Travaux pratiques

Chaines de Markov – chaines de Markov cachées

L'ensemble des fichiers nécessaires à la réalisation de ce TP se trouve dans le fichier « Archive TP TRIED 2020-2021.zip »

Ce TP nécessite l'installation sur votre machine du package python « pomegranate » :

Si vous utilisez anaconda sous windows: il faut au préalable ouvrir une fenêtre de commande anaconda en mode administrateur (menu démarrer-> anaconda -> anaconda prompt (faire un clic droit, puis choisir « executer en tant qu'administrateur »).
Dans la fenêtre de commande taper la commande suivante : conda install pomegranate

Si vous utilisez un autre environnement taper dans une fenêtre de commande : pip install pomegranate

`Voir doc :` https://pomegranate.readthedocs.io/en/latest/HiddenMarkovModel.html

Préambule :

Les parties 1 à 5 sont relativement guidées avec des questions précises. Dans la partie 6 sont volontairement plus vagues, à vous de prendre plus d'initiatives

La partie 6 est longue, elle doit être impérativement commencé au plus tard en milieu de matinée de la seconde séance. Ne pas faire la partie 5 le cas échéant.

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 convient de dire que le système est dans l'état « Sec » s'il ne pleut pas (état E0) et dans l'état « Pluie » s'il pleut (état E1). 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.

1. Déduire le modèle de Markow associé à ce problème **: E, A et π_0**
2. Représenter le graphe
3. Préliminaires : A l'aide de données pluviométriques on a ajusté les paramètres suivants α= 0.65, β= 0.02 et γ=0.9. On souhaite générer une séquence d'état de 100000 valeurs. Pour cela vous utiliserez la classe `HiddenMarkovModel`du package « pomegranate». La génération de la séquence s'effectue ensuite avec la méthode

sample(). Bien que cette classe soit dédiée aux chaines de Markov cachées, comment doit-on définir la matrice d'émission afin de générer une chaine de Markov ? Dans ce cas expliquer pourquoi la séquences des états est identique à la séquence des observables ? Le fichier partie1.py du dossier « Parties_1_et_2 » fournit le code (incomplet) permettant de générer la séquence d'observable et la séquence des états (qui sont donc identiques). Générer une séquence de 100000 individus. Cette séquence sera utilisée dans les questions suivantes.

4. Montrer que la probabilité qu'il pleuve vaut : $P(E_1) = \beta / (1+\beta - \alpha)$. Estimer empiriquement cette probabilité sur la séquence générée précédemment et vérifier avec la valeur théorique.
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 et les retourne ainsi que leur densité de probabilités empiriques et les classes associés :

```
from duree import duree
dureeSec,dureePluie,pdfSec,pdfPluie,binsSec,binsPluie =duree(StatesSequence)
```

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 de paramètre p et q :

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

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 1 indique l'absence de pluie tandis qu'une valeur 2 indique qu'il a plu.

La syntaxe python pour charger ce fichier est la suivante :

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

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', Etat nuageux 'Cloudy' et Etat très nuageux 'Very Cloudy'. 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 terme de pourcentage de pluie / sécheresse ? En terme de distribution de durée de pluie / sécheresse ?

3. On se propose d'améliorer les paramètres de 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 :

```
tmp= []              # On crée une liste composé de string 'sun' et 'rain' à la place des 1 et 2
for o in ObsMesure:
 if o ==1:
    tmp.append('sun')
 else:
    tmp.append('rain')
model.fit([tmp],algorithm='baum-welch',max_iterations=200)
```

Comparer les nouvelles matrices de transition (model.dense_transition_matrix()) et d'émission (ClearSky.distribution) 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 ?

4. Générer une nouvelle séquence de longueur suffisante à l'aide du nouveau modèle estimé et comparer les distributions 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 : froid ou chaud ?

