6.835 Project 4:
Multimodal Signal Fusion Using HMMs

Due: 6PM Wednesday, April 16, 2013

1 Introduction

The goal of this project is to explore various multimodal signal fusion techniques, using an HMM as the base learner. You will implement and compare four different fusion algorithms on an aircraft handling signals dataset containing two kinds of information: body joint information and hand shape information. As usual, the implementation will be in Matlab. Unlike previous mini projects, however, this mini project will take much longer to finish, mainly due to heavy computational requirements of the algorithms you will be using. It may not take much longer in terms of time to write the required code, but the testing and debugging might be much slower because of the longer run-times. So start now!

2 Getting Started

Download the packaged .zip file from Stellar. After extracting the file, start Matlab (you need version 7 or later) and load the dataset.

>> load NATOPS6.mat

The dataset contains six aircraft handling signals used in routine practice on the deck of an aircraft carrier, performed in a laboratory setting [4]. The six types of gestures are shown in Figure 1. The dataset includes upper body postures and the shapes of both hands. The body features are 3D angular velocities of four body joints – left/right elbows and wrists – represented as a 12-dimensional feature vector. The hand feature vector contains probability estimates of four predefined hand shapes – opened/closed palm, and thumb up/down, yielding an 8-dimensional feature vector.
Figure 1: Six aircraft handling signals from the NATOPS database [4] (#1: All Clear, #2: Not Clear, #3: Insert Chocks, #4: Remove Chocks, #5: Brakes On, #6: Brakes Off). Body movements are illustrated in yellow arrows and hand poses are illustrated with synthesized images of hands. Red rectangles indicate hand poses are important in distinguishing gesture pairs (#1-#2 and #5-#6).

Once you load the dataset, you will see a struct variable D that contains two cell-array type variables, `seqs` and `labels`; the `seqs` contain 2,400 gesture sequences and the `labels` contain corresponding labels. As detailed in [4], the NATOPS dataset was collected from 20 participants performing each gesture 20 times (400 samples per gesture). The 2,400 samples (6 gesture types x 400 samples per gesture) are compiled in the order of subject-gesture-trial, that is, the first 120 samples are from the first subject, and the first 20 samples of the first 120 samples are of gesture type #1 (all clear).

Each sample in `seqs` is a 20-by-$T_i$ matrix, where $T_i$ is the length of the $i$-th gesture sequence (i.e., each column in the matrix represents observation at each time frame). The table below lists the 20-dimensional features.
<table>
<thead>
<tr>
<th>Features</th>
<th>Joint Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>x(1:3,:)</td>
<td>left elbow angular velocity (x,y,z)</td>
</tr>
<tr>
<td>x(4:6,:)</td>
<td>left wrist angular velocity (x,y,z)</td>
</tr>
<tr>
<td>x(7:9,:)</td>
<td>right elbow angular velocity (x,y,z)</td>
</tr>
<tr>
<td>x(10:12,:)</td>
<td>right wrist angular velocity (x,y,z)</td>
</tr>
<tr>
<td>x(13,:)</td>
<td>p(opened palm</td>
</tr>
<tr>
<td>x(14,:)</td>
<td>p(closed palm</td>
</tr>
<tr>
<td>x(15,:)</td>
<td>p(thumb up</td>
</tr>
<tr>
<td>x(16,:)</td>
<td>p(thumb down</td>
</tr>
<tr>
<td>x(17,:)</td>
<td>p(opened palm</td>
</tr>
<tr>
<td>x(18,:)</td>
<td>p(closed palm</td>
</tr>
<tr>
<td>x(19,:)</td>
<td>p(thumb up</td>
</tr>
<tr>
<td>x(20,:)</td>
<td>p(thumb down</td>
</tr>
</tbody>
</table>

We have provided a Matlab script run.m that will run through all the experiments you will perform in this mini project. You will experiment with four different fusion algorithms using the functions below:

- Early fusion HMM, experiment_early_hmm.m
- Late fusion HMM, experiment_late_hmm.m
- Coupled HMM, experiment_coupled_hmm.m
- Co-training HMM, experiment_cotrain_hmm.m

The above functions will train and test each fusion algorithm, using these functions:

- trainHMM.m
- testHMM.m
- testLateHMM.m
- trainCHMM.m
- testCHMM.m
- trainCoHMM.m

Four utility functions are provided:

- split_data.m divides a dataset into four splits: train, validate, test, and unlabeled. It takes as an input labels, a 1-by-N cell array, that contain labels of N samples, and ratio, a 1-by-4 array, that specifies percentages of each split. It produces 4 arrays of indices, where the indices indicate elements from the dataset.
• **get_best_model.m** finds the best performing model based on the classification accuracy on the validation split. It takes as an input \( R \), a 1-by-\( K \) cell array that contains experimental results, and returns the index of the best performing model.

