Project 3: Gesture Recognition
ESE 650 Learning in Robotics, 2018 Spring

Dian Chen

March 1, 2018

Contents

1 Introduction

Gesture recognition is a classic type of classification problem, in which we're expected to output discrete labels to specify different gestures. However, typical classification algorithms such as K-Means, SVM, etc., are not the best choices to solve this problem since they ignore the temporal relations among data points in one gesture, and also fail to handle data sequences of different temporal length. Hidden Markov Models, on the other hand, are very capable of capturing temporal information in the data and thus are good candidates for gesture recognition tasks.

In this project, we're given labeled IMU data sequences corresponding to known gestures as training data; we are expected to train a series of Hidden Markov Models to predict gestures given any data sequences of the same form. One model is assigned to each gesture and during prediction the one that yields the highest probability stands for our prediction. General, non left-to-right models were used, which were trained using Baum-Welch algorithm, and cross-validated on the number of hidden states as well as the number of observation classes.

(Notice: the codes for generating the plots in this report are a bit different than those in the submitted package. The other parts stay untouched so I didn't re-submit the code package.)

2 Hidden Markov Model

A discrete Hidden Markov Model can be parameterized by the following parameters:

1. N, the number of hidden states;
2. M, the number of discrete observation classes;
3. $A_{N\times N} = \{a_{ij}\}$, the state transition matrix specifying the probability of transitioning from state i to state j;
4. $B_{N\times M} = \{b_{ik}\}$, the observation generation matrix specifying the probability of generating O_k in state i;

5. $\pi_{N\times 1} = \{\pi_i\}$, the prior probability that the initial state is state i.

Let's denote $\lambda = (A, B, \pi)$ as the parameter set that can fully describe a Hidden Markov Model. For this project we're trying to find a λ for each gesture model that can best explain the seen observations, and during prediction choose the one that yields the highest probability as our result. I chose to use same number of hidden states and shared observation classes among all models. Notice that the same N does not necessarily mean that different models have shared hidden states. The choice is simply for reducing the number of hyper-parameters to care about, and the hidden states of one model can be completely different from that of the other. However the observation classes are shared, which is required for a sequence being able to be examined by all models.

2.1 Left-to-Right vs. Non Left-to-Right

A left-to-right model means that the states can only proceed from left to right (the index only increases). Mathematically this poses some constraints on A and π:

$$a_{ij} = 0, \quad j < i$$

$$\pi_i = \begin{cases} 0, & i \neq 1 \\ 1, & i = 1 \end{cases}$$

Whereas as a general, non left-to-right model only has to satisfy the "sum to 1" probability constraint. The left-to-right model may be simpler in that it allows less possible transitions and has fewer parameters to solve. In my perspective, however, the left-to-right model may not be the best for our problem. The reasoning is that a left-to-right model assumes only on possible starting hidden state, but a gesture may not always have the same (or close enough) start (for example we can trace "8" starting from the top as well as starting from the bottom). Also, there may be hidden states that were actually visited more than once in a time elapse (consider tracing "3", we may have very similar states at the middle curve and the tail curve). Since the left-to-right model poses too strong assumptions, I chose to use the general, non left-to-right model which allows for more flexibility in our states.

2.2 Single Observations vs. Multiple Observations

In the paper the Baum-Welch algorithm gave treatments for both single observation sequence and multiple observation sequences. Since we're given multiple sequences for each of the gestures, incorporating information from all sequences of a particular gesture to build a corresponding model seems more reasonable and effective.

3 Model Training

3.1 Data Preprocessing

We were given raw IMU data, which are ordered by time and have 6 readings at each time stamp (3 from the gyro and 3 from the accelerometer). Before the data can be fed into the real training phase they have to be preprocessed.

3.1.1 IMU Data Filtering

The IMU measurements are unavoidably noisy, so an initial filtering was applied to the raw data to suppress the noises. A simple low-pass filter was used:

$$\mathbf{z}_t = \alpha \mathbf{z}_t + (1-\alpha)\mathbf{z}_{t-1}$$

This is a standard, simple procedure of dealing raw measurement series.

