Gesture Recognition

Abhijeet Singh

***Abstract*— This report describes the application of Hidden Markov Models (HMMs) to recognize arm motion gestures from IMU sensor data comprising of accelerometer and gyroscope readings. For each gesture in the training set, a separate HMM is trained which is then used to classify a new gesture into one of the trained classes.**

I. INTRODUCTION

Gesture recognition is the interpretation of human gestures by computers using some algorithm. This can be used by humans to interface with machines in general. The arm gestures we use (shown in Figure 2) are: Wave, Infinity, Eight, Circle, Beat3, Beat4. The sensor data was collected using a mobile with IMU, from which accelerometer and gyroscope readings were recorded. We use Hidden Markov Models (HMMs) to model each gesture using training data. The model parameters are then used to compute the log-likelihood of a new gesture to belong to the gesture class, which is used to finally predict the gesture.

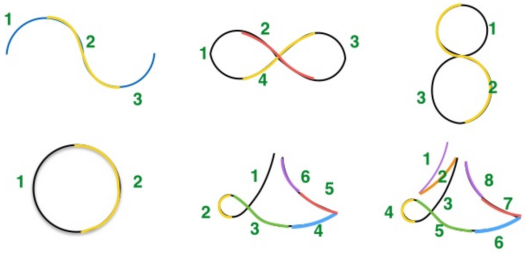

Fig. 1. Gestures

II. DESCRIPTION

Hidden Markov Model is a stochastic model based on the Markov assumption that the probability of the next event depends only on the current event, for all event instances. They consist of hidden states and observable states, where the hidden states determine the next hidden state (stochastically) and also the observed state. There are probabilities associated with each of the transitions which define the model. HMMs are generative models which model the joint distribution of the observed and hidden state.

*This work is part of the Project 3 of the ESE 650 class Learning in Robotics at University of Pennsylvania, taught by Prof. Daniel Lee

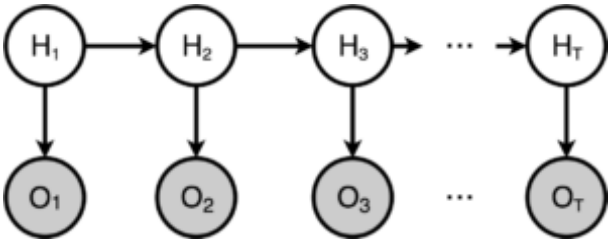

Fig. 2. Hidden Markov Model with H_i as the hidden states and O_i as the observed states

A. Problem Formulation

The sensor data for the gestures consists of consecutive accelerometer and gyroscope readings. These readings represent the state of the arm at each instant while performing the gesture. Each state is followed by another, making a pattern, unique to a particular gesture. Our task is to learn these patterns so that we can determine if a new gesture from the test set follows the same pattern or not.

This task can be effectively modeled using HMMs. The consecutive states forming a pattern can be represented by the hidden states of the HMM and the sensor readings as the observed states which basically describe these hidden states. Since the next hidden state in a HMM depends on the previous state, they can capture the various movements involved while performing the gesture.

In this way, the problem reduces to training a HMM for each gesture in the training set. For each HMM, one would need to estimate the transition probabilities and the emission probabilities that model that gesture. In terms of HMM parlance, this refers to the learning step of finding the correct model parameters (Problem 3 described in [1]) Once all the HMMs are trained for each gesture, we can calculate the likelihood of a new gesture belonging to classes trained in the previous step. The best likelihood can serve as a result of the classification for the new gesture (Problem 1 in [1]).

B. Technical Approach

The observations of the accelerometer and gyroscope are in a continuous vector space. We first discretize these observations into a fixed number of observation classes (say, M). This is done using the k-means clustering algorithm [2], where we use the 6 dimensional input data (consisting of three dimensions of acceleration and three dimensions of angular velocity readings from the IMU) to cluster into M clusters. We will perform all the computations in this discrete observation space.

