Projet n°2

Rapport : Apprentissage de modèles de Markov cachés et détection de mots clés.

Virgile CARTIER

MASTER 2

Langue & Informatique

Table des matières

Élaboration des ressources

Présentation de l’enregistrement

Notre corpus est un enregistrement audio de radio France 98 du 12 juillet de cette même année de 3 minutes exactement. Celui-ci comporte plusieurs locuteurs et, dans la mesure où il s’agit d’une émission de radio, ceux-ci sont dans des situations d’énonciation différentes.

De fait, les deux premières minutes de notre enregistrement sont composées de deux professionnels de la radio lisant un script ainsi que d’une publicité lue également ; aussi, notons que celle-ci est quelque peu polluée par un *jingle* de fond. Au reste, il y a dans ces deux premières minutes extrêmement peu de dysfluences – nous l’avons-vu dans la partie 1 de ce projet – la diction y est extrêmement claire et le flot de parole n’est pas vraiment naturel. Blanche-Benveniste soulevait dans son ouvrage *Approche de la langue parlée en français* [BBV00] que quand un locuteur est amené à parler, il a généralement idée de ce qu’il va dire : il actualise alors une structure syntaxique déjà construite par un lexique qu’il choisit *sur le moment*. Aussi, ce processus occasionne quelques flottements, répétitions, etc – ces *dysfluences*, propres à la parole spontanée.

En conséquence, ces deux premières minutes sont quelque peu atypiques, s’il s’agit bel et bien d’un enregistrement typique de la radio, ce n’est vraisemblablement pas un enregistrement caractéristique de l’oralité dans sa globalité.

Méthodologie du traitement de notre enregistrement

Comme nous l’avons évoqué, cet enregistrement est d’une durée de 3 minutes très exactement. Aussi, celui-ci est composé en majorité d’un script lu, notamment dans les deux premières minutes.

Ce sont ces mêmes deux premières minutes que nous allons découper en tours de parole et sur lesquelles nous générerons les modèles de Markov cachés pour mener nos expériences.

Étant confrontés à de la parole lue, les tours de parole ne sont pas nécessairement aisés à déterminer. Ainsi, nous avons choisi de les découper aux changements de locuteurs (notamment le cas du tour 1 au tour 2) ou encore aux respirations d'un locuteur (par exemple, du tour 4 au tour 5) dans ce qui est *a priori* un même flux de parole. Cela nous offre deux avantages : il s'agit d'une méthodologie que nous pourrons appliquer à toute notre séquence et nous retirons ainsi les respirations en dehors de notre jeu de données d'apprentissage.

Une fois ces tours de paroles découpés, nous les avons annotés manuellement avec le logiciel *Praat*, notamment en utilisant sa fonction d'*alignement d'intervalle* qui génère un équivalent phonétique de notre transcription. Nous l'avons méticuleusement reprise de façon à bien nous assurer de la qualité de cette transcription phonétique. Une fois réalisée, nous avons utilisé un script *Python* de façon à transposer les phonèmes *API*[1] en *Sampa* ; notons au passage que certains phonèmes seront ainsi perdus, *ɛ̃* et *œ̃*, soient les deux voyelles nasales équivalentes au son */ in /* du français (dont on trouve l'opposition dans la paire *brun* vs. *brin* par exemple) seront retranscrits sans distinction en *e~*. Aussi, il s'agit là de deux phonèmes très peu distincts ; au reste, une grande majorité de locuteurs ne les différencie pas.

Ceci fait, nous obtenons un fichier *lex0.txt* avec l'intégralité de nos tours de parole suivis de leur transcription phonétique :

- Tour_1 **e~ s t a~ a v E k v e R o n i k s a~ s o~ d i s E t 9 R v e~ t k a t R**

- **Tour_2** **b o~ Z u R s E j a n i k n o a e s i k O m m w a v u E t d E p R o f E s j O n E l d y f u t b O l e k u t e k a t R @ v e~ d i z H i t R a d j o f R a~ s l a R a d j o d @ l a k u p d y m o~ d**

- Tour_3 **d @ p H i l @ n 9 f Z H e~ o~ n a R @ s y d e d i z E n d e~ v i t e s y R k a t R @ v e~ d i z H i t R a d j o f R a~ s d e z O m d e f a m d e s p e s j a l i s t d e z a m a t 9 R u d e n e o f i t**

1 *Alphabet phonétique internationale*

- Tour_4 e t u s R E v E d y n f i n a l f R a~ s b R e z i l s E t f w a s a i E o~ E n i E

- Tour_5 e m e~ t n a~ s @ m a t S d y s j E k l i l f o l @ g a J e

- Tour_6 s @ n E s a~ d u t p a l @ p l y f a s i l m E s E t o p j e d y m y R k o~ R @ k O n E l @ m a s o~

- Tour_7 o~ n a t E l m a~ d i k E m e Z a k E a v E k o~ s t R H i y n e k i p d @ m a s o~ o~ n @ p 2 k @ s a~ R e Z H i R o Z u R d H i

- Tour_8 d @ l a m E m m a n j E R k o~ v a p a a f R a~ k f O R a v E k s e s o s i s

