

EMHMM-Toolbox: Eye-Movement analysis with Hidden Markov Models (HMMs)

Antoni B. Chan, City University of Hong Kong
Janet H. Hsiao, University of Hong Kong
Tim Chuk, University of Hong Kong
Cynthia Y.H. Chan, University of Hong Kong
Lan Hui, City University of Hong Kong

Copyright (c) 2021, City University of Hong Kong & University of Hong Kong
2021-Sep-9, v0.80

1 Quick Start

This is a MATLAB toolbox for analyzing eye movement data with hidden Markov Models (HMMs). The major functions of the toolbox are:

- 1) Estimating HMMs for an individual's eye-gaze data.
- 2) Clustering individuals' HMMs to find common strategies.
- 3) Visualizing HMMs and fixation data.
- 4) Statistical tests to see if two HMMs are different.
- 5) Using representative Holistic and Analytic models to compute HA scale value.
- 6) Computing entropy of HMMs.
- 7) Co-clustering individuals' HMMs (from different stimuli) to find common strategies
- 8) Visualizing co-clustering group HMMs.

You can find more information about the methodology in the below reference papers.

1.1 Brief Instructions

- 1) In MATLAB, run "setup" to setup the toolbox.
- 2) Note: if you are using Mac, you may need to disable Gatekeeper security for the MEX files in the toolbox. See Sec 3.1 in the documentation for more info.
- 3) In demo folder, run "demo_faces" for an example.
- 4) Also see "demo_faces_jov_clustering", "demo_faces_jov_compare", "demo_faces_duration", and "demo_conversion".
- 5) There are also demos in the "models" folder: "demo_PBR_model".
- 6) In "demo" folder run "demo_vb_faces" and "demo_vb_faces_duration" for demos using VBHEM for clustering, which automatically determines the number of clusters and states.
- 7) In "demo_cocluster" folder, run "demo_cocluster" for an example of using co-clustering. The script "brm_cocluster_analysis" will analyze the co-clustering results from the BRM journal paper.

More documentation and descriptions are in the "docs" folder. In particular, "docs/emhmm-tutorial.pdf" contains slides introducing how to use the toolbox, while "docs/emhmm-documentation.pdf" contains more detailed information.

1.2 References

If you use this toolbox, please cite the following papers:

- For learning HMMs for eye gaze data:
Tim Chuk, Antoni B. Chan, and Janet H. Hsiao, "Understanding eye movements in face recognition using hidden Markov models", *Journal of Vision*, 14(11):8, Sep 2014.
- For clustering HMMs with the VHEM algorithm:
Emanuele Coviello, Antoni B. Chan, and Gert R.G. Lanckriet. "Clustering hidden Markov models with variational HEM". *Journal of Machine Learning Research*, 15(2):697-747, Feb 2014.
- For representative Holistic/Analytic models and HA Scale:
Cynthia Y.H. Chan, Antoni B. Chan, Tatia M.C. Lee, and Janet H. Hsiao, "Eye Movement Patterns in Face Recognition are Associated with Cognitive Decline in Older Adults." *Psychonomic Bulletin & Review*, 25(6):2200-2207, Dec 2018.
- For clustering HMMs with VBHEM (selecting the number of clusters automatically):
Hui Lan, Ziquan Liu, Janet H. Hsiao, Dan Yu, and Antoni B. Chan. "Clustering Hidden Markov Models With Variational Bayesian Hierarchical EM." *IEEE Trans. on Neural Networks and Learning Systems*, to appear 2021.
- For co-clustering HMMs with VHEM or VBHEM:
Janet H. Hsiao, Hui Lan, Yueyuan Zheng, and Antoni B. Chan. "Eye Movement analysis with Hidden Markov Models (EMHMM) with co-clustering." *Behavior Research Methods*, April 2021.

1.3 Contact Info

Send comments, bug reports, feature requests to Antoni Chan (abchan@cityu.edu.hk).

1.4 Tutorial

The tutorial slides on how to install, setup, and use the toolbox are in the file "emhmm-tutorial.pdf".

1.5 Acknowledgements

This research was supported by the Research Grant Council of Hong Kong SAR: #17609117, #17402814 and HKU 745210H for J.H. Hsiao; CityU 110513 and G-CityU109/14 for A.B. Chan. We also thank the HKU Seed Funding Programme for Basic Research (Project numbers 201311159131 and 201811159165) to J.H. Hsiao. We also thank the Strategic Research Grant from City University of Hong Kong (Project No. 7005218) to A.B. Chan.

2 Table of Contents

1	Quick Start.....	1
1.1	Brief Instructions.....	1
1.2	References	2
1.3	Contact Info.....	2
1.4	Tutorial.....	2
1.5	Acknowledgements.....	2
2	Table of Contents	3
3	Setup	4
3.1	MEX files.....	4
3.2	MATLAB version and toolboxes	5
3.3	Parallel Computing Toolbox	5
4	Demo for Clustering HMMs	5
4.1	Reading data	5
4.2	Estimating and viewing individual HMMs.....	6
4.3	Clustering Individuals' HMMs	10
4.4	Analysis of HMMs	13
5	Other Demos.....	15
5.1	Computing entropy for individual HMMs.....	15
5.2	Eye fixation data with duration	17
5.3	Automatically estimating the number of groups when clustering.....	18
5.4	Other JOV Demos.....	19
5.5	Representative Holistic/Analytic Models and H-A Scale	19
6	Demo for Co-clustering HMMs	21
6.1	Reading Data	21
6.2	Estimating Individual HMMs	22
6.3	Co-Clustering HMMs	23
6.4	Analysis of HMMs	24
6.5	Co-clustering with VBHEM to automatically select the number of clusters	25
6.6	Other experiments.....	26
6.7	Other options	26
7	Frequently Asked Questions (FAQ).....	27
7.1	How do I make grayscale figures for printing?	27
7.2	How do I used pre-specified ROIs?	27
7.3	How to get the sequence of ROIs visited?	28

7.4	How to run a statistical test using symmetric KL divergence?	29
7.5	There are too many fixations on my plot. How do I make it look better?	29
7.6	How to change the location of the transition matrix in the plots?	29
7.7	How to change the order of the clusters from VHEM?	30
7.8	How to change the order of the ROIs in an HMM?	30
7.9	How do I remove fixations not on the stimuli in the image?	31
7.10	How to convert coordinates between face images of different datasets?	32
8	Function Usage.....	34
8.1	vbhmm_learn.....	34
8.2	vbhmm_learn_batch.....	36
8.3	vbhmm_set_hyperparam	37
8.4	vhem_cluster	38
9	Toolbox Contents.....	39

3 Setup

The toolbox includes a script for setting up the working environment of EMHMM. In MATLAB, change to the main directory of the toolbox and run the following to setup EMHMM.

```
>> setup
*** EMHMM - Eye-Movement analysis with HMMs ***
- current version: v0.80
- checking for new version...
- version up-to-date.
- checking MEX files...
- all OK
```

This will perform the following: 1) setup the MATLAB path to include the toolbox; 2) check if there is a new version of EMHMM available (requires an Internet connection); 3) check the MEX files and if necessary, compile and test them.

3.1 MEX files

EMHMM uses MEX files to speed up computations of some of the algorithms. MEX files are compiled functions that are native to specific computer platforms. *EMHMM includes the compiled code for macOS (High Sierra 10.13), Windows 7, and Linux (Ubuntu) for 64-bit processors. Hence for those systems you do not need to compile the MEX files.*

The toolbox will check if there are compatible MEX files when running “setup”. If MEX files are not found, the toolbox will attempt to compile the code for your computer automatically. Here are some tips:

- If you have never compiled a MEX file on your system before, you may need to run “mex -setup” to pick the compiler.
- To compile on Mac, you need to download and install Xcode from Apple. There could be some issues compiling on older versions of MATLAB when using a newer macOS.

These problems can be fixed by modifying a few files in Xcode – more info [here](#) and [here](#).

- If you need to re-compile the MEX files, run the command:
`emhmm_mex_check('all')`.

On **macOS**, there may be another problem with running the MEX files, which gives an error like *“file.mexmaci64 cannot be opened because the developer cannot be verified”*. This issue is caused by the GateKeeper security function on mac that prevents execution of unknown code. To resolve the problem, you need to run the following command in the Terminal:

```
sudo xattr -r -d com.apple.quarantine /path/to/emhmm-toolbox
```

This will remove the security lock on the MEX files in the toolbox. For more details, see [here](#).


Finally, you can still use EMHMM without compiled MEX files. However, the algorithms will run about 10-20 times slower.

3.2 MATLAB version and toolboxes

The EMHMM toolbox has been tested with MATLAB 2014b, 2018b, and 2020a, so it should work for versions 2014b and higher. It may work for earlier versions of MATLAB (down to 2011), but we have not tested it completely. The EMHMM toolbox uses the following MATLAB toolboxes:

- [Statistics toolbox](#)
- [Parallel computing toolbox](#) (optional)

3.3 Parallel Computing Toolbox

If the Parallel Computing toolbox is available, EMHMM will use “parallel for loops” (`parfor`) to speed up the calculations. The speed-up will depend on the number of CPU cores in your computer. To use this feature, you need to have a “parallel pool” created on your computer. MATLAB can be set to create pools automatically. Go to the MATLAB “Preferences”, and select “Parallel Computing Toolbox” and enable “automatically create a parallel pool”. You can also manually start a parallel pool by clicking on the small icon  on the bottom-left of the MATLAB window, and selecting “Start parallel pool”.

4 Demo for Clustering HMMs

In this section, we will go through one of the demo programs to show the functions of the toolbox. Change to the “demo” folder, run “demo_faces” to run a demo script. This demo will learn individual HMMs from their fixations, and then perform clustering to discover common eye gaze strategies.

```
>> cd demo
>> demo_faces
```

4.1 Reading data

This script will first load the fixation data from an Excel spreadsheet, `demodata.xls`. The spreadsheet has 4 columns: SubjectID, TrialID, FixX, and FixY, which correspond to the subject ID, trial number (ID), and X and Y fixation locations. The fixation data is

automatically separated according to the subject ID and trial number. In each trial, it assumes that the fixations occur in order. The output will look like this:

```
Reading demodata.xls
- found SubjectID in column 1
- found TrialID in column 2
- found FixX in column 3
- found FixY in column 4
- found 10 subjects:
1 2 3 4 5 6 7 8 9 10
  * subject 1 had 40 trials
  * subject 2 had 40 trials
  * subject 3 had 40 trials
  * subject 4 had 40 trials
  * subject 5 had 40 trials
  * subject 6 had 40 trials
  * subject 7 had 40 trials
  * subject 8 had 40 trials
  * subject 9 had 40 trials
  * subject 10 had 39 trials
```

The program found 10 subjects with 39-40 trials each.

4.2 Estimating and viewing individual HMMs

Next, an individual HMM will be estimated from each subject's fixation data. We use the variational Bayesian EM algorithm to estimate the HMMs. It can automatically select the number of ROIs, as well as estimate all the hyperparameters from the data, by finding the model with highest log-likelihood of the data. In this case, we try $K=[1, 2, 3]$ ROIs. For each K , the VB algorithm is run 50 times with random initializations, and then the hyperparameters are estimated for each unique result. The run with the highest log-likelihood is kept. The best model over all the candidate K s will be selected. The `vbhmm_learn_batch` command will take data for all subjects, and learn an individual HMM for each subject. The command looks like this:

```
>> [hmms] = vbhmm_learn_batch(data, K, vbopt);
```

where `vbopt` is a structure containing the options for running the VB-EM algorithm (see the demo script, and details below). `hmms` is a cell array containing the HMM for each subject.

The output for one subject looks like this (when Parallel Computing is not available):

```

=== running Subject 10 ===
[Subject 10] -- vbhmm K=1: (seed=1000).
[Subject 10] (K=1) optimizing trial 1: -729.844.....LL=-724.012
[Subject 10] (K=1) best run=1; LL=-724.012
[Subject 10] (K=1) {alpha0=1; epsilon0=1; v0=5.497; beta0=5.564e+08; W0=0.001332; mu0=[166;219.8]; }
[Subject 10] -- vbhmm K=2:
(seed=1000).....10.....20.....30.....40.....50
[Subject 10] (K=2) optimizing trial 1: -730.401.....LL=-723.247
[Subject 10] (K=2) best run=1; LL=-723.247
[Subject 10] (K=2) {alpha0=1.533e+08; epsilon0=6.049e-10; v0=4.911; beta0=1.778; W0=0.002266; mu0=[164.8;218.8]; }
[Subject 10] -- vbhmm K=3:
(seed=1000).....10.....20.....30.....40.....50
[Subject 10] (K=3) optimizing trial 1: -736.514.....LL=-729.707
[Subject 10] (K=3) optimizing trial 3: -738.425.....LL=-731.333
[Subject 10] (K=3) optimizing trial 29: -737.579.....LL=-730.541
[Subject 10] (K=3) best run=1; LL=-729.706
[Subject 10] (K=3) {alpha0=4.29; epsilon0=0.05854; v0=7.114; beta0=0.6597; W0=0.001853; mu0=[169.8;209.3]; }
[Subject 10] best model overall: K=2; L=-722.553
[Subject 10] {alpha0=1.533e+08; epsilon0=6.049e-10; v0=4.911; beta0=1.778; W0=0.002266; mu0=[164.8;218.8]; }

```

Different values of K are tried, and then the hyperparameters are optimized for a few trials. Finally the best model with highest log-likelihood (LL) is selected. The dots and numbers indicate the trials being run, or each iteration in hyperparameter estimation.

