CR 09 Hidden Markov models for time series analysis
Lab Session – **Bayesian decision and EM algorithm in HMC**
S. Derrode

L'idée de ce TP est de programmer certains des algorithmes vus en cours sur les chaines de Markov cachées (HMC). L'objectif est d'appliquer la décision bayésienne (par le biais des critères MPM et MAP), après avoir appris automatiquement les paramètres du modèle HMC par le biais du principe EM (« *Expectation-Maximization* »).
En téléchargeant le fichier zip du TP à l'URL http://perso.ec-lyon.fr/derrode.stephane/Teaching.php, vous disposerez de 4 fichiers python :

- ***Func.py :*** contient tous les algorithmes pour la décision bayésienne et pour l'apprentissage EM. Certaines fonctions sont vides : à vous de les programmer selon les indications données ci-dessous.
- ***SimulHMC.py :*** permet de simuler *N* observations et *N* états d'une chaine de Markov cachée dont les paramètres sont fixés au début du programme principal. Les trajectoires simulées sont stockées dans le sous-répertoire '*sources*'.
- ***SupervisedHMCRestoration.py :*** permet la restauration optimale au sens Bayésien lorsque les paramètres de la HMC sont connus.
- ***UnsupervisedHMCRestoration.py :*** même objectif que ci-dessous, mais avec un apprentissage automatique des paramètres selon le principe EM appliqué au cas HMC.

Simulation d'une HMC – *SimulHMC.py*

- Vérifiez que vous comprenez bien les algorithmes servant à simuler les états de la chaine de Markov et des observations correspondantes.
- Exécutez le programme *SimulHMC.py*, en sélectionnant la matrice de transition *t4* (chaines de Markov à 2 états), et les gaussiennes de moyennes 100 et 110, et d'écart-types 6 et 3. Dans le répertoire '*results*', vous pouvez observer des extraits graphiques des 2 signaux générés.
- Ouvrez le fichier '*sources/XY_T4.out*' avec un éditeur de texte et observez le format des données enregistrées. Ce sont ces données qui seront utilisées en entrée des 2 prochains programmes.

Restauration supervisée – SupervisedHMCRestoration.py

- Vérifiez que les paramètres dans le programme principal sont bien identiques à ceux utilisés pour la simulation (on est ici dans le cas de la restauration à paramètres connus). Et vérifiez que le programme lit bien le fichier que vous avez simulé par le programme précédent (i.e. '*XY_t4.out*').
- Programmez la méthode '*getBeta(…)*' du fichier *func.py*, grâce à l'algorithme *backward* présenté en cours.
- Programmez la méthode '*getMAPClassif(…)*' du fichier *func.py* qui implémente l'algorithme de Viterbi.
- Lancez le programme et vérifiez que vous obtenez un résultat similaire au suivant :

```
Confusion matrix for MPM:
 [[476.   7.]
 [ 11. 506.]]
Global error rate for MPM: 0.018
By class error rate for MPM: [0.01449275 0.0212766]

Confusion matrix for MAP:
 [[476.   7.]
 [  8. 509.]]
Global error rate for MAP: 0.015
By class error rate for MAP: [0.01449275 0.01547389]
```

Restauration non supervisée – UnsupervisedHMCRestoration.py

Il s'agit d'apprendre automatiquement les paramètres du modèle à partir de l'algorithme EM décrit en cours, avant d'appliquer la décision Bayésienne critère MPM.

- Analysez le squelette de l'algorithme, qui est très similaire à ce qui a été fait pour le modèle de mélange.
- Programmez la méthode '*UpdateParameters(…)*', appelée par la fonction '*EM_Iter(…)*' du fichier *func.py*.
- Lancez le programme. Après 20 itérations, vous devriez observer un résultat comparable à

```
--->iteration= 0
Initial estimations:
  Confusion matrix for MPM =
 [[200. 283.]
 [  0. 517.]]
  Global Error rate for MPM:  0.283
  Class Error rate for MPM:  [0.58592133 0.        ]
--->iteration= 1
--->iteration= 2
--->iteration= 3
…
--->iteration= 18
--->iteration= 19
Final estimations:
  Confusion matrix for MPM =
 [[473.  10.]
 [  6. 511.]]
  Global Error rate for MPM:  0.016
  Class Error rate for MPM:  [0.02070393 0.01160542]
```

- Voici les courbes de convergence obtenues dans ce cas (sauvegardées dans le répertoire '*results*') :