Andrew Moran
andrewmo@mit.edu
3/18/15
6.835: Project 3

1. After examining the gestures with the visualization GUI, there were some noticeable differences between how gestures of the same type were performed. Some of these key differences were related to the positions/orientations of other joints that were not directly involved with the gesture and/or the global starting and ending positions of the gesture. For example, in gesture #9, the right hand points at different locations in front of the user (upper left, lower right, etc.). For gestures #8-9, rotation of the hands had different starting positions. In addition, some rotating gestures involved twisting the body as well, while others mainly moved only their arms. Also, the zooming gestures #6-7 had different hand starting positions when they completed the gesture. Some started at angles 0,π while others were at π/4, 5π/4 (however, they were generally always 180 degrees apart). These are only a few differences out of many noticed in the visualization GUI.

2. The "chance performance", without looking at the test data would reflect our performance when randomly guessing what to label a gesture. Since there are nine gestures total, the probability of labeling a gesture correctly with a random guess would be 1 in 9 chance (.11%).

3. The nearest neighbor algorithm can only be performed on fixed-length gestures because it requires calculating the L2 norm on gestures. As a result of calculating the norm, it would give another vector/matrix that represents the Euclidean distance between the two vectors/matrices. If they were not the same length, then there would be components missing and no way to calculate the distance from an extra dimension that's defined in one vector but not in the other. Therefore, it is necessary to normalize the vectors so they are of fixed length to get an accurate Euclidean distance between the two in question.

   Below was my approach to how I completed the `normalize_frames(data, nsamples)` function. First, I needed to determine whether frames needed to be added or removed if the number of columns in the provided `DxN` matrix were less than or more than `nsamples` respectively. Next, I created a base ratio that only gave the remainder (digits after the decimal point when dividing `nsamples/N`). The modulus operator was necessary to maintain precision. This ratio is a threshold that says "Should I add/skip at this ratio*`[1,N]` frame?". Next, for each `DxN` matrix, I looped through all the original frames `N`. If skipping frames, I only kept a frame if the index*ratio is less than the original ratio. If adding, I add all the original frames but repeat the same frame if index*ratio is less than the original ratio. Below illustrates my idea and how it best normalizes 12 frames given `nsamples`.

n		i = 1	2	3	4	5	6	7	8	9	10	12	12
7	Remove frames with *Remainder < 7/12*	**7/12**	**2/12**	**9/12**	**4/12**	**11/12**	**6/12**	**1/12**	**8/12**	**3/12**	**10/12**	**5/12**	**0/12**
14	Repeat frames with *Remainder < 2/12*	**2/12**	**4/12**	**6/12**	**8/12**	**10/12**	**0/12**	**2/12**	**4/12**	**6/12**	**8/12**	**10/12**	**0/12**

4. Below summarizes the accuracies of `test_nn(normalize_frames(gestures,n) 0.4)`. The accuracy for n=20 was .6593 and the best accuracy was at n=28 with .6889.

n=10	12	14	16	18	**20**	22	24	26	**28**	30
.6519	.6296	.6741	.6667	.6667	**.6593**	.6593	.6593	.6444	**.6889**	.6667

For the implementation of label classifying, I first normalized all the frames in the training set to match the fixed-length of the test matrix. Then for each `Dxn` matrix in the normalized training set, I calculated the L2 norm from the test matrix. The gesture with the `Dxn` training matrix that provided the minimum distance from the test matrix was then used label that test matrix. I used Matlab's built-in `norm` function.

Nearest neighbor for gesture recognition has some of its pros and cons. The pros are that it's simple and powerful. No added parameters needed, simply calculated the minimum distance. Also, there is no training involved. The cons are that it could be expensive and slow. As the number of examples increases and with many dimensions, the amount of computations increases linearly. To determine the nearest neighbor for a gesture matrix, must compute all the distance to all the m training examples, which could be costly.

5. Below compares the accuracies after changing the `nbHiddenStates` and `nbGaussMixtures` parameters for the `experiment_hmm` function.

Type	# Hidden States	# Gaussians	Accuracy
Default	6	3	0.7630
Best	7	7	**0.7926**

The best result I came up with is `nbHiddenStates=`7, `nbGaussMixtures=`7, `accuracy=`0.7926. Exceeding 8 states resulted in singular matrices that could not be inverted in the HMM calculations. I originally looped through all possible hidden states and Gaussians from one to ten, however, ran into singular matrix errors. Then, I looked at the accuracy for all hidden states with just one Gaussian mixture. This gave me a ballpark of where adding Gaussian mixtures could potentially improve the current number of hidden states used. I found that with `nbGaussMixtures` =1, the results of having `nbHiddenStates=`6 had an `accuracy` =0.6519, where as `nbHiddenStates=`7 had an `accuracy` =0.7407. There and behold I found that with 7 states, accuracy was optimized with 7 Gaussians at `accuracy` =0.7926.