When Parallel Computing is available, then the `vbhmm_learn_batch` function will learn HMMs on several subjects in parallel to save time. The output will look like this:

```

Starting parallel pool (parpool) using the 'local' profile ...
connected to 6 workers.
=== running Subject 1 ===
=== running Subject 2 ===
=== running Subject 3 ===
=== running Subject 4 ===
=== running Subject 5 ===
=== running Subject 6 ===
[Subject 1] -- vbhmm K=1: (seed=1000).
[Subject 1] (K=1) optimizing trial 1: -1122.78.....LL=-1104.64
[Subject 1] (K=1) best run=1; LL=-1104.64
[Subject 1] (K=1) {alpha0=1; epsilon0=1; v0=830.3; beta0=Inf; W0=1.658e-06; mu0=[155.9;210]; }
[Subject 2] -- vbhmm K=1: (seed=1000).
[Subject 2] (K=1) optimizing trial 1: -1005.12.....LL=-993.392
[Subject 2] (K=1) best run=1; LL=-993.392
[Subject 2] (K=1) {alpha0=1; epsilon0=1; v0=987.8; beta0=Inf; W0=3.339e-06; mu0=[160.1;200.2]; }
[Subject 3] -- vbhmm K=1: (seed=1000).
[Subject 3] (K=1) optimizing trial 1: -916.899.....LL=-903.895
[Subject 3] (K=1) best run=1; LL=-903.895
[Subject 3] (K=1) {alpha0=1; epsilon0=1; v0=17.37; beta0=1.058e+15; W0=0.00011; mu0=[164.6;198.4]; }
[Subject 4] -- vbhmm K=1: (seed=1000).
[Subject 4] (K=1) optimizing trial 1: -1010.87.....LL=-996.495
[Subject 4] (K=1) best run=1; LL=-996.495
[Subject 4] (K=1) {alpha0=1; epsilon0=1; v0=1184; beta0=5.788e+221; W0=2e-06; mu0=[172.5;205.6]; }
[Subject 5] -- vbhmm K=1: (seed=1000).
[Subject 5] (K=1) optimizing trial 1: -953.233.....LL=-941.716
[Subject 5] (K=1) best run=1; LL=-941.716
[Subject 5] (K=1) {alpha0=1; epsilon0=1; v0=147.7; beta0=6.402e+99; W0=2.1e-05; mu0=[138.4;209.5]; }
[Subject 6] -- vbhmm K=1: (seed=1000).
[Subject 6] (K=1) optimizing trial 1: -820.548.....LL=-812.185
[Subject 6] (K=1) best run=1; LL=-812.185
[Subject 6] (K=1) {alpha0=1; epsilon0=1; v0=5.537; beta0=8.425e+14; W0=0.0007937; mu0=[157.1;242.1]; }
[Subject 1] -- vbhmm K=2:
(seed=1000).....10.....20.....30.....40.....50
[Subject 1] (K=2) optimizing trial 1: -1080.29.....LL=-1073.53
[Subject 2] -- vbhmm K=2:
(seed=1000).....10.....20.....30.....40.....50
[Subject 3] -- vbhmm K=2:
(seed=1000).....10.....20.....30.....40.....50
[Subject 4] -- vbhmm K=2:
(seed=1000).....10.....20.....30.....40.....50
[Subject 4] (K=2) optimizing trial 1: -975.826.....LL=-969.663
[Subject 5] -- vbhmm K=2:
(seed=1000).....10.....20.....30.....40.....50
[Subject 6] -- vbhmm K=2:
(seed=1000).....10.....20.....30.....40.....50
...

```

Since there are 6 workers in the pool, then 6 subjects will be run in parallel, and the output messages are mixed together.

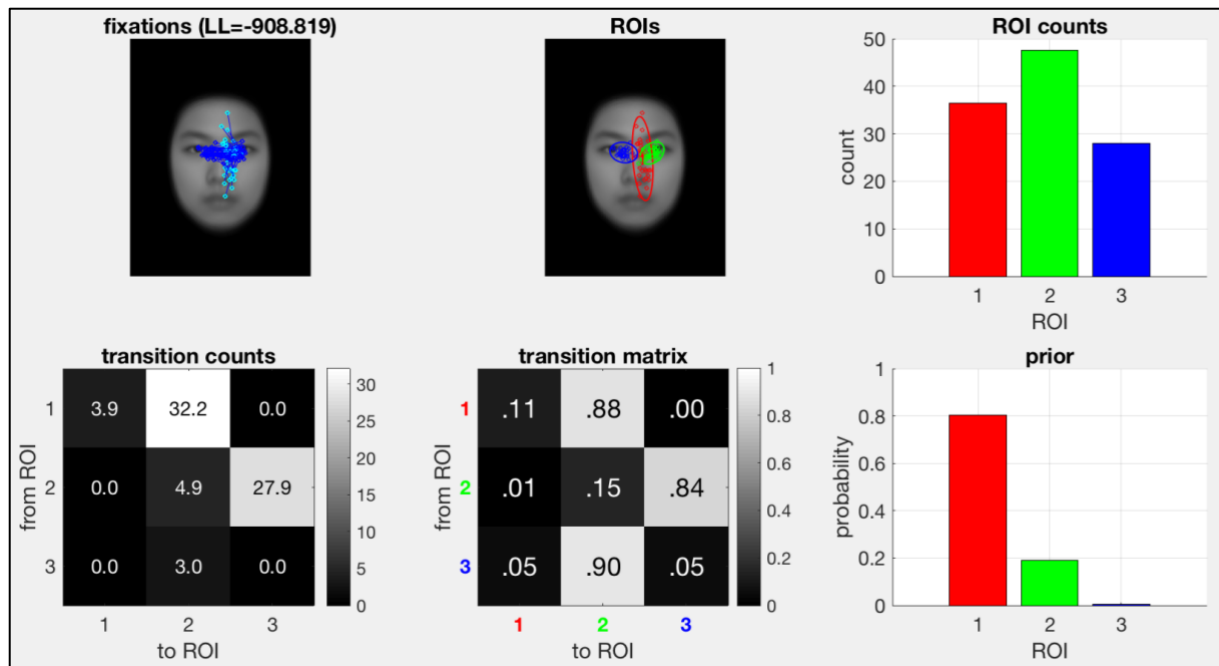
To visualize an HMM and the data, we use the following 2 commands (in the demo script):

```

vbhmm_plot(hmms{4}, data{4}, faceimg)
figure, vbhmm_plot_compact(hmms{4}, faceimg)

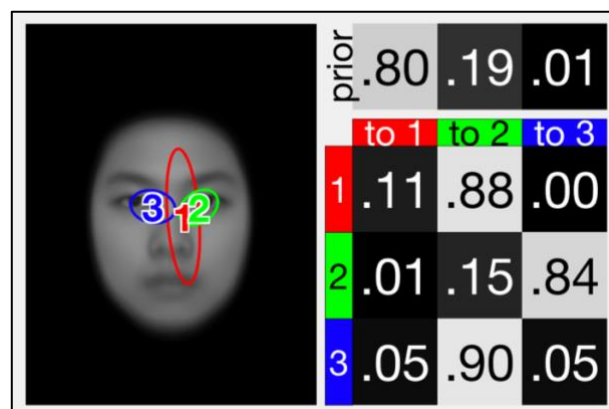
```

The first command looks at the HMM of the first subject. "faceimg" contains the name of the image file of the face to put in the background. The figure from `vbhmm_plot` looks like this (Figure 1 from the demo):



The top-left plot shows the fixation sequences, and the cyan “o” is the first fixation. The top-middle plot shows the emission densities (ROIs) and the corresponding fixations. The top-right plot shows the total number of fixations in each ROI. On the bottom, the left and middle plots show the transition counts and transition matrix. In the transition matrix (bottom-middle), each row is a probability distribution. I.e., each row is normalized to sum to 1. Finally, the bottom-right shows the prior probability, i.e., the probability of the ROI of the first fixation. The ROIs are automatically sorted according to the most likely fixation path: ROI 1 is the most probable initial fixation; ROI 2 is the most likely next fixation, given a fixation in ROI 1; etc. In this example, the subject has 80% probability to look at ROI 1 (red region). Then has an 88% of transitioning to the green ROI 2. In the green ROI 2, the subject will then look at blue ROI 3 (84% probability). When in the blue ROI 3, the subject tends to look back at the green ROI 2 (90% probability). So the subject tends to look back and forth between the eyes.

The second command makes a compact plot of the HMM and doesn't show the data. The figure looks like this (Figure 2 from the demo):

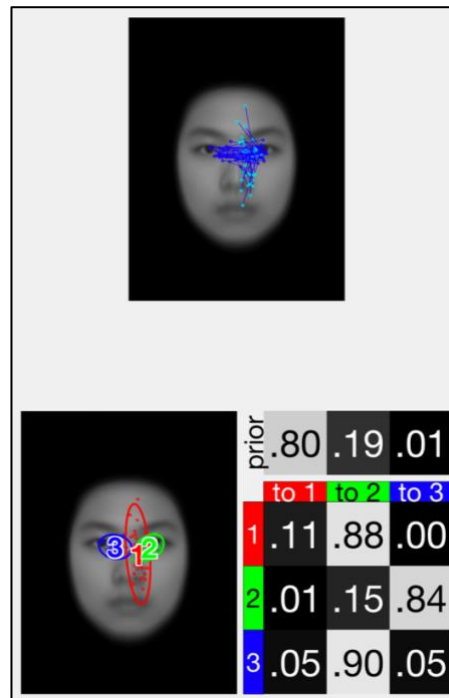


On the left side are the ROIs (labeled with colors and numbers). The right-top shows the prior probabilities, and the right-bottom shows the transition matrix. The rows/columns are labeled with colors and numbers to indicate the corresponding ROIs.

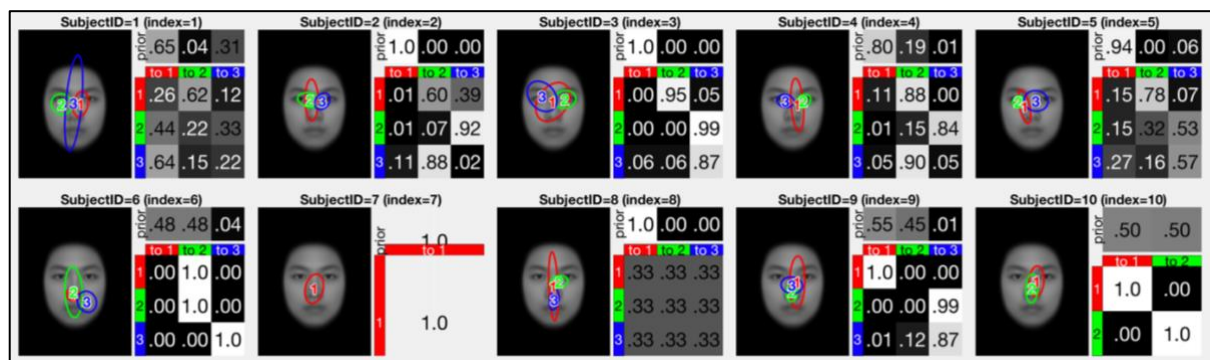
We can also visualize the fixations on the HMM using the following code:

```
>> figure
>> subplot(2,1,1)
>> plot_fixations(data{mys}, faceimg, [], 's');
>> subplot(2,1,2)
>> vbhmm_plot_compact(hmms{mys}, faceimg, 'r', data{mys});
```

Here is the resulting figure, which shows a separate plot of the fixations, as well as the fixations on the ROIs (Figure 3 in the demo). Each color of the fixation represents the ROI that it was assigned to.