- Tour_9 o~ n @ s a t a k p a o b R e z i l l a f l 9 R o f y z i l a m E j 9 R a t a k d @ s @ m o~ d j a l l @ b R e z i l

- Tour_10 o R a s a~ d u t f O R t a f E R f a s a l a m E j 9 R d e f a~ s l a f R a~ s

- Tour_11 l e t a~ S a~ Z a~ k a t R @ v e~ d 2 u k a t R @ v e~ s i s

- Tour_12 o~ v a~ t E n o t R m i l j 2 d @ t E R e~ Z e n j a l e k R e a t i f m E b a t y

- Tour_13 o~ m E t o Z u R d H i a~ n e k s E R g n o t R m a~ t a l n o t R s o l i d a R i t e e n o k a l i t e d e f a~ s i v

- Tour_14 l a v i k t w a R s @ R a p 2 t E t R o b u

- Tour_15 a~ t u k a l e b l 2 n o~ p a d @ k o~ p l E k s a n u R i R p a z a~ k O R p u R a v w a R y n S a~ s i l f o p a R t i R a e g a l i t e e s @ n E p a e v i d a~

- Tour_16 i l f o t a b s o l y m a~ u b l i e k @ l @ b R e z i l E S a~ p j o~ d y m o~ d

- Tour_17 k a t R f w a S a~ p j o~ d y m o~ d i l f o s @ d e b a R a s e d @ s e~ k a~ t H i t s w a s a~ t d 2

- Tour_18 s w a s a~ t d i s e k a t R @ v e~ k a t O R z

- Tour_19 R E j e p @ l e R o n a l d o g a R i n S a u z i k o d e k a R t

- Tour_20 s @ d i R k @ l @ Z o n p 2 t o s i E t R l a k u l 9 R d @ l E s p w a R

- Tour_21 k @ l @ b l 2 d w a t E t R s E l d @ l a v i k t w a R a~ n e~ m o i l f o t i k R w a R

- **Tour_22** l a k u p d y m o~ d Z @ v 2 d i R m w a 9 Z e n e R a l m a~ l @ f u t m e~ t e R E s E p a d y t u m E s E t k u p d y m o~ d l a m a v R E m a~ e~ t e R E s e e d o~ k d @ p H i 9 e~ m w a Z @ v w a l e m a t S Z e s H i v i t u l e m a t S p a R s k @ l a f R a~ s Z @ s a~ k @ l a f R a~ s v a p 2 f E R Z y s t m a~ p u v E f E R k E l k @ S O z p u R s @ 9 m o~ d j a l e E l l a p R u v e

- Tour_23 d a v i d m a R s @ l e~ b o~ s w a R

- Tour_24 b o~ s w a R f R a~ k b a l a~ Z e b o~ s w a R a t u s

- Tour_25 o Z u R d H i d u z Z H i j E m i l n 9 f s a~ k a t R @ v e~ d i z H i t l a f R a~ s 9 a~ t j E R t u R n o t u R d y b a l o~ e v u n a l e 9 s a~ d u t p a m @ k o~ t R @ d i R

- Tour_26 w i d @ v a~ s a t e l e u d @ v a~ e~ n e k R a~ d a~ z e~ t a k s i d a~ z 9 b a R u e~ R E s t o R a~ a~ n e k u t a~ l a R a d j o

De ceux-ci, le **tour_2** et le **tour_22** sortent du lot :

- Le premier est un *jingle* de publicité où la parole prononcée est *noyée* dans un fond sonore musical et d'ambiance *football* (chants de supporters et autres *hourras* peuvent être entendus).
- Le second est le seul tour de parole non scripté de ce corpus d'apprentissage ; il s'agit de la réponse spontanée d'un interviewé – en ce sens, nous entendons divers dysfluences. De plus, d'autres sons parasites s'entendent pendant que le locuteur parle.

Hypothèses et théories

Ce découpage a cela de particulier que nos modèles de Markov cachés apprendront sur des phonèmes prononcés de façon claire et limpide. Par contre, ceux-ci seront alors utilisés sur la deuxième partie de notre enregistrement qui présente trois problèmes majeurs :

- Il s'agit en majorité de parole spontanée, en effet, c'est une interview.
- L'interviewée et l'intervieweuse sont deux femmes, pourtant notre corpus d'apprentissage est composé de paroles prononcées par des hommes.
- Elles se trouvent dans un lieu où la qualité audio n'est clairement pas équivalente à celle d'un studio de Radio France ; vraisemblablement un monastère. En conséquence, la qualité du son chute drastiquement.

Pour toutes ces raisons, nous pouvons supputer que notre corpus d'apprentissage n'est pas nécessairement approprié pour une reconnaissance phonétique parfaite de notre corpus de test. Néanmoins, il devrait être possible d'y reconnaître quelques mots clefs – aussi, nous pourrons les sélectionner de façon à illustrer ces problématiques : un segment de notre corpus de test, les 24 premières secondes, sont des paroles prononcées par un animateur radio, elles seront vraisemblablement plus aisées à reconnaître. En conséquence, nous tâcherons de sélectionner nos mots-clefs selon ces trois

critères : prononcés par un homme uniquement, par un homme et une ou des femmes, par une ou des femmes uniquement.