3.1.2 Discretization via K-Means

Since I chose to use discretized observation, the IMU data which lie in the continuous space should be clustered into M classes. A one-time K-Means clustering was applied to the filtered IMU data in the training set all gathered together, and the classifier was fixed and saved. The IMU data from the cross validation set and test set would be clustered according to this classifier. After this a sequence of T by 6 continuous measurements is converted to a long series of observation class labels (T by 1).

One implementation-wise trick is that, to be able to reproduce the entire results, the random seed of the K-Means classifier is set to a constant, specifically, 0.

3.2 Baum-Welch Algorithm

Baum-Welch algorithm is essentially a type of EM algorithm. The Expectation Step involves a forward pass and backward pass, which cleverly circumvent direct calculation of $P(O \mid \lambda)$ which is on the order of $O(TN^T)$. The Maximization Step updates the parameter λ according to the results in E-Step.

The full implementation of Baum-Welch algorithm along with tricks and comments are described in this section.

3.2.1 Initialization

Since I'm using general non left-to-right models, any initialization with positive, normalized entries in A, B and π should be valid. However, after some tryouts I found that random initialization gave better performance than uniform initialization. This is very likely because random initialization made it possible to not stuck in local optima.

3.2.2 Forward Pass with Scaling

In the forward pass, we compute an α matrix of shape $N \times T$. We first initialize the first column of α to be:

$$\alpha_1(i) = \pi b_i(O_1)$$

Since I chose to use scaling to deal with the numeric underflow problem of the diminishing probability, this first column should be normalized as follow:

$$c_1 = \frac{1}{\sum_{i=1}^{N} \alpha_1(i)}, \quad \hat{\alpha}_1(i) = c_1 \alpha_1(i)$$

For each time afterwards, we populate the columns of α according to the following induction rules:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \hat{\alpha}_t(i) a_{ij}\right] b_j(O_{t+1})$$

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)}, \quad \hat{\alpha}_t(i) = c_1 \alpha_t(i)$$

In practice, it's possible that a certain column of $b(O_t)$ has all zero entries (when the model sees an unknown observation from training) which will lead to all zeros and infinities for α_t and c_t onwards. Numerically this would break the execution of code. Theoretically, on the other hand, this is not a reasonable behavior because an outlier has the capability to cause the entire sequence to have zero probability. A proper way to handle this is to manually assign a very small, uniform probability to the column of α encountering unknown observation. I.e.,

$$\alpha_t = \text{np.max}(\alpha_t, \text{thresh})$$

The thresh can be set to something like 10^{-6}. This not only makes the execution numerically stable but also make sense when dealing with unseen observations. The outlier may cause the probability to plummet but data points later in the sequence can bring the probability back to normal. For a series data points that corresponds to truly unseen observations, the probability would only stay low as expected, but without blowing the execution.

3.2.3 Backward Pass with Scaling

In the backward pass we're populating another matrix β which also has the shape $N \times T$. The backward pass with scaling proceeds very similarly as the forward pass, with an initialization on column T:

$$\beta_1(i) = 1, \quad \hat{\beta}_1(i) = c_1 \beta_1(i)$$

And marching backwards according to induction rules:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j), \quad \hat{\beta}_t(i) = c_t \beta_t(i)$$

Similarly, to deal with the unseen observations, I chose to also threshold the probability at a very low value such that:

$$\beta_t = \text{np.max}(\beta_t, \text{thresh})$$

3.2.4 Parameter Update

Having computed $\hat{\alpha}$ and $\hat{\beta}$ in the E-Step, we should use them to compute other two values:

$$\gamma_t(i) = \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{P(O \mid \lambda)} = \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{i=1}^{N} \hat{\alpha}_t(i)\hat{\beta}_t(i)}$$

$$\xi_t(i,j) = \frac{\hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{P(O \mid \lambda)} = \frac{\hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}$$

And the parameters the models corresponding to each training sequence can be updated as:

$$\bar{\pi}_i = \gamma_1(i)$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\bar{b}_j(k) = \frac{\sum_{t=1,\text{s.t.}O_t=v_k}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}$$

This concludes the Maximization Step.