The demo script also plots all the subjects in one figure (Figure 4 from the demo):



The demo script saves the individual HMMs to the file: models_demo_faces_individual.mat

4.3 Clustering Individuals' HMMs

Given the 10 individuals' HMMs, the demo script then computes the overall eye gaze pattern by clustering the individuals into one group. Here the variational HEM (VHEM) algorithm is used to cluster the HMMs. The algorithm tries 3 different initialization methods, running each one 100 times. The run with the highest log-likelihood is kept. Here is the command used in the demo:

```
>> [all_hmms1] = vhem_cluster(hmms, 1, 3, hemopt)
```

First we use K=1 groups (i.e., put all into one group) and S=3 hidden states. `hemopt` is a structure containing the options for running VHEM. The output of the demo script looks like this:

```
=== Clustering (1 group) ===
Checking input HMMS: done
+ set seed to 1001
auto initialization: trying baseem: VHEM Trial: .....
.....

Best run is 77: LL=-901.647
auto initialization: trying gmmNew: VHEM Trial: .....
.....

Best run is 76: LL=-901.644
auto initialization: trying gmmNew2: VHEM Trial: .....
.....

Best run is 8: LL=-901.651
best init was gmmNew; LL=-901.644

all_hmms1 =

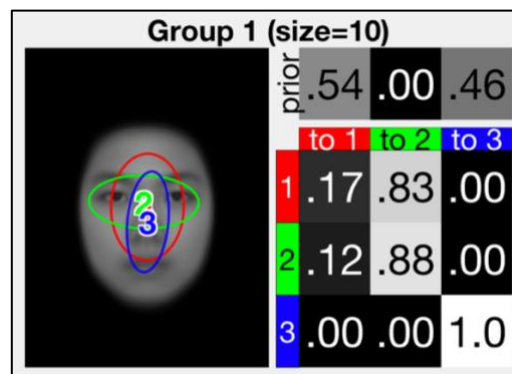
    struct with fields:

        Z: [10x1 double]
        LogLs: [1x29 double]
        LogL: -901.6440
        label: [1 1 1 1 1 1 1 1 1 1]
        groups: {[1 2 3 4 5 6 7 8 9 10]}
        group_size: 10
        hmms: {[1x1 struct]}
        hemopt: [1x1 struct]
        emhmm_version: 'v0.75'
```

We can plot the group HMM using the following command:

```
>> vhem_plot(all_hmms1, faceimg);
```

The corresponding plot of the group HMM is here (Figure 5 from the demo):



For the overall HMM for all subjects, ROI 1 is large and cover most of the face features. This is mainly because there are different strategies used by each subject, and merging them all together yields a “washed out” HMM, which lacks specific details. One interesting thing is that subjects’ mainly look to the eyes for the second fixation (green ROI), after looking somewhere on the face (red ROI). This indicates that a common strategy of subjects looking somewhere on the face first, and then turning to the eyes. Alternatively, 46% of the time, the subjects looked at only the nose and mouth (blue ROI).

To discover the different strategies among the subjects, next the demo script clusters the individual HMMs into 2 groups using VHEM, by changing to K=2.

```
>> [group_hmms2] = vhem_cluster(hmms, 2, 3, hemopt)
```

Here is the output:

```
=== Clustering (2 groups) ===
Checking input HMMS: done
+ set seed to 1001
auto initialization: trying baseem: VHEM Trial: .....
.....

Best run is 44: LL=-879.151
auto initialization: trying gmmNew: VHEM Trial: .....
.....

Best run is 79: LL=-878.838
auto initialization: trying gmmNew2: VHEM Trial: .....
.....

Best run is 44: LL=-878.839
best init was gmmNew; LL=-878.838

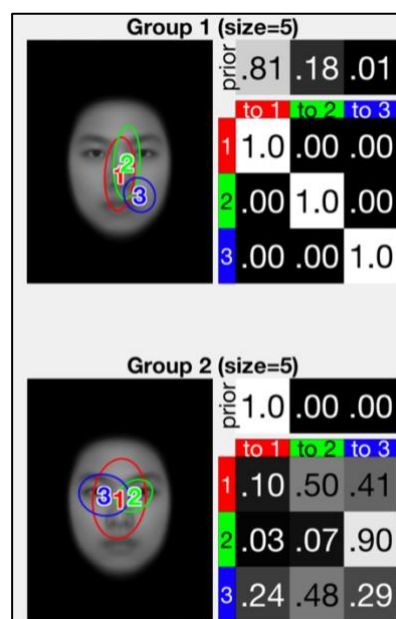
group_hmms2 =

  struct with fields:

      Z: [10x2 double]
      LogLs: [1x37 double]
      LogL: -878.8376
      label: [2 2 2 2 2 1 1 1 1 1]
      groups: {[6 7 8 9 10] [1 2 3 4 5]}
      group_size: [5 5]
      hmms: {[1x1 struct] [1x1 struct]}
      hemopt: [1x1 struct]
      emhmm_version: 'v0.75'

Group membership (indicies):
  group 1 = [6 7 8 9 10]
  group 2 = [1 2 3 4 5]
Group membership (SubjectID):
  group 1 = 6, 7, 8, 9, 10,
  group 2 = 1, 2, 3, 4, 5,
```

Here is the plot of the two group HMMs (Figure 6 from the demo):

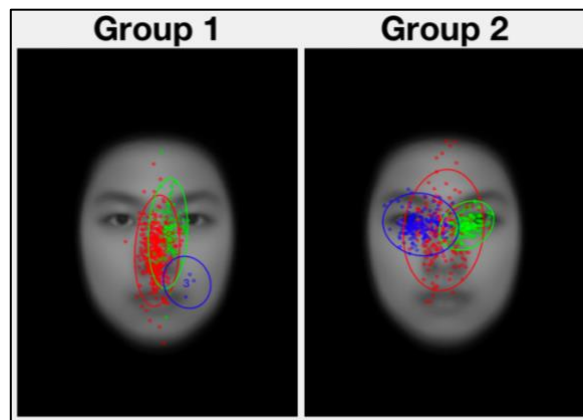


After clustering into two groups, we see two separate strategies. For Group 1 (top), they tend to look at the center-line of the face (red and green ROIs), and very infrequently at the corner of the mouth (blue ROI). Hence, Group 1 is representative of a holistic strategy. For Group 2 (bottom), they tend to look between the eyes (green and blue ROIs), and thus this is representative of an analytic strategy.

Finally, we can look at the fixations of subjects assigned to each group using the function `vhem_plot_fixations`:

```
>> vhem_plot_fixations(data, group_hmms2, faceimg);
```

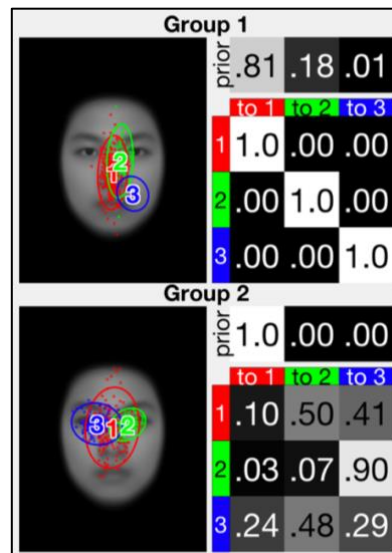
Here is the example figure (Figure 9 from the demo)



Adding the 'c' option will also show the transition matrices:

```
>> vhem_plot_fixations(data, group_hmms2, faceimg, 'c');
```

Here is the example figure (Figure 10 from the demo):



4.4 Analysis of HMMs

Next, the demo script analyzes the two group HMMs. The following command finds the most probable ROI sequences from the first group HMM, using length=3 sequences:

```
>> [seqs, seqs_pr] = stats_stateseq(group_hmms2.hmms{1}, 3);
```

Here is the output from the demo:

```
*** top-5 most probable ROI sequences ***
prob      : hmm1 ROI seq
-----
0.8067 : 1 1 1
0.1849 : 2 2 2
0.0084 : 3 3 3
0.0000 : 1 2 2
0.0000 : 1 1 2

prob      : hmm2 ROI seq
-----
0.4465 : 1 2 3
0.1934 : 1 3 2
0.1163 : 1 3 3
0.0954 : 1 3 1
0.0488 : 1 1 2
```

For hmm1, the most likely ROI sequence is 1-1-1, with probability 80.7%, followed by 2-2-2 (18.5%). For hmm2, the most likely ROI sequences is 1-2-3, with probability 44.7%, followed by 1-3-2 with probability 19.3% and 1-3-3 with probability 11.6%.

Next, we test whether the two group HMMs are different. We take the data for subjects in Group 1, and calculate the log-likelihood of this data under the Group 1 HMM and under the Group 2 HMM. The average difference between the log-likelihoods is an estimate of the Kullback-Leibler (KL) divergence, which is a dissimilarity measure for probability distributions. A KL divergence of 0 means that the two distributions (HMMs) are the same. We run a paired t-test on the two lists of log-likelihoods to see if the average log-likelihood difference is significantly different from 0. Since KL divergence is not symmetric, we need to do this twice: once using the data from Group 1, and once using the data from Group 2. The code for computing test on hmm1 is:

```
>> data1 = data(group_hmms2.groups{1});
>> [p, info, lld] = stats_ttest(group_hmms2.hmms{1}, ...
    group_hmms2.hmms{2}, data1);
```

The first line collects the data for subjects in group 1. The second line computes the test.

The test results are below:

```
*** t-test ***
- test group hmm1 different from group hmm2: t(4)=8.15202; p=0.000616171
- test group hmm2 different from group hmm1: t(4)=6.41628; p=0.00151614
```

In both cases, the two HMMs are significantly different ($p=0.0006$ and $p=0.0015$).

The script then calculates the mean log-likelihood of each subject under the two group HMMs. A correlation test can be used with the log-likelihood data and behavioral data, to see if a particular eye fixation strategy is correlated with behavior. The command for calculating the mean log-likelihoods for hmm1 is:

```
>> [mLL1] = stats_meanll(group_hmms2.hmms{1}, data);
```

The output of the demo shows both hmms:

```

*** get mean LL for each subject ***
- mean LL for each subject under group hmm1:

mLL1 =
  -11.5196   -9.9474  -12.3179  -10.6476  -12.3742   -8.5738   -8.1270
  -8.8334   -8.4829   -8.2036

- mean LL for each subject under group hmm2:

mLL2 =
   -9.2684   -8.4376   -8.8677   -8.5927   -8.8478  -10.1217   -9.1723
  -9.8693   -9.2937   -9.0687

```

Finally, the script computes the “AB scale” for each subject. The AB scale is a quantitative measure of the similarity of each subject to either cluster 1 (denoted as A) or cluster 2 (denoted as B). It is computed as:

$$AB = \frac{LLA - LLB}{|LLA| + |LLB|}$$

where LLA and LLB are the log-likelihoods of the subject’s data under cluster A and cluster B. The AB scale value is positive if the subject is closer to cluster A, and negative if closer to cluster B. The script output looks like this:

```

- AB scale values:

AB =
  -0.1083   -0.0821   -0.1629   -0.1068   -0.1662    0.0828
   0.0604    0.0554    0.0456    0.0501

```

The AB values for the first 5 subjects are less than 0 (more similar to cluster B), and likewise, those of the last 5 subjects are greater than 0 (more similar to cluster A). This is consistent with the groupings found in the previous section.

5 Other Demos

The “demos” folder contains a few more demos. This section contains more information about these other demos.

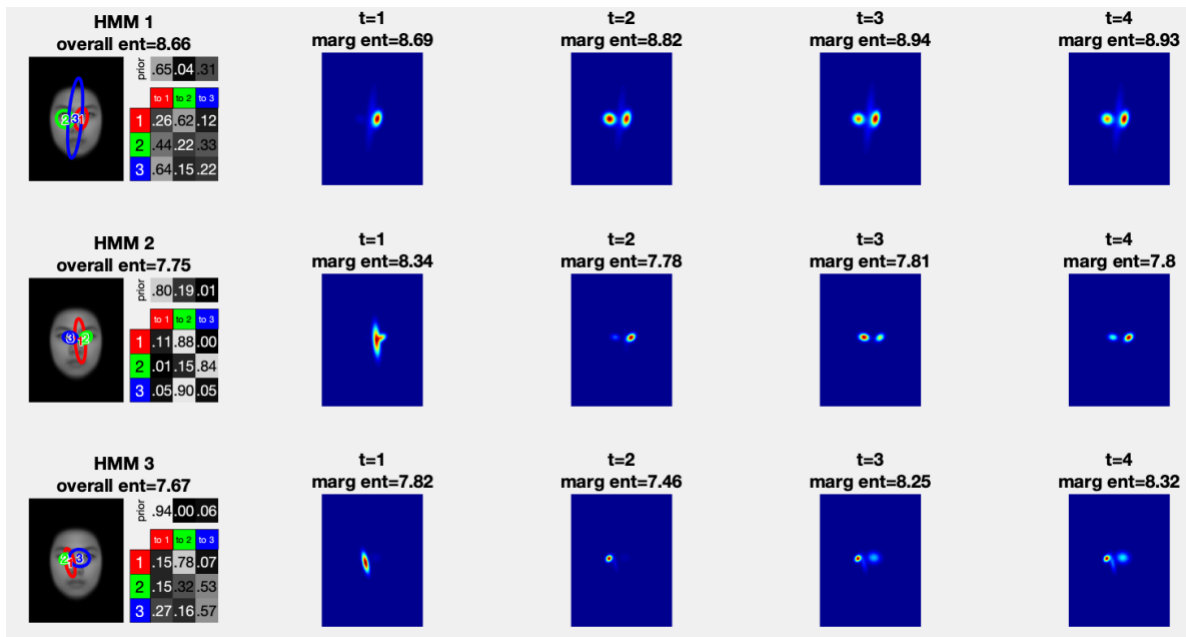
5.1 Computing entropy for individual HMMs

Entropy is a measure of uncertainty or randomness in a random variable. Higher values of entropy indicate more uncertainty, while lower values mean more deterministic (the outcome is more certain). An entropy of 0 means completely deterministic. Various types of entropy can be computed for the individual HMMs for analysis:

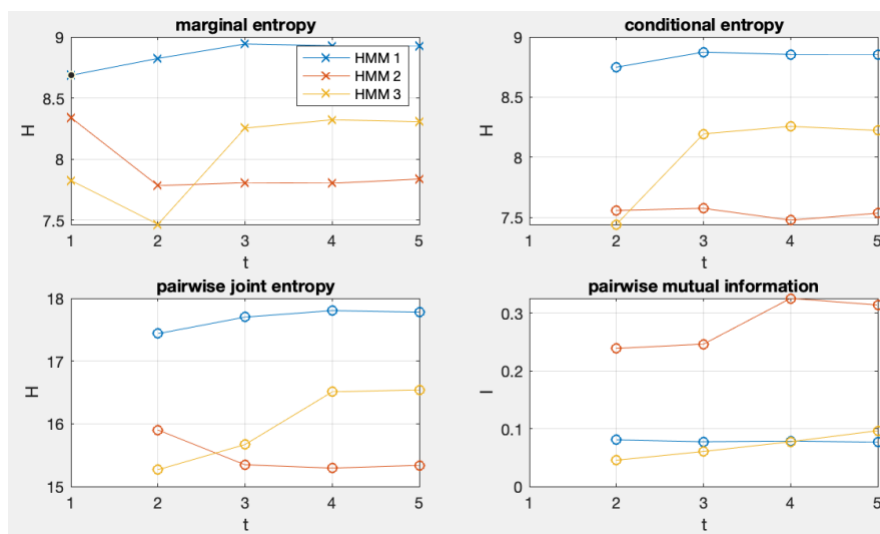
- The *overall entropy* based on fixation sequence data, which is a measurement of the overall uncertainty of the model.
- The *marginal entropy*, which is the entropy of the fixation distribution at time t . This is a measure of the uncertainty of the fixation at time t .
- The *conditional entropy*, which is the entropy of the fixation distribution at time t , given the fixation at time $t-1$, averaged over the fixation distribution at time $t-1$. This is a measure of the uncertainty of the next fixation when the previous fixation is known.

