# Gesture Recognition Using Hidden Markov Model

## I. Introduction

This project aims to recognize the gesture using IMU sensor readings from gyroscopes and accelerometers. Generally, we train a set of Hidden Markov Models, one for each gestures, using single or multiple data sequences. Then we can predict the label of any observation sequence by comparing its log-likelihood under each gesture's model.

## II. Pre-Processing

In order to apply the hidden Markov model to the IMU data, first we need to discretize the raw observation. In this project, a simple K-means clustering method is taken to quantize the 6-dimension IMU reading. Note that all observation sequences are concatenated and clustered together, so that they will have compatible physical meaning among all different gesture models. Experiments show that, using a proper cluster number (number of discrete observations), it is sufficient to provide a satisfactory prediction result without other more complicated pre-process.

## III. Gesture Recognition with Hidden Markov Model

The core task of this project is to a Hidden Markov Model for each gesture, and use these models to calculate the likelihoods of an observation sequence.
It is noticed that the same gestures always start from the same point, so we will use left-right HMM, as suggested, to represent a gesture. Which means $\pi_1 = 1$, $\pi_i = 0, i = 2, \cdots, N$ holds through training, and the state transition matrix should be initialized as a upper triangular matrix.
In the following sections, we will introduce the inference and learning methods of HMM. Most of the equations are from Rabiner's paper [1].

### A. Inference

Given a trained Hidden Markov Model $(\pi, A, B)$, we can calculate the forward probability $\alpha_t(i) = P(O_1, O_2, \cdots, O_t, q_t = S_i|\lambda)$ iteratively:

$$\begin{gathered}\alpha_1(i) = \pi_i b_i(O_1) \\ \alpha_{t+1}(j) = (\sum_{i=1}^{N} \alpha_t(i) a_{ij}) b_j(O_{t+1})\end{gathered} \tag{1}$$

the likelihood of the observation sequence can be computed as:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{2}$$

Although the backward probability is not needed in the inference step, it will be useful in the Baum-Welch algorithm in the training step. The backward probability is defined as:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \cdots, O_T | q_t = S_i, \lambda)$$

$\beta$ can be computed as:

$$\begin{gathered}\beta_T(i) = 1 \\ \beta_t(j) = \sum_{i=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)\end{gathered} \tag{3}$$

One thing we should consider is that $\alpha$ and $\beta$ are computed iteratively, their value can be very small as $t$ gets larger. In order to avoid underflow, the probabilities need to be rescaled every time after computation. More specifically, we need to normalize $\alpha$ every time after the update. Equation (1) becomes:

$$\begin{gathered}\alpha_1(i) = \pi_i b_i(O_1), \quad c_1 = \frac{1}{\sum_{j=1}^{N} \alpha_1(j)} \\ \hat{\alpha}_1(i) = c_1 \alpha_1(i) \\ \alpha_{t+1}(j) = (\sum_{i=1}^{N} \hat{\alpha}_t(i) a_{ij}) b_j(O_{t+1}) \\ c_t = \frac{1}{\sum_{j=1}^{N} \alpha_t(j)} \\ \hat{\alpha}_{t+1}(j) = c_t \alpha_{t+1}(j)\end{gathered} \tag{4}$$

And the log likelihood is:

$$\log P(O|\lambda) = -\sum_{t=1}^{T} \log c_t \tag{5}$$

Similarly, we need to rescale $\beta$ as well. Rabiner's paper assumes that the scale of $\alpha$ and $\beta$ are similar, so that we can use $c_t$ as the scale factor of $\beta$. However, sometimes $\beta$ could has larger scale than $\alpha$ and the algorithm will overflow. So we choose to normalize $\beta$ independently using the same method as above:

$$\begin{gathered}\hat{\beta}_T(i) = 1/N \\ \beta_t(j) = \sum_{i=1}^{N} a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j) \\ d_t = \frac{1}{\sum_{i=1}^{N} \beta_t(i)} \\ \hat{\beta}_t(j) = d_t \beta_t(j)\end{gathered} \tag{6}$$

Another method to avoid underflow is to take logarithm of the probabilities and variables during the computation. The equations can be re-write as follows:

$$\log \alpha_1(i) = \log \pi_i + \log b_i(O_1)$$

$$\log \alpha_{t+1}(j) = \log \sum_{i=1}^{N} \exp(\log \alpha_t(i) + \log a_{ij}) + \log b_j(O_{t+1}) \tag{7}$$

$$\log P(O|\lambda) = \log \sum_{i=1}^{N} \exp(\log \alpha_T(i)) \tag{8}$$

$$\log \beta_T(i) = 0$$

$$\log \beta_t(j) = \log \sum_{i=1}^{N} \exp(\log a_{ij} + \log b_j(O_{t+1})) + \log \beta_{t+1}(j) \tag{9}$$

## B. Training

To train a HMM, we use the Baum-Welch method, or equivalently an EM method. In each iteration, the expectation and maximization steps can be described as below:
***E-step***:

1) Compute forward and backward probabilities $\alpha_t, \beta_t$ and $\log P(O|\lambda)$ using Equations (5) to (7).
2) Compute $\gamma_t(i) = P(q_t = S_i|O, \lambda)$:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{10}$$

3) Compute $\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j|O, \lambda)$:

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)} \tag{11}$$

***M-step***

1) Update $A$

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{12}$$

2) Update $B$:

$$b_i(l) = \frac{\sum_{\substack{t=1 \\ s.t. O_t = v_l}}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \tag{13}$$

Note that since we are using left-to-right model, the prior $\pi$ does not need to be updated.
One trick during training is to replace all the 0 values with a small number after updating the emission matrix. The reason is that if there are zero value in the emission matrix, the probability of the training sequence could be 0 and the algorithm might fail.
If we have multiple observations sequence and would like to train the model using all sequences together, we could use a generalized version of Baum-Welch algorithm. Suppose we have a set of sequences $O = [O^1, O^2, \cdots, O^K]$, the probability we need to maximize becomes:

$$\prod_{k=1}^{K} P(O^k|\lambda) \sim \sum_{k=1}^{K} \log P(O^k|\lambda) \tag{14}$$

The generalized algorithm can be written as:
***E-step***: For each observation sequence $O^k$ in $O$, compute $\alpha^k$, $\beta^k$, $\log P(O^k|\lambda)$, $\gamma^k$, and $\xi^k$ as above.
***M-step***

$$a_{ij} = \frac{\sum_{k=1}^{K}\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{k=1}^{K}\sum_{t=1}^{T-1} \gamma_t(i)} \tag{15}$$

$$b_i(l) = \frac{\sum_{k=1}^{K}\sum_{\substack{t=1 \\ s.t. O_t = v_l}}^{T} \gamma_t(i)}{\sum_{k=1}^{K}\sum_{t=1}^{T} \gamma_t(i)} \tag{16}$$

Same as the inference step, we can write the Baum-Welch method in logarithms. The adapted equations will not be specified in the report. For this project, we mainly use the scaling method instead of the logarithm method. However, the latter version is also implemented and the details can be found in the code.

## IV. Experiment and Result

### A. Cross validation and Parameters Selection

In order to select a proper setting and hyper-parameters, we choose LOOCV(leave-one-out cross validation) to evaluate the performance of the models. The most straight-forward metric of the performance is the test accuracy.However, since there are only 32 training sequences (7 for "wave", "infinity", "eight", "circle" each, and 2 for "beat3", "beat4" each), the accuracy may not be of significant difference. We introduced another metric to evaluate the confidence of the prediction - the ratio between the maximum and second-maximum log-likelihood:

$$\mathbf{1}(S_{est} = S_{gt})\frac{\log P_1 - \log P_2}{|\log P_2|} = \mathbf{1}(S_{est} = S_{gt})(1 - \frac{\log P_1}{\log P_2})$$

while $S_{est}$ is the estimated state, $S_{gt}$ is the ground-truth state, $P_1$, $P_2$ are the first and second maximum likelihood of all 6. We can see that, if the estimation is wrong, the confidence is 0; Otherwise the confidence reflects how much the probability of the correct model is larger than the rest.

First, I compared the performance of HMMs trained using only one sequence per model and the HMMs trained using multiple sequences. The training accuracy of the single-sequence-trained model has an accuracy of 0.97 and the average confidence of 0.87; While the multi-sequence-trained model has accuracy 1.0 and average confidence 0.89 From the result, we can see that both can correctly classify most of the sequences, the first one is faster to train and could prevent from overfitting, while the second one provides a more robust result. The confusion matrices are shown in Figure. 1 and 2.

Second, for HMM, it is critical to select the proper $N$ (number of hidden states) and $M$ (number of difference

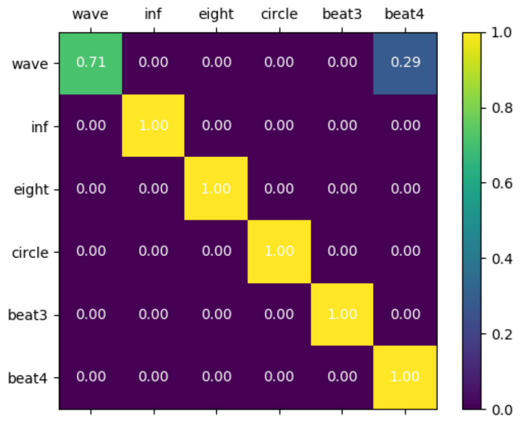

Fig. 1: Confusion matrix of training data using single-sequence-trained HMM (M=30, N = 10)

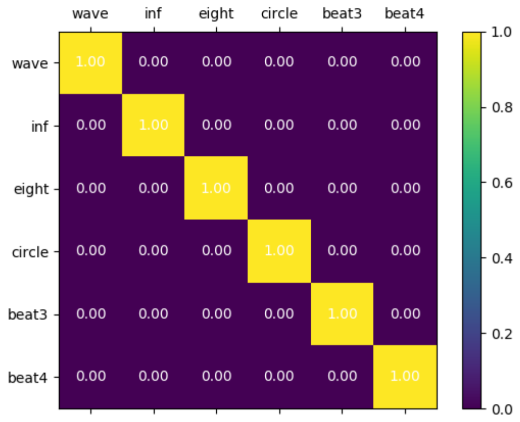

Fig. 2: Confusion matrix of training data using multi-sequence-trained HMM (M=30, N = 10)

observations) value. In the experiment, I tested on a set of different combinations of $N$ and $M$, the results are demonstrated in Figure. 3.