Segmentation et apprentissage des machines de Markov phonétiques

Premiers traitements HTK

La fonction HCopy

Nos tours de paroles ont été étiquetés en phonèmes Sampa, nous allons maintenant générer nos machines de Markov.

Pour ce faire, nous allons lancer les scripts *Perl runParamApp* et *runParamTest*

Ceux-ci vont exécuter la commande *HCopy* de *HTK* suivante :

```
HCOPY  -C $hcopyConf -D $refDir$signal/$liste[0].wav refDir$param/$liste[0].mfc
```

- -C spécifie le nom du dossier de configurations, soit *configs/hcopyWAV*
- -D permet d'afficher les paramètres de configurations. [HTK05]

Ici, nous utilisons *HCopy* de façon à paramétrer nos fichiers *wav* en en générant les *mfc* correspondant « *Since input files may be coerced, HCopy can also be used to convert the parameter kind of a file, for example from WAVEFORM to MFCC, depending on the configuration options* ». [BBV00]

Les *MFC* ou « *Mel Frequency Cepstral Coefficients* » sont des « *representation of a the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency*. » [ZZS01]. Comme évoqué, *HTK* nous permet de générer automatiquement les équivalents *.mfc* de nos *.wav* ; notons que nous n'avons pas modifier le fichier de configuration.

Il faut préciser comme variable la liste correspondant à la transcription phonétique de nos tours de paroles, dans nos fichiers *lex0.txt*.

Ainsi, ce script *Perl* va peupler notre dossier *param/apprentissage* des .mfc et va également générer une liste correspondant au chemin de tous ces fichiers dans *donnees/Locuteur/lists. runParamTest* fera la même chose pour notre *.wav* de test (*seqTest* dans nos fichiers).

La fonction HVite

Nous allons maintenant générer nos .lab correspondant aux phonèmes de notre enregistrement d'apprentissage. Deux méthodes sont possibles :

- les créer manuellement via nos annotations *Praat*
- les générer via l'utilisation de la commande *HTK HVite* « *the recogniser tool HVite can be used to perform a forced alignment of the training data*. » [HTK05]

Nous utiliserons la deuxième méthode mais nous pourrons utiliser *Praat* afin de vérifier si la transcription est correcte – pour ce faire, nous utiliserons un script *Perl* écrit par *Carolina Rodriguez* qui nous permet de générer les .*TextGrid* équivalant, ceux-ci pourront être ouverts avec leur fichier son correspondant afin de vérifier si les annotations sont correctement réalisées. Les *.lab* se présentent comme suit :

```
0 1100000 e# -389.613953
1100000 2200000 s -384.210815
2200000 2700000 t -174.678513
2700000 4700000 a# -779.906250
4700000 6100000 a -542.677551
6100000 6400000 v -116.416138
6400000 7600000 E -447.877960
7600000 7900000 k -114.473907
7900000 9000000 v -396.526642
9000000 9300000 e -117.016510
9300000 10800000 R -590.669556
10800000 11100000 o -125.452148
11100000 11500000 n -139.434494
11500000 11900000 i -156.917084
```

Ces données correspondent à l'intervalle de prononciation d'un phonème donné en microsecondes [µs] :

Début et Fin pour le phonème *s*

```
11900000 12400000 k -204.666946
12400000 13400000 s -342.216095
13400000 14500000 a# -410.707825
14500000 15600000 s -376.635712
15600000 17300000 o# -605.419556
```

Aussi, pour générer les *.lab* correspondant à nos données, nous utiliserons le script *Perl runAlign* qui permet de généraliser cette commande *HVite* à notre jeu de données.

Pour ce faire, nous devons renseigner les dossiers où sont rangés nos données, notamment nos *.mfc*, ainsi que la liste comprenant notre transcription phonétique. Il sera alors généré après exécution des fichiers *.lab* dans le répertoire *donnee/Locuteur/lab/*.

Après vérification des annotations sous *Praat*, nous pouvons attester de la précision de cette fonction. En cela, nous devons noter que la segmentation phonémique d'un flux de parole est une tâche relativement complexe : de fait, il n'est pas évident de précisément trouver la *frontière* entre un phonème et son voisin ; certaines seront évidentes, d'autres se fondent dans un flux de parole naturelle et, bien que les sons s'entendent distinctement à l'oreille, retrouver leur moment exact d'articulation peut s'avérer particulièrement ardu. Après analyse des résultats proposés par la fonction *HVite d'*HTK, nous trouvons l'alignement proposé tout à fait acceptable. Nous pourrons d'ailleurs éventuellement en juger selon les résultats obtenus.