- The *pairwise joint entropy*, which is the entropy of the joint distribution of fixations at time $t-1$ and time t .
- The *pairwise mutual information* of fixations at time $t-1$ and time t . This is a measure of the dependence between two fixations; a value of 0 means the two fixations are statistically independent (knowing one does not tell us anything about the other), and higher values indicate more dependence between the fixations.

In the “demos” folder, the script “demo_faces_entropy.m” shows how to compute entropy for a set of individual HMMs. The script assumes that “demo_faces.m” has already been run. The script generates two demo figures. The first figure shows 3 HMMs and the marginal distributions of fixations at time $t=1$ to $t=4$ (one HMM on each row). The overall entropy and marginal entropies are displayed above each figure.



The second figure shows a plot of various types of entropy over time.

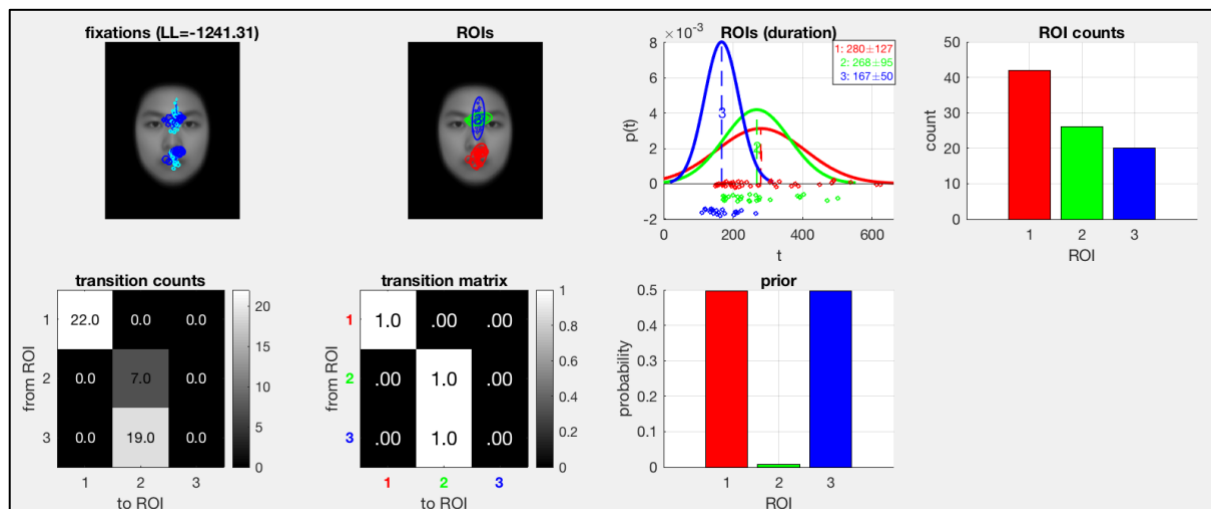


HMM1 has the highest entropy over time, indicating a less consistent eye gaze strategy (i.e., more uncertainty in the fixations). HMM2 has low marginal and conditional entropy, which indicates a consistent strategy. This is evident when examining the transition matrix of

HMM2, since the transition probabilities are quite saturated. Finally, HMM3 has low marginal and conditional entropy in the first 2 fixations, and then it increases. This indicates a consistent strategy for the first 2 fixations, followed by a less consistent strategy afterwards.

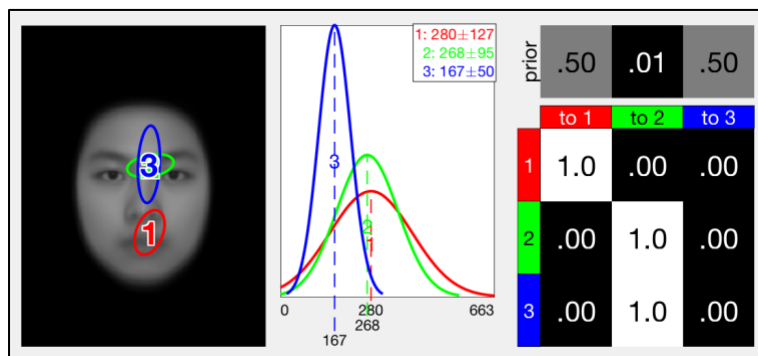
5.2 Eye fixation data with duration

The toolbox can also learn HMMs for eye fixation data consisting of both fixation location (x,y) and duration (time in milliseconds). An example can be found in the “demo_faces_duration.m” script. The script loads the data from “jov_duration.xls”. The Excel spreadsheet now has a 5th column with the header “FixD”, which is for the fixation duration. The commands used to learn a subject’s HMM and group HMMs are the same. Here is an example of the 6th subject’s HMM using fixation location and duration:



The 3rd subplot shows the distribution of fixation duration for each ROI. For this subject, either they will have a long fixation on the mouth (red ROI), or a quick fixation to the upper face (blue ROI) and then looking at the nose for longer (green ROI). The fixation duration data is shown at the bottom of the ROI duration plots. Note that the x-axis values are the durations, and the y-values are randomly selected to better visualize the data. The legend in the upper-right of the subplot shows the mean and standard deviation of the durations for each ROI.

Here is a compact plot of the same HMM:



More details can be found in the demo script “demo_faces_duration.m”.

5.3 Automatically estimating the number of groups when clustering

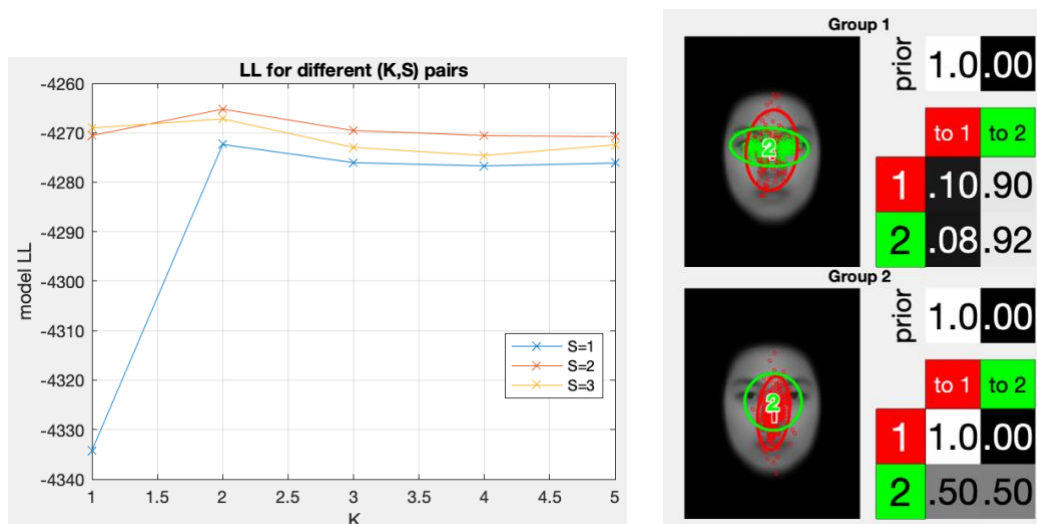
The number of groups and number of states can be automatically determined for clustering using the variational Bayes (VB) framework. A demo is shown in “demo_vb_faces.m”. In this demo, learning the individual HMMs is the same as “demo_faces.m”, and clustering is now performed using the “vbhem_cluster” function. The following code in the demo runs the VBHEM algorithm for clustering:

```
K = 1:5; % automatically select from 1 to 5 clusters
S = 1:3; % automatically select from 1 to 3 ROIs (states)

% initial hyperparameters (similar to vbhmm_learn)
vbhemopt.alpha0 = 1;
vbhemopt.eta0 = 1;
vbhemopt.m0 = vbopt.mu0;
vbhemopt.W0 = 0.001;
vbhemopt.lambda0 = 1;
vbhemopt.v0 = 10;
vbhemopt.epsilon0 = 1;
vbhemopt.learn_hyps = 1; % automatically estimate the hyperparameters
vbhemopt.seed = 1001;
vbhemopt.tau = 5;

% perform VB clustering
group_vbhmm = vbhem_cluster(hmms, K, S, vbhemopt);
```

Here a range of values are tried for K (number of clusters) and S (number of ROIs/states). The vbhemopt options are the initial hyperparameters used during optimization, which are similar to the ones used for VBHMM (vbopt structure). Different values of K and S are used, and the model with highest marginal log-likelihood is selected. The below figure on the left shows the model marginal log-likelihood (LL) for different values of K and S. The model with largest LL has K=2 and S=2, and thus is selected and shown below right.



The script will display the selected values of K and S, as well as the grouping.

```
** overall best model: K=2; S=2; L=-4265.23; after pruning K=2, S=[2 2] **
best init was gmmNew estimated hyps: {alpha0=5.776e+08; eta0=2.154e-08;
epsilon0=0.2215; v0=6.226; lambda0=6.975; W0=0.0005311; m0=[162.3;207.2]; }
Group membership (indicies):
group 1 = [1 2 3 4 5]
group 2 = [6 7 8 9 10]
```

See “demo_vb_faces_duration.m” for the demo using VBHEM on fixation and duration data.

5.4 Other JOV Demos

There are more demos for clustering and comparing HMMs in the demo scripts “demo_faces_jov_clustering.m” and “demo_faces_jov_compare.m”.

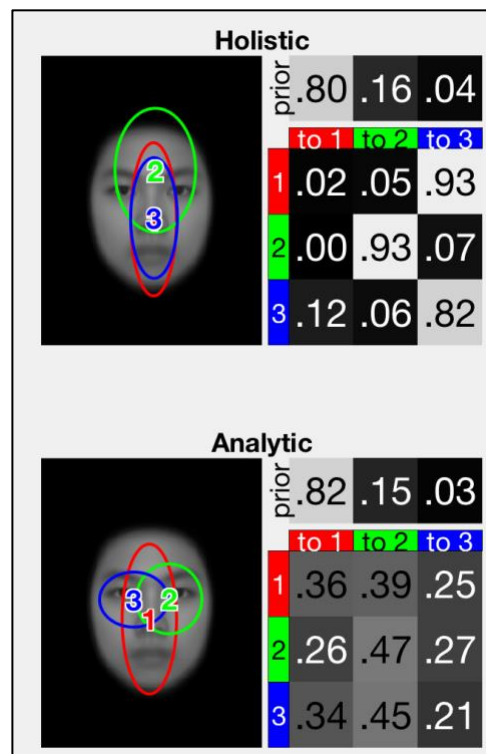
5.5 Representative Holistic/Analytic Models and H-A Scale

In this demo we show the representative models and H-A scale used in the paper:

Cynthia Y.H. Chan, Antoni B. Chan, Tatia M.C. Lee, and Janet H. Hsiao, “Eye Movement Patterns in Face Recognition are Associated with Cognitive Decline in Older Adults.” *Psychonomic Bulletin & Review*, to appear 2018.

The representative model and demo script are in the “models” directory. Change to the “models” directory, and then run `demo_PBR_model`.

The script first loads the representative models and plots them. The top is the Holistic representative model, where fixations tend to be on the center of the face. The bottom is the Analytic representative model, where fixations tend to be on the eyes and mouth.



