ESE650 Project 3: Gesture Recognition

Joe Trovato

February 26, 2015

Introduction

The gesture recognition problem involves addressing a few key issues: feature generation, state tracking, classification of different gestures. Feature Generation is handled mostly by the IMU data provided. Additional processing, such as orientation estimation, can be used to augment the original features. State tracking is the most important part of the gesture recognition pipeline. State tracking allows a gesture to be decomposed into seperate parts or states. This portion is handled by a Hidden Markov Model (HMM). This model, specifically a left-right HMM, solves the time dialation problem as well as models the sequential state progression of gestures. Before optimal performace is achieved, these models must be trained. Specifically, a different HMM model is train for each gesture using the Baum-Welch Algorithm, the EM procedure for HMM parameter optimization. Finally, a new input sequence is classified by taking the Maximum Likelihood Estimate (MLE) over the different gesture models. Using this pipeline, I was able to acheive very accurate classifcation over new instances of the 6 training gestures.

Algorithm

Data Preprocessing

The data supplied for training did not need to be scaled or biased as in the last project. This significantly reduced preprocessing and allowed me to rely more heavily on the raw measurements. I implemented a simple low-pass filter in an attempt to reduce noise, but saw no improvement in recognition accuracy.

Feature Generation

Feature generation is an area of this project I left realatively unexplored. The possibilities of feature generature are enormous. For instance, orientation encodes information that accelerometers and gyros often hide. My UKF from project 2 was slightly too unreliable to generate orientation estimates that I could include in my features. And the results I saw were accurate to the point

Mapping Sensor Data to Observations: K-Means

One of the largest issues was sanitizing the dirty, continuous data from the IMU. The standard HMM framework requires discrete states and discrete observations in order to form the transition probability matrix and the emmission probability matrix. To get our data in this form, I used Kmeans clustering with K=45 over the entire dataset. The entire dataset means all training data for every gesture. This way, the centroids of KMeans define a discrete alphabet of observations for any possible input data. When classifying a new gesture, the same centroids must be used and each data point in the new trial must be assigned to a cluster, which will represent the observation seen at that timestep. Since this assignment is basically 1-nearest neighbor, I used a 1nn to classify new observations.

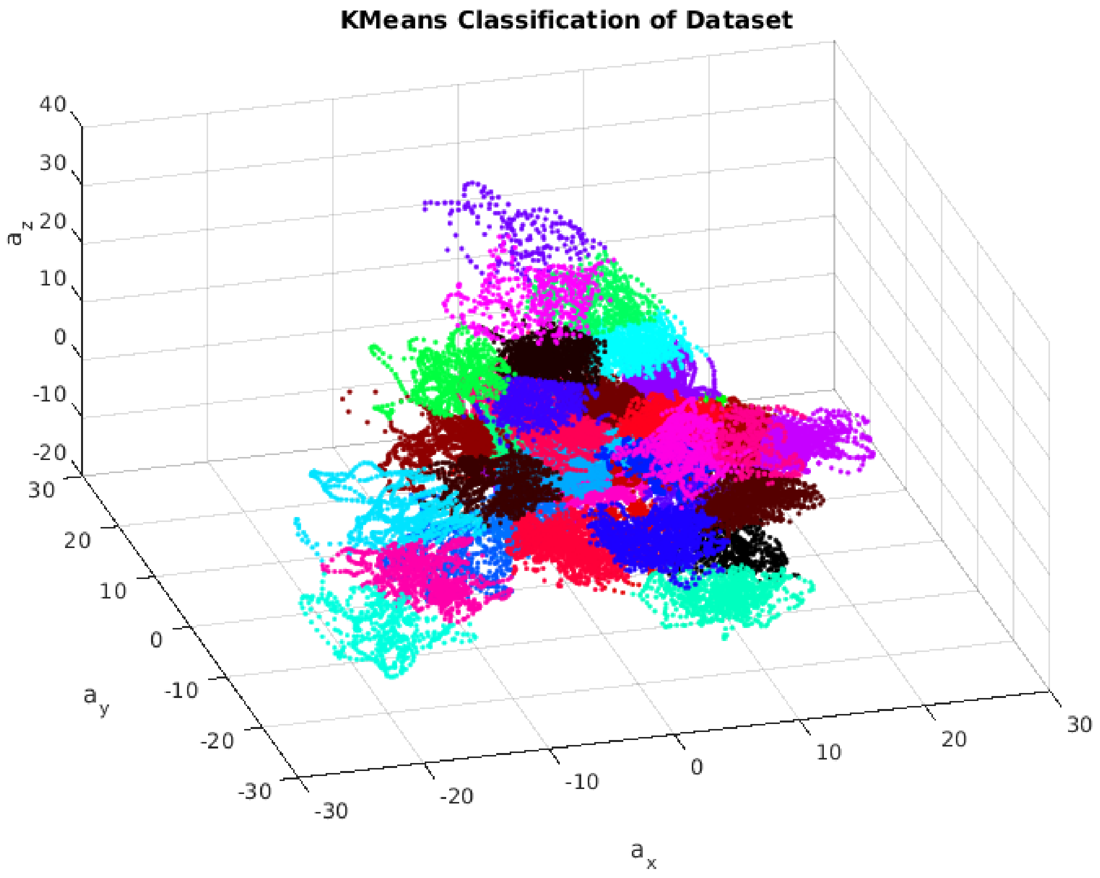

Figure 1: The entire dataset clustered using KMeans (K=45). These clusters represent discrete observations.

