UFR des sciences
Master recherche *Génie Informatique*

Séance 2

Modélisation de séquences par Modèles de Markov Cachés Discrets

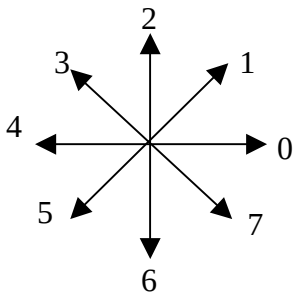
---

Au cours de cette séance de Travaux Pratiques vous allez mettre en œuvre des modèles de Markov cachés discrets pour modéliser puis reconnaître des séquences correspondant à des tracés de chiffres manuscrits.

1. Acquisition des données et codage

Les données seront acquises selon la méthode utilisée lors du TP N° 1. Toutefois les données doivent faire l'objet d'un pré-traitement pour permettre l'utilisation de modèles discrets.

Rappel : les modèles discrets travaillent sur des séquences de symboles d'un alphabet défini préalablement. L'alphabet utilisé sera constitué des 8 directions de Freeman qui seront codées selon la convention suivante :



Programmer sous Matlab une fonction de pré-traitements permettant de transformer une séquences de points représentés par leurs coordonnées en une séquence de directions de Freeman. Le codage se fera en attribuant la direction la plus proche.

2. Mise en œuvre des modèles de Markov Cachés Discrets

Vous allez utiliser la bibliothèque de traitements réalisée par Kevin Murphy (1998) et disponible sur le site suivant http://www.ai.mit.edu/~murphyk/Software/hmm.html

Pour cela vous installerez la bibliothèque sous le répertoire C:\Matlab\HMM
Vous installerez les bibliothèques KPMStats, KPMtools et netlab

a- prise en main de la bibliothèque HMM
Utilisez le programme donné en exemple pour mettre en œuvre l'apprentissage d'un modèle, de Markov Discret à l'aide de l'algorithme EM.

b- Apprentissage de séquences de chiffres
Définir et mettre en œuvre l'entraînement d'un modèle de Markov Caché discret pour chaque classe de chiffre

Tester les performances des modèles ainsi appris pour effectuer la reconnaissance de chiffres manuscrits. Pour cela on choisira de prendre une décision par maximum a posteriori.

$$P(Ci/O)=\frac{P(O/Ci)}{P(O)}P(Ci) \quad \propto \quad P(I/Ci)$$

Ce qui revient à une décision par maximum de vraisemblance lorsque les classes sont équi-probables car la probabilité de l'observation P(O) est la même quelle que soit la classe considérée.

La vraisemblance de chaque modèle sera d'abord calculée à l'aide de l'algorithme forward grâce à la fonction **dhmm_logprob( ).**

Comparer avec une décision prise en calculant la vraisemblance du meilleur chemin en utilisant l'algorithme de Viterbi, grâce à la fonction **viterbi_path( )**

Evaluer l'influence du nombre d'état des modèles sur les performances en reconnaissance.

Annexe 1 : Exemple de manipulation des HMM discrets

```
%  demo_HMM.m
%  Exemple d'utilisation de l'algorithme EM pour l'apprentissage de
%  modèles de Markov cachés discrets sur des séquences de longueurs
%  différentes
%
%  SUJET de TP du MASTER Système d'Acquisition et de Traitement de l'Information
%  Université de ROUEN
%
%  Le TP met en oeuvre la boite à outils développée par Kevin Murphy
%           http://www.ai.mit.edu/~murphyk/Software/hmm.html
%
%  pour cela il est nécessaire d'installer les boîtes à outils KPMStats, KPMtools et netlab
%
%
%  Année 2005                              Thierry.Paquet@univ-rouen.fr
%

O = 3; % nombre d'observations
Q = 2; %nombre d'états

% Génération aléatoire des paramètres du "vrai" modèle
prior0 = normalise(rand(Q,1));
transmat0 = mk_stochastic(rand(Q,Q));
obsmat0 = mk_stochastic(rand(Q,O));

% génération de 10 séquences de longueur 5 observations
T = 5;       % les séquences ont toutes la même longueur
nex1 = 10;   % nombre d'exemples
data1 = dhmm_sample(prior0, transmat0, obsmat0, T, nex);

% generation de 5 séquences de longueur 4 observations
T = 4;
nex2 = 5;
data2 = dhmm_sample(prior0, transmat0, obsmat0, T2, nex);

% fusion des séquences générées dans un tableau de cellules
% de longueur variable
% on transforme les tableaux en tableaux de cellules
% chaque cellule est une séquence
DATA1 = NUM2CELL(data1,1);
DATA2 = NUM2CELL(data2,1);
% on fusionne les tableaux de cellules de longueur variable
data = {DATA1{[1:nex1]} DATA2{[1:nex2]}}
% le nombre d'éléments du tableau des séquences
numex = length(data)

% Génération aléatoire d'un premier modèle
prior1 = normalise(rand(Q,1));
transmat1 = mk_stochastic(rand(Q,Q));
obsmat1 = mk_stochastic(rand(Q,O));

% estimation du meilleur modèle par l'algorithme EM en 5 itérations
[LL, prior2, transmat2, obsmat2] = dhmm_em(data, prior1, transmat1, obsmat1, 'max_iter', 5);

% affichage de la vraisemblance après 5 itérations
LL

% calcule de la vraisemblance des données pour le modèle obtenu
loglik = dhmm_logprob(data, prior2, transmat2, obsmat2)
```

Annexe 2

Principales fonctions de la boîte à outils nécessaires

function [LL, prior, transmat, obsmat] = dhmm_em(data, prior, transmat, obsmat, varargin)

```
% LEARN_DHMM Find the ML/MAP parameters of an HMM with discrete outputs using EM.
% [ll_trace, prior, transmat, obsmat] = learn_dhmm(data, prior0, transmat0, obsmat0, ...)
%
% Notation: Q(t) = hidden state, Y(t) = observation
%
% INPUTS:
% data{ex} or data(ex,:) if all sequences have the same length
% prior(i)
% transmat(i,j)
% obsmat(i,o)
%
% Optional parameters may be passed as 'param_name', param_value pairs.
% Parameter names are shown below; default values in [] - if none, argument is mandatory.
%
% 'max_iter' - max number of EM iterations [10]
% 'thresh' - convergence threshold [1e-4]
% 'verbose' - if 1, print out loglik at every iteration [1]
% 'obs_prior_weight' - weight to apply to uniform dirichlet prior on observation matrix [0]
%
% To clamp some of the parameters, so learning does not change them:
% 'adj_prior' - if 0, do not change prior [1]
% 'adj_trans' - if 0, do not change transmat [1]
% 'adj_obs' - if 0, do not change obsmat [1]
```

function path = viterbi_path(prior, transmat, obslik)

```
% VITERBI Find the most-probable (Viterbi) path through the HMM state trellis.
% path = viterbi(prior, transmat, obslik)
%
% Inputs:
% prior(i) = Pr(Q(1) = i)
% transmat(i,j) = Pr(Q(t+1)=j | Q(t)=i)
% obslik(i,t) = Pr(y(t) | Q(t)=i)
%
% Outputs:
% path(t) = q(t), where q1 ... qT is the argmax of the above expression.
```

function [loglik, errors] = dhmm_logprob(data, prior, transmat, obsmat)

```
% LOG_LIK_DHMM Compute the log-likelihood of a dataset using a discrete HMM
% [loglik, errors] = log_lik_dhmm(data, prior, transmat, obsmat)
%
% data{m} or data(m,:) is the m'th sequence
% errors  is a list of the cases which received a loglik of -infinity
```