Next, we select the number of hidden states (say, N) for the HMM. Note that the values of N and M can be determined

by cross validation as well.

The first job is to find the correct HMM parameters - the transition probabilities from one hidden state to another (represented by matrix A) and the emission probabilities from one hidden state to an observation state (represented by matrix B). These parameters encode each "pattern" of the gesture we see. This parameter estimation is performed using the Baum-Welch algorithm [1].

The Baum-Welch algorithm is essentially the Expectation-Maximization algorithm in the context of HMMs. In the E-step, we compute the posterior probabilities of the hidden random variables using the parameter estimates in the previous iteration. In the M-step, the model parameters are computed to maximize the expected likelihood under the above posterior probabilities.

The E-step involves the computation of posterior probabilities like $\gamma_t(i)$ and $\xi_t(i,j)$. $\gamma_t(i)$ represents the probability of being in hidden state i at time instant t, given the entire observation sequence. $\xi_t(i,j)$ represents the probability of transitioning from a hidden state i to j at time instant t.

$$\gamma_t(i) = P(HiddenState_t = i | X_{1:T})$$

$$\xi_t(i,j) = P(HiddenState_t = i, HiddenState_{t+1} = j | X_{1:T})$$

Both these variables can be easily computed using the matrices A, B and another set of variables called the forward variable ($\alpha_t(i)$) and the backward variable ($\beta_t(i)$). The forward variable represents the joint probability of seeing the observation sequence upto time t and being in state i at time t, while the backward variable is the probability of observing the sequence after time t given the current state to be i. These variables can be directly computed using the forward-backward procedure described in [1]. From those values, γ and ξ are computed as follows -

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_k^K \alpha_t(k)\beta_t(k)}$$

$$\xi_t(i,j) = \frac{\alpha_t(i)A(i,j)B(j,X_{t+1})\beta_t(j)}{\sum_m^K \sum_n^K \alpha_t(m)A(m,n)B(n,X_{t+1})\beta_t(n)}$$

Once we have the above posterior probabilities, we can proceed to the M-step of the EM algorithm. This involves the re-computation of A and B matrices -

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

$$B_{ix} = \frac{\sum_{t=1}^{T} \gamma_t(i).(x_t == x)}{\sum_{t=1}^{T} \gamma_t(i)}$$

If the training set includes multiple observation sequences for each gesture, the above equations will be summed over all such observation sequences in the numerator and denominator. We also compute the vector of initial state probabilities (π) apart from the matrices A and B, which is calculated directly by averaging the γ values for the first state.

hidden state.

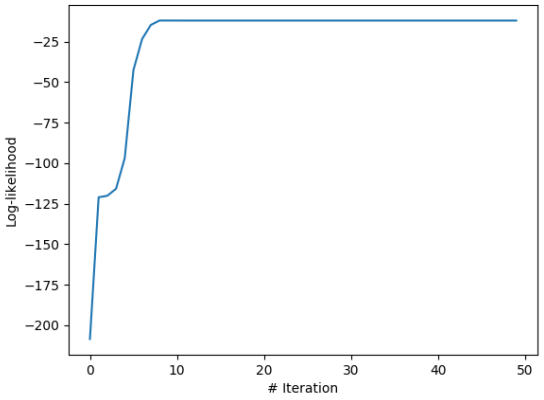

Fig. 3. Convergence in a typical run of Baum-Welch

The E-step and M-step is repeated until the log-likelihood of the observed sequence converges. This log-likelihood can be calculated using the forward variables as follows -

$$P(X_{1:T}) = \sum_i \alpha_T(i)$$

The above equation can also be used to calculate the likelihood of a new observation sequence under the learned parameters A, B, π. After we learn the model parameters for each HMM (corresponding to each gesture), we can test it on a new observation sequence. To do this, we calculate the likelihood of the observation under each model, and classify the new observation sequence as the gesture having the maximum likelihood.

