Travaux pratiques

Chaines de Markov – chaines de Markov cachées

Ce TP nécessite l'installation sur votre machine du package python « pomegranate» version 1.1.2 .

Installation

Ce package s'appuie sur le package pytorch.

- Si vous utilisez anaconda le plus simple pour ne pas modifier l'environnement que vous utilisez dans d'autres cours est de créer un autre environnement en installant « miniconda » puis en installant tous les packages requis :
  1. Installer miniconda depuis le site officiel : https://docs.anaconda.com/miniconda/install/
  2. Ouvrir la fenetre de commande de miniconda et installer les modules suivants :
     conda install anaconda ::matplotlib
     conda install anaconda ::spyder
     conda install anaconda ::scipy
     conda install anaconda ::scikit-learn
     conda install anaconda ::seaborn
     conda install conda-forge ::pomegranate

     Vous pouvez ensuite lancer spyder à partir de la fenetre de commande : spyder

- Si vous utilisez vscode
  pip install pomegranate

Vous pouvez lancer le petit script « TestPackage.py » pour vérifier l'installation

Documentation du package pomegranate :
https://pomegranate.readthedocs.io/en/latest/

https://github.com/jmschrei/pomegranate

Préambule : Les parties 1 et 2 sont relativement guidées avec des questions précises. A la partie 3 qui concerne la reconnaissance de mot vous devez au contraire prendre des initiatives. Cette partie est longue, il faut impérativement lire toute la partie Introductive AVANT de venir en TP.

La partie 4 est en bonus !

Les parties en rouges sont à préparer avant la séance de TP !

Chaines de Markov

Parite 1: pleut-il ?

On souhaite modéliser les périodes de précipitation / sécheresse à l'aide d'une petite chaine de Markov à 2 états. Pour cela, on fait l'hypothèse que la prévision de la pluie en un lieu donné à l'instant t+5 minutes dépend fortement de la connaissance à l'instant t de l'état (pluie/ non pluie) à ce même lieu. Plus précisément, on fait l'hypothèse que s'il

pleut maintenant, il pleuvra dans 5 minutes avec une probabilité de α et s'il ne pleut pas maintenant la probabilité qu'il pleuve dans 5 minutes est β. On note par E0 l'état « Sec » s'il ne pleut pas et par E1 l'état « Pluie ». La probabilité que le modèle soit dans l'état « sec » à t=0 est noté γ. Ce problème peut donc se mettre sous la forme d'une chaîne de Markov à deux états. A l'aide de données pluviométriques on a déduit les paramètres suivants α= 0.65, β= 0.02 et γ=0.9.

1. Déduire le modèle de Markow associé à ce problème **: E, A et π_0**
2. Représenter le graphe
3. Montrer que la probabilité qu'il pleuve vaut : $P(E_1) = \beta / (1+\beta - \alpha)$.
4. On va estimer empiriquement cette probabilité sur une séquence générée par ce modèle afin de la comparer à la valeur théorique précédente. Le script « exercice1aCompeleter.py » fournit un exemple de code pour générer une chaine de Markov cachée. La ligne de code « obsSeq,statesSeq = model.sample(1) » retourne la séquence des observables ainsi que la séquence des états. Afin de simuler une chaine de Markov et non une chaine de Markov cachée, comment faut-il définir la matrice d'émission B afin que la séquence des observables soit identique à celle des états ? Compléter le code. Générer une séquence de taille suffisante afin que la probabilité estimé empiriquement soit un « bon » estimateur de P(E1).
5. On définit la durée d'un événement de pluie D_{pluie} comme étant la durée où le modèle reste dans l'état E_1. De même on définit la durée d'une période sèche D_{sec} comme étant la durée où le modèle reste dans l'état E_0. Montrez que le modèle à 2 états implique que la probabilité qu'un évènement de pluie dure D suit une loi géométrique de la forme :

$$P(D_{pluie} = D) = pq^{D-1}$$

Déterminer les expressions de q et p.