## B. Result on Test Data

Table. 1 gives the test result using final submitted model (trained on all training samples, $N = 10$, $M = 30$):

## References

[1] Lawrence R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, PROCEEDINGS OF THE IEEE, 1989

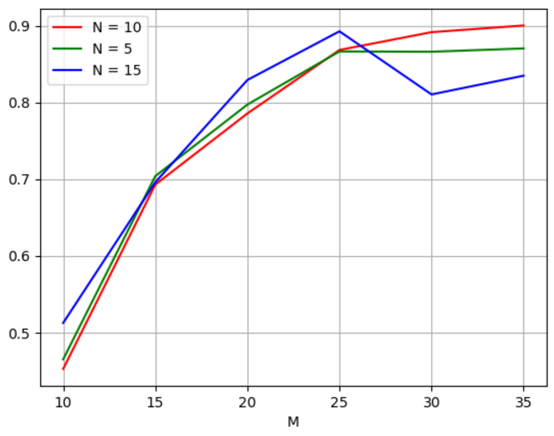

Fig. 3: Confidence of prediction for training data under different N and M

| dataset | prediction | confidence |
|---|---|---|
| test 11 | beat4 | 0.5579 |
| test 12 | inf | 0.6654 |
| test 13 | beat4 | 0.2569 |
| test 14 | eight | 0.9765 |
| test 15 | inf | 0.6154 |
| test 16 | circle | 0.9887 |
| test 17 | wave | 0.9614 |
| test 18 | beat3 | 0.1004 |

TABLE I: Prediction on test set

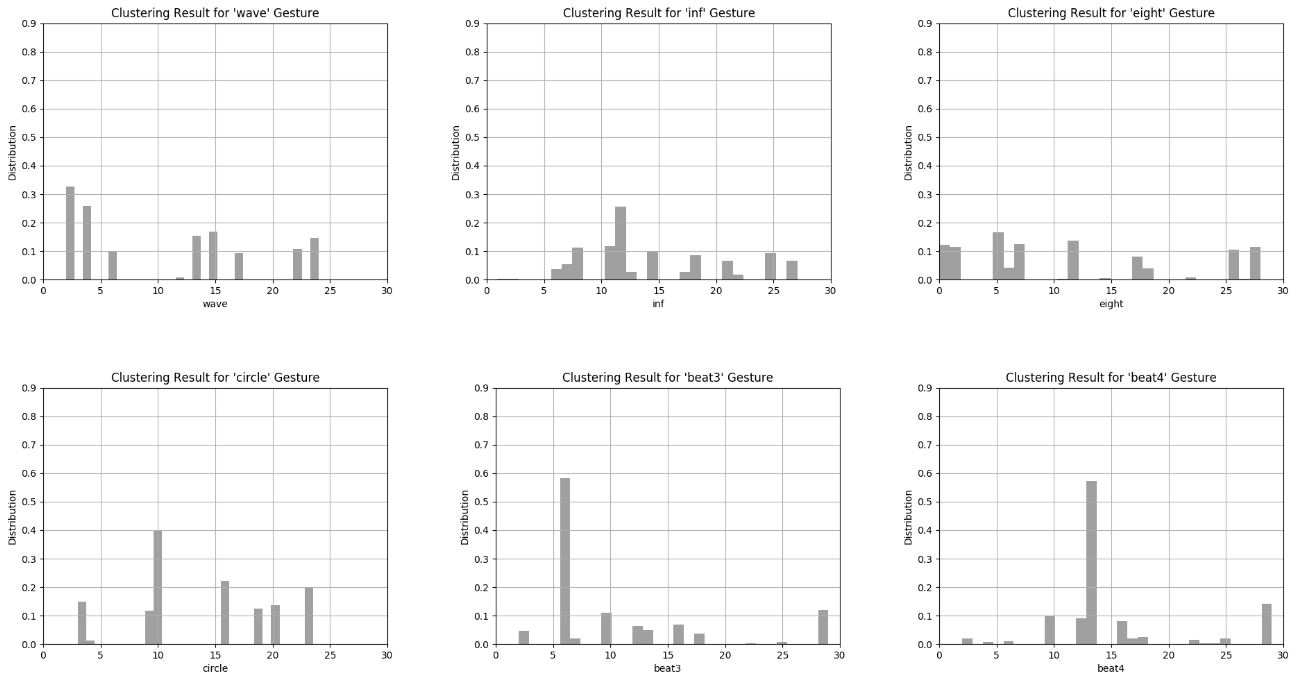

Fig. 4: Histogram of Quantized IMU Data Distribution using K-Means Clustering (M=30)