3.2.5 Convergence

In the scaled forward pass we have a by-product c_t, from which we can get the likelihood of the observation sequence conditioned on the current parameter λ:

$$P(O \mid \lambda) = \frac{1}{\prod_{t=1}^{T} c_t}$$

To avoid numeric underflow we can actually use the log likelihood:

$$\log\left[P(O \mid \lambda)\right] = -\sum_{t=1}^{T} \log c_t$$

The convergence criteria is that, when the difference of log likelihood between two successive estimations becomes sufficiently small, we deem the estimation converged.

3.2.6 Integration

For G kinds of gestures, we have several training sequences for each of them. I.e., we have $L_1, L_2, \ldots, L_G$ sequences for gesture 1, gesture 2, ..., gesture G. I chose to use each training sequence to train a single observation Hidden Markov Model using all the above rules, and for each gesture a final Hidden Markov Model is obtained by averaging the parameters of the single observations models.

$$\bar{a}_{ij} = \frac{\sum_{l=1}^{L} \bar{a}_{ij}^{l}}{L}$$

$$\bar{b}_j(k) = \frac{\sum_{l=1}^{L} \bar{b}_j^{l}(k)}{L}$$

$$\bar{\pi}_i = \frac{\sum_{l=1}^{L} \bar{\pi}_i^{l}}{L}$$

This is assuming each sequence has equal likelihood in the multiple sequence update rules (eq. 109 and eq.110 in Rabiner paper). Within a particular gesture model this is a reasonable approximation. In the end we can obtain G models, each corresponding to a particular gesture.

3.3 Prediction

3.3.1 Highest Log Likelihood

The prediction procedure using HMMs is a bit different than usual classification problems. Usually we feed the data into one model and the model spits out the label. Here, after training G separate models, we feed our sequence into each model in our model zoo and see which one yields the highest log likelihood. The numeric label of the results is thus obtained by taking the argmax across a bunch of log likelihoods:

$$l^* = argmax(\log P_l)$$

3.3.2 Normalized Likelihood

In this project I calculated another likelihood, which is the log likelihood normalized by time:

$$\log P' = \frac{\log P}{T}$$

After that the time-normalized likelihood is computed as

$$P' = e^{\log P'}$$

To make it a valid probability the time-normalized likelihoods should again normalized to satisfy the "sum to 1" rule. The intuition of this calculation is that longer data sequence tend to diminish the overall probability, making the log likelihoods of sequences of different length hard to interpret. So, we'd better take the total time into account.

3.3.3 Confidence of Prediction & Unknown Gesture

It is possible that out of the model zoo we get two or even multiple log likelihoods that are very close. To evaluate how certain my model zoo is about the predicted label, I defined two kinds of confidence metric.

1. Top-2 Difference
   This metric is simply defined as the difference between the largest log likelihood and the second largest log likelihood:

$$\text{confidence} = \log P_{max} - \log P_{second}$$

2. Likelihood Ratio
   This metric is computed as the ratio of the difference between the top likelihood and second largest likelihood, in the top likelihood (all normalized likelihoods):

$$\text{confidence} = \frac{P'_{max} - P'_{second}}{P'_{max}}$$

These two are both reasonable metrics depicting the confidence. I.e., if our model is very confident about the guess, the gap between the log likelihoods or the ratio of difference should be high. In my implementation, if the confidence of the prediction is below some threshold, meaning that there are fuzzy guesses, the predicted label is set to "Unknown Gesture".

3.4 Cross-Validation

There are two important hyper-parameters that need to be tuned via cross-validation: the number of hidden states N, and the number of observation classes M. In this project, we're given 2 sequences for "beat3", 2 sequences for "beat4", 7 sequences for "circle", 7 sequences for "eight", 7 sequences for "infinity" and 7 sequences for "wave". I chose to hold out one or two sequences in each gesture class as cross validation set and the left ones were used as training data. The data division was:

Gesture	# of Training Data	Training Data	# of CV Data	CV Data
beat3	1	beat3_31	1	beat3_32
beat4	1	beat4_31	1	beat4_32
circle	5	circle_12 circle_13 circle_14 circle_17 circle_32	2	circle_18 circle_31
eight	5	eight_01 eight_02 eight_04 eight_07 eight_32	2	eight_08 eight_31
infinity	5	inf_11 inf_13 inf_16 inf_32 inf_112	2	inf_18 inf_31
wave	5	wave_01 wave_02 wave_03 wave_05 wave_32	2	wave_07 wave_31