Les fonctions HInit, HRest, HERest et HHEd, l'apprentissage

Maintenant que nos *.lab* ont été générés, nous pouvons procéder à l'apprentissage de nos modèles de Markov cachés. Pour se faire, nous allons exécuter le script *Perl runApprentissage* qui va lancer sur notre jeu de données les fonctions *HInit, HRest, HERest, et HHEd* :

- *HInit* utilise l'algorithme de Viterbi « *l'agorithme de Viterbi permet de corriger, jusqu'à un certain point, les erreurs survenues lors d'une transmission à travers un canal bruité. Son utilisation s'appuie sur une connaissance du canal bruité, c'est-à-dire la probabilité qu'une information ait été modifiée en une autre, et permet de simplifier radicalement la complexité de la recherche du message d'origine le plus probable. D'exponentielle, cette complexité devient linéaire*. (…) *Cet algorithme a pour but de trouver la séquence d'états la plus probable ayant produit la séquence mesurée*. » [FOR73]. « *HInit is used to provide initial estimates for the parameters of a single HMM using a set of observation sequences. It works by repeatedly using Viterbi alignment to segment the training observations and the recomputing the parameters by pooling the vectors in each segments.* » [HTK05]
- *HRest* est particulièrement semblable à *HInit* à l'exception près que celle-ci n'utilise pas l'algorithme de Viterbi « *Its operation is very similar to HInit except that* (…) *it expects the input HMM definition to have been initialised and it uses Baum-Welch re-estimation in place of Viterbi training*. » [HTK05]. Cette fonction sera exécutée à partir du dossier hmm.0, soit la résultante de *HInit* – les résultats obtenus seront placés dans hmm.1.
- *HERest* va, quant à lui, mettre à jour tous les HMMs dans un système en utilisant toutes les données d'entraînement « *On startup, HERest loads in a complete set of HMM definitions. Every training file must have an associated label file which gives a transcription for that file.* » [HTK05]. Cette fonction va être exécutée à partir du dossier hmm.1, soit la résultante de *HRest* – les résultats obtenus seront placés dans hmm.2.
- HHEd va affiner les données d'apprentissage davantage « *HHEd takes as input a set of HMM definitions and outputs a new modified set, usually to a new directory.* » [HTK05]. En effet, il va utiliser les résultats stockés dans le dossier hmm.2 pour en générer de nouveaux stockés dans hmm.3 – la fonction *HERest* leur sera enfin appliquée finissant ainsi le traitement de nos données d'apprentissage.

Nous pouvons dès alors nous atteler à mettre en œuvre ces HMMs sur notre jeu de Test.

Décodage acoustico-phonétique et détection de mots-clés

Décodage de phonèmes

Maintenant que nos données ont été traitées avec les fonctions *HTK*, nous pouvons maintenant traiter notre fichier de test de façon à en reconnaître les phonèmes grâce aux HMMs appris sur notre jeu d'apprentissage. Comme nous l'avons supposé ci-dessus, nous estimons que les résultats seront potentiellement mauvais ; de fait, notre jeu de données d'apprentissage est composé pour l'essentiel de paroles lues par un homme – avec une voix particulièrement grave soit dit en passant – avec une qualité d'enregistrement digne d'un studio de radio France. Notre jeu de test, quant à lui, est pour l'essentiel constitué d'une interview, les locutrices sont des femmes – même si un homme parle les 20 premières secondes – et les qualités d'enregistrement sont *de terrain*, quelques bruits parasitent l'entretien.

Aussi, nous allons pour se faire utiliser la fonction *HVite* d'*HTK* pour générer des résultats transcrits grâce aux HMMs appris sur l'apprentissage.

Fonction HVite

HVite va nous permettre de mettre en œuvre tous nos HMMs, cette fonction va générer un fichier de résultat (.*rec*) que nous pourrons comparer avec notre transcription manuelle. « *HVite is a general-purpose Viterbi word recogniser. It will match a speech file against a network of HMMs and output a transcription for each*. » [HTK05].

Pour ce qui est de notre transcription manuelle, nous avons utilisé la même méthode telle qu'expliquée ci-dessus. Voici la commande utilisée :

```
HVITE $hviteConf -H $refDir/$HMM -i dap.rec -S $refDir/$paramList -w tmp/net2
$fileDic $hmmList
```

Nous allons générer le fichier *dap.rec* ainsi que le fichier *net2*, la liste de référence utilisée est une liste des phonèmes Sampa disponibles *lists/dap.net*.

Visualisation des résultats avec HResults

Maintenant que nous obtenons notre fichier, nous pouvons commencer la comparaison et évaluer notre modèle.

```
HResults -S liste_matrice.txt -S test phonesHTK.lst
```

Nous utiliserons la fonction *HResults* pour ce faire qui est une fonction permettant la comparaison et la lecture des résultats générés par la fonction *HVite* « *HResults is the HTK performance analysis tool. It reads in a set of label files (typically output from a recognition tool such as HVite) and compares them with the corresponding reference transcription files*. »

Ici, *liste_matrice.txt* est un fichier texte comportant le chemin vers le répertoire où sont stockés nos résultats (*rec/dap.rec*) – test est le nom du document où se trouve la transcription manuelle de notre jeu de test.