Training Hidden Markov Models

In order to accurately distinuguish gestures from each other, HMMs of each gesture must be trained. With a model defined by π, A, and B the Viterbi Algorithm can then be used to estimate the log likelihood that a certain model generated a specific observation sequence. Backtracking, the Baum-Welch Algorithm, as described in the Rabiner paper, was used to train the model parameters for each gesture. The Baum Welch algorithm is a specifc case of the Expectation-Maximization (EM) algorithm. The expectation step involves using the forward backward algorithm to estimate probabilites of being in a specific state. The Maximization algorithm involves using statistics over the estimates produced in the E-Step to update the values for π, A, and B.

My additions to the algorithm were the convergence checks between iterations and underflow scalings. Using the Viterbi Algorithm, I was able to get a log likelihood estimate of the probability that the current iteration's model produced the training observation sequence. If the difference in log probability was less than some $\epsilon = 0.000001\%$ of the previous log probability then the algorithm converged. An edge case occured when rounding error caused the log probability to decrease (which is not possible according to EM). So whenever this situation occured, I also halted the algorithm.

Another issue was underflow. When multiplying probabilities, my matix elements quickly went to 0 and *Nan* causing the whole algorithm to crash. This was solved by scaling the probabilities before storing them and them storing the scale and probability seperate. This way probabilities can be added directly and there scales can be used to get back to the correct value. Similarly, logs were used during the viterbi algorithm calculations to prevent underflow when calculating the likelihood of the model and the predicted state sequence.

Classification: MLE over HMMs

After the HMMs were trained, classification was relatively simple. The gesture was detemined by taking the MLE of the observations given the different gesture models. The model with the highest probability of generating the sequence was then used as the label. The equation is below:

$$gesture = argmax_{A,B,\pi} \quad P(observations|A_i, B_i, \pi_i)$$

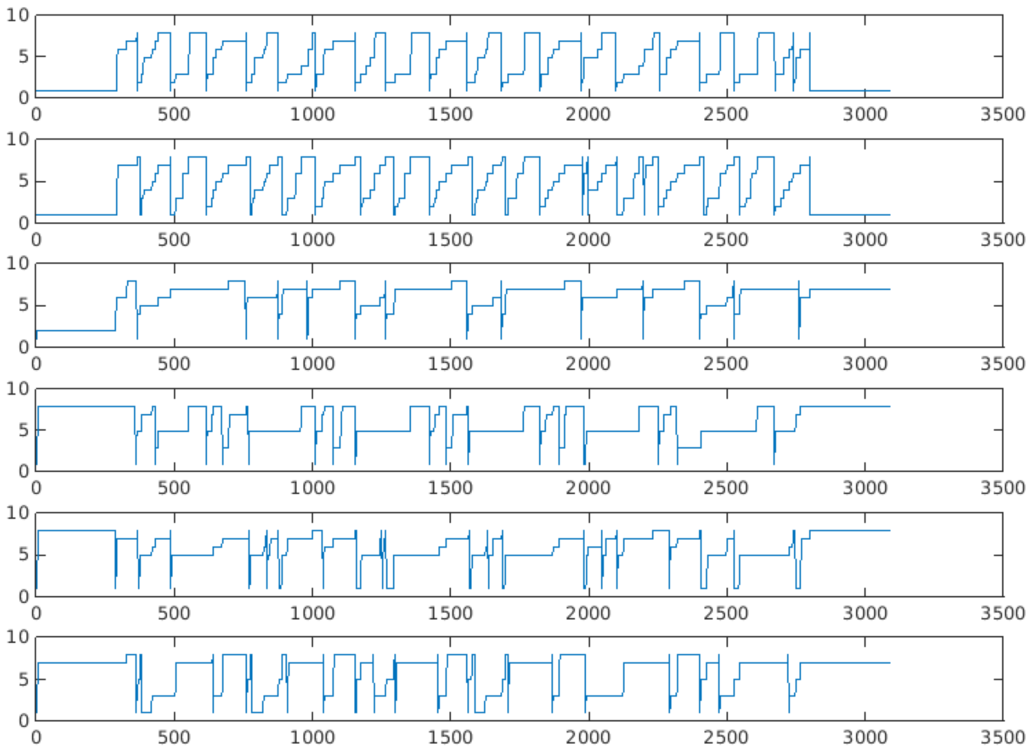

Figure 2: The MLE state sequence for each gesture HMM

Evaluation: Confidence and Cross Validation

Because this project has no visual output or verification, evaluation is extremely import in assesing the performance of the algorithm. First, to get a better idea of the how good the classification was, I implemetned a confidence metric. The equation is below. Note that the core fraction must be inverted becuase the log likelihoods are all negative. This gives and idea of how 'good' a particular classification is. For example, because the 3 beat and 4 beat gestures are similar, these gesture have a lower confidence than a more unique element. I also implemented a simple rank to verify this. All of this information is printed to the console when running on test data.

$$confidence = (\frac{largest\ likelihood}{2nd\ largest\ likelihood})^{-1}$$

With these evaluation metrics in place, I was able to do a simple 5-fold cross validation on the dataset we were given. I held out a random file from each gesture folder and trained on the rest. With this I was able to tune K for the inital KMeans clustering and N, the number of states in each HMM. After some small tweaks, I saw 100% accuracy rate over the 5 folds. After decreasing my training data percentage, I saw slight errors in classification, but was convinced of my algorithm's performace.

Results

We'll see when the test set comes out...

Demo Instructions

1. specify test directory by editing test_dir_name at top of test_script.m
2. set train = 0 to load previously trained models or set train=1 to train from the training data.
3. run demo.m, classifications should be output to console