Frans Fourie

Intelligent Autonomous Systems

IMU Hand Gesture Recognition

Contents

Table of Tables

Table of Figures

Introduction

This report documents the creation of a model that can be trained to identify specific hand gestures. The data used to train the model and form predictions are the IMU data from a handheld device held by a person performing the gestures. Three machine learning algorithms where used to create the model namely K-means (K-means clustering), KNN (K-nearest neighbors) and HMM (Hidden Markov model). K-means is used initially on all training data to find clusters and identify cluster centers. KNN is then used to match all the training data to their closest cluster center and save the results. These results are then used to train the HMM's that can be used for predictions. In order to then predict the gesture of testing data the data needs to be fed to the KNN algorithm for classification to a cluster. By making use then of the classified testing data and the trained HMM's it is possible to predict the gesture of the test data. Predictions are ranked using the loglikelihood with the gesture model that returns the highest loglikelihood being the prediction of the gesture. By then looking at the models returning the 2nd and 3rd largest loglikelihoods we can also make a 2nd and 3rd most likely gesture guess.

Problem Formulation

Given IMU data from a handheld device that was held by a person performing hand gestures create a set of HMMs that can be used to identify the same hand gestures given new IMU data.

Technical Approach

In the following section the technical approach is documented. This include the featured analysis, HMM model and test results.

Feature analysis

In order to solve the problem and be able to successfully identify new gestures the first step was to identify smaller gestures inside the larger gestures that acted as building blocks for the main gestures. These building blocks would be the smaller gestures and by taking different combinations of them in different orders you would be able to reconstruct all the main gestures. To identify these features of the main gestures the K-means learning algorithm was used.

Using the K-means algorithm to identify the features of the training data proved effective. The implementation of K-means would take the training data as input as well as a value for the number of clusters to create and output a csv file containing the values of the cluster centers found. The program would load all the different training data files and combine them into one large dataset that would be the input data to the algorithm. Deciding on the number of clusters to use was hard considering one could not visually graph the data and identify clusters using that method. Cross validation was necessary to identify a good value for the cluster count. The problem was deciding what to use as the metric for the cross validation as there was not a single obvious score to use. Things that were considered where counting the loglikelihood of all the top predictions for each file together, but this would not work. As using cross validations would mean that you would have different files each iteration and there was no obvious way to compensate for some gestures yielding much higher loglikelihood scores than other gestures. The second consideration was using the top 3 predictions for each gesture and counting them together, but this had the same problem as the previous idea. Thus, inspection was used to determine

the number of clusters to use. But in order to use inspection to determine the number of clusters the KNN algorithm had to be implemented first.

The KNN algorithm was used to match data points to their closest cluster center. The algorithm takes as input the cluster centers that where created by the K-means algorithm as well as the data files to classify. The KNN algorithm then finds the closest center to each of the data point using the Euclidean distance. The program then generates as output an output file corresponding to each input file where the output file is a one-dimensional array with the number of the cluster that each data point belongs too.

Now by running the K-means algorithm on the training data files containing the continues datasets for the gestures it was possible to experiment and identify an appropriate number for the number of clusters. By running K-means on the training data and then feeding the data into the KNN algorithm with the found cluster center file corresponding files where generated listing the cluster each gesture belonged too. Now to use inspection to determine the number of clusters to use it is necessary to think about what type of output is expected and how the output of the KNN algorithm should look. Considering that the logic is that clusters represent smaller gestures that can be combined to create the large gestures it holds that the output files are expected to show this. Thus, when looking at the output files it is expected to see a group of data points which are in the same cluster before a next group of data points belonging to the same cluster and so on. The logic was to use as many clusters as possible to increase the number of features identified. So, by starting at a large number for number of clusters and then looking at the output files and seeing there is erratic changes between cluster centers between datapoints without clear groups of data points being visible indicating to lower the number of centers. By following this approach until clear groups of data points where visible allowed the identification of 50 clusters as an appropriate amount to be used in the K-means algorithm.

The K-means algorithm works by starting of taking random data points within the dataset as the initial cluster centers. The Euclidian distance is then calculated between every data point and every data center. The average value of all the datapoints belonging to a cluster is then taken as the new cluster center. The new centers are then used, and the process repeated. This is done for 100 iterations or until convergence.

Hidden Markov Model

Given that in this problem the datasets are sequential with the observational data being known but nothing is known about the states it is a perfect application for HMMs. Assuming there is a model underlying this sequentially observed data that model is a Markov Model. Thus, by using the given data it is possible to find the Markov Model that underlies the data. After construction of the model it is possible to take new data and match it to the model seeing how good of a match to the model the new data is. Given a strong match between the data and a model it is likely that, that data has the same model underlying and can thus be classified to that model's classification.