La fonction « duree » disponible dans le fichier duree.py calcule les durées des périodes sèches et pluvieuses d'une série temporelle. Elle retourne 6 arrays contenant les durées de sécheresse et de pluie, **les densités de probabilités empiriques de sécheresse et pluie et les classes associés** aux probabilités de sécheresse et de pluie :

```
from duree import duree
dSec,dPluie,pdfSec,pdfPluie,binsSec,binsPluie=duree(obsSeq)
```

Vérifier que la loi de probabilité empirique des durées de pluie obtenues sur la série simulée est bien en adéquation avec la loi théorique. Vous pouvez utiliser la ou les méthodes de votre choix pour comparer les deux distributions : on peut par exemple comparer la moyenne et la variance théoriques avec la moyenne et variance empirique. Vous pouvez également représenter en échelle semi-log sur une même figure la loi de probabilité empirique des durées de pluie et celle estimée par la formule théorique ci-dessus. On rappelle que pour une loi géométrique X de paramètre p et q :

$$E\{X\} = 1/p \qquad V\{X\} = q/p^2$$

On considère des mesures de pluie réalisées en région parisienne à l'aide d'un pluviomètre. Le fichier « RR5MN.mat » contient une série de mesures collectées sur une période de 761 jours (environ 2 ans) à la résolution de 5 minutes. Une valeur 0 indique l'absence de pluie tandis qu'une valeur 1 indique qu'il pleut.

La syntaxe python pour charger ce fichier est la suivante :

```
import scipy.io.matlab as mio
ObsMesure = mio.loadmat('RR5MN.mat')['Support'].astype(np.int8).squeeze()-1
```

6. Charger le fichier puis comparer les caractéristiques statistiques des durées de pluie obtenues sur les mesures avec celles obtenues avec le modèle de Markov à 2 états. Le modèle de Markov à 2 états est-il un bon modèle pour modéliser les périodes de pluie ?

7. Faire une étude identique en considérant les durées des périodes sèches. Préciser notamment l'expression théorique de $P(D_{sec} = D)$.

8. Conclure quant à l'utilisation d'une chaine de Markov à 2 états pour générer un support de pluie. L'hypothèse d'un modèle à deux états est-elle pertinente pour modéliser les périodes de pluie et de sécheresse ?

Chaines de Markov cachées

Partie 2 : peut-il (suite) ?

On modifie le modèle précédent pour le transformer en une chaine de Markov cachée. On fait l'hypothèse que l'état du système représentera la présence ou non de nuage. La sortie du modèle (ie. l'observable) représentera le fait qu'il pleuve ou non (sec/pluie). On supposera un modèle à 3 états : Etat ciel clair sans nuage 'Clear Sky' E_0, Etat nuageux 'Cloudy' E_1 et Etat très nuageux 'Very Cloudy' E_2. En présence de ciel clair la probabilité de pluie est nulle alors qu'en présence de ciel très nuageux elle sera au contraire forte. Une étude préliminaire a permis d'estimer le modèle suivant :

E = { Clear Sky, Cloudy, V. Cloudy}

$$T = \begin{pmatrix} 0.9 & 0.1 & 0 \\ 0.5 & 0.4 & 0.1 \\ 0 & 0.35 & 0.65 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 \\ 0.8 & 0.2 \\ 0 & 1 \end{pmatrix} \gamma_0 = (0.5,\ 0.4, 0.1)$$

1. Représenter le graphe. Que signifie d'un point de vue « physique » la valeur 0 de la première ligne colonne 3 de la matrice de transition ? Même question pour la valeur 0 dans la matrice d'émission

2. Générer une séquence de longueur suffisante, puis en comparant (par la/les méthode(s) de votre choix) avec la série expérimentale, répondre à la question : cette modélisation permet-elle de bien représenter le support de la pluie en termes de pourcentage de pluie / sécheresse ? En termes de distribution de durée de pluie / sécheresse ? Comparer aux résultats obtenus à l'exercice précédent

3. On se propose d'améliorer ce modèle en ajustant les paramètres des matrices de transition et d'émission à l'aide de l'algorithme de Baum Welch et de la série expérimentale. La méthode « fit() » permet de faire cela :

```
X=np.reshape(ObsMesure[:100000],(1,-1,1)).astype('int32')   # ajuste les parametres sur les 100000 premieres valeurs
model.fit(X)
model.ends[0],model.ends[1],model.ends[2]=-46.0, -46.0, -46.0
obsSeq,statesSeq = model.sample(1)
obsSeq= obsSeq[0][:,0]
statesSeq= statesSeq[0]
```