Voici ce que nous obtenons :

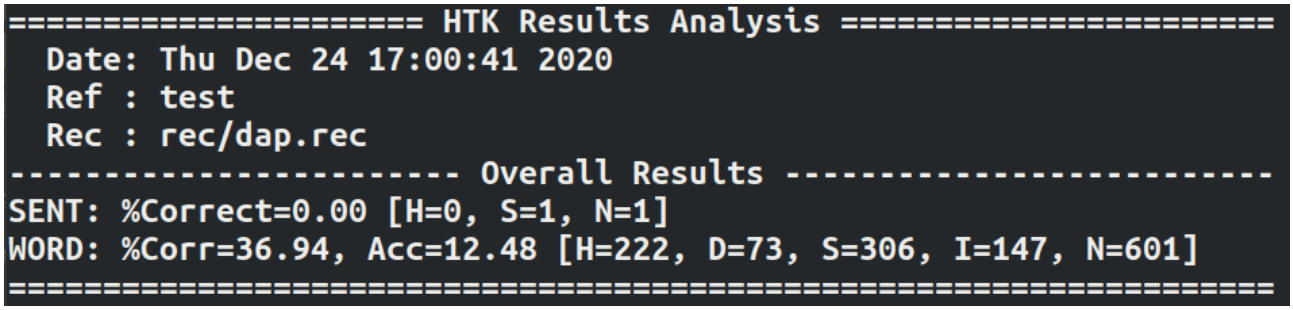


Figure 1: résultats HResults sur notre séquence de Test

Globalement, les résultats obtenus ne sont pas extraordinaires, H = 222, soit 222 labels corrects, D = 73, soit 73 suppressions, S = 306, soit 306

substitutions et enfin I = 147, soit 147 insertions – le tout sur N = 601, soit 601 le nombre total de labels dans le fichier de transcriptions manuelles.

Pour les raisons évoqués plus haut, il semble que le jeu de données sur lequel est conçu nos HMMs n'est peut-être pas le plus approprié dans le but de reconnaître les phonèmes de notre séquence de test – aussi, cela n'est pas surprenant et il est tout à fait possible que nous obtenions de meilleurs résultats concernant la détection de mots-clés.

Détection de Mots-clés

Conception des mots-clés et premiers résultats

Pour nos mots-clés, nous avons choisi des termes que nous retrouvons dans notre jeu d'apprentissage : *foot*, *match*, *radio*, *France* autant que ceux trouvés uniquement dans notre jeu de test : *personne*, *sœur*, *règlement*.

Nous estimons que les premiers seront plus facilement reconnus.

Pour cela, nous exécutons le script *runDetections1.pl*, celui-ci utilise la fonction *HVite* d'*HTK* que nous avons déjà vu. Il nous faudra écrire une liste (*lists/lex1.txt*) comportant les mots-clés que nous recherchons dans notre séquence de test, une autre liste lui donnant accès à certains *objets* avec lesquels les HMMs annoteront la séquence (soit nos mots-clés ou un phonème) – il s'agit des documents *lists/keywordsplus.dic* et *lists/keywordsplus.net*.

En outre, cet algorithme génère dans le dossier *tmp/net2* un document qui nous permettra de *récompenser* nos HMMs ; nous l'utiliserons en effet pour créer une nouvelle liste *lists/keywordsplus.net2* qui nous permettra d'utiliser un nouveau script, celui-ci prenant en compte de potentielles récompenses.

Soit, keywordsplus.net2 qui se présente avec différentes variables comme suit :

- N = nombre de nœuds
- L = nombre de transitions

- I = l'état, avec W = la valeur du nœud
- J = les transitions de l'état S à l'état E

En définitive, cela fonctionne comme un automate, il s'agira de favoriser certaines transitions en leur donnant un poids, ces fameuses *récompenses*.

Voyons nos résultats *au naturel*, pour ce faire, il s'agit d'exécuter une commande *HResults* :

```
HResults -w -f label.lst EN.rec
```

Décryptons cette commande : il s'agit de détecter nos mots-clés contenus dans *label.lst* dans le fichier *EN.rec* généré par *runDétections2*. Pour comparer ce fichier, nous avons manuellement conçu un fichier *EN.lab* correspondant aux mots-clés et leurs intervalles (en microsecondes) dans notre séquence de test.

```
---------------------- Figures of Merit -----------------------

     KeyWord:    #Hits     #FAs  #Actual      FOM
        foot:        0        0        2     0.00
       match:        0        0        3     0.00
     seconde:        0        0        2     0.00
    personne:        0        0        2     0.00
       soeur:        0        0        1     0.00
    probleme:        0        0        1     0.00
       radio:        0        0        1     0.00
      France:        0        0        1     0.00
   reglement:        0        0        1     0.00
      finale:        0        0        1     0.00
     Overall:        0        0       15     0.00
```

Figure 2: FOM de nos mots-clés sur notre séquence de Test

Nous pouvons le constater, nos résultats sont particulièrement mauvais, aucun mots-clés n'a été détecté. Aussi, nous devrons utiliser les *récompenses* pour essayer de les améliorer – pour ce faire, nous devrons y aller à tâtons.

En outre, nous sommes aussi à même d'améliorer nos résultats en augmentant notre jeu de données d'apprentissage, ainsi, nous avons échangé avec un camarade nos tours de parole segmentés de façon à améliorer nos résultats.