Table 1: Data Division for Training and Cross-Validation

In the cross validation I considered 6 possible combinations, with $N = 10, 15, 20$ and $M = 30, 40$. The prediction accuracy was considered priority, which ruled out the model zoo with $N = 10, M = 30$ (which incorrectly labeled "beat3" as "beat4"). The other 5 zoos all got 100% on the 10 cross-validation sequences so the prediction confidence was compared next. The one with the highest cumulative confidence (10 confidence values summed up) was picked.

4 Results

All the plots are available at this google drive link:

`https://drive.google.com/open?id=1eKZc0U60g1HDAdRmPsZceUIR3sf7i0zU`

4.1 Visualization of Transition Matrices

The visualization of the transition matrices of the model zoo with $N = 20$ and $M = 40$ is as follow (Fig. 1):

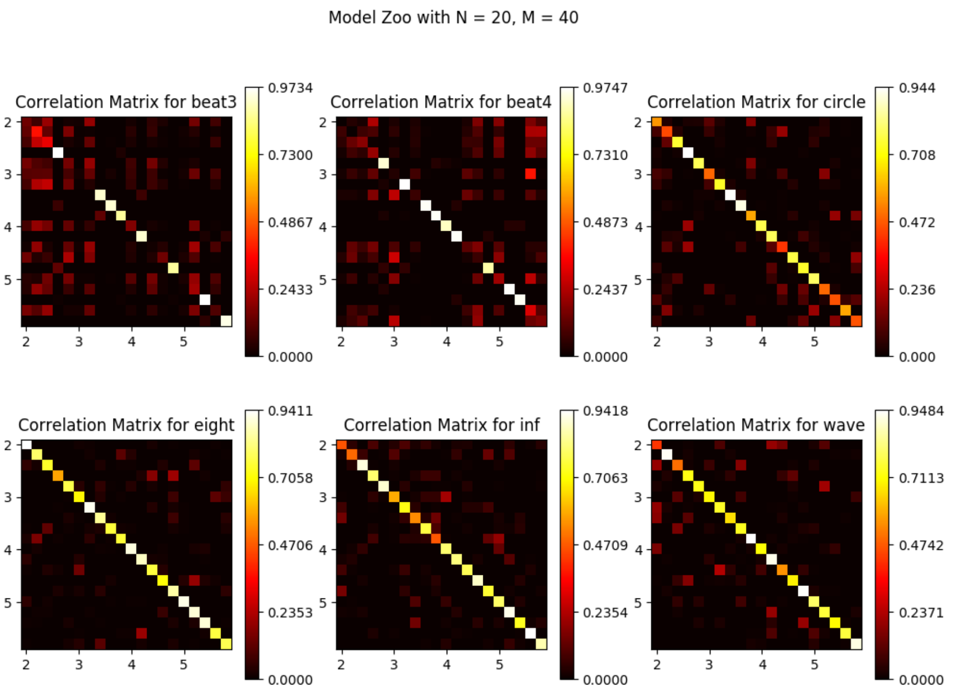

Figure 1: Transition Matrices with $N = 20, M = 40$

4.2 Prediction Results

The prediction results of all the test data are showed in Fig. 2 to Fig. 9.

5 Discussion

5.1 Test Results

During the in-class demonstration I got 7 out of 8 predictions right (one "beat3" being mislabeled as "beat4"), while the guy who performed the demo and claimed to use left-to-right models got all 8 predictions right. He suggested that may because the left-to-right models are simpler than non left-to-right ones and thus less prone to over-fitting. I agree with him on the possibility of over-fitting our data since we were given so few sequences for "beat3" and "beat4", which are already complex and similar to each other in themselves. However, as argued in the model selection section, I still believe that non left-to-right models are more suitable for this gesture recognition task (please refer back for detailed explanation). The only failure on "beat3" may result from insufficient training data, convergence in local optima, and imperfect choice on hyper-parameter N and M.