On souhaite avoir une idée de la température moyenne annuelle durant une période d'étude particulière qui s'est déroulée il y a 5000 ans. Malheureusement le thermomètre n'existait pas !
De façon plus précise, on souhaite savoir qu'elles sont parmi les années de la période considérée celles qui ont été chaudes et celles qui ont été froides. On notera par l'état 'chaud' le fait d'être une année chaude et par l'état 'froid' le fait d'être une année froide.
La croissance des arbres implique que chaque année une nouvelle couche d'écorce se forme. L'épaisseur de cette dernière dépend en partie de la température. On se propose donc d'estimer la température via la mesure de

l'épaisseur des anneaux de troncs d'arbres retrouvés au fond de certains marécages et ayant poussés il y a 5000 ans. On distinguera 3 épaisseurs différentes 'fine', 'moyenne' et 'epaisse'.
Des études ont permis de montrer que la probabilité qu'une année chaude soit suivie d'une autre année chaude est de 0.6 tandis que la probabilité qu'une année froide soit suivie par une autre année froide est de 0.7. De plus, d'autres études ont montré que pour une année chaude les probabilités que les troncs aient des épaisseurs fines, moyennes ou épaisses sont respectivement 0.1, 0.4, 0.5 tandis que pour une année froide elles sont respectivement de 0.6, 0.3, 0.1.

1. Mettre ce problème sous la forme d'une chaine de Markov cachée et déterminer les matrices de transition et d'émission
2. Les paléontologues qui ont découvert de tels arbres vieux de 5000 ans ont mesuré les épaisseurs d'anneaux durant 10 années successives : O_{10} = fine, moyenne, fine, épaisse, moyenne, fine, moyenne, moyenne, moyenne, moyenne.

a. Qu'elle est la probabilité de la séquence O_{10} sachant le modèle décrit ci-dessus (méthode log_probability) ? Commenter.
b. Calculer la séquence d'état la plus probable de température et la comparer avec la séquence qu'on pourrait déduire de façon intuitive sans utiliser le modèle Markov. Discuter

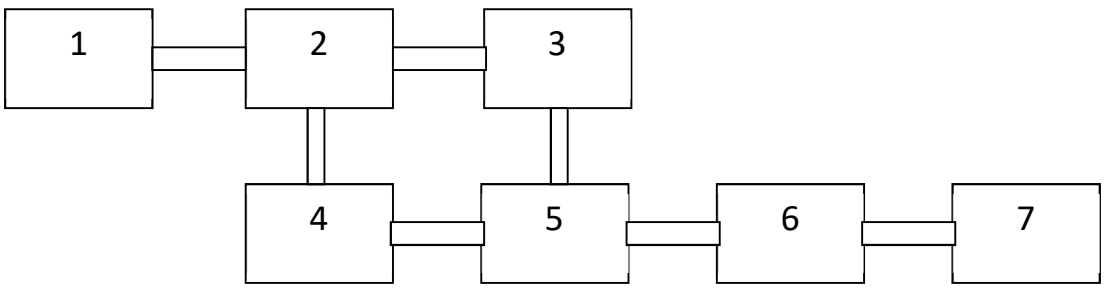
Partie 4 : Système de surveillance

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 » (10000 mesures par exemple). Générer la série des valeurs observées et la série des états dans deux tableaux distincts :

```
observables,etats = model.sample(Nsamples, path=True)
```

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



Plan de l'appartement

7. On note p_n le vecteur ligne contenant la probabilité de chacun des états à l'instant n :
$$p_n = \left(p_n(E_1),\ p_n(E_2), \dots,\ p_n(E_N) \right)$$
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**.

Partie 5 : pile ou face ?

On suppose un jeu de pile (P) ou face (F) composé de 3 pièces de monnaies. Les pièces sont différentes. Les probabilités d'obtenir le côté face sont les suivantes :
Pièce 1 : P(f) = 0.5
Pièce 2 : P(f)= 0.75
Pièce 3 : P(f) = 0.25
Le jeu commence toujours par le lancer de la pièce 1. Le joueur lance ensuite, soit la même pièce, soit une autre pièce selon les probabilités de transition suivantes :
Pièce n°1 : P(1,1) = 0.5, P(1,2) = 0.4
Pièce n°2 : P(2,1) = 0.3, P(2,2) = 0.4
Pièce n°3 : P(3,1) = 0.1, P(3,2) = 0.2
On souhaite modéliser le jeu à l'aide d'une chaine de Markov cachée.

