ESE 650 Project 3: Gesture Recognition

James Yang

February 26, 2015

1 Introduction

In this project, we were tasked with creating a gesture recognition algorithm that can classify some basic motion gestures. I trained a discrete-observation Hidden Markov Model on K-Means partitioned accelerometer and gyroscope data recorded on an Android phone. The HMM was used to calculate the probability that a certain motion sequence represented a particular gesture. The HMM trained on six different gestures: circle, eight, infinity, wave, 3-beat, and 4-beat.

2 Generating Observations with K-Means

The first part of my algorithm takes raw sensor data and partitions them into clusters using the K-means algorithm. Figure 1 shows a sample of the resultant discretizations. The number of observations was experimented with, and 47 observations provided the best results. While typically running the HMM with 47 observations gave somewhat consistent results, results were generally sensitive to the positions of the resultant cluster centers.

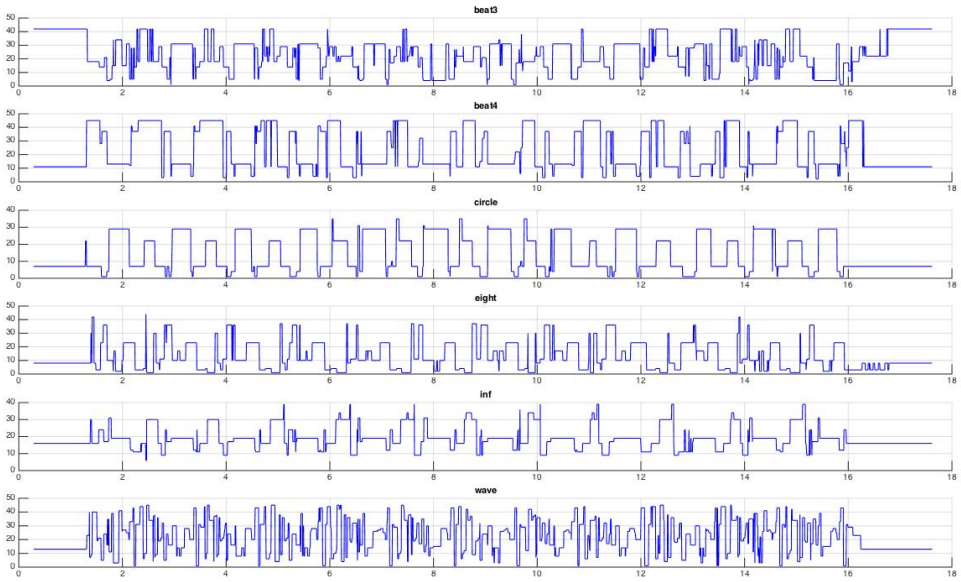

Figure 1: A sample observation set.

3 Hidden Markov Model

The HMM was implemented as outlined in [1]. The following is an outline of some particular aspects of my implementation.

3.1 Initial State

For initial state, better results were obtained by using the left-right model for the state transition matrix A. A was initialized as a band diagonal indicating 50% chance of transitioning from one state to the next. The observation matrix B was initialized as uniform, and the initial class priors π were initialized as having 100% chance of being in the first state as it was given that each motion sequence would start at the beginning.

3.2 Scaling for Underflow

A significant problem during implementation was underflow, where double precision was insufficient for carrying meaningful probability values. Instead of calculating probabilities directly, each iteration of α and β was scaled by a normalization factor such that $\sum_i^N \alpha_t(i)/\exp Z_{\alpha_t} = 1$ and $\sum_i^N \beta_t(i)/\exp Z_{\beta_t} = 1$. The formula for α_{t+1} and β_{t+1} are as follows:

$$\alpha_{t+1} = \hat{\alpha}_{t+1} \exp\left(Z_{\alpha_{t+1}}\right)$$

$$\hat{\alpha}_{t+1} = A^T \hat{\alpha}_t b\left(O(t)\right)$$

$$Z_{\alpha_{t+1}} = Z_{\alpha_t} + \ln \sum_i^N \hat{\alpha}_{t+1}$$

$$\beta_{t+1} = \hat{\beta}_{t+1} \exp\left(Z_{\beta_{t+1}}\right)$$

$$\hat{\beta}_{t-1} = A \cdot b\left(O(t)\right) \hat{\beta}_t$$

$$Z_{\beta_{t-1}} = Z_{\beta_t} + \ln \sum_i^N \hat{\beta}_{t+1}$$

3.3 Termination Metric

Termination of the algorithm was determined when the magnitude of the rate of change in log probability was below an empirically determined threshold, 0.005. Effectively, given a sequence of log probabilities $\log P_t$, termination was determined if the following condition was satisfied:

$$|\log P_t - \log P_{t-1}| < 0.005$$

This permitted the log probability rate of change to be negative, effectively allowing for the EM algorithm to look past potential local minima and continue optimizing. This typically resulted in a 10% decrease in training error.

3.4 Leave-One-Out Cross Validation

While actual averaging of cross-validated solutions was not implemented, a single dataset for each gesture was removed from the training set, and later tested on. Using this method, the three primary parameters, number of hidden states N, number of unique observations M, and termination threshold h, were tuned to provide the best testing error. Upon finding decent values, they were further tuned to achieve as close to perfect training error as possible on the full dataset.

4 Running the Code

To train on the full dataset, run hmm_train.m. To test on the full dataset, run hmm_test.m. To train and test with LOOCV, run hmm_train_and_test.m.

References

[1] L. R. Rabiner. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.* In A. Waibel and K.-F. Lee, editors, Readings in Speech Recognition, pages 267-296. Kaufmann, San Mateo, CA, 1990.