• **build_confmat.m** builds a confusion matrix. It takes as an input `prediction` and `groundtruth`, each of which is a 1-by-\( N \) array, and returns an \( N \)-by-\( N \) confusion matrix.

• **plot_confmat.m** prints a confusion matrix. It takes as an input `confmat`, an \( N \)-by-\( N \) matrix, obtained from `build_confmat`. Values in the diagonal of the plotted confusion matrix show per-class classification accuracy. Off-diagonal values show the percentage of times one label was confused with another.

We have also provided you with three files (`fwdback.m`, `learn_params_dbn_em.m`, and `mixgauss_prob.m`) that contain bug fixes of the FullBNT package. These will automatically override existing BNT functions as long as they are in the same directory as you are working in. In other words, you don’t need to overwrite the existing files with the provided files; just keep them in your working directory.

### 3 Unimodal Gesture Recognition

In this first part of the project, we will experiment with a standard HMM using body or hand features alone. Follow the steps in `run.m` (Part 1a and 1b).

To use `run.m`, copy sections from it and paste them into Matlab for execution. Start by copying and pasting the initializations: all the lines above “Part 1a”. This sets a number of parameters. Then, to do Step 1a, copy the lines between Step 1a and Step 1b, paste into Matlab command line environment, etc.

**Question 1** [2 points]: From the experiments using body features (Part 1a) and hand features (Part 1b), what are the best classification accuracies you obtained? What parameter values did you validate and what was your strategy for finding the best parameter value?

**Question 2** [4 points]: For each experiment, the `run.m` script will automatically generate a confusion matrix for the best performing model. Submit and interpret the two confusion matrices you obtained: For each modality explain which pairs of gestures are confused the most and *why* you think they were. See the gesture pairs in Figure 1 and use them in explaining the *why*. 
4 Early versus Late Fusion HMM

We will compare two simple modality fusion algorithms, early fusion and late fusion [1]. Follow the steps in run.m (Part 2a and 2b).

Question 3 [2 points]: Follow the steps in Part 2a. What is the best classification accuracy you obtained using early fusion HMM? What parameter values did you try and what was your strategy for finding the best parameter value?

Question 4 [8 points]: Implement testLateHMM.m to perform late fusion, and explain your implementation with pseudocode in your writeup. Note that you must follow the type signature of the function, as the function’s input and output parameters are used in the function experimentLateHMM.m.

Implementation detail: The input variables to the function are seqs, a 1-by-N cell array of test samples, labels, a 1-by-N cell array of test labels, hmm, a 1-by-2 cell array of trained HMMs, each of which is a set of HMMs from each modality, featureMap, a 1-by-2 cell array of indices defining each modality, and weightsMV, a 1-by-K cell array of modality weights we want to try out (these weights determine the relative weight given to each modality during fusion). To implement the function, use hmm{1} (and hmm{2}) to obtain estimates of log-likelihoods of all samples using only body features (and only hand features). Then, for each weight value in weightsMV, compute the weighted average of the two log-likelihoods, obtain predicted labels by taking the max of log-likelihoods of each sample, and finally compute the accuracy of the estimates. Your return variable stat must contain three field values, Ystar, a 1-by-N predicted labels, Ytrue, a 1-by-N ground truth labels, and accuracy the classification accuracy value.

Question 5 [2 points]: Follow the steps in Part 2b. What is the best classification accuracy you can get using late fusion HMM? What parameter values did you validate and what was your strategy for finding the best parameter value? Which weight value tends to give you the best performance in terms of the classification accuracy? Why do you think that weight value give the best result?

Question 6 [4 points]: Describe the differences between early and late fusion algorithms in terms of the underlying assumptions, how the classifiers are trained and then used to test new samples.