1. Quel sont les états du modèle, en déduire la dimension de la matrice de transition A. Quels sont les valeurs observables ? En déduire la dimension de la matrice d'émission B.
2. Déduire les matrices de transition (A) et d'émission (B).
3. Représenter le graphe de cette chaine de Markov.
   Un joueur obtient la séquence suivante : « FPF ». Représenter le treillis associé.
4. On souhaite calculer $P(O_T/\lambda)$. Estimer cette probabilité à la main en envisageant tous les chemins possibles.
5. Recalculer $P(O_T/\lambda)$ en utilisant à la main la méthode forward. Evaluer la quantité de calcul réalisé dans les 2 cas.
6. Vérifier votre calcul de $P(O_T/\lambda)$ à l'aide de la méthode forward() . Comparer avec le résultat obtenu précédemment.
7. Que vaut $P(O_T/\lambda)$ pour la séquence « FFF » puis pour la séquence « PPP » ?. Pourquoi à votre avis $P(O_T/\lambda)$ est supérieure pour la première séquence ?
8. On considère à nouveau la séquence « FPF »
   a. Calculer $\delta_n(j)$ et montrez que $\delta_n(j) = \begin{matrix} & n & & \\ j & \begin{matrix} 0.5 \\ 0 \\ 0 \end{matrix} & \begin{matrix} 0.125 \\ 0.05 \\ 0.0375 \end{matrix} & \begin{matrix} 0.03125 \\ 0.0375 \\ 0.0065 \end{matrix} \end{matrix}$.
   Vous pouvez faire le calcul à la main ou écrire un petit programme python

   b. Appliquer l'algorithme de Viterbi et estimer « à la main » la séquence optimale des états.
9. En utilisant la méthode « predict » vérifier votre résultat précédent.
10. On suppose maintenant que les paramètres du modèle sont inconnus et que l'on cherche à les déterminer. Pour cela on supposera que l'on dispose d'une liste de N séquences chacune de longueur T =100 (générées au préalable avec sample). On estime alors les paramètres sur ces séquences à l'aide de l'algorithme de Baum-Welch (méthode fit()).
    a. Estimer les nouveaux paramètres lorsque une seule séquence est disponible (N=1). Visualiser les matrices de transition et d'émission et commenter (attention les états ne sont pas forcément dans l'ordre de départ).
    b. Refaire la même simulation avec N=10. Conclure quant à l'estimation des paramètres du modèle.

Partie 6 : reconnaissance vocale de mots isolés

Une entreprise souhaite développer un répondeur « intelligent » capable de « comprendre » des mots (en nombre limité) prononcés par des utilisateurs. 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.
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), le répondeur 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 peut être 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+1 = 129 features / frame.
2. Méthode Filter : à partir du spectre obtenu par la méthode Spectrum précédente on calcule les énergies obtenues par 26 filtres de Mel (cf. cours). 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 discrète des énergies obtenues précédemment par la méthode Filter avec les filtres de Mel (cf. cours). On obtient donc un vecteur de 12 features / frame

On dispose suivant la méthode d'extraction de caractéristiques employée de vecteurs notés o_i de dimension 256, 26 ou 12 pour chaque frame. On définit la séquence d'observable $O_T = o_1 o_2 \dots o_T$ comme étant une succession de vecteurs o_i contenant chacun les features d'une frame.
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 une partie de ces enregistrements pour apprendre les Q chaines de Markov (une chaine par mot). Les autres enregistrement serviront aux tests

C. Phase de reconnaissance d'un mot inconnu

On suppose que l'on dispose 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 $\mathbf{O_T} = \mathbf{o_1 o_2 \dots o_T}$

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

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

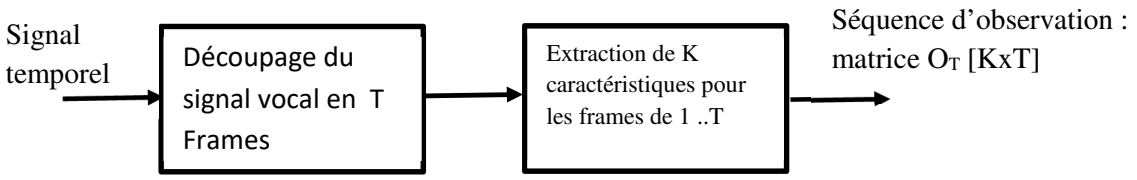
$$q_{mot} = \underset{q \in [1\,..\,Q]}{\operatorname{argmax}} P(\mathrm{O_T}|\lambda_q) P(q)$$

