
<s> ## [ZTRM->EXCEPTION]
le ll|ee*eu*oe*| [DETMS]
petit pp|ee*eu*oe*|ttii [AMS]
chat chaa [NMS]
</s> ## [ZTRM->EXCEPTION]
<FIN> ???? []
```

Le format de la sortie de `lia_phon` est alors converti au format de graphes reconnu par `HTK`. Ce dernier est formé d'expressions régulières encapsulées dans des parenthèses. Ces expressions sont construites à partir de séquences de mots et des métacaractères suivants :

|	: pour désigner des variantes de prononciations
[ ]	: pour encapsuler des caractères optionnels
{ }	: pour désigner zéro ou un répétition
< >	: pour désigner une ou plusieurs répétitions
« »	: pour désigner des boucles sensibles au contexte

Différentes options sont disponibles pour la construction du graphe :
Au niveau des short pauses :

- phonétisation sans short pauses (uniquement les short pauses données par `lia_phon`)
  `(##ll[ee|eu|oe]pp[ee|eu|oe]ttiichaa##)`
- phonétisation avec short pauses obligatoires entre chaque mots
  `(##ll[ee|eu|oe]sppp[ee|eu|oe]ttiispchaa##)`
- phonétisation avec short pauses optionnelles entre chaque mot
  `(##ll[ee|eu|oe][sp]pp[ee|eu|oe]ttii[sp]chaa##)`

Au niveau des mots :

- les mots ne peuvent pas être sautés ou répétés
  `(##ll[ee|eu|oe]pp[ee|eu|oe]ttiichaa##)`
- les mots peuvent être sautés mais pas répétés
  `(##[ll[ee|eu|oe]][pp[ee|eu|oe]ttii][chaa]##)`
- les mots peuvent être sautés et répétés
  `(##{ll[ee|eu|oe]}{pp[ee|eu|oe]ttii}{chaa}##)`
- Conversion au format `HTK` mise dans un fichier *.syn :

Conctruction du graphe phonétique Le graphe phonétique est alors converti au format net à partir de la fonction HTK `HParse`.

```
HParse $syn_name $net_name
```

Modèle bigram Dans le cas où le texte du signal de parole traité n'est pas connu, on utilise un modèle de langage comme, décrit précédemment. Ce modèle est appris à partir de la phonétisation du corpus d'apprentissage.

4.1.2 Acoustique

Conversion du format audio et de la fréquence d'échantillonnage, normalisation par niveau vocal actif, calcul des coefficients MFCC.

Conversion du format audio Le format audio ainsi que la fréquence d'échantillonnage doivent correspondre à celles utilisées pour le corpus d'apprentissage. Si ce n'est pas le cas, le format et la fréquence d'échantillonnage sont convertis par superVP.

Niveau vocal actif Si temps réel, pas de normalisation. Si pas de temps réel, meilleurs résultats si normalisation. La normalisation est effectuée en considérant le niveau vocal actif déterminé par l'algorithme `p56` (voir l'article).

Calcul des coefficients mel cepstraux Définition des coefficients mel cepstraux. Les coefficients mel cepstraux sont calculés à partir des log-amplitudes des bancs de filtre $\{m_j\}$

$$c_i = \sqrt{\frac{2}{N}} \sum_{j=1}^{N} m_j \cos\left(\frac{\pi i}{N}(j - 0.5)\right)$$

où N est le nombre de canaux du banc de filtres.

4.2 Décodage

Après prétraitement, le signal de parole peut être segmenté. On utilise pour cela la fonction HVite de HTK en précisant le nom du fichier audio et le nom du graphe phonétique utilisé. Dans le cas ou le texte est disponible, on utilisera le graphe déduit de ce texte. Dans le cas où le texte n'est pas connu, on utilise le graphe correspondant au modèle bigram. Le décodage est réalisé par l'algorithme de viterbi sur le graphe phonétique donné. Des options permettent de pondérer l'importance de l'information *a priori* lors du décodage. Le décodage peut également être effectué par liste.

4.3 Apprentissage

Initialisation : soit flat-start, soit boot-strap à partir de signaux de parole segmentés à la main. Puis, embedded training réparti sur plusieurs machines. Puis calculs des sommes sur une machine. Split de gaussiennes, triphones.