Next, we load our new data, and then normalize the coordinates to be the same as the images used to learn the representative model (in the demo, the coordinates are the same, so no normalization is actually needed). For more information about converting coordinates between face images, please see the next demo.

We compute the log-likelihoods of the new data under the representative models.

```

*** get mean LL for each subject ***
Column 1 refers to the LL of the Holistic pattern
Column 2 refers to the LL of the Analytic pattern

LL_all =

-10.3345  -9.3578
-9.8063   -9.1211
-9.5015   -9.2958
-9.3297   -9.2799
-8.5411   -9.0032
-9.6745  -10.6560
-9.2844   -9.0184
-9.6761   -9.2645

```

We also compute the Holistic-Analytic (HA) scale, which is:

$$HA = \frac{LLH - LLA}{|LLH| + |LLA|}$$

where LLH is the log-likelihood of the subject's data under the Holistic model, and LLA is the log-likelihood under the Analytic model.

```

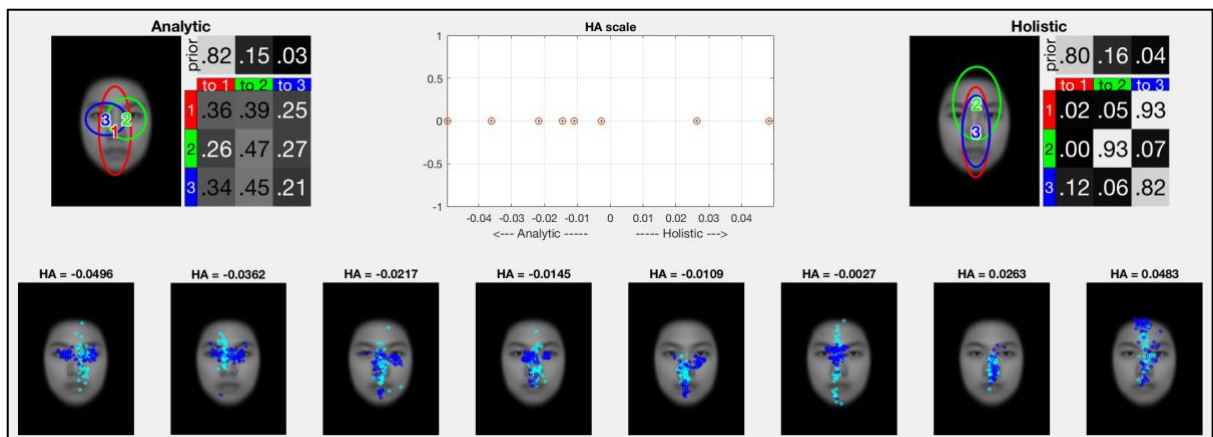
*** H-A scale ***

HAscale =

-0.0496
-0.0362
-0.0109
-0.0027
 0.0263
 0.0483
-0.0145
-0.0217

```

Here is the generated plot showing the Analytic and Holistic representative models (top-left and top-right), the HA scale (top-center). A few subjects on the scale are plotted on the bottom. These are highlighted in the HA scale plot with red circles. Positive HA values indicate a more holistic pattern, while negative HA values indicate a more analytic pattern.



Finally, we compute the cluster assignments of the subjects.

```
*** subject assignments to HMMs ***  
  
*** groups ***  
group 1 - Holistic: 5 6  
group 2 - Analytic: 1 2 3 4 7 8
```

In the demo script, the “datamode” variable can be changed to select one of the other datasets: all JOV data and the demo dataset.

6 Demo for Co-clustering HMMs

The previous demos assume that the layout of the images are similar to each other (e.g., the faces are aligned), so that aggregating fixations across different images makes sense. In some experiment designs, this is not possible, e.g., in scene viewing experiments where each image has a different layout. To handle images with different layouts, we apply co-clustering to group individuals that share similar eye gaze patterns among all images. With co-clustering, representative HMMs are estimated for each image separately, and individuals are grouped together if they have similar eye gaze (on each image) to other group members.

The demo for co-clustering is in the “demo_cocluster” directory, and called “demo_cocluster.m”. Change to this directory to run the demo code. We next go through each part of the co-clustering demo.

```
>> cd demo_cocluster  
>> demo_cocluster
```

Note that all the options for running the co-clustering experiment are stored in the `OPT` structure in the script. The description below is for the default experiment (specified by `OPT.datasource=0`).

6.1 Reading Data

The data expected for co-clustering are the fixations, the image stimuli (for visualization), and the valid box region for each image.

The script first loads the fixation data from an Excel spreadsheet, “test-data/test-data.xlsx”. The spreadsheet should contain 5 columns: SubjectID, TrialID, StimuliID, FixX, and FixY, which correspond to the subject ID, trial number (ID), the name of the image stimuli, and the X and Y fixation locations. Note that the TrialID must be unique across all stimuli for a subject, i.e., different stimuli cannot have the same TrialID. The Excel file is specified like this:

```
OPT.fixationfile = 'test-data/test-data.xlsx';
```

The image stimuli are stored in a directory, and we assume all the images have the same size. The following code specifies both:

```
OPT.imgdir       = 'test-data/';  
OPT.imgsize      = [1280; 768];
```

The valid box region of each image is stored in another Excel file, which has the following columns: StimuliID, Xlo, Xhi, Ylo, Yhi, which correspond to the stimuli name, and the lower

and upper X and Y coordinates of the valid box. During processing, all fixations outside of the valid box region will be ignored.

```
OPT.imginfo      = 'test-data/test-info.xlsx';
```

Finally, we need to specify the output directory for storing the results, including visualizations and saved models.

```
OPT.outdir       = 'test-data-results/';
```

The script uses the “read_xls_fixations2” function to read in the fixation data. It then removes fixations that are outside the ROI boxes (outside the valid stimuli). Finally, it sorts the stimuli by name. The output of the script while reading the data looks like this:

```
Reading test-data/test-data.xlsx
- found SubjectID in column 1
- found TrialID in column 2
- found FixX in column 4
- found FixY in column 5
- found StimuliID in column 3
- found 10 subjects:
Subject1 Subject2 Subject3 Subject4 Subject5 Subject6 Subject7 Subject8 Subject9
Subject10
  * subject 1 had 50 trials
  ...
  * subject 10 had 50 trials
- found 0 empty rows in the Excel file
Preprocessing fixations using test-data/test-info.xlsx:
- Subject 4 Trial 6: removing 1 fixations (19 remaining)
...
- Subject 10 Trial 50: removing 1 fixations (19 remaining)
Sorting for co-clustering:
NOTE: stimuli order is changed
- Found 5 stimuli: Image1.jpg Image2.jpg Image3.jpg Image4.jpg Image5.jpg
  * subject 1 trials:  10  10  10  10  10
  * subject 2 trials:  10  10  10  10  10
  * subject 3 trials:  10  10  10  10  10
  * subject 4 trials:  10  10  10  10  10
  * subject 5 trials:  10  10  10  10  10
  * subject 6 trials:  10  10  10  10  10
  * subject 7 trials:  10  10  10  10  10
  * subject 8 trials:  10  10  10  10  10
  * subject 9 trials:  10  10  10  10  10
  * subject 10 trials: 10  10  10  10  10
```

It shows how many fixations are read, how many are removed due to being outside of the valid box region, and how many trials are read for each image and subject. *Note that it is okay for some subjects to not have trials on some images.*

Finally, if you want to see the fixations plotted on each image, you can set the following option: `OPT.SAVE_FIXATION_PLOTS = 1`. Visualizations of the fixations will then be saved in the directory “1_fixations_stimuli” in the specified output directory.

6.2 Estimating Individual HMMs

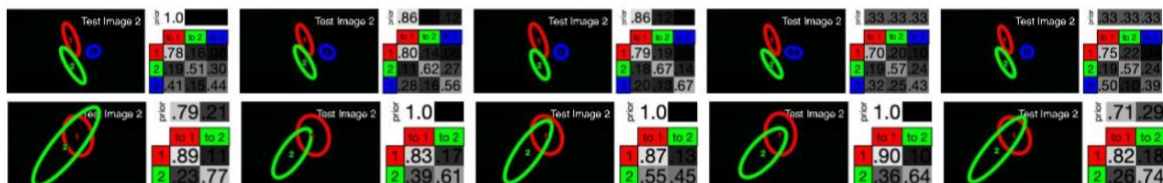
Next, the script estimates the individual HMMs for each subject and stimuli using `vbhmm_learn_batch` (as in the previous demos). The parameters are specified through the `OPT` structure:

```
% individual HMM settings (set vbopt like this)
OPT.S      = 2:4; % number of ROIs to try
OPT.vbopt.numtrials = 50; % set vbopt fields like this
```

Here we set the number of ROIs (S), and the number of VB trials (vbopt.numtrials). Other vbopt options can be set in a similar way.

Visualizations of the individual HMMs for each stimulus are saved into the sub-directory “2_indhmms_stimuli” of the output directory. The HMMs are saved in a mat file “individual_hmms.mat” in the output directory. Here is an example visualization of the individual HMMs for the 2nd test image in the demo.

2nd test image



3rd test image



6.3 Co-Clustering HMMs

Next, the script clusters the individual HMMs using co-clustering. In the demo, the options are set like this.

```
OPT.HEM_K = 2;
OPT.HEM_S = [];
OPT.SWAP_CLUSTERS = [];
OPT.vb_coccluster = 0;
```

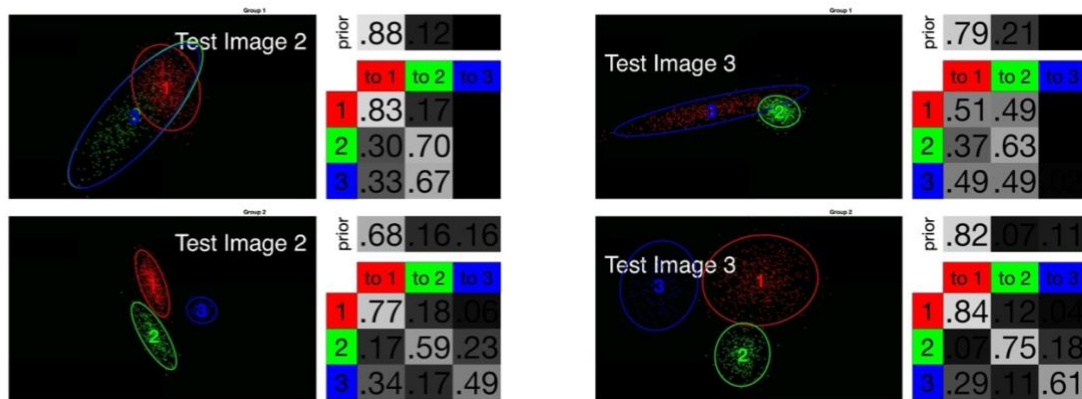
The HEM_K option sets the number of groups to 2. The HEM_S option is the number of ROIs to use in the representative HMMs. Here HEM_S=[] means that the numbers of ROIs for the representative HMMs are set as the median number of ROIs in the individual HMMs for each stimuli. *Note that each stimuli can use different S for their representative HMMs.* The vb_coccluster=0 option indicates that we will be using standard co-clustering. Finally, the SWAP_CLUSTERS can be set to swap the cluster IDs at the end, if you want a particular cluster to be the first one.

The function `vhem_coccluster` performs the co-clustering, and the models are saved in “cogroup_hmms.mat” in the output directory. Visualizations of the representative HMMs are saved into various subdirectories of the output directory:

- “3_grphmms_all/” – all representative HMMs combined into several figures.
- “4_grphmms_stimuli/” – the representative HMM for each stimuli.
- “5_grphmms_fixations/” – the representative HMMs and fixations for each stimuli.
- “6_grphmms_cluster/” – the representative HMMs and cluster members (individual HMMs) for each stimuli.

In the sub-directories “4_grphmms_stimuli” and “5_grphmms_fixations”, the images are sorted by the difference between the representative HMMs (via KL divergence), with largest

differences first. This can be turned off by setting “`OPT.SORT_BY_DIFF = 0;`” Here are the learned group HMMs for the 2nd and 3rd images:



Note that visualizing the representative group HMMs and the cluster member HMMs (the images in “6_grphmms_cluster”) takes a while to process. This visualization can be skipped by setting “`OPT.SAVE_GROUP_MEMBERS_PLOTS = 0;`”.

6.4 Analysis of HMMs

After estimating the co-clustering group HMMs, similar analysis can be performed as in other demos. Specifically, for each stimuli, we compute the log-likelihood of an individual’s data under the representative HMMs for that stimuli:

```
for i=1:Nsubjects
    for j=1:Nstimuli
        for k=1:HEM_K
            if ~isempty(alldataC{j,i})
                % the avg LL for each subject of each stimuli under each model
                LL(i,j,k) = mean(vbhmm_ll(cg_hmms{j}.hmms{k}, alldataC{j,i}));
            else
                LL(i,j,k) = 0;
            end
        end
    end
end
```

Then we compute the AB scale value for each individual, which quantifies how similar the individual’s eye fixations are to the group HMMs.

```
% (assumes K=2)
% LL of subject under model 1 and 2
LL1 = sum(LL(:, :, 1), 2); % sum over stimuli
LL2 = sum(LL(:, :, 2), 2);

% compute AB scale
% since some stimuli are missing, then normalize
AB = (LL1 - LL2) ./ (abs(LL1) + abs(LL2));
```

Here are the computed AB values in the demo:


```
AB =  
-0.2545  
-0.2533  
-0.2519  
-0.2717  
-0.2305  
0.2461  
0.2343  
0.2376  
0.2464  
0.2520
```

The AB values for the first 5 subjects are negative, and thus those subjects are more similar to Group 2. Likewise, the AB values for the last 5 subjects are positive, so they are more similar to Group 1.