III. Discussion

There were several practical considerations we had to take while implementing the above algorithm.

A. Calculations in log space

As can be seen from the equations in the previous sections, all of them involve repeated multiplication of probabilities. Since the probabilities are small values, they underflow while performing the calculations and everything results in zeros. This was tackled by performing all the computations in log-space. To transform the equations, simply take log on both sides, and use the logsumexp() routine to calculate log sums. This way, all the computations are in log-space, including the final computation of log-likelihood.

B. Multiple initializations of Baum-Welch

Since the EM algorithm converges to a local maxima, we can try multiple initializations of the EM algorithm to find a better local maxima. This can be done by running the EM algorithm several times, with different initialization values of the matrices A, B and choosing the result with the best final log-likelihood.

C. State transition matrix form

The format of the state transition matrix is also important while running the Baum-Welch algorithm. The full state transition matrix is equivalent to saying that the observations can go from one hidden state to any hidden state. A more restricted format can be of left to right HMMs where the states are constrained to go from one state to the next one only, in sequence. This can be important when we believe that the underlying model does not need to a full matrix, for example in gesture recognition, there is one way to make a gesture and one would not move from one state to another arbitrarily. This has important consequences when we have limited amount of training data, or the algorithm for the full transition matrix is overfitting. In such a scenario, we would like to reduce the complexity of the training algorithm, which can be done by restricting the transition matrices with heuristics as above. In our implementation as well, we see that we get good results by restricting the transition matrix. We don't strictly follow a left to right model, but allow some skip connections (in the same order) as well. Additionally, we also allow the model to go back to the first state from the last, to help in recognizing gestures that are repeated in single observation sequence. Our matrix looks something like the randomly initialized matrix shown in figure 4. The entries with 0 indicate that those transitions are restricted. It is sufficient to initialize the matrix with 0s at the right place, and the algorithm will automatically ensure that they remain 0 until the final step of convergence.

```
array([[0.205, 0.379, 0.416, 0.   , 0.   , 0.   , 0.   , 0.   ],
       [0.   , 0.231, 0.654, 0.115, 0.   , 0.   , 0.   , 0.   ],
       [0.   , 0.   , 0.186, 0.348, 0.466, 0.   , 0.   , 0.   ],
       [0.   , 0.   , 0.   , 0.156, 0.451, 0.393, 0.   , 0.   ],
       [0.   , 0.   , 0.   , 0.   , 0.911, 0.07 , 0.018, 0.   ],
       [0.   , 0.   , 0.   , 0.   , 0.   , 0.503, 0.435, 0.062],
       [0.157, 0.   , 0.   , 0.   , 0.   , 0.   , 0.33 , 0.513],
       [0.183, 0.619, 0.   , 0.   , 0.   , 0.   , 0.   , 0.198]])
```

Fig. 4. Random initialization of state transition matrix A, with N = 8. Entry (i, j) represents the probability of transitioning from state i to state j

D. Cross-validation results

As mentioned previously, we need to find the values of N and M - the number of hidden states and the number of discrete observation states. These numbers have to be chosen such that they represent the data we are using to train our HMMs. It can simply be done by using hold-out cross validation, where we fix some data as validation set and train on the rest of the data. The results on the validation set indicate which parameter combination should be chosen.

Table I shows the results of cross validation with different values of N and M. The state transition matrix is not made zero for any state combination. Table II shows the results for the same values of N and M, but with a state transition matrix closer to the one depicted in Figure 4.