4.4 Description des Packages

4.4.1 Utils

getFieldGenConf Extrait la ligne du fichier general conf correspondant au champ passé en entrée. le tag de sélection inverse, si mis à 1, extrait les champs différents du champ passé en entrée.
Entrée : path de la configuration générale + champ correspondant à la ligne à extraire + tag de sélection inverse
Sortie : nom du(des) champ(s) correspondant à la ligne à extraire (exemple : `"FORCE|ALLOw|NAT"`)
Prototype : `getFieldGenConf($gen_conf,$field,$neg)`.

getInfoGenConf Retourne la valeur du champ passé en entrée.
Entrée : path de la configuration générale + nom du champ dont on souhaite extraire la valeur
Sortie : nom du champ dont on souhaite extraire les valeurs dans le fichier de configuration générale
Prototype : `getInfoGenConf($gen_conf,$field)`.

getFileFormat Extrait le format du fichier passé en entrée
Entrée : nom du fichier
Sortie : format du fichier
Prototype : `getFileFormat($input_file)`.

getFileName Extrait le nom du fichier passé en entrée
Entrée : nom du fichier
Sortie : nom du fichier sans l'extension
Prototype : `getFileName($input_file)`.

checkMonoMLF Vérifie quels sont les monophones compris dans le fichier MLF donnée en argument.
Entrée :path du fichier mlf, liste des monophones, répertoire de destination
Sortie : liste des monophones inclus dans le MFL+liste des monophones non inclus dans le MLF
Prototype : `checkMonoMLF($mlf_path,$monophones_list,$dir)`.

createMLF Créer un fichier MLF (Master Label File HTK) à partir d'une liste de fichiers de labels donnée en entrée
Entrée : liste des fichier de label à inclure dans le fichier MLF + répertoire de destination
Sortie :path du fichier mlf
Prototype : `createMLF($lab_list,$dir)`.

4.4.2 Langage

filtre Filtre la chaîne de caractère passée en entrée afin d'enlever les caractères provoquant les erreurs de phonétisation de Liaphon.
Converti Est en début de phrase en est
Filtre la ponctuation
Filtre les tirets mal interprétés
Ajoute un point à chaque fin de phrase
Entrée : chaîne de caractère non filtrée
Sortie : chaîne de caractère filtrée
Prototype : `filtre($sentence)`.

buildGraph Construction du graphe phonétique sous la forme d'une chaîne de caractère interprétable par HTK
signification des tags :
$lia_tag :
0 : phonétisation liaphon sur la phrase complète (prise en compte de la structure de la phrase)
1 : phonétisation liaphon par mot avec les différentes variantes de prononciation
$sp_tag :
0 : phonétisation sans short pauses (uniquement les short pauses données par liaphon)
1 : phonétisation avec short pauses obligatoires entre chaque mots
2 : phonétisation avec short pauses opitonnelles entre chaque mot
$words_tag :
0 : les mots ne peuvent pas être sautés ou répétés
1 : les mots peuvent être sautés mais pas répétés
2 : les mots peuvent être sautés et répétés
Entrée : phrase à partir de laquelle construire le graphe phonétique + tags liaphon

Sortie : chaîne de caractère correspondant au graphe interprété par HTK
Prototype : `buildGraph($sentence,$lia_tag,$sp_tag,$words_tag)`.

phonetise Crée le graphe phonétique au format HTK (net).
-filtrage du texte avec la fonction filtre
-phonétisation à l'aide de la fonction buildgraph
-création du fichier syn
-création du fichier net en utilisant HParse
Entrée : nom du fichier texte, tags liaphon, tag binaire
Sortie :nom du fichier net
Prototype : `phonetise($txt_file,$lia_tag,$sp_tag,$words_tag,$binary_tag)`.

createNETList Crée les net des fichiers inclus dans la liste donnée en entrée et renvoie la listes des paths de ces fichiers
Entrée : liste des fichiers textes, répertoire de destination des fichiers net, tag lia, tag binaire
Sortie : nom de la liste NET
Prototype : `createNETList($txt_list,$net_dir,$lia_tag,$sp_tag,$words_tag,$binary_tag)`.

