ESE650 Learning in Robotics, 2018 Spring
Project 3: Gesture Recognition
Wudao Ling

Introduction

In this project, we need to recognize 6 gestures based on raw IMU data (time, accelerometer and gyroscope). I began with K-means on sensor data, in order to quantize them as discrete observations. Furthermore, I built Hidden Markov Models for each gesture and trained with multiple observation sequences. At last, HMMs together with K-means model can predict gesture accurately.

1 Sensor Data Quantization

In order to obtain discrete observations for Hidden Markov Model, I need to cluster IMU data. First I leave out time data because HMM is sequential and actual time doesn't matter that much. Then I tried different feature extraction like norm and angle of acceleration, however after experimentation, the raw data with 6 features work the best. I think this strategy might be a good fit if acceleration is with respect to world frame, not body frame. At last, I used K-means model in scikit-learn to fit all raw data and saved the model.
The number of clusters M is also the number of observation classes. With K-means model, both training and testing data could be predicted as discrete observations from 1 to M.

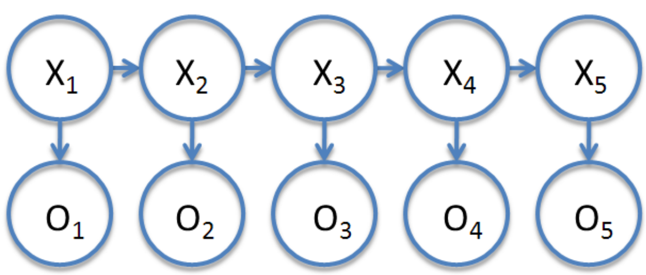

Figure 1: Structure of Hidden Markov Model

2 Hidden Markov Model

HMM is a generative model for time series data. This dynamic Bayesian network models the unobserved hidden states and their emission(observation).
HMM has following parameters:

- N, the number of hidden states (S)
- M, the number of observations (O)
- $A_{N\times N}$, transition, where $A_{ij} = P(S_{t+1} = i|S_t = j)$, $1 \leq i, j \leq N$.
- $B_{N\times M}$, emission, where $B_j(k) = P(O_t = k|S_t = j)$, $1 \leq j \leq N, 1 \leq k \leq M$
- π_N, initialization, where $\pi_i = P(S_1 = i)$, $1 \leq i \leq N$.

N and M are pre-determined and specific for applications, generally they are selected by cross-validation. Other parameters consist of model $\lambda = (A, B, \pi)$. A also decide whether HMM is ergodic or left-to-right.

The 3 basic problems in HMM are:

- Problem 1: Given observation sequence O and model λ, how to efficiently compute the likelihood $P(O|\lambda)$
- Problem 2: Given observation sequence O and the model λ, how to choose a corresponding state sequence S that explain observations the best
- Problem 3: How to adjust $\lambda = (A, B, \pi)$ to maximize $P(O|\lambda)$.

I will focus on problem 1 and problem 3 in this project.

2.1 Log space computation

Basically all factors in HMM are probabilities that are less than 1, which imply HMM tend to underflow due to limited machine precision. I used log space to solve this, an alternative is scaling.
With log space, multiplication and division change to addition and subtraction. Sum become scipy.special.logsumexp, because direct exponential operation may underflow.
For consistency concern, this report will be written like normal. The log space implementation could be found in my code.