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

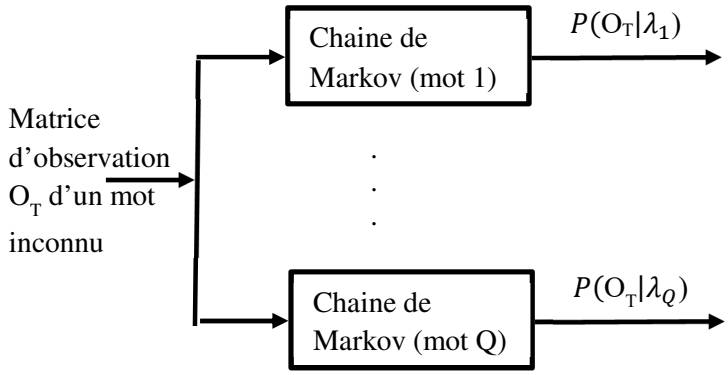
$$q_{mot} = \underset{q \in [1\,..\,Q]}{\operatorname{argmax}} P(\mathrm{O_T}|\lambda_q)$$

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

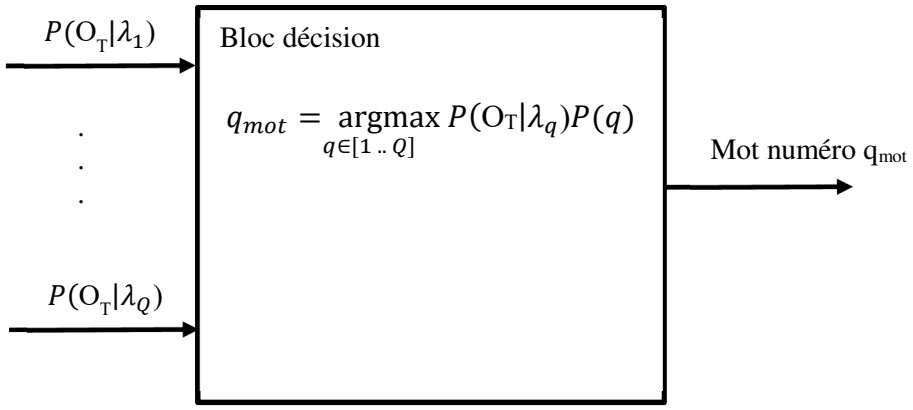
- **Extraction des caractéristiques :**



- **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 :

$$s_n = (F_1, F_1, \dots., F_K)^T$$

La probabilité $b(s_n)$ d'émettre le symbole s_n sera 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 6' 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 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. Vous pouvez écouter les enregistrements des différents mots en double-cliquant sur le fichier .wav correspondant.

La figure 1 du programme exemple1.py est composée de 4 subplot. Le premier permet de visualiser le signal temporel du premier enregistrement du mot 'apple' (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 figure 2 montre les histogrammes des 12 premières Features de 'apple' pour la méthode 'spectrum. Il est possible d'afficher les histogrammes des autres features en modifiant les variables featStart et featStop. Les figures 3 et 4 sont identiques mais pour les méthodes filter et mfcc.
La figure 5 montre les deux premières features (Fx=0 et Fy=1) de 'apple' pour la méthode 'spectrum'. 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 Fx et Fy pour sélectionner d'autres composantes
Les figures 5 à 7 sont identiques mais pour les méthodes filter et mfcc.

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. Pour cela, lancer le programme exemple1.py et comprendre les différents affichages obtenus. Ce programme contient 3 méthodes permettant d'afficher les features sous diverses formes :

- plotOneWord : affiche les caractéristiques d'un enregistrement pour un mot donné
- histFeatures : affiche les histogrammes de chaque composante des vecteurs symboles pour un mot et une méthode d'extraction donnés
- plotFeatureXY : affiche les composantes X et Y des features associés à un mot et une méthode d'extraction sous la forme d'un nuage de points.

Vous pouvez modifier les variables suivantes afin de réaliser l'étude :
- label : contient le nom d'un des 7 mots disponibles ('apple', 'banana', 'kiwi', 'lime', 'orange', 'peach', 'pineapple').
- Fx et Fy : contient les numéros des 2 composantes de feature à afficher par la méthode 'plotFeatureXY'.
- featStart et featStop : utilisés par la méthode 'histFeatures' qui affiche les histogrammes des features de la composante featStart à featStop

Indications : Vous pouvez par exemple dans un premier temps à l'aide de la méthode « plotOneWord » comparer les spectrogrammes de différents mots. Idem avec la méthode filter (« Cepstogrammes ») et la méthode mfcc. Choisir quelques mots avec des sons graves, d'autres avec des sons aigus, des mots courts, des mots longs, etc…
Vous pouvez également comparer les histogrammes à l'aide de la méthode histFeatures pour différents mots et méthodes d'extraction. Comparer aussi les nuages d'individus pour différents couples de composantes pour

différents mots et méthodes à l'aide de la méthode plotFeatureXY (attention, le nombre de feature est tres différent selon les méthodes (129, 26, 12).