Résultats finaux avec un jeu de données plus complet

Pour étoffer notre jeu de données d'apprentissage, il nous faudra revenir en arrière – c'est-à-dire relancer certains scripts après avoir placé ces nouvelles données au bon endroit – c'est-à-dire étoffer notre *lists/lex0.txt* des nouveaux tours de parole que nous aurons placés dans notre document *donnees/Locuteur/wav/apprentissage*.

Ceci fait, nous relançons nos scripts de façon à reconstruire nos HMMs grâce à ces nouvelles données – c'est-à-dire dans un premier *runParamApp* de façon à créer nos nouveaux *mfc* ainsi que pour mettre à jour *donnees/Locuteur/lists/lex0.txt*. Une fois fait, nous lançons *runApprentissage* sans faire de *runAlign* au préalable dans la mesure où nos *.lab* ont déjà été manuellement alignés tout comme ceux de Nicolas ; il serait contre-productif de les recréer.

Regardons si notre détection phonémique est maintenant plus précise ; pour cela, relançons `HResults -S liste_matrice.txt -S test phonesHTK.lst :`

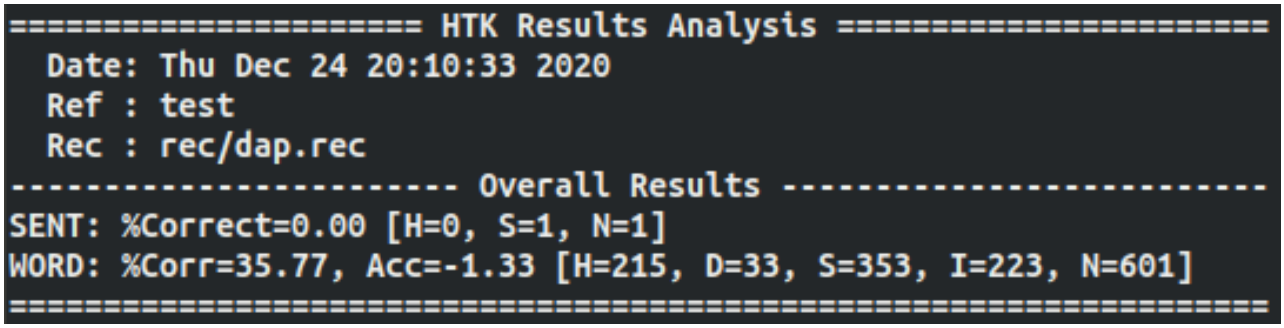
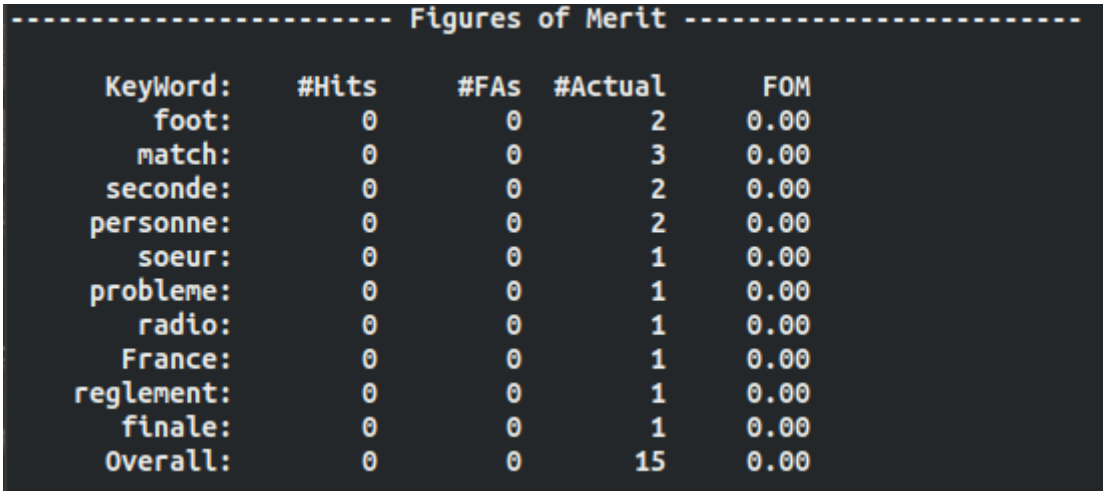


Figure 3: résultats HResults sur notre séquence de Test avec complémentation des données

Nous pouvons le remarquer, plus de données ne se traduit pas nécessairement en de meilleurs résultats. Ici, nous perdons quelques pourcentages de précisions.

Voyons ce qu'il en est avec notre liste de mots-clés. Il nous faut relancer les scripts *runDetections1*, reconduire les manipulations déjà évoquées puis lancer le script *runDetections2* pour enfin vérifier les résultats.