TABLE I

% CORRECT CLASSIFICATIONS WITH FULL STATE TRANSITION MATRIX

N↓/M→	15	20	24	28	30	34	36
6	0.88	0.55	0.66	0.77	0.66	0.55	0.77
7	0.88	0.66	0.88	0.66	0.77	0.66	0.77
8	0.88	0.77	0.77	0.77	0.66	0.77	0.55
9	0.88	0.88	0.77	0.77	0.77	0.77	0.66
10	0.88	0.77	0.88	0.66	0.66	0.77	0.55
11	0.88	0.77	0.66	0.77	0.77	0.66	0.66
12	0.88	0.77	0.88	0.77	0.55	0.66	0.66

TABLE II

% CORRECT CLASSIFICATIONS WITH PARTIAL STATE TRANSITION MATRIX

N↓/M→	15	20	24	28	30	34	36
6	0.88	0.77	0.66	0.66	0.77	0.66	0.55
7	0.88	0.88	0.77	0.77	0.66	0.66	0.55
8	0.88	0.77	0.77	0.77	0.77	0.55	0.66
9	0.88	0.88	0.66	0.77	0.77	0.77	0.55
10	0.88	0.88	0.66	0.66	0.77	0.55	0.66
11	0.88	0.66	0.77	0.66	0.77	0.55	0.66
12	0.88	0.88	0.77	0.66	0.55	0.66	0.66

There are two important observations from the tables I, II. First, we note that there is not much difference in the number of correct classifications made by the full matrix and the partial matrix similar to left to right HMM. Second, we see that in general, the performance is not that great for the range of values of M shown in the table, but is better for the lowest value of M (i.e. 15) in the table. This indicates that we should test with lower values of M.

The results for lower values of M and N are shown in the table III. These results are the average of two hold-out cross validation sets where different examples were held out for cross validation in the two scenarios.

TABLE III

% CORRECT CLASSIFICATIONS WITH PARTIAL STATE TRANSITION MATRIX

N↓/M→	6	7	8	9
3	0.93	0.93	0.93	0.89
4	0.93	1	0.93	0.87
5	0.93	0.93	0.94	0.67
6	0.93	0.79	0.81	0.73
7	0.93	0.86	0.81	0.80
8	0.93	0.93	0.93	0.83
9	0.93	0.87	0.93	0.94

We see a lot better performance for these values of N and M. The values in the range 0.93-0.94 indicate that only one example out of the validation set was misclassified. For many, this misclassification was related to the Beat 3 and Beat 4 classes for which we had only 2 observation sequence, and if one is kept in test, there is only one sequence left to train. Assuming that the final training will be done on all the observation sequences, there will be enough data to make the above parameters a good enough choice.

In fact, we choose the parameters $(N, M) = (8, 8)$ for the submission. This is because the obvious choice $(4, 7)$ has

very few states and it was felt that with more data, this could be quite less complex to model the gesture data effectively.

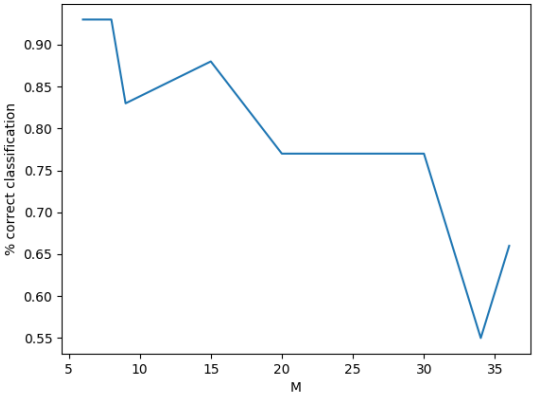

Fig. 5. % correct classification for N = 8

IV. Results

This section contains the results on the test set. The results for each of the files is shown in the following list -

- test11.txt - Beat 4
- test12.txt - Infinity
- test13.txt - Beat 3
- test14.txt - Eight
- test15.txt - Infinity
- test16.txt - Circle
- test17.txt - Wave
- test18.txt - Beat 4

References

[1] Rabiner, Lawrence R. "A tutorial on hidden Markov models and selected applications in speech recognition." Proceedings of the IEEE 77.2 (1989): 257-286.

[2] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.