2.2 Forward-backward procedure

Assuming the length of observation sequence is T, the brute-force solution to problem 1 has M^T complexity. In contrast, forward-backward procedure solve this with M^2T (dynamic programming):

Forward procedure

- Define $\alpha_t(i) = P(O_1, O_2, ..., O_T, S_t = i)$.
- Initialize $\alpha_1(i) = \pi_i B_i(O_1)$, $1 \leq i \leq N$.
- Induction $\alpha_{t+1}(i) = \big[\sum_{j=1}^{N} \alpha_t(j) A_{ij}\big] B_i(O_{t+1})$, $1 \leq t \leq T-1, 1 \leq j \leq N$.
- Termination $P(O|\lambda) = \sum_i \alpha_T(i)$

Backward procedure

- Define $\beta_t(i) = P(O_{t+1}, O_{t+2}, ..., O_T | S_t = i)$.
- Initialize $\beta_T(i) = 1$, $1 \leq i \leq N$.
- Induction $\beta_t(i) = \sum_{j=1}^{N} \beta_{i+1}(j) A_{ji} B_j(O_{t+1})$, $T-1 \geq t \geq 1, 1 \leq i \leq N$.

2.3 Baum-Welch Algorithm

Baum-Welch is the EM algorithm for HMM, and it helps to solve problem 3: training HMM. Assuming there are K observation sequences for a HMM to train.

Definition

- $\xi_t(i, j) = P(S_t = j, S_{t+1} = i | O)$.
- $\gamma_t(i) = P(S_t = i | O)$.

E-step

For observation sequence k:

- run Forward-backward procedure to obtain α^k and β^k
- $\xi_t^k(i, j) = \frac{\alpha_t^k(j) A_{ij} B_i(O_{t+1}^k) \beta_{t+1}^k(i)}{\sum_i \sum_j \alpha_t^k(j) A_{ij} B_i(O_{t+1}^k) \beta_{t+1}^k(i)}$.
- $\gamma_t^k(i) = \frac{\alpha_t^k(i) \beta_t^k(i)}{\sum_i \alpha_t^k(i) \beta_t^k(i)}$.

M-step

- $\pi_i = \frac{1}{K} \sum_k \gamma_1^k(i)$.
- $A_{ij} = \frac{\sum_k \sum_t \xi_t^k(i,j)}{\sum_k \sum_t \gamma_t^k(j)}$.
- $B_j(k) = \frac{\sum_k \sum_t \gamma_t^k(j) st. O_t^k = k}{\sum_k \sum_t \gamma_t^k(j)}$.

The updated model parameters need to be normalized.
Baum-Welch Algorithm will be run iteratively until it reaches maximum iteration or converges to a local minima (training data likelihood doesn't decrease anymore).

3 Training

3.1 Validation

The dataset contains 2 instances for Beat3, Beat4 and 7 instances for other gestures. I split 1 instance of Beat3, Beat4 and 2 instances of other gestures for validation. All the rest instances are used to train HMM. Also I manually changed the validation set for "cross-validation".

3.2 Model

After trial and validation, I found $N = 10$ and $M = 15$ lead to robust performance regardless of initialization.
For model initialization, π is uniform, while elements of A and B are random numbers in interval $(0, 1)$. A is a lower triangle matrix (elements above diagonal are all zeros) because my HMM model is left-to-right. These model parameters are normalized after initialization.
For HMM training, max iteration number is 50 and log likelihood tolerance is 0.1. This choice also result from trial and validation.

3.3 Result

At the end, my models could achieve 100% accuracy on both training set and validation set. Thus I could be confident in chosen parameters and trained final HMMs on all data.

4 Testing

Instances: [test11, test12, test13, test14, test15, test16, test17, test18]
Prediction: [beat4, inf, beat3, eight, inf, circle, wave, beat3]
Max log likelihood: [-1295.71364916, -1265.89492647, -inf, -1035.49680571, -1211.96914108, -693.02656902, -596.4222537, -921.25004991]

References

[1] L.R.Rabiner, *A tutorial on hidden markov models and selected apllications in speech recognition.* In A. Waibel and K.-F. Lee, editors, Readings in Speech Recognition, pages 267-296. Kaufmann, San Mateo, CA, 1990.

[2] *Hidden Markov Models*, CIS520 Machine Learning, University of Pennsylvania, https://alliance.seas.upenn.edu/ cis520/wiki/index.php?n=Lectures.HMMs