createNETListJ

Crée les net des fichiers inclus dans la liste donnée en entrée et renvoie la listes des paths de ces fichiers + les grammaires réduites aux phrases (pour Julien)
Entrée : liste des fichiers textes, répertoire de destination des fichiers net, tag lia, tag binaire
Sortie : nom de la liste NET
Prototype : `createNETListJ($txt_list,$net_dir,$lia_tag,$sp_tag,$words_tag,$binary_tag,$monophonesDA`

4.4.3 French

lia2htk Converti la chaîne de caractère du format liaphon ver le format HTK.
Entrée : chaîne de caractère au format liaphon
Sortie : chaîne de caractère convertie au format HTK
Prototype : `lia2htk($sequence)`.

htk2Xsampa Converti la chaîne de caractère du format HTK au format XSampa.
Entrée : chaîne de caractère au format HTK
Sortie : chaîne de caractère convertie au format XSampa
Prototype : `htk2Xsampa($sequence)`.

writeFrenchMonophoneListHTK Créer la liste des phonèmes (monophones.list) utilisés en français au format HTK.
Entrée : répertoire vers lequel créer la liste
Sortie : path complet de la liste
Prototype : `writeFrenchMonophoneListHTK($dir)`.

writeFrenchMonophoneDAPListHTK Créer la liste des phonèmes (monophones.list) utilisés en français au format HTK et rajoute !ENTER à la fin de la liste, nécessaire pour le décodage utilisant le modèle bigram.
Entrée : répertoire vers lequel créer la liste
Sortie : path complet de la liste
Prototype : `writeFrenchMonophoneDAPListHTK($dir)`.

writeFrenchMonophoneListLIA Créer la liste des phonèmes (monophones.list) utilisés en français au format LIA (sera utilisé à la place du format dit HTK par la suite).
Entrée : répertoire vers lequel créer la liste
Sortie : path complet de la liste
Prototype : `writeFrenchMonophoneListLIA($dir)`.

writeFrenchPhonGramHTK Créer le lexique (phongram.list) nécessaire au décodage utilisant le modèle bigram.
Entrée : répertoire vers lequel créer le lexique au format HTK
Sortie : path complet du lexique
Prototype : `writeFrenchPhonGramHTK($dir)`.

writeFrenchPhonGramLIA Créer le lexique (phongram.list) nécessaire au décodage utilisant le modèle bigram.
Entrée : répertoire vers lequel créer le lexique au format LIA
Sortie : path complet du lexique
Prototype : `writeFrenchPhonGramLIA($dir)`.

convertLabFormat Converti le fichier de label passé en entrée selon le tag donné en entrée.
Correspondance des tags :
0 : format Xsampa phones
1 : format XSampa phones + marquage centre phones
Entrée : fichier lab au format HTK + format_tag
Sortie : path du fichier lab converti selon le tag.
Prototype : `convertLabFormat($lab_file,$lab_format_tag)`.

4.4.4 Acoustic

writeHCopyConfig Ecrit la configuration nécessaire à HCopy(calcul des MFCC) en extrayant les champs nécessaire de la configuration générale.
Entrée : Répertoire de destination + path de la config générale
Sortie : path de la config Hcopy (HCopy_conf)
Prototype : `writeHCopyConfig($dir,$gen_conf)`.

checkAudioFile

Vérifie et converti la Fréquence d'échantilonnage ainsi que le format du fichier audio pour que celui-ci corresponde aux caractéristiques du corpus sur lequel les modèles ont été entrainé
Entrée : nom du fichier audio + path de la configuration générale
Sortie : nom du fichier wav
Prototype : `checkAudioFile($input_file,$wav_file_SR)`.

computeMFCC Calcul des coefficients mfcc. Procède de la manière suivante :
-conversion du fichier audio (format+Fe)
-Normalisation du fichier par aslnorm si le tag est activé (uniquement en offline)
-calcul des coefficients MFCC par HCopy
Entrée : fichier audio + configuration HCopy + Fe du corpus d'entrainement + tag de normalisation
Sortie : path du fichier mfcc calculé
Prototype : `computeMFCC($input_file,$HCopy_conf,$wav_file_SR,$norm_tag)`.