5.2 Normalizing Log Likelihood by Time

Due to the varying length of sequences fed into the model, the longer the sequence, the smaller the likelihood. So while the log likelihood doesn't make much sense at first look, normalizing it by

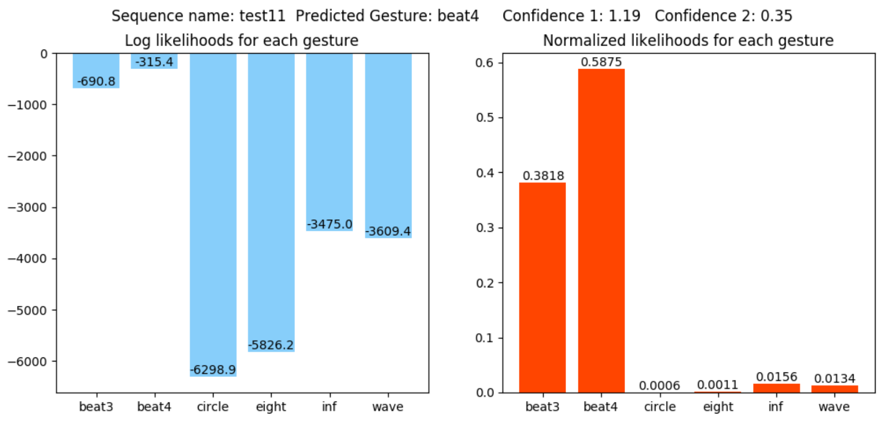

Figure 2: Prediction Result for test11

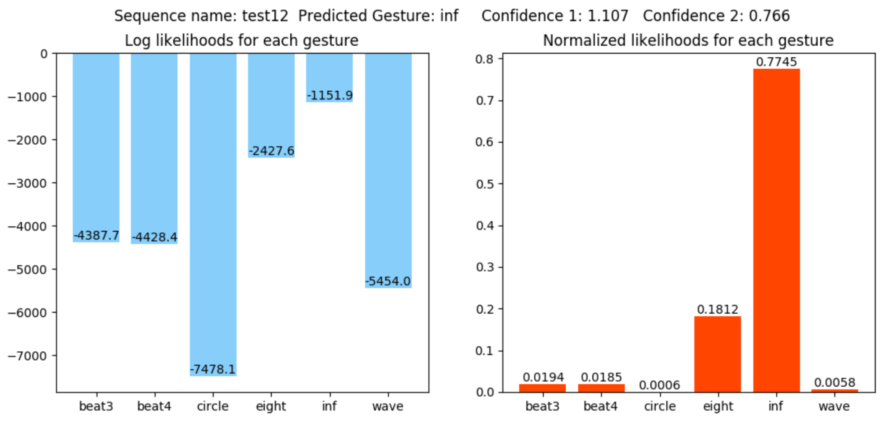

Figure 3: Prediction Result for test12

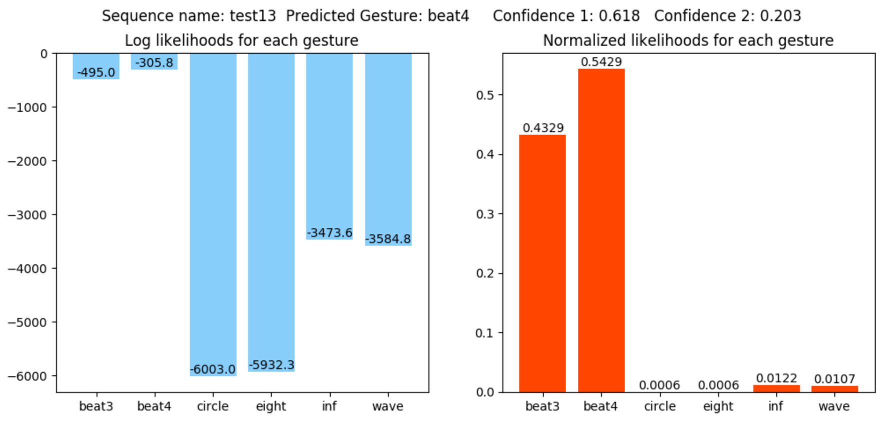

Figure 4: Prediction Result for test13

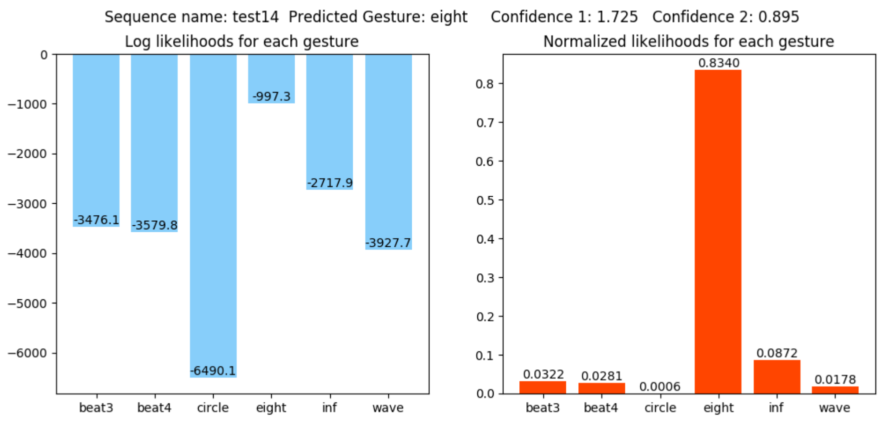

Figure 5: Prediction Result for test14

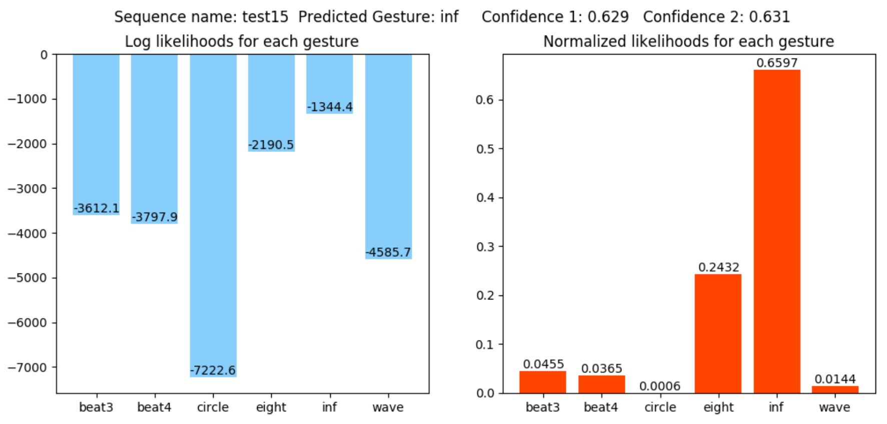

Figure 6: Prediction Result for test15

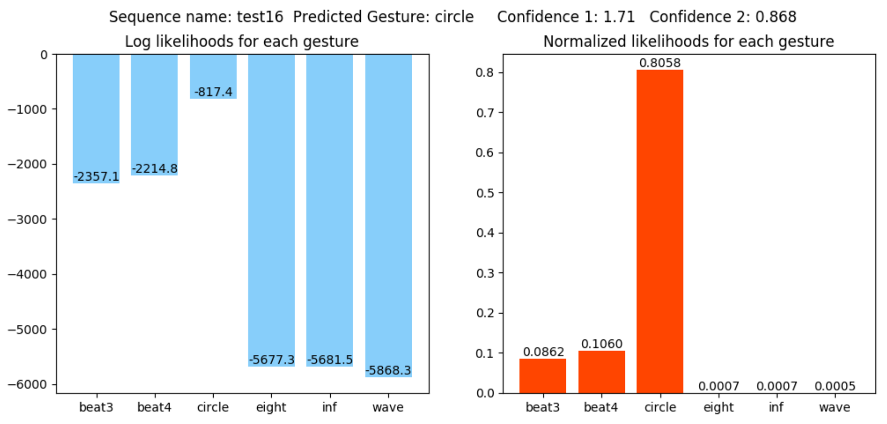

Figure 7: Prediction Result for test16

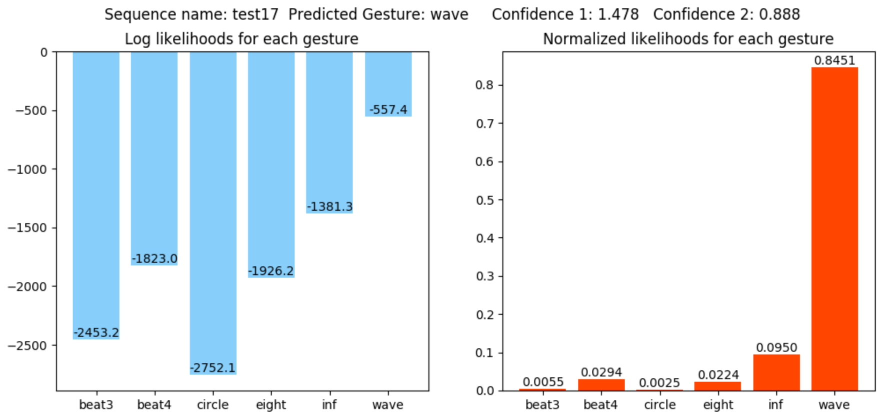

Figure 8: Prediction Result for test17

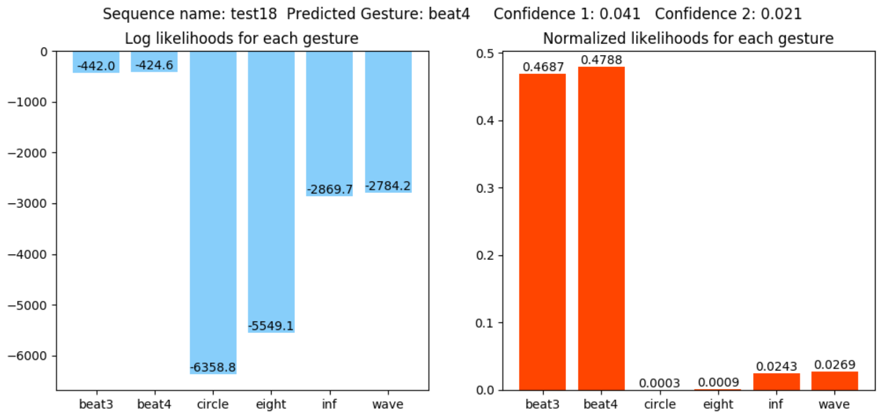

Figure 9: Prediction Result for test18

time is a clever and reasonable manipulation. As we can see from the red bar plots in the result section, the probabilities make much sense and we can spot the predicted label more easily.