Comparer les nouvelles matrices de transition et d'émission aux matrices initiales et discuter des changements obtenus. Peut-on réellement continuer d'interpréter les 3 états comme étant : Etat ciel clair, Etat nuageux et Etat très nuageux ?

```
print('Matrice de transion : ',model.edges.exp())
print('Matrice d emission :',d0.probs,d1.probs,d2.probs)
```

4. Générer une nouvelle séquence de longueur suffisante à l'aide du nouveau modèle et comparer les distributions des durées obtenues par ce modèle avec les distributions obtenues avec les mesures. A votre avis, cette nouvelle modélisation pourrait-elle être intéressante pour la modélisation du support de la pluie ? Justifier.

Partie 3 : reconnaissance vocale de mots isolés

Introduction

Le but de cette partie est de développer une méthode permettant d'identifier des mots (en nombre limité). Ces mots sont définis dans un petit dictionnaire de Q mots. Pour cela, on se propose de développer un modèle de reconnaissance vocale à base de chaines de Markov cachées à émission gaussienne.

Le principe est le suivant :

- Il s'agit pour chacun des Q mots du dictionnaire d'apprendre une chaine de Markov λ_q.
- En présence d'un mot inconnu (sous la forme d'une séquence d'observable notée O_T), l'algorithme devra pour chaque chaine de Markov λ_q estimer la probabilité de d'observer le mot inconnu $P(O_T|\lambda_q)$, puis de déduire quel mot du dictionnaire le mot inconnu est le plus proche.

L'ensemble du processus de reconnaissance vocale d'un mot sera divisé en 3 parties A, B et C.

A. Etape d'extraction de caractéristiques de l'enregistrement vocal d'un mot : il est difficile d'utiliser directement une série temporelle du signal vocal avec une chaine de Markov. Des caractéristiques doivent en être extraites (voir dernière partie du polycopié de cours et le lien suivant : https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html. Pour cela, dans un premier temps, l'enregistrement temporel d'un mot est découpé en T tranches (appelées trames ou frames) d'une durée typique de 10 à 30 ms (20 ms dans le TP). On extrait ensuite pour chaque frames un ensemble de caractéristiques. Plusieurs techniques d'extraction de caractéristiques sont couramment utilisées. Dans ce TP nous allons utiliser 3 méthodes d'extraction :

1. **Méthode Spectrum** : pour chaque frame son spectre est calculé à l'aide d'une transformée de Fourier rapide (fft) sur 256 valeurs. Le spectre étant pair, on obtient donc un vecteur de 256/2 = 128 features / frame.
2. **Méthode Filter** : à partir du spectre obtenu par la méthode Spectrum précédente on calcule les énergies obtenues par les 26 filtres de Mel (cf. cours et lien ci-dessus). On obtient ainsi un vecteur de 26 features / frame.
3. **Méthode MFCC** (Mel-Frequency Cepstral Cœfficients) : on calcule 12 coefficients basés sur la transformée en cosinus discréte des énergies obtenues précédemment par la méthode Filter (cf. cours et lien ci-dessus). On obtient donc un vecteur de 12 features / frame

On dispose suivant la méthode d'extraction de caractéristiques utilisée d'un vecteur notés o_i de dimension 256, 26 ou 12 pour chaque frame. On définit la séquence d'observable $O_T = o_1 o_2 \ldots o_T$ comme étant une suite de vecteurs o_i contenant chacun les features d'une frame (dans le cours la séquence d'observable était une suite de valeurs discrètes scalaires).

Dans ce TP, afin de ne pas trop alourdir les temps de calcul on dispose seulement des enregistrements de 7 mots. Chacun des mots a fait l'objet de 15 enregistrements dans des conditions différentes. On dispose donc au total de 15 x 7 = 105 fichiers audios (format *.wav) contenant 105 séquences audio. Les fichiers sont stockés dans le dossier 'audio'.

B. Apprentissage : Pour chacun des Q = 7 mots on dispose de 15 enregistrements (observables). Il s'agira d'utiliser ces enregistrements pour apprendre les Q chaines de Markov (une chaine par mot).