createMFCCList Calcul des MFCCs (avec normalisation ou non) des fichiers inclus dans la liste donnée en entrée et renvoie la liste des paths de ces fichiers.
Entrée : liste de fichiers audio, path de la configuration générale, répertoire de travail, tag de normalisation
Sortie : nom de la liste MFCC
Prototype : `createMFCCList($audio_file_list,$HCopy_conf,$wave_file_SR,$mfcc_dir,$norm_tag)`.

4.4.5 Decoding

writeHViteConfig

Ecrit la configuration nécessaire à HVite(décodage Viterbi) en extrayant les champs nécessaire de la configuration générale.
Entrée : répertoire de destination + path de la config générale
Sortie : path de la config Hcopy (HCopy_conf)
Prototype : `writeHViteConfig($dir,$gen_conf)`.

alignSolo Alignement d'un fichier audio relativement au net donné en entrée (ce graphe peut correspondre au modèle de langage ou au graphe phonétique sur la phrase)
Entrée : configuration HVite + path modèle à utiliser + fichier net + grammaire bigram + liste des monophones + fichiers mfcc + tag format
Sortie : path du fichier de labels
Prototype : `alignSolo($HVite_conf,$model_in,$net_file,$phon_gram,$monophones_list,$mfcc_file,$format`

alignMulti Alignement d'une liste de fichiers audio relativement à la liste de fichiers net donné en entrée (ce graphe peut correspondre au modèle de langage ou au graphe phonétique sur la phrase)
Entrée : configuration HVite + path modèle à utiliser + liste de fichiers net + grammaire bigram + liste des monophones + liste de fichier mfcc + tag format
Sortie : path du fichier de labels
Prototype : `alignMulti($HVite_conf,$lab_dir,$model_in,$net_list,$phon_gram,$monophones_list,$mfcc_li`

4.4.6 Training

buildProto Crée un modèle de HMM prototype
Entrée : nombre d'états, répertoire de destination
Sortie : répertoire de destination
Prototype : `buildProto($nstates,$dir)`.

initialiseProto (Flat Start) Initialise le modèle HMM prototype par la variance et la moyenne globale et recopie le moèle prototype pour chacun des modèles de la liste des monophones.
Entrée : fichier de configuration HCompV + liste MFCC + répertoire de modèle utilisé + liste des monophones
Sortie : path du fichier hmmmdef
Prototype : `initialiseProto($HCompV_conf,$mfcc_list,$model_dir,$monophones_list)`.

initialiseProtoHRest Crée le fichier MACROS.
(nom à modifier) (ne crée que le fichier Macro pour l'utilisation de Hinit+HRest)
Entrée : fichier de configuration HCompV + liste MFCC + répertoire de modèle utilisés + liste des monophones
Sortie : path du fichier hmmmdef
Prototype : `initialiseProtoHRest($HCompV_conf,$mfcc_list,$model_dir)`.

HInit Initialisation de chaque modèle HMMs de phonèmes par k-means.
Entrée : fichier de configuration HCompV + liste de monophones contenus dans les phrases segmentées manuellement + liste de monophones non contenus dans les phrases segmentées manuellement + path du fichier mlf manuel + répertoire du modèle in + répertoire du modèle out
Prototype : `HInit($HCompV_conf,$mono_in_list,$mono_out_list,$mlf_path,$mfcc_list,$model_dir,$model_`

HRest Effectue une initialisation par HRest (voir HTK book)
Entrée : fichier de configuration HCompV_conf + liste de monophones contenus dans les phrases segmentées manuellement + liste de monophones non contenus dans les phrases segmentées manuellement + path du fichier mlf manuel + répertoire du modèle in + répertoire du modèle out
Sortie : path des modèles
Prototype : `HRest($HCompV_conf,$mono_in_list,$mono_out_list,$mlf_path,$mfcc_list,$model_dir,$model_`

splitMixture Fonction permettant d'ajouter une ou plusieurs gaussienne au mélange
Entrée : répertoire du modèle initial + répertoire du modèle final, liste des monophones, nombre d'états, nombre de gaussiennes souhaitées dans le mélange
Prototype : `splitMixture($model_dir,$model_dir_out,$monophones_list,$nstates,$nmix)`.

5 Perspectives