6. The first-order Markov assumption declares that when determining the next state for a process, it only depends on the previous (single) step only. There are some pros and cons when using a first-order Markov process. When using the first-order Markov model assumption, it allows the HMM to learn temporal dynamics because

it captures maximum likelihood (ML) calculation only from the previous state. Another benefit is that if the model is not known a priori, the states could still be estimated using ML. If the dynamics were truly governed by a first-order process, then any higher ordered models will closely correlate to the first-ordered one. However, not all systems are first order. Therefore, as a disadvantage, the maximum likelihood model may not be the best depiction of the system. Also, much calculation would be involved for higher ordered and continuous processes as compared to first-order discrete ones.

7. I believe that each latent variable represent an individual gesture. As mentioned before, our process of gesture recognition takes use of features as 3D positions of your body joints. So the observation is the current pose of your body. The hidden state that causes this observation would most likely be the gesture you are attempting to complete. So a hidden state would be a gesture (e.g. pan left, rotate right), and that causes an observation of a sequence of body poses (given as a set of matrices).

8. The structure of `trainHMM` and train_`hmm` are fairly similar. They start off initializing the parameters used to feed into the `mhmm_em` function. Initialization of some of the variables requires constructing stochastic matrices, multivariate Gaussians, and normalizations. The main function to analyze is the `mhmm_em` function. As the name suggests, it completes the main algorithm used in HMM, which is the Expectation-Maximization (EM) algorithm. This takes into consideration the prior probabilities and computes the maximum likelihood given the previous states and Gaussians. These likelihoods are then updated according to the M state. Below describes the input and output values of the `mhmm_em` function.

   Inputs
   Let Q = hidden states and M = Gaussian mixtures
   - `prioir0` = prior (initial) starting probability. Pr(Q(1) = j)
   - `transmat0` = state transition matrix. The probability of transitioning to a new state given the current one. Pr(Q(t+1)=j | Q(t)=i)
   - `mixmat0` = Gaussian mixture matrix. Probability of each Gaussian given current state. Pr(M(t)=k | Q(t)=j)
   - `mu0` = mean of Gaussian. E[Y(t) | Q(t)=j, M(t)=k]
   - `sigma0` = covariance of Gaussian. Cov[Y(t) | Q(t)=j, M(t)=k]

   Outputs
   - `ll` – log likelihood. Probabilities utilizing prior and posterior when completing the EM step for HMM
   - `prior` – adjusted prior
   - `transmit` – adjusted transition matrix
   - `mu` – mean of final multivariate Gaussian
   - `sigma` – covariance of final multivariate Gaussain
   - `mixmat` – adjusted Gaussian mixture matrix

9. We have now expanded out features so that out matrices include velocities in addition to the original positions. Therefore our new matrices are of size `2DxN`. Now we want to compute the best accuracy adjusting the window w, `nbHiddenStates`, `nbGaussMixture`. First I started computing the best window for the default 6 states and 3 mixtures. The best w=7. Afterwards, I experimented with changing all the three parameters. I started with my previous best and additional tested further but only changing these parameters slightly. Changing all parameters was very time consuming. If running this job in parallel, I could have been more rigorous. Shown below, I got the best accuracy with default parameters.

Default

w	Response
0	HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.711111
1	HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.807407
2	HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.851852
3	HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.874074
4	HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.851852
5	HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.837037
6	HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.911111
7	**HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.955556**
8	HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.925926
9	HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.874074
10	HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.814815
11	HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.733333
12	HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.762963
13	HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.785185
14	HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.792593
15	HMM: nbHiddenStates=6, nbGaussMixtures=3, accuracy=0.807407

Previous Best

w	Response
0	HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.740741
1	HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.718519
2	HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.844444
3	HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.837037
4	HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.911111
5	HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.918519
6	HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.911111
7	**HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.925926**
8	HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.911111
9	HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.851852
10	HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.800000
11	HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.844444
12	HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.792593
13	HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.785185
14	HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.772325
15	HMM: nbHiddenStates=7, nbGaussMixtures=7, accuracy=0.807407