Question 7 [4 points]: Submit and interpret the two confusion matrices you obtained (both from early fusion and late fusion). Which approach (early versus late) performed better? Pick the confusion matrix that performed better and compare it to the two confusion matrices you obtained in Question 2. What differences do you see? Do you see a better classification accuracy on those gesture pairs that were confused the most in unimodal approach? Why do you think the performance has improved?
Coupled HMM for Mid-Level Fusion

The previous section compared two simple modality fusion approaches: early and late fusion. In this section you will implement and try out more sophisticated model, a coupled HMM [2], that performs mid-level fusion. Figure 2 illustrates the difference between the standard HMM and a coupled HMM. To start, follow the steps in run.m (Part 3).

![Figure 2: Graphical representation of the standard HMM (left) and a coupled HMM (right).](image)

**Question 8 [10 points]:** Implement the function `chmm = make_chmm(N,Q,X)` (located inside `trainCHMM.m`) that generates the graph structure of a coupled HMM. Explain your implementation with pseudocode in your writeup. Note that you must follow the type signature of the function, as the function’s input and output parameters are used in the function `trainCHMM.m`.

**Hint 1:** The FullBNT package provides documentation (located under ./bnt/docs). See the documentation `usage.dbn.html` to learn how to construct a graph structure, and type `help mk.dbn` in Matlab command line to learn how to make a Dynamic Bayesian Network using that function.

**Hint 2:** For debugging purpose, we have provided you with a file `chmm_2_3_3.mat` that contains a variable named `chmm`, the output of the function with parameter values `chmm = make_chmm(2,[3,3],[12,8])`. Your output must be the same as the provided variable when using the same parameter values.

**Question 9 [4 points]:** Follow the step in Part 3 and run an experiment using coupled HMM. What is the best classification accuracy you obtained using coupled HMM? What parameter values did you validate and what was your strategy for finding the best parameter value? 

---

1You may notice that it takes so much longer to train a coupled HMM than a standard HMM. This is mainly because the graph structure of a coupled HMM contain many loops, and
Question 10 [2 points]: Submit and interpret the confusion matrix you obtained, comparing to the confusion matrices you obtained so far. Does coupled HMM tend to perform better than the early/late fusion HMMs? Why do you think it did (or did not)?

Question 11 [4 points]: Describe the differences between coupled HMM and early/late fusion HMMs in terms of the underlying assumptions, how the classifiers are trained and then used to test new samples.

6 Co-training HMM

Question 12 [6 points]: What are the two assumptions that a co-training algorithm makes? In the context of the NATOPS dataset, do those assumptions make sense?

Question 13 [16 points]: Implement trainCoHMM.m that performs co-training of HMMs. We have provided you with pseudocode for implementing co-training algorithms (Figure 3). Note that you must follow the type signature of the function, as the function’s input and output parameters are used in the function experiment_cotrain_hmm.m.

Given:
- a set $L$ of labeled training examples
- a set $U$ of unlabeled examples

Create a pool $U'$ of examples by choosing $n$ examples at random from $U$

Loop for $k$ iterations:
- Use $L$ to train a classifier $h_1$ that considers only the $x_1$ portion of $x$
- Use $L$ to train a classifier $h_2$ that considers only the $x_2$ portion of $x$
- Allow $h_1$ to label $p$ positive and $n$ negative examples from $U'$
- Allow $h_2$ to label $p$ positive and $n$ negative examples from $U'$
- Add these self-labeled examples to $L$
- Randomly choose $2p + 2n$ examples from $U$ to replenish $U'$

Figure 3: Pseudocode of a co-training algorithm.

Implementation detail: The input variables to the function are `seqs`, a 1-by-N cell array of training samples, `labels`, a 1-by-N cell array of labels, `useqs`, a
1-by-M cell array of unlabeled samples, and **params** containing training parameters. There are three parameters you need during co-training: **params.maxiter.cotrain** that specifies the maximum number of iterations in co-training, **params.initN.cotrain** that specifies the initial pool size, and **Ny.cotrain** that specifies the number of samples per class to be added by each classifier in each iteration. Your return variable **coHmm** is a 1-by-2 cell array of trained HMMs, each of which is a set of HMMs from each modality.

**Question 14 [2 points]**: Follow the step in Part 4 and run an experiment using co-training HMM. What is the best classification accuracy you obtained using co-training HMM? What parameter values did you try out and what was your strategy for finding the best parameter value? Submit and interpret the confusion matrix you obtained.

*Note: Please do NOT submit NATOPS6.mat file.*

**References**