C. Phase de reconnaissance d'un mot inconnu

En présence de l'enregistrement vocal d'un mot inconnu, la reconnaissance du mot sera divisée en trois phases :

1. **Etape extraction des caractéristiques de l'enregistrement vocal inconnu** : On applique au mot inconnu un des 3 traitements d'extraction de caractéristique défini à la section A pour obtenir la séquence d'observable $\mathbf{O_T = o_1 o_2 \ldots o_T}$

2. **C**alcul de $P(O_T|\lambda_q)$: à partir de la séquence d'observation $\mathbf{O_T}$ du mot inconnu on calcule la probabilité $P(O_T|\lambda_q)$ pour chacune des q = *1 .. Q* chaines de Markov apprises précedemment.

3. Bloc décision : On cherche parmi les Q probabilités calculées précedemment laquelle maximise :

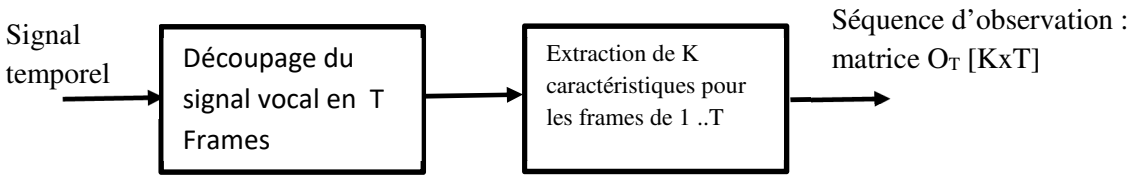
$$q_{mot} = \underset{q\in[1\,..\,Q]}{\operatorname{argmax}} P(\mathrm{O_T}|\lambda_q)P(q)$$

Dans notre cas on supposera que tous les mots sont équiprobables. La formule précédente devient donc :

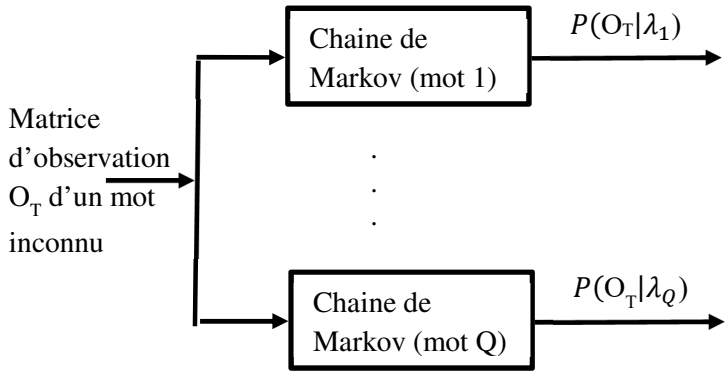
$$q_{mot} = \underset{q\in[1\,..\,Q]}{\operatorname{argmax}} P(\mathrm{O_T}|\lambda_q)$$

L'ensemble du processus est résumé à la figure ci-dessous.

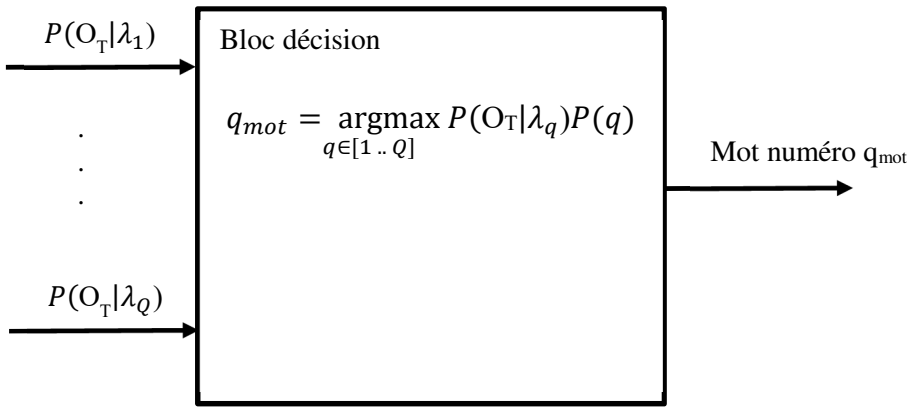
- **Extraction des caractéristiques :**



- **Chaines de Markov**



- **Décision**



Remarque concernant les chaines de Markov utilisées dans ce TP : Concernant l'émisson d'un symbole, il y a deux différences importantes par rapport au cours :