The script also runs a statistical test to check if the group models are significantly different, similar to “demo_faces.m”, except that now we calculate the average log-likelihood of each subject over all the stimuli, under the Group 1 and Group 2 models. Then, a paired t-test is run on the two lists of log-likelihoods to see if the average log-likelihood difference is significantly different from 0. A similar test is also run using the fixations from Group 2. The result is shown here.

```
=== group difference (overall) ===  
modell1 vs modell2 using grp1 data: p=5.702e-07, t(4)=56.93, cohen_d=25.46  
modell1 vs modell2 using grp2 data: p=9.738e-06, t(4)=27.96, cohen_d=12.5
```

Finally, for each stimuli, we test whether the Group 1 and Group 2 representative HMMs are different, and show the number of stimuli with significantly different groups.

```
=== group test (each stimuli, symmetric KL) =====  
t-test for symmetric KLD  
comparison between group models:  
  number of sig diff models (combined): 5/5  
saving SKLD stats: test-data-results/skld_stats
```

6.5 Co-clustering with VBHEM to automatically select the number of clusters

We can also run co-clustering with VBHEM to automatically select the number of clusters and states. Change the following line: `OPT.datasource = 1;` This will select a different set of experiment parameters that runs VBHEM co-clustering. Specifically, the parameters are now:

```
OPT.HEM_K = 1:5;  
OPT.HEM_S = [];  
OPT.SWAP_CLUSTERS = [];  
OPT.vb_cocluster = 1; % use vb co-clustering  
OPT.vbhemopt.numtrials = 50;
```

Here we specify a range of values for the number of clusters K to try. We also set `vb_cocluster` to 1, which tells the script to run VB co-clustering. Finally, we can set the options for `vbhem_cocluster` using the `OPT.vbhemopt` structure.

The function to perform VB co-clustering is `vbhem_cocluster`. It will try different values of K and then selects the one that yields the highest model log-likelihood. The group HMMs are saved into the file “`vbcogroup_hmms.mat`”. Here is the output of the script where it selects the best model.

```
best model: K=3; L=-113921
Hyperparameters:
{alpha0=0.06897; eta0=0.4447; epsilon0=0.4928; v0=3.416; lambda0=0.08744;
W0=0.0002876; m0=[639;383.2]; }
Pruning model:
removing hmms: 1
1: removed states 3;
removing hmms: 1
1: removed states 3;
removing hmms: 1
1: removed states 3;
removing hmms: 1
2: removed states 3;
removing hmms: 1
after pruning: K=2; S=[2 2 2 3 2;3 3 3 2 2]
```

Note that originally it selects K=3, but 1 of the clusters is empty. Thus, it is automatically removed, and the final model has K=2 clusters. Also note that some states (ROIs) were pruned.

The visualization of the group HMMs and the analysis of group HMMs are the same as the above co-clustering version.

6.6 Other experiments

The same demo_cocluster script can be used for running three additional experiments. These are selected by changing the value of `OPT.datasource` appropriately.

- Full VB co-clustering experiment (`OPT.datasource=10`) – this will run a full VB co-clustering experiment on the data from the BRM journal paper. There are 120 stimuli and 61 subjects. Computing the individual HMMs takes about 3.5 hours on a desktop PC, and running VB co-clustering takes about 1.2 hours.
- Full co-clustering experiment (`OPT.datasource=11`) – this will run a full co-clustering experiment on the data from the BRM journal paper. It will take about 3.5 hours on a desktop PC to compute the individual HMMs, and a few minutes to perform co-clustering.
- Replicate results from BRM journal paper (`OPT.datasource=12`) – this will run VB co-clustering on a set of pre-computed individual HMMs computed for the BRM journal paper.

Finally, the script “brm_cocluster_analyze.m” will run the analysis (statistical tests and regression analysis) from the BRM journal paper on the results from experiment “10” or “12”.

6.7 Other options

Here are some other useful options in the co-clustering script:

- `OPT.DEBUG_MODE` – setting this to 1 will set the script into debugging mode, where only the first 10 subjects and first 5 stimuli are used for testing purposes. Only 5 trials of VBHMM or co-clustering are used. This option is used for testing that the code can properly read in the files before running the main experiment (with `DEBUG_MODE` set to 0).
- `do_indhmms` – Setting this to 1 will make the script compute the individual HMMs and save them. Setting this to 0 will make the script load the individual HMMs from

the previous run. This option is useful for saving time when the clustering results need to be recomputed.

- `do_cocluster` – Similarly, setting this to 1 will make the script compute the co-cluster HMMs, while setting it to 0 will make the script reload the previous run. This is useful for saving time when the HMM analysis needs to be recomputed.

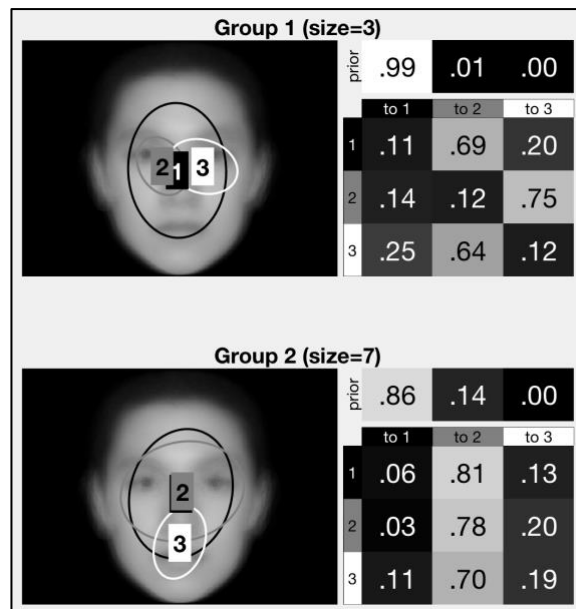
7 Frequently Asked Questions (FAQ)

7.1 How do I make grayscale figures for printing?

By default EMHMM will produce color figures. Grayscale figures (e.g., for printing) can be created by toggling the color mode:

```
>> emhmm_toggle_color
** emhmm is set to grayscale figures
```

EMHMM will now generate ROIs using different gray levels. Here is an example figure generated from `demo_faces`:

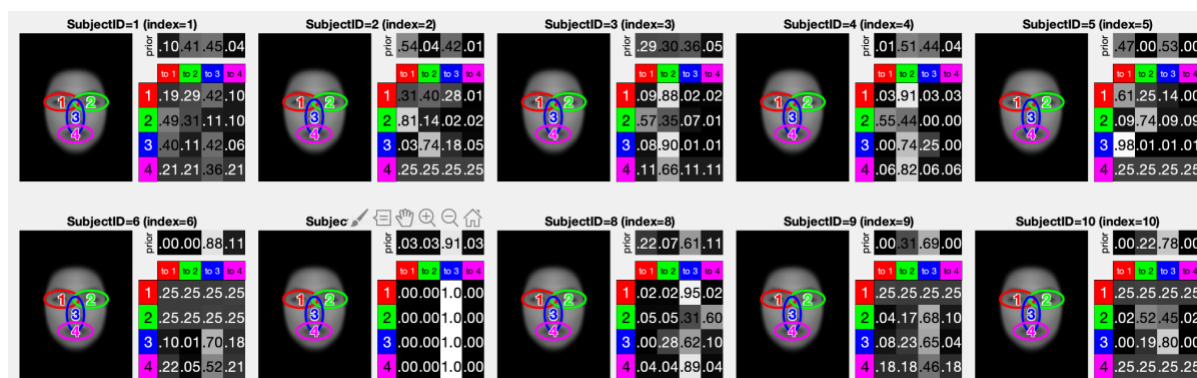


7.2 How do I use pre-specified ROIs?

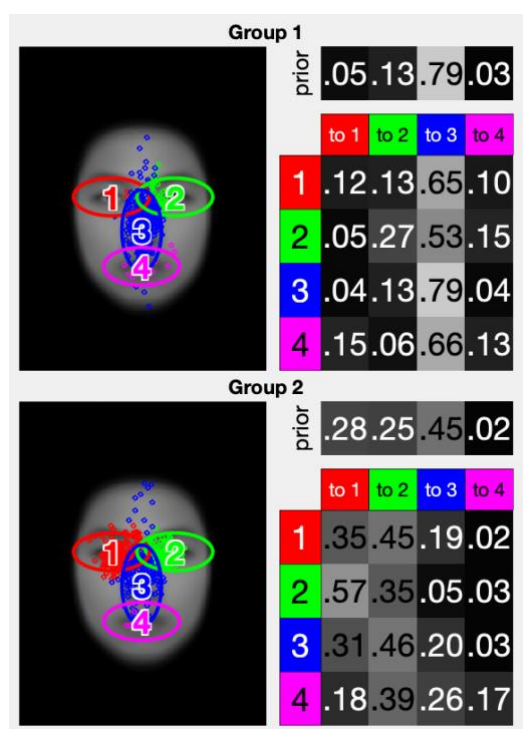
Normally the ROIs are learned from the data for each individual. The toolbox also supports using pre-specified ROIs if it is required for your analysis. In “`demo_faces.m`”, fixed ROIs can be specified by uncommenting the below lines:

```
>> vbopt.fixed_rois = {[120,195,80,40], [202,195,80,40], ...
                      [159,237,40,80], [159,285,80,40]};
>> K = length(vbopt.fixed_rois);
```

The ROIs are specified as a cell array with each entry as `[x,y,w,h]`, which correspond to the center location (x,y) and extent (width, height) of the ROI. The width and height correspond to 2 standard deviations of the Gaussian ROI. Here are examples of the individual HMMs learned using fixed ROIs:



Here is the clustering result using VHEM, also with fixed ROIs:



Note that the clustering result are the same as when the ROIs are learned from the data.

7.3 How to get the sequence of ROIs visited?

The most likely ROI state sequence can be inferred from a fixation sequence using the Viterbi algorithm on the HMM. For the individual HMMs, you can use the `vbhmm_map_state` function. For example, using the `demo_faces` data:

```
>> rseq = vbhmm_map_state(hmms{1}, data{1})
>> rseq{1}
ans =
    1    1    2
```

The variable `rseq` is a cell array containing the ROI sequence for each fixation sequence in `data{1}` with respect to `hmms{1}`. In this example the ROI sequence for the first fixation sequence is 1-1-2.

The `vhem_plot_fixation` can also return the ROI sequences for each subject's data according to the most likely group HMM. As an example,

```
>> [hfig, rseq] = vhem_plot_fixations(data, group_hmms2, faceimg);
```

The variable `rseq` is a cell array, where the i -th entry contains the list of ROI sequences for the most likely group for the i -th subject, i.e., `group_hmms.label(i)`.

7.4 How to run a statistical test using symmetric KL divergence?

The following line will run a t-test for the symmetric KL divergence between the two group HMMs from “demo_faces.m”.

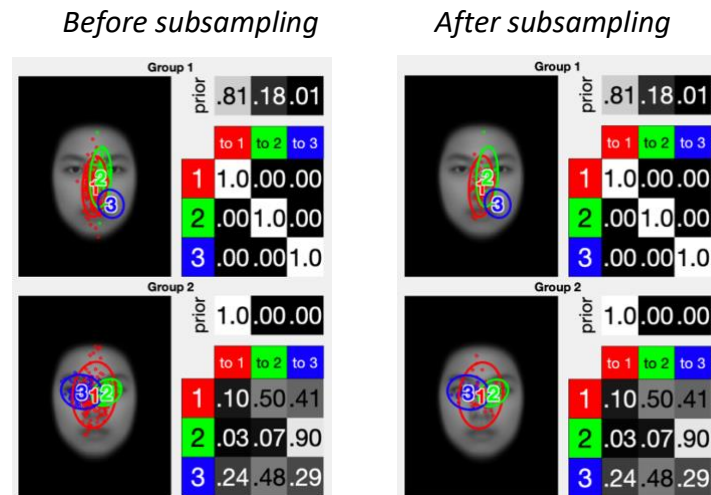
```
>> [p,info, lld] = stats_ttest_skl(group_hmms2.hmms{1}, group_hmms2.hmms{2}, data1, data2)
```

7.5 There are too many fixations on my plot. How do I make it look better?

When using `vhem_plot_fixations`, the fixation data can be subsampled to make the figure easier to read. Here is an example:

```
>> vhem_plot_fixations(data, group_hmms2, faceimg, 'c', 5)
```

The last argument “5” tells the function to only plot one-fifth of the fixation data.