Encore une fois, les résultats obtenus sont particulièrement mauvais :

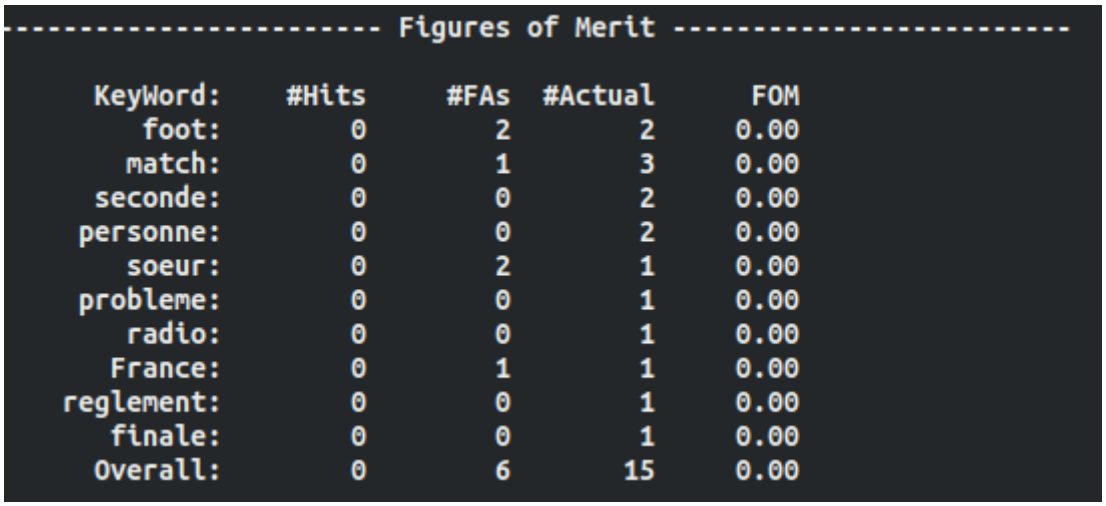


```
---------------------- Figures of Merit ------------------------

     KeyWord:    #Hits     #FAs  #Actual      FOM
        foot:        0        0        2     0.00
       match:        0        0        3     0.00
     seconde:        0        0        2     0.00
    personne:        0        0        2     0.00
       soeur:        0        0        1     0.00
    probleme:        0        0        1     0.00
       radio:        0        0        1     0.00
      France:        0        0        1     0.00
   reglement:        0        0        1     0.00
      finale:        0        0        1     0.00
     Overall:        0        0       15     0.00
```

Figure 4: FOM de nos mots-clés sur notre séquence de Test avec complémentation des données

Il nous faudra nécessairement avoir recours aux *récompenses* pour obtenir de meilleurs résultats, pour ce faire, nous allons modifier notre fichier *keywordsplus.net2* de façon à favoriser les transitions aboutissant aux mots-clés à détecter. Aussi, dans la mesure où les résultats obtenus sans la complémentation de nos données avec celles de Nicolas ne sont pas meilleurs, nous allons revenir à notre jeu de données d'apprentissage originel ; voyons nos résultats :



```
---------------------- Figures of Merit ------------------------

     KeyWord:    #Hits     #FAs  #Actual      FOM
        foot:        0        2        2     0.00
       match:        0        1        3     0.00
     seconde:        0        0        2     0.00
    personne:        0        0        2     0.00
       soeur:        0        2        1     0.00
    probleme:        0        0        1     0.00
       radio:        0        0        1     0.00
      France:        0        1        1     0.00
   reglement:        0        0        1     0.00
      finale:        0        0        1     0.00
     Overall:        0        6       15     0.00
```

Figure 5: FOM avec les récompenses pour nos mots-clés réglées à l = 10

En réglant nos points à l = 10, nous pouvons constater que nos HMMs repèrent bel et bien ces mots-clés, néanmoins, ceux-ci ne sont pas situés au bon endroit, il s'agit donc de faux positifs.

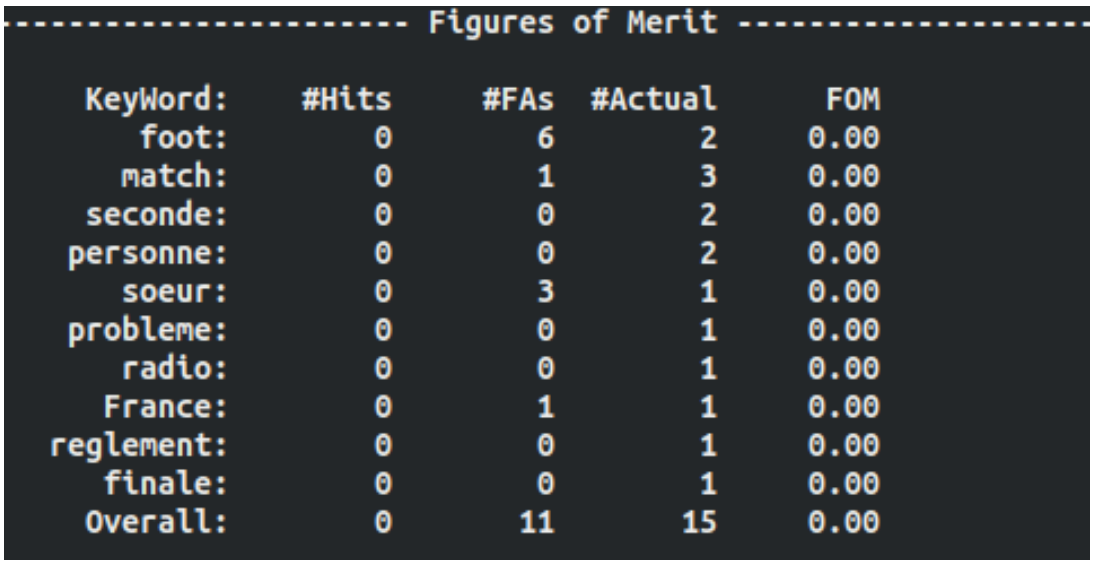


Figure 6: FOM avec les récompenses pour nos mots-clés réglées à l = 15

Nous pouvons constater ici une augmentation des faux positifs, cela dit, les mots correspondants sont toujours nuls. Augmentons encore nos *récompenses*.

