SMAC with HMM
Toolbox Manual

This document is intended as a basic user-guide and implementation overview of the toolbox “**S**canpath **Mo**deling **A**nd **C**lassification with HMM”. This is a beta-version. If you have any problem or question, please contact one of the authors: Antoine Coutrot <acoutrot@gmail.com>; Janet Hsiao <jhsiao@hku.hk>; Antoni Chan <abchan@cityu.edu.hk>. This toolbox is subject to the ”GNU General Public License 3.0”.
If you use this toolbox, please cite
[1] Antoine Coutrot, Janet H. Hsiao, Antoni B. Chan, *Scanpath modeling and classification with Hidden Markov Models,* Behavior Research Methods, 2017.
[2] Tim Chuk , Antoni B. Chan, Janet H. Hsiao. *Understanding eye movements in face recognition using hidden Markov models*, Journal of Vision 14(11):1–14, 2014.

CONTENT

File name	**Type**	**Description**
Stimuli	folder	Subset of the stimuli from Coutrot’s and Koehler’s dataset
vbhmm	folder	Core functions for variational HMM computation
PRML_functions	folder	RVM and AdaBoost functions from Bishop’s textbook
EyeData_Coutrot	.mat struct	Eye positions from Coutrot’s dataset
EyeData_Koehler	.mat struct	Eye positions from Koehler’s dataset
Plot_HMM_Coutrot	.m script	Plot HMM from scanpaths recorded on Coutrot’s dataset
Plot_HMM_Koehler	.m script	Plot HMM from scanpaths recorded on Koehler’s dataset
Compute_HMM_descriptors_Coutrot	.m script	Compute HMM parameters from EyeData_Coutrot.mat and save them in example_HMM_descriptor_Coutrot.mat
Compute_HMM_descriptors_Koehler	.m script	Compute HMM parameters from EyeData_Koehler.mat and save them in example_HMM_descriptor_Koehler.mat
HMM_descriptor_Coutrot	.mat struct	HMM gaze features learned from EyeData_Coutrot.mat
HMM_descriptor_Koehler	.mat struct	HMM gaze features learned from EyeData_Koehler.mat
Classif_from_HMM_descriptors_Coutrot	.m script	Classification based on HMM_descriptor_Coutrot.mat
Classif_from_HMM_descriptors_Koehler	.m script	Classification based on HMM_descriptor_Koehler.mat
classifier	.m function	Leave-one-out classification on selected variables with selected method
pad_with_ghost_states	.m function	Pad HMM so they all have the same number of states
extract_scanpath	.m function	Extract eye positions of a given observer on a given stimulus in a given condition

The vbhmm folder contains a subset of the EMHMM toolbox available at
http://visal.cs.cityu.edu.hk/research/emhmm/

In this manual, the toolbox is explained with Coutrot's dataset, but the same rules apply to any data.
Let's begin with **Plot_Coutrot_HMM**. Plot_Coutrot_HMM computes and plots HMMs learned from EyeData_Coutrot. There are two possibilites:

1- Compute one HMM per scanpath.

In this example, we learn HMMs from 9 observers (one each) watching stimulus 'istim' in 'with_os' auditory condition. K represents the possible state numbers (here K=1:3). For each HMM, the optimal state number is determined by the variational approach. The states are then sorted from left to right. We constrained the covariance matrices to be all identical circular distributions (vbopt.do_constrain_var = 1). The size of the circles can be tuned with vbopt.do_constrain_var_size (see initialize_HMM_computation.m in vbhmm folder).

extract from Plot_Coutrot_HMM.m

```
%% Learn 1 HMM per scanpath
for isub=1:9
    % Extract scanpath of observer isub
    scanpath = extract_scanpath(example_EyeData_Coutrot,'with_os',isub,istim,K);
    if ~isempty(scanpath{1,1})
        % Compute corresponding HMM
        vbopt=initialize_HMM_computation(frame);
        vbopt.do_constrain_var=1; %Tie covariance matrices (identical circle distributions)
        [hmm,~] = vbhmm_learn(scanpath, K, vbopt);%Learn 1 HMM from each scanpath

        % sort states from left to right
        hmm = sort_hmm_state(hmm);

        if isplotHMM
            subplot(3,3,isub)
            plot_hmm_state(hmm,scanpath,frame)
            s=sprintf('observer %u',isub);
            title(s)
        end
    end
end
```

White dots: eye positions downsampled at 25 Hz. With Koehler's data, it would have been fixation points.
Colored circles: state distributions sorted from left to right. The HMM of most observers have 3 states, but some have 2 (e.g. observer 1).

hmm is a structure with the following fields. K is the number of states.

hmm.prior: state priors (Kx1).
hmm.trans: transition matrix (KxK).
hmm.pdf: means and covariances of emissions (Kx2).
hmm.LL: model log-likelihood.
hmm.gamma: posterior probabilities for each eye position to belong to each state (KxN, with N the number of samples).
hmm.M: number of eye positions transitioning from one state to the other (KxK).
hmm.model_LL: log-likelihood of models with $K=1:K^{max}$ (here $K^{max}=3$).
hmm.model_bestK: optimal number of states.