- On utilisera ici des chaines de Markov cachées à émission gaussienne (GaussianHMM) et non à émission discrète. En effet, les features sont des valeurs continues et non un ensemble fini de valeurs discrètes.
- Les observables o_n ne sont pas des scalaires mais des vecteurs de dimension K contenant les features émises à chaque frame.

Les symboles émis par la chaine de Markov s_n sont donc des vecteurs de dimension K représentant les caractéristiques (notées Feature ou Feat ou F dans le TP) d'une frame à l'instant n :

$$s_n = \begin{pmatrix} F_1 \\ F_2 \\ \dots . \\ F_K \end{pmatrix}$$

La probabilité $b(s_n)$ d'émettre le symbole s_n sera donc définie à l'aide d'une loi normale multidimentionnelle de dimension K. Cette loi est définie par un vecteur $\boldsymbol{\mu}$ de dimension K contenant les espérances et par sa matrice de variance-covariance $\boldsymbol{\Sigma}$ de dimension [KxK]. De plus, les paramètres de la loi normale dépendent de l'état E_i ($i \in [1\,..\,N]$) dans lequel on se trouve mais aussi de la chaine de Markov q ($q \in [1\,..\,Q = 7]$) considérée :

$$b_i^q(s) = \frac{1}{(2\pi)^{N/2}|\Sigma_i^q|^{1/2}} e^{-\frac{1}{2}(s-\mu_i^q)^T(\Sigma_i^q)^{-1}(s-\mu_i^q)}$$

1) Etude qualitative des séquences :

Dans tout ce qui suit, la durée d'une frame est fixée à 20 ms et le chevauchement entre 2 frames est de 10 ms.

Le dossier 'partie 3' contient tous les fichiers nécessaires. Le sous dossier « audio » contient 7 sous dossiers intitulés : apple, banana, …. Chacun de ces sous-dossiers contient 15 fichiers audio contenant des enregistrements de chacun des 7 mots (enregistrés par la même personne dans différentes conditions).

Le programme Ex3_exemple1.py permet de charger les 105 enregistrements (7 mots x 15 enregistrements par mot), d'extraire les features pour les 3 méthodes considérées (mfcc, filter et spectrum) et de faire différents affichages afin de comparer les 3 méthodes. Vous pouvez écouter les enregistrements des différents mots en double-cliquant sur le fichier .wav correspondant.

Les différentes méthodes utilisées sont décrites ci-dessous :
Méthode plotOneWord : La figure 1 du programme exemple1.py est composée de 4 subplots. Le premier permet de visualiser le signal temporel du premier enregistrement du mot 'banana' (record =0). Les autres subplot affichent les features obtenues pour chaque trame pour les 3 méthodes décrites plus haut : 1. Spectrogramme (méthode 'spectrum'), 2. les énergies obtenues par les filtres de Mel (méthode 'filter'), 3. les coefficients de Mel (méthodes 'mfcc'). La variable myWord permet de sélectionner un mot tandis que la variable record permet de sélectionner 1 enregistrement parmi les 15 disponibles.

Méthode CorrFeatures : La figure 2 montre pour le mot sélectionné et pour chacune des 3 méthodes la matrice d'autocorrélation obtenue à partir des features correspondantes

Méthode histFeatures : Les figures 3 à 6 montrent les histogrammes des Features du mot banana pour les 3 méthodes. Pour chaque figure l'histogramme rouge représente la première partie du mot, le vert la partie centrale et le bleu la fin du mot.