5.3 Transition Matrices

The visualization of transition matrices is very interesting and worth some discussion. First, for all 6 models we can see clear high probabilities on the diagonals, which conforms with the intuition that hidden states in a gesture tend to evolve sequentially in time, with scattered possibilities to deviate the "main road". Secondly, for matrices of "beat3" and "beat4", we can see that the highlights on the diagonals are less as intense, and there are more scatterings at other probability entries. One possible reason is that these two gestures are by nature more complex and have more obscure jumpings between hidden states; another reason may be that we had too few training sequences that they haven't quite converged to the optimal models.

5.4 Confidence Metrics

In classification tasks like this, a properly defined confidence metric would benefit the analysis of our models. In this project I've defined two confidence metrics, which both make sense while explaining the results. For example, when "beat3" or "beat4" sequences are fed in, the model zoo would have much lower confidence on the prediction, which inherently comes from the close likelihoods. For other kinds of sequences there will be a clear gorge in the log likelihood plot and a clear spike in the normalized likelihood plot. The fuzziness of predicting "beat3" and "beat4" is again showed in various ways.
Another usage of confidence metrics is to determine whether or not to label the data as "Unknown".

5.5 Other Comments

Aside from the theoretical part, we should also take care of many practical, implementation-wise problem, such as zero probabilities, scaling, or more generally, numeric stability during execution. We should also be careful that the K-Means classifier should be trained only once and fixed after that.

Reference

1. L R. Rabiner "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition"

2. `https://www.youtube.com/watch?v=9dp4whVQv5s`, HMM Lecture by Pieter Abbeel