2- Compute one HMM for a group of scanpaths.

In this example, we learn one HMM from a group of 19 observers (one each) watching stimulus ‘istim’ in ‘with_os’ auditory condition.

extract from Plot_Coutrot_HMM.m

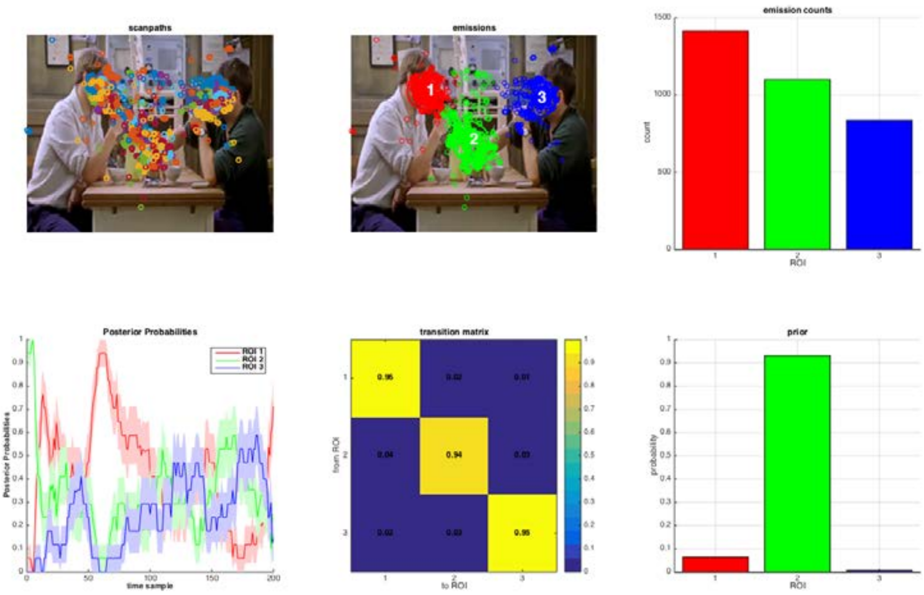
```
%% Learn 1 HMM from a group of scanpaths
%Select observers
observers=1:19;
% Extract their scanpath
scanpath = extract_scanpath(example_EyeData_Coutrot,'with_os',observers,istim,K);
%remove empty cell array contents
scanpath=scanpath(~cellfun('isempty',scanpath));

if ~isempty(scanpath{1,1})
   % Compute corresponding HMM
   vbopt=initialize_HMM_computation(frame);
   vbopt.do_constrain_var=1; %Tie covariance matrices (identical circle distributions)

   [hmm,~] = vbhmm_learn(scanpath, K, vbopt); %Learn HMM from all scanpaths

   % sort states from left to right
   hmm = sort_hmm_state(hmm);

   if isplotHMM
      vbhmm_plot(hmm,scanpath,frame_name)
   end
end
```



3-state HMM modeling 19 scanpaths on an image from Coutrot’s dataset. From top to bottom and left to right. **Scanpaths**: eye positions of the same color belong to the same observer. **Emissions**: 3 states have been identified. **Emission counts**: number of eye positions associated with each state. **Posterior probabilities**: temporal evolution of the probability of being in each state. Shaded error bars represent standard error from the mean. **Transition matrix**: probability of going from state (or Region Of Interest) i to j, with $(i,j)\in[1..3]^2$. **Priors**: initial state of the model.

Scanpath classification is a two-step process.

First, one HMM is computed for each scanpath (1st case of Plot_Coutrot_HMM described above). If needed, HMMs are padded with 'ghost states' (null priors and transition matrix coefficients) so they all have the same number of states. This is taken care of by the script '**Compute_HMM_descriptors_Coutrot.m**'. HMM parameters are saved in the structure '**HMM_descriptor_ Coutrot.mat**'.