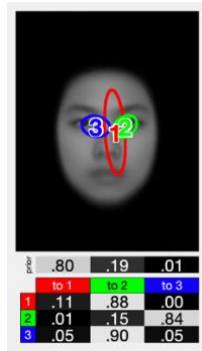


7.6 How to change the location of the transition matrix in the plots?

The transition matrix can be placed below the image in the HMM plots by using the ‘b’ option. Here are some examples:

```
>> figure, vbhmm_plot_compact(hmms{mys}, faceimg, 'b');
>> vhem_plot_fixations(data, group_hmms2, faceimg, 'b');
>> vhem_plot(group_hmms2, faceimg, 'b');
```

Here are example figure using `vbhmm_plot_compact` with the ‘b’ option:

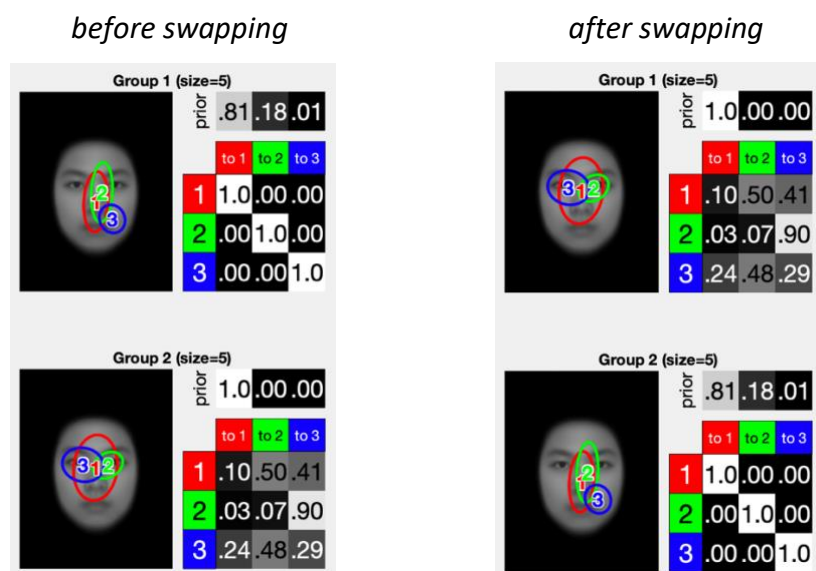


7.7 How to change the order of the clusters from VHEM?

You can swap the order of the HMM clusters produced by VHEM using the `vhem_permute_clusters` function. Here is an example that swaps the two clusters in `demo_faces`:

```
>> swap_group_hmms = vhem_permute_clusters(group_hmms2, [2 1]);
```

Here are the example figures before and after swapping:



7.8 How to change the order of the ROIs in an HMM?

You can change the order of the ROIs in an HMM using the `vbhmm_standardize` function:

```
>> hmm_new = vbhmm_standardize(hmm, 'f');
```

Here the 'f' means to re-order the ROIs so that the mostly likely first fixation is ROI1, the most likely 2nd fixation is ROI2, etc. Other options can be used, such as:

- 'e' – sort by overall number of fixations in the ROI
- 'p' – sort by the probability of the first fixation in the ROI.
- 'l' – sort ROIs by location, from left to right.
- 'r' – sort ROIs by location, from right to left.

You can also use the `vbhmm_permute` function to specify the reordering of the ROIs of an HMM.

```
>> hmm_new = vbhmm_permute(hmm, [3 1 2]);
```

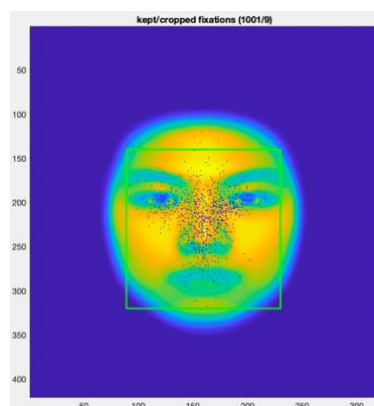
This will map ROI-3 in the old HMM to ROI-1 in the new HMM, ROI-1 to ROI-2, and ROI-2 to ROI-3.

7.9 How do I remove fixations not on the stimuli in the image?

When reading in data from an Excel file, the `read_xls_fixations` function can ignore all fixations that are outside of some specified regions. This is helpful for removing some fixations that are not on the stimuli (e.g., face). To set a rectangular region, you specify the top-left and bottom-right coordinates of the rectangle inside a cell array:

```
>> [data, SubjNames, TrialNames] = ...  
    read_xls_fixations(xlsname, '', {'r', [89,140,230,320]}, faceimg);  
...  
- ignored 9 fixations outside cropfix  
...
```

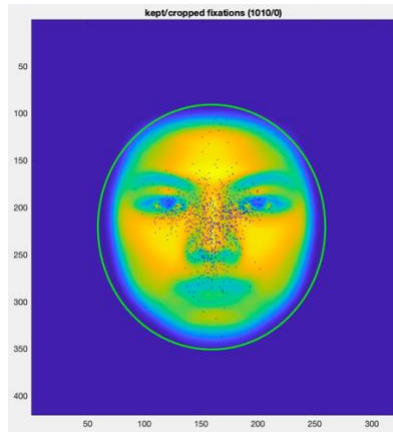
In this example the top-left coordinates are (89,140) and the bottom-right ones are (230,32). The 'r' indicates that this is a rectangular region. The function will plot the region and fixations for verification after reading in the data:



An elliptical region is specified by center coordinates and the width and height. For example, if the elliptical region is centered at (159,221) and has width 200 and height 260:

```
>> [data, SubjNames, TrialNames] = ...  
    read_xls_fixations(xlsname, '', {'e', [159,221,200,260]}, faceimg);
```

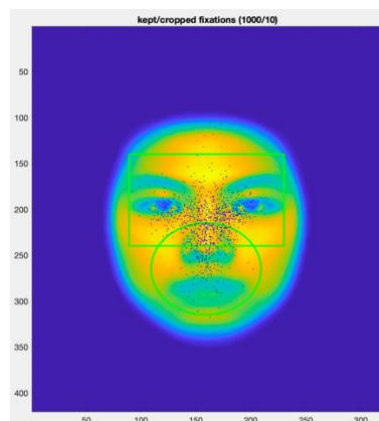
The 'e' indicates this is an elliptical region. The verification figure looks like this:



Finally, multiple regions can be specified by concatenating extending the cell array. For example, this will create a rectangular and an elliptical region, and fixations not inside at least one region will be ignored.

```
>> [data, SubjNames, TrialNames] = ...
    read_xls_fixations(xlsname, '', ...
        {'r', [89,140,230,240], 'e', [159,265,100,100]}, faceimg);
...
- ignored 10 fixations outside cropfix
...
```

Here is the figure:

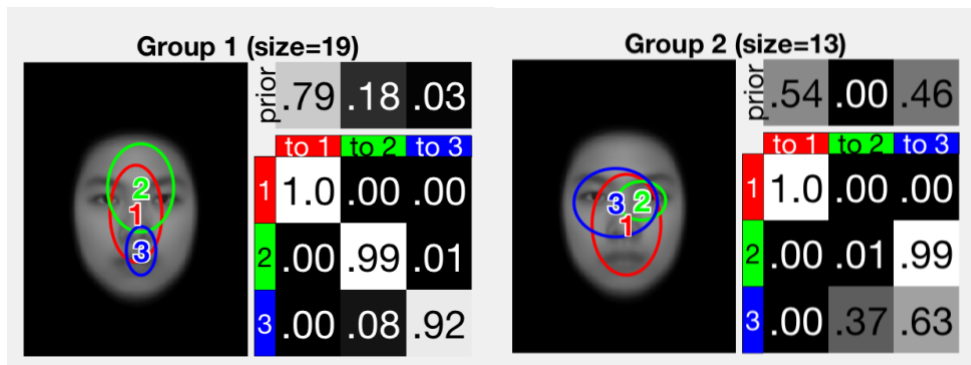


7.10 How to convert coordinates between face images of different datasets?

The learned HMMs depend on the coordinate system of the images used to collect the data. For example the location of the eyes, mouth, and nose are particular to one dataset, and could change for another dataset. The demo `demo_conversion` shows how to convert from the coordinates of one face image to another face image, so that the HMMs can be used to analyze data collected from other image sets.

IMPORTANT NOTE: normalization of new data to an old model is only valid when the presentation sizes (visual angle) of the old and new data are the same.

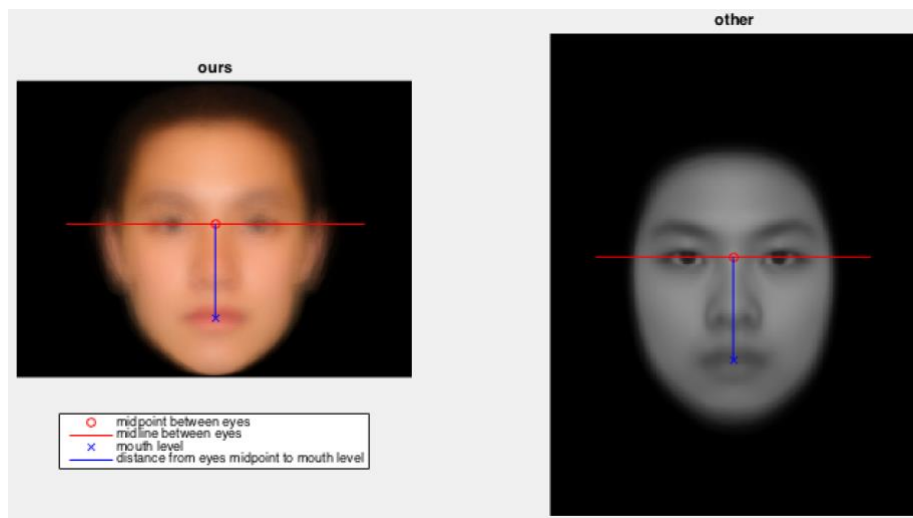
The group HMMs learned from the JOV data look like this:



However, our new data is collected from images with a different size:



To convert from our data to the JOV data, we first define two things on the images: 1) the coordinates of the midpoint between the eyes (called C); 2) the distance between the eyes midpoint and the mouth level (called D). Here is a picture showing these points:



For these images, we have:

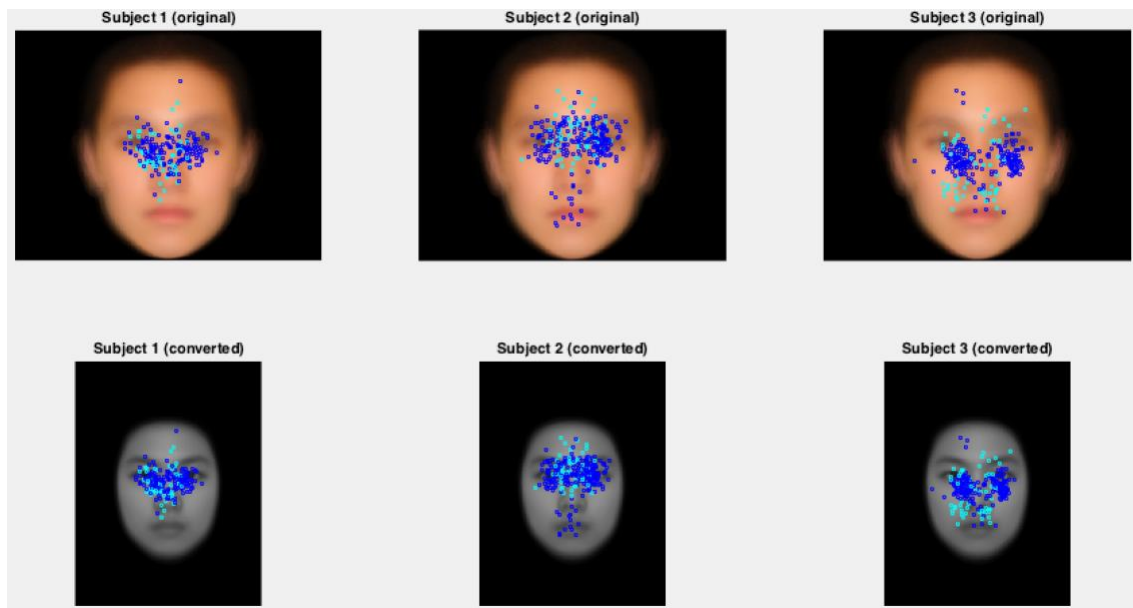
```
% for our images
our_C = [258, 186]; % midpoint between eyes
our_D = 121;       % distance from eyes midpoint to mouth level

% for the other images from JOV:
other_C = [160, 195]; % midpoint between eyes
other_D = 90;         % distance from eyes midpoint to mouth level
```

Using this information, we convert the coordinates from our images to the other images:

```
>> data_s = convert_face_coord(data, our_C, our_D, other_C, other_D);
```

The following figure shows the fixations using the original coordinates and the converted coordinates on their respective images:



Finally, we use the converted data and compute the log-likelihoods with the JOV models:

```
>> [mLL1] = stats_meanll(model.group_hmms2.hmms{1}, data_s);
>> [mLL2] = stats_meanll(model.group_hmms2.hmms{2}, data_s);
```

Here is the result from the demo script:

```
- mean LL for each subject under hmml:

mLL1 =

    -9.4891    -9.8699   -10.2966   -10.3456   -11.2027    -9.5222    ...
    -9.7196    -9.6563   -10.2170    -9.6101

- mean LL for each subject under hmm2:

mLL2 =

    -8.9928    -9.6833    -9.5378    -9.9843   -10.9729    -9.2440    ...
    -9.5662    -9.1458    -9.8409    -9.2563
```

8 Function Usage

In this section, we provide more details about the main function calls and their input parameters. More detailed help can be found in MATLAB for each function by using the help command on the function (e.g., `help vbhmm_learn`).