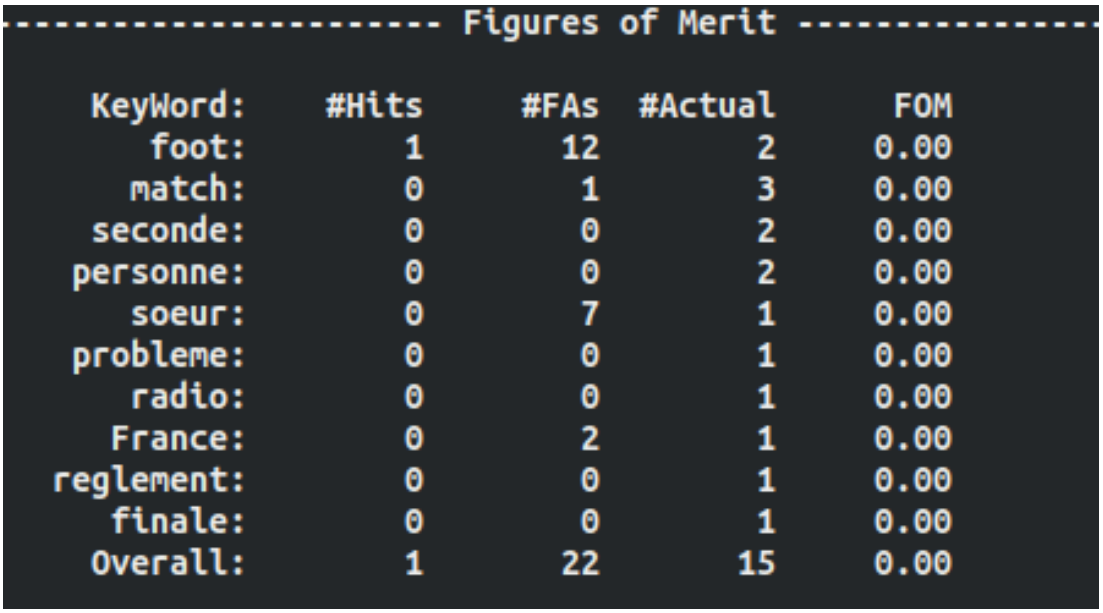


Figure 7: FOM avec les récompenses pour nos mots-clés réglées à l = 20

Enfin, nous réussissons à repérer l'un de nos mots-clés, *foot*. Malheureusement, nous pouvons constater que cela s'accompagne d'une augmentation drastique des faux positifs (le double).

Aussi, nous pensons qu'il sera quelque peu compliqué de réussir à trouver nos mots-clés, nous pourrions augmenter la valeur des *récompenses*

liées aux potentiels mots-clés détectés mais cela s'accompagnera forcément d'un nombre considérable de faux positifs.

Conclusion et perspectives

Comme nous l'avions supputé, les HMMs que nous avons générés n'ont pas enregistré de résultats particulièrement satisfaisants concernant la détection de phonèmes. Cela dit, la grande surprise a été de constater que, quelque soit nos efforts, les mots-clés restaient presque impossibles à détecter pour notre système – même en ajoutant à notre jeu de données celle d'un de nos camarades, les résultats restaient particulièrement faibles. Aussi, le jeu des *récompenses* s'avérait malheureusement peu utile dans la mesure où celui-ci tendait surtout à accroître les faux positifs.

Aussi, les raisons que nous avions évoquées au début de ce devoir semblent avoir été vérifiées après expérience ; ces trois critères ont eu raison de nos HMMs : d'homme à femme, de studio d'enregistrement à conditions *sauvages*, de parole lue à parole spontanée, la différence semble trop grande.

Aussi, nous pourrions très certainement améliorer ces résultats en suivant diverses stratégies ; par exemple, faire une distinction entre un phonème prononcé en début de mots, au même prononcé en milieu ou en fin de mot. Aussi, nous pourrions éventuellement pré-traiter nos données de façon à réduire le poids des dysfluences, ou encore réduire également le poids des bruits parasites ; et pour ce faire, sans doute, des méthodes que nous ne connaissons pas encore.

Bibliographie

BBV00: Claire Blanche-Benveniste, Approches de la langue parlée en français, 2000
HTK05: Steve YOung, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Gareth Moore, Julian odell, Dave Ollason, Dan Povey, Valtcho Valtchev, Phil Woodland, The HTK Book, 2005
ZZS01: Zheng, F. Zhang, G. Song, Z., Comparison of different implementations of MFCC, 2001
FOR73: G.D. Forney, The viterbi algorithm, 1973