Méthode plotFeatureXY : La figure 7 montre les features 2 et 5 de banana pour les 3 méthodes. La couleur des points indique si la frame est plutôt située en début du mot, au milieu ou à la fin (Rouge : début, Vert : milieu, Bleu : fin). Il est possible de modifier les arguments I et J pour sélectionner d'autres features

Méthode TSNEFeatures : La figure 8 montre la TSNE calculée sur l'ensemble des features pour un mot donné ('apple' par défaut). La couleur des points indique si la frame est plutôt située en début, milieu ou fin du mot

Vous pouvez modifier les variables suivantes afin de réaliser l'étude :
- myword : contient le mot choisi parmi les 7 mots disponibles ('apple', 'banana', 'kiwi', 'lime', 'orange', 'peach', 'pineapple').
- record : numéro de l'enregistrement (0..14). Utile uniquement pour la figure 1
- X et Y : contient les indices des 2 composantes de feature à afficher par la méthode 'plotFeatureXY'.

Le but de cette première partie est de réaliser une analyse qualitative permettant de comparer entre elles les 3 méthodes d'extraction pour différents mots (4 – 5 pages maximum).

Indications : Vous pouvez par exemple dans un premier temps à l'aide de la méthode « plotOneWord » comparer les features obtenus de différents mots. Choisir des mots avec des sons graves, d'autres avec des sons aigus, des mots courts, des mots longs, etc…
Vous pouvez également comparer les corrélations des features pour les 3 méthodes d'extraction pour différents mots. Comparer aussi les nuages d'individus pour différents couples de composantes pour différents mots et méthodes vous pourrez également vous aider de la visualisation des features à l'aide de la TSNE.
Toute autre initiative est la bienvenue

Conclure en formulant des hypothèses sur la ou les méthodes d'extraction des features qui vous semble la meilleure pour différencier les mots les uns des autres.

2) Apprentissage d'une Chaine de Markov cachée associée à un mot

Le script Ex3_exemple2.py permet d'apprendre un modèle de HMM pour un mot donné. La classe GaussianHMM permet de créer et d'entrainer une chaine de Markov avec la méthode de Baum-Welch à partir des séquences de features stockées dans une liste fournie en argument. Le script présente un exemple basique permettant d'apprendre une chaine de Markov cachée à 3 états pour le mot 'apple' et la méthode 'mfcc'. La variable Nstates permet de choisir le nombre d'états de la chaine. Dans cet exemple, seules les 3 premières features (featStart =0, featStop = 2) sont utilisées dans les séquences d'observation.

- La méthode plotGaussianConfidenceEllipse permet d'afficher les features des composantes Fx et Fy d'un mot ainsi que les ellipses à 95% de confiance des gaussiennes associées à chacun des états. Le numéro au centre des centroïdes permet d'identifier l'état.
- La méthode log_prob estime le log de la densité de probabilité de la séquence observée.
- La méthode predict permet d'estimer la sequence d'état optimale par l'algorithme de Viterbi
- La méthode getTrans permet de récupérer la matrice de transition
- la méthode getPi0 permet de récupérer les probabilités initiales des états
- les méthodes getCov et getMu permettent de récupérer les matrices de covariance et les vecteurs moyennes associées aux gaussiennes

Lancer le programme et comprendre le script, les sorties sur la console et les figures.

Analyse

Choisir quelques mots très différents et faire une étude pour les 3 méthodes d'extraction des features (penser que la méthode spectrum a beaucoup de features (128)) :
Vous pouvez par exemple faire varier le nombre de features apprises et comparer le 'mappage' des individus par les ellipses différents plan (Fx, Fy). Faire pour différents mots.
Vous pouvez également faire varier le nombre d'états. Le nombre d'états optimal est-il identique pour tous les mots ? Justifier.
Vous pouvez également comparer les valeurs (densités) de probabilité obtenus sur les 15 enregistrements de chaque mot pour chaque méthode.

Vous pouvez également comparer pour chaque méthode l'ordre de grandeur entre les variances et les covariances des matrices de covariances. Qu'observe-t-on ?
Etudier la séquence des états optimale pour chacun des 15 enregistrements. La chaine obtenue est-elle du type left-to-right ? Mettre les séquences en relation avec la matrice de transition. Vous pourrez par exemple tester pour des HMM ayant un nombre d'états compris entre 2 et 5.
Au final, que peut-on en conclure ? Y a-t-il une ou plusieurs méthode(s) d'extraction qui semblent mieux convenir ? Est-ce en accord avec vos conclusions de la partie précédente ? Y a t-il Un nombre d'état optimal ?
Tout autres commentaires sont les bienvenues

