ECE5242 Project 2: Gesture Recognition
Code Due Date: **3/3/2022 at 9:25am** on Canvas, <NetID>_project2.zip
Report Due Date: **3/7/2022 at 11:59pm** on Canvas, <NetID>_project2.pdf

In this project, you will use IMU sensor readings from gyroscopes and accelerometers to train a set of Hidden Markov Models (HMMs) and recognize different arm motion gestures. You should then be able to classify unknown arm motions in real-time using your algorithm.

Training Data Download: Now available!
Repeated gestures at: https://cornell.box.com/v/ECE5242Proj2-train
Single gestures (same format as test): https://cornell.box.com/v/ECE5242Proj2-train-additional
Test Data Release: 3/7/2022 9:25am, https://cornell.box.com/v/ECE5242Proj2-test
The data sets contain the raw IMU readings collected from a consumer mobile device and corresponding labels that describe the arm motions associated with the movements. You can find the coordinate system used here:
http://developer.android.com/reference/android/hardware/SensorEvent.html
*Data format for each column (7 columns in total) in the IMU data is ts , Ax, Ay, Az, Wx, Wy, Wz (Time (millisecond), 3x Accelerometer (m/s2), 3x Gyroscope (rad/sec))

Upload: on Canvas:
(1) Code (due 3/3/2022 9:25am, <NetID>_project2.zip) : Do not include training/test data. Include all your training and testing code. Include your trained models. Your readme should give specific, clear instructions what file paths to update and how to run your code in order to apply your trained models to test data (without needing to re-train your HMM on training data). Your readme should also give a brief description of all files you included.
(2) Write-up (due 3/7/2022 11:59pm ,<NetID>_project2.pdf) : Write a project report including the following sections: Introduction, Problem Formulation, Technical Approach, Results and Discussion. **For each gesture model, you must include your training plot of your log-likelihood per epoch in your results.**
*Clearly presenting your approach in the report and having good algorithm performance are equally important.

Collaboration:

From the syllabus:

> All work that is handed in for this course should be performed solely by the individual students. However, you are allowed to discuss approaches and methods with your colleagues, but any collaborative influences should be properly documented in your reports. This applies to code that you hand in as well—submitted work must be original. Violators will receive a failing grade and be reported to the appropriate disciplinary office; see the following site for more details: http://theuniversityfaculty.cornell.edu/academic-integrity/"

If you do get useful ideas from other classmates in your discussions, please list the classmate(s) near the beginning of your writeup along with a tiny summary of the idea/help. **You should not let other students see your code and you should not look at another student's code.** If you do need to look up some syntax for a minor coding issue online, please include the url of the resource you used in a comment directly above the relevant lines of code in your submission. If you include example images from online sources in your writeup, please cite the source directly next to the image.

Project Steps:

1) VISUALIZE AND QUANTIZE. Visualize the raw data and experiment with discretizing the raw sensor information. You should discretize your data observations using k-means clustering to transform your continuous observations to discrete observations. We recommend 50-100 observation clusters. You can then try to manually split the recordings of repeated gestures into individual gestures by inspecting the time series of the observations.

2) IMPLEMENT HMM. You will use hidden markov models (HMMs) to classify gesture data. You will learn one HMM for each gesture. The HMM can compute the probability of a sequence of observations being generated by that gesture. The classification prediction across gestures will simply by the argmax of all the gesture models, that is, the model that assigns the highest probability to the sequence of observations.

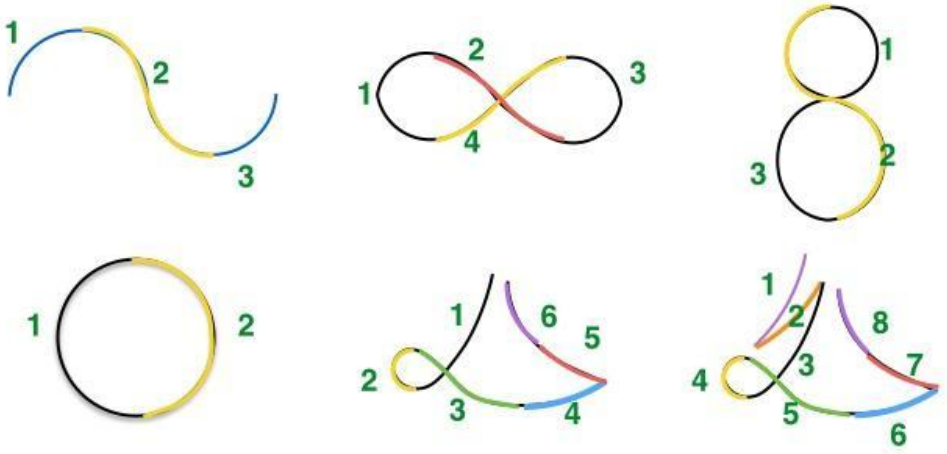
Within your HMM, a hidden state will correspond to a component of a gesture (like "swooping down", "curving left", "curving right", etc.). Each hidden state has some discrete probability distribution for the (discretized) observations you might see in that state. For example: during "swooping down", the discretized IMU data might have some distribution of probability split between several different observation clusters, due to noise or variation between examples. The observation model is the probability of a measurement from a particular hidden state falling within a particular observation cluster. We recommend using 10-20 hidden states and 50-100 observation clusters.