Options are passed to the main functions, `vbhmm_learn`, `vbhmm_learn_batch`, and `vhem_cluster`, using a structure where each field represents one parameter setting. Values are not specified in the structure will be automatically filled-in using default values.

8.1 [vbhmm_learn](#)

estimate a single HMM from eye fixation data.

Usage:

```
[hmm, L] = vbhmm_learn(data, K, vbopt)
```

INPUTS	Description
data	<p>Nx1 cell array, where each element is a fixation sequence and N is the number of sequences.</p> <p>data{i} is a TxD matrix, where T is the sequence length, and D is the dimension of a single fixation point (typically 2 for x and y coordinates).</p>
K	The number of hidden states (ROIs) to use. If given a vector, the K will be automatically selected among the entries in the vector.
vbopt	structure containing options (see below)
vbopt.alpha0	<p>The Dirichlet distribution concentration parameter for the prior distribution of the initial fixation. Large values encourage a uniform prior, while small values encourage a concentrated prior (default=0.1). It should be a positive number.</p> <p>Another way to think of it is in terms of "virtual" samples. A typical way to estimate the probability of something is to count the number of samples that it occurs and then divide by the total number of samples, i.e. $P = (\# \text{ times it occurred}) / (\# \text{ samples})$. The alpha parameter of the Dirichlet adds a "virtual" sample to this estimate, so that the probability estimated is $P = (\# \text{ times it occurred} + \alpha) / (\# \text{ samples})$. Hence, for small alpha, the probability estimate will just follow the data, while for large alpha, it forces all the probabilities to be very similar (i.e., uniform).</p>
vbopt.epsilon0	The concentration parameter for the Dirichlet distribution on the rows of the transition matrix (default=0.1). The meaning is similar to alpha above, but for the probabilities in the transition matrix.
vbopt.mu0	<p>prior mean of the ROIs (default = [256;192]). Typically, it should be at the center of the image. Alternatively, you can use the average fixation location.</p> <p>When fixation duration is included in the data, then the default is [256;192;250].</p>
vbopt.W0	Inverse variance of the inverse Wishart distribution (default=0.005). This parameter determines the variance (width) of the ROI prior. For example, $W=0.005$ means that the variance of the ROI prior is $1/0.005=200$. Hence, the ROI prior has a standard deviation of 14 pixels, i.e., the ROI ellipses have width of $14*4 = 56$ pixels.

	A vector can be specified, in which case the (x,y,d) widths of the ROIs can be different.
<code>vbopt.v0</code>	The degree-of-freedom of the inverse Wishart, $v > D-1$. (default=10). Larger values give preference to diagonal covariance matrices.
<code>vbopt.beta0</code>	The Wishart concentration parameter (default=1). Large values encourage estimated ROIs to be similar to the prior ROI (mean & W), while small value ignores the prior.
<code>vbopt.seed</code>	Seed for random number generator used when initializing the model. The seed is required for reproducible results.
<code>vbopt.learn_hyps</code>	estimate the hyperparameters (alpha0, epsilon0, etc) for each subject by maximizing the log-likelihood of the data. Use "1" to indicate all hyperparameters should be estimated.
<code>vbopt.numtrials</code>	The number of trials to run when using 'random' initialization (default=50).
<code>vbopt.showplot</code>	<ul style="list-style-type: none"> • 1 means show a plot of the HMM (default). • 0 means don't show a plot.
<code>vbopt.bgimage</code>	Filename of the background image for plotting.
<code>vbopt.verbose</code>	<ul style="list-style-type: none"> • 0 means don't show any messages. • 1 means show a few messages showing progress. (default) • 2 means show lots of messages.
<code>vbopt.sortclusters</code>	Sort the ROIs by: <ul style="list-style-type: none"> • 'f' = most likely fixation path (default) • 'p' = larger prior probabilities • 'e' = larger emission probabilities
<code>vbopt.fixed_rois</code>	Instead of learning the ROIs from data, use a fixed set of specified ROIs. The ROIs are specified by their mean (x,y) and size (width,height), corresponding to 2 standard deviations from the mean. The ROIs are listed in a cell array: {[x1, y1, width1, height1], ...}.

OUTPUTS	Description
<code>hmm</code>	Structure containing all the HMM parameters and other information. Use "help vbhmm_learn" for more details.
<code>L</code>	The log-likelihood of the data for the hmm.

8.2 [vbhmm learn batch](#)

Given a batch of data (data for several subjects), estimate an HMM for each subject. The options are largely the same as `vbhmm_learn`.

Usage:

```
[hmm_batch, Ls] = vbhmm_learn_batch(data_batch, K, vbopt)
```

INPUTS	Description
data_batch	Sx1 cell array, one cell for each subject. Each cell entry contains a cell array of N fixation sequences. I.e., the i-th trial of the s-th subject is: data{s}{i} = [TxD] matrix, where each row is a fixation (x,y) or (x,y,t). Duration t should be in milliseconds. Note: the sequence length T, and number of sequences N can be different for each subject/trial.
K	The number of hidden states (ROIs) to use. Same as vbhmm_learn.
vbopt	structure containing options. Most of the options are the same as vbhmm_learn, except the below ones.
vbopt.seed	Seed for random number generator used when initializing the model. The seed is required for reproducible results.
vbopt.learn_hyps	Estimate the hyperparameters (alpha0, epsilon0, etc) for each subject by maximizing the log-likelihood of the data. When used with vbhmm_learn_batch, each subject will have its own set of hyperparameters. This option assumes that each subject has a distinctive eye gaze strategy, and thus each subject should have its own set of hyperparameters.
vbopt.learn_hyps_batch	Estimate common hyperparameters for all subjects, i.e., all subjects share the same set of hyperparameters. Using this option assumes that all subjects have similar eye gaze strategies, and thus their HMMs are from the same distribution.

OUTPUTS	Description
hmm_batch	Structure containing all the HMM parameters and other information. Use “help vbhmm_learn” for more details.
Ls	Vector of data log-likelihood of the data for each hmm.

8.3 [vbhmm_set_hyperparam](#)

This is a helpful function to automatically set the hyperparameters mu and W for vbhmm_learn/vbhmm_learn_batch using the data.

Usage:

```
[vbopt] = vbhmm_set_hyperparam(vbopt, data, img, opt)
```

INPUTS	Description
<code>vbopt</code>	Existing <code>vbopt</code> structure
<code>data</code>	the same format as <code>vbhmm_learn</code> or <code>vbhmm_learn_batch</code> .
<code>img</code>	the template image (or filename)
<code>opt</code>	Option settings; ‘c’ = set μ as the center of the image, and W so that the prior ROI is 1/8 the width of the image. For duration, set the mean as 250ms and standard deviation as 25ms. ‘d’ = use a data-driven approach. Set the μ as the mean fixation location and duration. Set W according to the inverse variance of the data. Assumes that the ROI is circular in x-y dimensions.

OUTPUTS	Description
<code>vbopt</code>	New options structure with μ and W set.

8.4 [vhem_cluster](#)

Cluster HMMs into groups using VHEM and estimate representative HMMs for each group.

Usage:

```
[group_hmms] = vhem_cluster(hmms, K, S, hemopt)
```

INPUTS	Description
<code>hmms</code>	Nx1 cell array of HMMs, learned with <code>vbhmm_learn_batch</code> or <code>vbhmm_learn</code> .
<code>K</code>	The number of groups to cluster the data
<code>S</code>	The number of states (ROIs) in each group HMM. Use [] to automatically select the number of states as the median number of states in the input HMMs.
<code>hemopt</code>	structure containing options (see below)
<code>hemopt.tau</code>	virtual sequence length (default=10). It can be set as the median sequence length in the data. Use <code>"get_median_length(data)"</code> to get the median length.
<code>hemopt.trials</code>	number of trials with random initialization to run (default=100)
<code>hemopt.sortclusters</code>	Sort the ROIs (same meaning as <code>vbopt.sortclusters</code>).
<code>hemopt.seed</code>	Seed for random number generator used when initializing the model. The seed is required for reproducible results.
<code>hemopt.verbose</code>	<ul style="list-style-type: none"> 0 means don't show any messages.

	<ul style="list-style-type: none"> • 1 means show a few messages showing progress (default) • 2 means show lots of messages.
--	--

OUTPUTS	Description
group_hmms	A structure containing the group HMMs and other information. Use “help vhem_cluster” for more details.
group_hmms.hmms	A 1xK cell array, where each entry is a group representative HMM.
group_hmm.LogL	The log-likelihood score of the group HMMs.
group_hmm.label	A 1xN vector, where each entry is the cluster assignment for the i-th input HMM.
group_hmm.groups	A 1xK cell array, where each entry contains the group members for that group.

9 Toolbox Contents

The contents of the emhmm toolbox are listed below. Here only the important functions for the user are listed.

Directory/File name	Description
setup.m	Setup MATLAB path for the toolbox, and check for updates.
demo/	
demo_faces.m	Simple example for learning individual HMMs and clustering HMMs.
demo_faces_duration.m	Simple example using fixation location and duration.
demo_faces_jov_clustering.m	Another example of learning and clustering HMMs.
demo_faces_jov_compare.m	Example of comparing HMMs learned from correct and incorrect face recognition trials.
demo_face_entropy.m	Example of computing HMM entropy values.
demo_vb_faces.m	Example of selecting the number of groups automatically while clustering.
demo_vb_faces_duration.m	Example of selecting the number of groups automatically while clustering with fixation duration.
demo_conversion.m	Example showing conversion of fixation coordinates between two images.
demo_cocluster/	
demo_cocluster.m	Example for co-clustering HMMs learned over different stimuli
brm_cocluster_analyze.m	Analyze the co-clustering results from the BRM’2021 paper
models/	
demo_PBR_model.m	Example computing HA scale using representative models from the PBR paper.
src/hem	
vhem_cluster.m	Cluster HMMs with VHEM algorithm
vhem_plot.m	Visualize the group HMMs
vhem_plot_clusters.m	Visualize the group HMMs and cluster members
vhem_plot_fixations.m	Visualize the fixations belonging to group HMMs
vhem_permute_clusters.m	Change the order of the HMM groups.
src/hmm	
vbhmm_learn.m	Learn HMM from data
vbhmm_learn_batch.m	Learn a batch of HMMs from batch of data.
vbhmm_plot_compact.m	Visualize an HMM using a compact format

vbhmm_plot.m	Visualize an HMM
vbhmm_kld.m	Calculate the KL divergence between 2 HMMs
vbhmm_ll.m	Calculate the log-likelihood of data for an HMM
vbhmm_entropy.m	Calculate the overall entropy of an HMM
vbhmm_entropy_marginal.m	Calculate the marginal entropy of an HMM
vbhmm_entropy_conditional.m	Calculate the conditional entropy of an HMM
vbhmm_entropy_joint2.m	Calculate the pairwise joint entropy of an HMM
vbhmm_mutualinfo2.m	Calculate the pairwise mutual information of an HMM
vbhmm_set_hyperparam.m	Automatically set some hyperparameters using the data
vbhmm_remove_empty.m	Remove empty ROIs (those with < 1 fixation) in an HMM.
vbhmm_map_state.m	Compute the most probable state (ROI) sequence from fixation data
vbhmm_permute.m	Permute (re-order) the states of an HMM.
vbhmm_standardize.m	Standardize the order of states in an HMM.
src/cocluster	
vhem_cocluster.m	Co-clustering of HMMs with VHEM
src/vbcocluster	
vbhem_cocluster.m	Co-clustering of HMMs with VBHEM, which automatically selects the number of clusters.
src/vbhem	
vbhem_cluster.m	Clustering of HMMs with VBHEM, which automatically selects the number of clusters and number of states.
src/stats	
computeCohen_d.m	Compute Cohen's d (effect size)
stats_ttest.m	Run a t-test to compare log-likelihoods of two HMMs.
stats_meanll.m	Calculate mean log-likelihood of a subject's data
stats_stateseq.m	Calculate the probability of all state sequences of a particular length.
stats_ttest_skl.m	Run a t-test to compare two HMMs via symmetric KL divergence
src/util	
emhmm_version.m	Toolbox version number and history
emhmm_check_updates.m	Check if a new version of EMHMM is available
emhmm_mex_check.m	Check MEX files and compile & test if necessary.
emhmm_toggle_color.m	Toggle between grayscale and color figures.
get_median_length.m	Compute the median length of sequences in data.
convert_face_coord.m	Convert fixation data to a new image size.
read_xls_fixations.m	Read fixations from an Excel spreadsheet in standard format.
read_xls_fixations2.m	Read fixations from an Excel spreadsheet for co-clustering.