3) Apprentissage d'une Chaine de Markov cachée pour chaque mot du dictionnaire

Il s'agit dans cette partie d'apprendre Q=7 chaines de Markov pour chacun des mots du dictionnaire. Compte tenu du nombre très restreint d'enregistrements disponibles, vous utiliserez pour l'apprentissage de chaque mot les 10 premiers enregistrements mais en les bruitant (mettre l'argument noise à la valeur 500 lors de la lecture des enregistrements) et les 5 derniers enregistrements pour la validation :

```
words=Words(rep='audio',name='audio',numcep=numcep,lowfreq=lowfreq, highfreq=None, winlen=winlen,
winstep=winstep, nfilt=nfilt, nfft=nfft, noise=500)
```

A partir du programme Ex3_exemple2.py apporter les modifications nécessaires afin de créer une liste contenant les 7 Modèles de Markov.
Faire quelque chose du genre :

```
models=[]
for word in listeDesMotsDisponibles:
   liste=words.getFeatList(label=word,methode=methode,featStart=featStart,featStop=featStop, dataset='train')
   modele= GaussianHMM(liste=liste, Nstates=Nstates)
   models.append(modele)
```

Compléter le code afin de charger les enregistrements de validation (faire : dataset='val') et calculer pour chacun la (log) probabilité $P(O_T|\lambda_q)$ de ce mot pour chacune des Q =7 chaines de Markov (utiliser la méthode log_prob) puis compléter le code afin de déduire pour chaque enregistrement le mot le plus probable.

Modifier le code en ajoutant une boucle pour renouveler l'expérience précédente un « grand » nombre de fois (20 fois par exemple). A partir des résultats obtenus, calculer la matrice de confusion ainsi que le F1-Score (utiliser dans le package sklearn.metrics les fonctions confusion_matrix et f1_score).

Vous pouvez faire varier les hyperparamètres (nombre d'états, nombre de features utilisées) afin d'en déduire une configuration optimale pour la méthode mfcc.
Y a-t-il des mots où l'algorithme se trompe régulièrement ? Si oui propposer une explication.
Refaire l'étude en utilisant les méthodes 'filter' et 'spectrum'
Conclure quant à la méthode d'extraction de features la plus efficace. A votre avis les résultats obtenus sont-ils généralisables ?

4) Evaluation des performances

On souhaite améliorer la généralisation des performances de notre système de reconnaissance vocal. On ne dispose malheureusement pas d'une base de mots de taille suffisante. On va augmenter la taille de notre base en ajoutant des versions bruitées des enregistrements d'origine :

```
for i in range(10):          # On crée une base de 15*7*10 mots bruités
    if i==0:
        words=Words(rep='audio',name='audio',numcep=numcep,winlen=winlen,winstep=winstep,nfilt=nfilt,nfft=nfft,
        noise=500)
    else:
```

```
words=words+Words(rep='audio',name='audio',numcep=numcep,winlen=winlen,winstep=winstep,nfilt=nfilt,
nfft=nfft ,noise=500)
```

Utiliser uniquement la méthode mfcc

Afin d'évaluer les performances on mettra en œuvre une méthode de type k-fold cross validation dans laquelle on scinde la base en k partitions. Pour cela vous pouvez utiliser la méthode Kfold du package sklearn en prenant k=3.
Refaire l'étude de la partie 3 en prenant comme indicateur de performances la moyenne des 3 F1-score obtenu
Discuter de l'ensemble des résultats obtenus et conclure.
A votre avis l'algorithme développé est-il généralisable à d'autres personnes ? Justifier

Bonus : Si vous souhaitez compléter la base de mot, d'autres fichiers de mots sont disponibles dans les répertoires audio1 et audio2