Write down the corresponding model parameters that will need to be learned from the training data. Use the Baum-Welch algorithm to learn the model parameters. **Read the Rabiner HMM Tutorial** posted under the Files tab (Reading Materials folder) on Canvas. Additional hints on the notation used in that paper are available at http://alumni.media.mit.edu/~rahimi/rabiner/rabiner-errata/. **Be sure to include a plot of your log-likelihoods generated during your training process in your writeup.**

3) TESTING. You should then make sure that you can apply your saved models to input sensor readings from unknown gestures. You can then compute the log likelihood under the different HMM models, and then show how confidently you can classify the unknown motion as a particular gesture by comparing the log-likelihoods from the different models. We will release test data on the due date of the code. Please include the results of your classification of the test set in the report including the associated log-likelihoods**. Accuracy of the test set predictions will be worth 50% of the points of the coding portion of the project** (which is half the total points including the report). **Include the top 3 predictions for each test gesture** (we'll give partial credit if your 1st is wrong but 2nd is right).

Tips:

1) The data contains six different motions: Wave, Infinity, Eight, Circle, Beat3, Beat4. The names indicate their corresponding motions. (Disregard the numbers by the motions). The first set of training data contains motions repeated multiple times. The second set of train data contains each motion performed once. The test data will contain motions only performed once each.



2) Experiment and visualize your work while discretizing the raw data. This will give you an idea how to use the data and what features you are extracting. Feature extraction, as you've learned in Project 1, is a very important part of the learning process. One way to get discrete observations from your continuous features is to run k-means clustering, and each new state will be assigned to its closest cluster. (Advanced note: You don't *have* to do vector quantization if you want to use a continuous emission (eg: gaussian emission) HMM. But we strongly suggest doing vector quantization to make your HMM training easier.)

3) Choose the number of hidden states (N) and observation classes (M) with cross validation. A good starting range to think about is N between 10 and 20 and M between 50 and 100.

4) Try different initialization strategies

5) Like in Project 1, you can set a maximum number of iterations or use a threshold to detect convergence of the learning algorithm.

6) Be aware about underflow issues. You can either use do your calculations in logspace or use the scaling procedure described by Rabiner in Section V of the paper.

7) Consider emission probabilities of 0.
If you never see a particular observation during gesture training, the emission probability of that observation would be set to 0 during Baum-Welch. And then, if during test-time you see that observation at some timepoint in your test data, your model thinks the probability of the entire trajectory is 0.

Why is that a problem?
This is a problem if you want to compare two models to see which one is more likely and both return a probability of 0. You can't compare two 0's to figure out which is bigger.

What can you do about it:
One solution is to insist that your emission model never return exactly 0 probability. For example, you could add 1e-8 to your emission probability model table to ensure that no probability is exactly zero. Intuitively, this would be adjusting your model to say "it's really really unlikely to see this observation, because we've never seen it in the training data" rather than "it's impossible to see this observation, because I've never seen it in the training data".

8) If you're struggling to see if your HMM training is correct, first check to make sure your log-likelihood training plots make sense to you. Additionally, you can come up with example toy data to see if your algorithm converges correctly. You could use something like the following, very simple toy data:

1) The data is of a process that stays in hidden state 0 for 1000 steps, then in hidden state 1 for 1000 steps, then in hidden state 0 for 1000 steps.

2) Each hidden state has emission probabilities:

In state 0: always emit observation with label 0

In state 1: emit observation with label 0 half the time and 1 half the time.

The python code to generate the associated observations for that toy example is:

```
X=np.concatenate((np.zeros((1000,1)),np.random.randint(0,2,size=(1000,1)),np.zeros((1000,1))))
```

The model that best fits this data should look something like:
Transition matrix:

[[0.999, 0.001],
[0.001, 0.999]]

Emission matrix: (row index indicates which state you're in, column index indicates which observation you're emitting).

[[1.0, 0.0],
[0.5, 0.5]]

Since there's randomness involved in the test data (and since there's the possibility that your Baum-Welch gets caught in a local maximum), it's OK if your results aren't exactly the optimal values, but being often close to those optimal values is a good sign that your HMM training algorithm is probably working.

9) Use Matrix Multiplication

After you're done getting your code to work, you may find that your code is very slow. If you do, this tip may be useful for you to speed up your code without changing any of the mathematical calculations it does. In python, matrix multiplication using numpy's dot product function is much much MUCH faster than if you write matrix multiplication by hand using a for loop. (About a thousand times faster). So, if you have any places in your code that look at all like:

```
new = np.zeros((50,100))
for i in range(50):
   for j in range(100):
       for k in range(50):
           new[i,j] += A[i,k] * B[k,j]
```

You really should replace that with something that looks like

```
new = A.dot(B)
```

10) Consider using a left-to-right HMM transition structure to simplify training (only if you've split apart the repeated training gestures into individual training gestures).