Compute_HMM_descriptors_Coutrot.m

```
            end
            %% Auditory Condition 2: Without Orginal Soundtrack

            % Extract current scanpath
            scanpath_wos = extract_scanpath(example_EyeData_Coutrot,'without_os',isub,istim,K);
            if ~isempty(scanpath_wos{1,1})
                % Compute corresponding HMM
                vbopt=initialize_HMM_computation(im);
                vbopt.do_constrain_var=1;
                [hmm_wos,~] = vbhmm_learn(scanpath_wos, K, vbopt);

                % sort states from left to right
                hmm_wos = sort_hmm_state(hmm_wos);

                % add 'ghost states' if K < Kmax so all gaze descriptor vectors have the same dimension
                hmm_wos=pad_with_ghost_states(hmm_wos,max(K),im);

                %Extract gaze_descriptor vector from HMM parameters:
                %priors, transition matrix coefficients, state centres and state covariances
                gaze_descriptor_wos(isub,:) =extract_hmm_parameters(hmm_wos);

                if isplotHMM
                    subplot(1,3,2)
                    plot_hmm_state(hmm_wos,scanpath_wos,im)
                    title WithoutSound
                end
            end
        end

        gaze_descriptor_ws(isnan(gaze_descriptor_ws(:,1)),:)=[];
        gaze_descriptor_wos(isnan(gaze_descriptor_wos(:,1)),:)=[];

        HMM_descriptor_Coutrot.(im_name_struct).with_os.gaze_descriptor=gaze_descriptor_ws;
        HMM_descriptor_Coutrot.(im_name_struct).without_os.gaze_descriptor=gaze_descriptor_wos;

    end
    save('HMM_descriptor_Coutrot','HMM_descriptor_Coutrot')
```

Second, HMM parameters are normalized (zscore) and regularized ($W^{regul} = (1\text{-}\lambda)W + \lambda I$, with λ=1e-5). Regularization is necessary when padding has been applied, since null coefficients lead to singularities.
Users can then choose the classes to classify as well as the classification method in **Classif_from_HMM_descriptors_Coutrot.m**. Are available: Linear Discriminant Analysis (*LDA*), Quadratic Discriminant Analysis with a diagonal covariance matrix estimate (*diagquadratic*), Mahalanobis distances with stratified covariance estimates (*mahalanobis*), Support Vector Machine with linear kernel (*SVM*), Relevance Vector Machine (*RVM*), *AdaBoost* with 100 weak classifiers, and *Random Forest* with 200 trees. All these methods are called in **classifier.m**, where their parameters (e.g. kernels) can be tuned.
A correct classification score (number of correctly classified scanpaths / total number of scanpaths) is computed for each stimulus via a leave-one-out procedure if cross_validation=1, via a k-fold cross-validation otherwise.

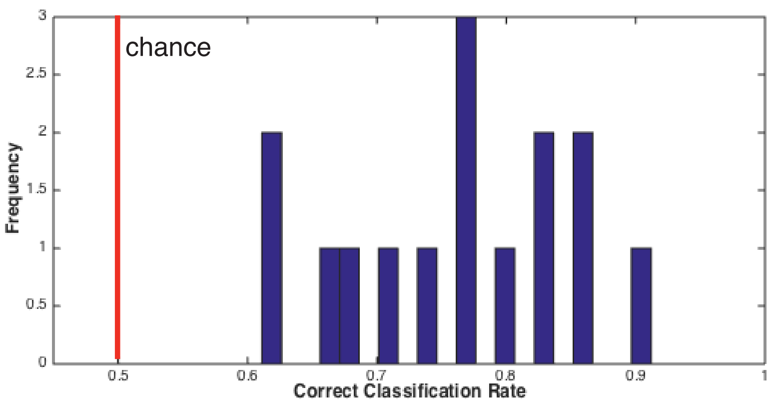
Classif_from_HMM_descriptors_Coutrot.m

```
%% Choose classes to classify
gaze_descriptors={regul_gaze_descriptor_ws, regul_gaze_descriptor_wos};
categoric_var={'with_os', 'without_os'};

%% Select type of classifier
classifier_type='LDA';
%  classifier_type='diagquadratic';
% classifier_type='mahalanobis';
%classifier_type='SVMBinary';
%classifier_type='SVMMultiClass';
%classifier_type='AdaBoostBinary';
%classifier_type='AdaBoostMultiClass';
% classifier_type='RVM';%Only for 2-class problems
% classifier_type='AdaBoost';%Only for 2-class problems
% classifier_type= 'RandomForest';

%% k-fold cross-validation
cross_validation=1;
% % if cross_validation==1
% %     leave-one-out
% % else
% %    'k'-cross_validation
% % end
try
    [lda_stats, success_rate] = classifier(categoric_var, gaze_descriptors,classifier_type,cross_validation);

    %        %LDA 1st Eigen vector: absolute values and normalization
    %        lda_stats.eigenvec(:,1)=abs(lda_stats.eigenvec(:,1));
    %        LDA_1st_Eigen_vect(:,istim)=lda_stats.eigenvec(:,1)/sum(lda_stats.eigenvec(:,1));
catch
    fprintf('stimuli %u could not be classified\n',istim)
    success_rate=NaN;
    manova_stats=NaN;
end
```



Correct classification rate for the 15 stimuli from Coutrot’s dataset