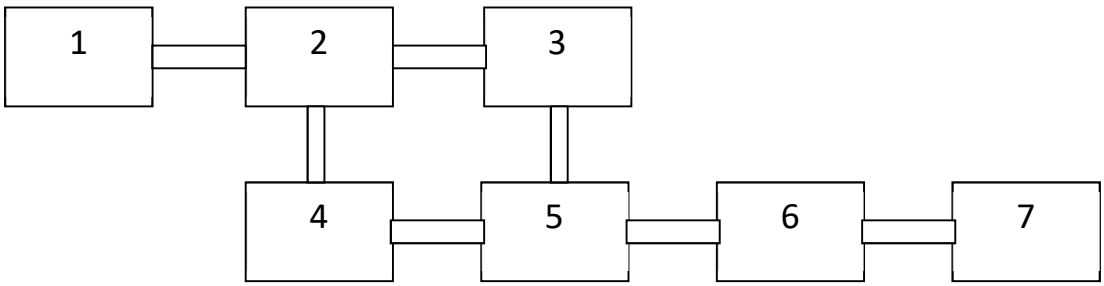
Bonus

Partie 4 : Système de surveillance

Un système de surveillance permet de détecter en temps réel la position géographique d'une personne dans un appartement constitué de 7 pièces. La porte d'entrée se située dans la pièce 1. Des détecteurs de présence donnent la position de la personne à l'intérieur de l'appartement à intervalle de temps régulier de T=10 secondes. Les détecteurs de présence ne sont cependant pas parfaits et parfois ils fournissent une information erronée due aux portes entre les pièces. On estime que 50% du temps le système de détection fournit une information correcte et que durant 50% du temps il localise par erreur la personne dans une pièce adjacente.

1. Comment peut-on modéliser ce problème à l'aide d'une chaine de Markov Cachée ? Faire le graphe. Préciser les variables d'états, les observables.
2. On estime qu'en T secondes la personne à 60% de chance de rester dans la même pièce et que sinon elle se déplace dans une pièce adjacente. Déduire les paramètres du modèle $\lambda(T, B, \pi)$ personne+appartement+détecteur.
3. On souhaite simuler le déplacement d'une personne dans l'appartement sur un intervalle de temps « long ». Générer la série des valeurs observées et la série des états.

4. Vérifier empiriquement sur les séries simulées le pourcentage de fois où les détecteurs se sont trompés. Si on se fie uniquement à l'information fournie par les capteurs, le système est-il fiable pour connaitre la position de la personne ?
5. Estimer théoriquement à partir de ce modèle la probabilité qu'une personne reste au moins 20 secondes dans une pièce, puis au moins 30 secondes. Comparer avec les probabilités empiriques estimées sur la série d'observation simulée.
6. Représenter un histogramme des états du modèle à partir de la série simulée. Estimer empiriquement la probabilité de chacun des états $P(E_1)$, $P(E_2)$, $P(E_3)$, … $P(E_7)$. Commenter. Vous comparerez ces valeurs à celles obtenues à la question suivante



7. On note p_n le vecteur ligne contenant la probabilité de chacun des états à l'instant n :

$$p_n = \left(p_n(E_1),\ p_n(E_2), \ldots,\ p_n(E_N) \right)$$

On note p* le vecteur p_n lorsque n tend vers l'infini. On montre que sous certaines conditions (cf. cours) le vecteur p* tend vers un vecteur constant (distribution stationnaire).

a. Montrer que $p_n = \pi_0 A^n$. Calculer A^n pour n grand (>30) et en déduire une estimation de p*. Comparer avec les résultats obtenus à la question précédente.

b. Montrer que p*(A – I)= 0. En déduire que p* est le vecteur propre gauche associé à la valeur propre λ=1. Calculer les valeurs et vecteurs propres gauche de A (il suffit de calculer les vecteurs propres droite de la transposée de A). En déduire p* (attention faire en sorte que la somme du vecteur propre soit égale à 1) et vérifier la consistance avec les résultats trouvées précédemment.

c. De quel type est la matrice de transition ? p* dépend-il de π_0 ? Dans quels cas p* aurait-il pu dépendre de π_0 ?

8. **Conclusion** : A partir de la séquence d'observation générée précédemment, estimer la séquence optimale des états à partir de la séquence d'observation.
Estimer empiriquement sur cette séquence d'état le pourcentage de temps où l'état estimé ne correspond pas à l'état réel. Comparez ce résultat à celui obtenu à la question 4. **L'ajout d'une modélisation a-t-elle amélioré les performances du système ? Discuter des avantages et des inconvénients**.