Au final, émettre des hypothèses sur la ou les méthodes qui vous semblerait la ou les meilleures pour différencier les mots les uns des autres.

2) Apprentissage d'une Chaine de Markov cachée:

Pour apprendre un modèle de HMM vous utiliserez la classe GaussianHMM présente dans le fichier TpHmmUtilit.py. Cette classe permet de créer et d'entrainer une chaine de Markov avec la méthode de Baum-Welch à partir de séquences de features stockées dans une liste fournie en argument. Le fichier exemple2.py 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. Dans cet exemple, seules les 5 premières features (featStart =0, featStop = 4) sont utilisées.
- 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.
- 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 les sorties sur la console et la figure.
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) :
Vous pouvez par exemple faire varier le nombre de features apprises et comparer le 'mappage' des individus par les ellipses pour différentes valeurs de Fx et Fy (penser que la méthode spectrum a 129 features). 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) qui semblent mieux convenir ? Un nombre d'état optimal ?

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. Pour chaque mot on utilisera les 15 enregistrements disponibles pour l'apprentissage. A partir du programme exemple2.py et pour la méthode d'extraction 'mfcc' apporter les modifications nécessaires afin de créer une liste de 7 Modèles de Markov.
Faire quelque chose du genre :

```
for label in words.getLabels() :
   liste=words.getFeatList(label=label, ….
   Models.append(GaussianHMM(liste=liste, Nstates=Nstates)
…
```

- En tirant aléatoirement un mot parmi les 105 mots disponibles calculer la (log) probabilité $P(O_T|\lambda_q)$ de ce mot pour chacune des Q chaines de Markov (q=1 … Q). Déduire le mot le plus probable.
- Renouveler l'expérience un grand nombre de fois et tirer une première conclusion.
- Refaire les 2 questions précédentes en utilisant les méthodes 'filter' et 'spectrum'

4) Evaluation des performances

On souhaite évaluer les 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(5):              # On crée une base de 105*5 mots bruités
    if i==0:
        words=Words(rep='audio',name='audio',numcep=numcep,winlen=winlen,winstep=winstep,nfilt=nfilt,nfft=nfft,
        noise=True)
    else:
        words=words+Words(rep='audio',name='audio',numcep=numcep,winlen=winlen,winstep=winstep,nfilt=nfilt,
        nfft=nfft ,noise=True)
```

Afin de ne pas pénaliser le temps d'apprentissage on utilisera dans un premier temps les 3 premières features et la méthode 'mfcc':

```
featStart=0
featStop=2
```

On souhaite mettre en œuvre une méthode de type k-fold cross validation dans laquelle on scinde la base en k partitions. On choisit un ensemble de k-1 partitions pour l'apprentissage et 1 partition pour l'ensemble de test. On recommence la procédure apprentissage/test avec un ensemble d'apprentissage de k-1 partitions différentes et ainsi de suite.

Pour cela vous pouvez utiliser la méthode Kfold du package sklearn en prenant k=3. La méthode splitListe permet de créer 2 sous listes à partir d'une liste:

```
kf = KFold(n_splits=3,shuffle=True)     # On coupe en 3 subsets
  for Itrain,Itest in kf.split(indices):
….
            liste=words.getFeatList(label=label,methode=methode,featStart=featStart,featStop=featStop)
            listeTrain,listeTest= liste.splitListe(Itrain,Itest)
….
```

L'objectif est de calculer la matrice de confusion. Pour cela il faut apprendre les 7 chaines de Markov sur les données d'apprentissage (listeTrain). Dans un second temps il faut pour chaque enregistrement de la base de test calculer sa log probabilité pour les 7 chaines de Markov et en déduire le mot correspondant.

- Calculer la matrice de confusion, puis l'afficher.
- Refaire la même chose avec les 2 autres méthodes.
- Augmenter le nombre de features utilisé et refaire l'étude.
- Conclure
- Discuter de l'ensemble des résultats obtenus et conclure. A votre avis l'algorithme développé est-il généralisable à d'autres personnes ? Si oui dans quelles conditions ?

Remarque : D'autres fichiers de mots sont disponibles si nécessaire dans les répertoires audio1 et audio2