To obtain the hidden parameters of the HMMs the Baum Welch algorithm was used. The algorithm required as input the quantized data found during the feature analysis as well as initial Pi, A and B matrixes. These matrixes are the A matrix which is the state transition matrix which gives the probability of moving from one state to another. The B matrix is the observation matrix which gives the probability that a state is observed given a specific observation. Lastly the pi matrix is the initial state distribution.

To construct these initial parameters, it was first necessary to decide on the amount of hidden states that would be used for the model. Experimenting with different amounts of hidden states and looking how

they effected prediction likelihoods helped narrow down the range in which to choose the value. In the final range of hidden state values to choose from the values increased the prediction accuracy of some models while decreasing the accuracy of others. Thus, an average value was chosen but that leant towards maximizing the difference between predictions of beat3 and beat4 since they are the closest to each other and it would be preferred to increase the algorithms ability to distinguish between these two specific gestures. This led to the number states being selected as 10. Then the input parameter A which is in this implementation a 10 by 10 matrix was initialized with equal probability (1/10) in each entry in the matrix. The input parameter B which in this implementation is a 10 by 50 matrix is also initialized with equal probability meaning each entry in the matrix has value (1/50) given that 50 was chosen as the number of observation classes in the k-means implementation. Lastly the initial state distribution or pi matrix is simply initialized as a zero array of length 10 with the first element being set to 1 for an initial guess. The reason for using equal probabilities in the matrixes for initialization is because it places the initial parameters in a default state where all models to be computed should have equal chance of being close to the initial values. If the initial values are chosen as specific values closely matching a specific model the problem could arise that, that model is found faster but that now there are models far away which will take more iterations to find.

Given the initial parameters the Baum Welch algorithm starts with the forward backward calculations calculating the initial alpha, beta and c values. A small value of 10^(-10) is added to the c value here to ensure that there is always a small probability and not zero as a zero probability leads to Nan's which makes calculating the loglikelihood impossible. After these three values are calculated they are passed to the expectation step of the algorithm where gamma and xi is calculated. These values in turn are passed to the Maximization step where the A, B and pi matrixes gets updated before the next iteration of the algorithm. To calculate the loglikelihood error and check for convergence the sum of the logs of all the values in the c array is taken.

The program creates a model for each one of the gestures producing the A, B and pi matrix for each model. These matrixes get saved so they can be loaded later by the prediction program and used in making the predictions.

Algorithm 1: The Baum-Welch algorithm

Initialization: Θ_0, $\{O_{1:T}\}$

Looping:

for $l = 1, \ldots, l_{\max}$ **do**

1. Forward-Backward calculations:

$$\alpha_1(i) = \pi_i b_i(O_1),\ \beta_T(i) = 1,$$

$$\alpha_t(i) = \left[\sum_{j=1}^{K} \alpha_{t-1}(j) a_{ji}\right] b_j(O_t),\ \beta_t(i) = \sum_{j=1}^{K} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$$

$$\text{for } 1 \le i \le K,\ 1 \le t \le T-1$$

2. E-step:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{K} \alpha_t(j)\beta_t(j)},\ \xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}\beta_t(j))}$$

$$\text{for } 1 \le i \le K,\ 1 \le j \le K,\ 1 \le t \le T-1$$

3. M-step:

$$\pi_i = \frac{\gamma_1(i)}{\sum_{j=1}^{K} \gamma_1(j)},\ a_{ij} = \frac{\sum_{t=1}^{T} \varepsilon_t(i,j)}{\sum_{k=1}^{K}\sum_{t=1}^{T} \varepsilon_t(i,k)},\ w_{kd} = \frac{\sum_{t=1}^{T} \gamma_t(k,d)}{\sum_{t=1}^{T}\sum_{r=1}^{D} \gamma_t(k,r)}$$

$$\text{for } 1 \le i \le K,\ 1 \le j \le K,\ 1 \le k \le K, 1 \le d \le D$$

end

Result: $\{\Theta_l\}_{l=0}^{l_{\max}}$

Figure 1: Baum Welch algorithm equations

The above figure shows the equations that was implemented in the programs in order to create HMMs. It shows the definitions for alpha, beta, gamma and xi as they where taken and used in the program.

Optimization

A couple of things was done to optimize the speed at which the entire model could be trained, and new predictions could be made faster. The first optimizations to increase speed was to use thresholds for convergence for termination in combination with max iterations criteria to terminate the K-means algorithm and the Baum Welch algorithm. The second optimization that showed major increases in performance especially in the K-means algorithm was vectorization of the computations done. A user aimed optimization was splitting the model into 4 programs, the K-means program, the KNN program, the

Baum Welch program and the predicting program. By splitting the programs in this way, it is possible to run predictions on new testing data without having to retrain the models.

Results

The following are the results obtained when creating and running the model.

Model results

To create the models the loglikelihood was observed until convergence. When convergence was reached the model parameters (A,B,Pi) was saved.

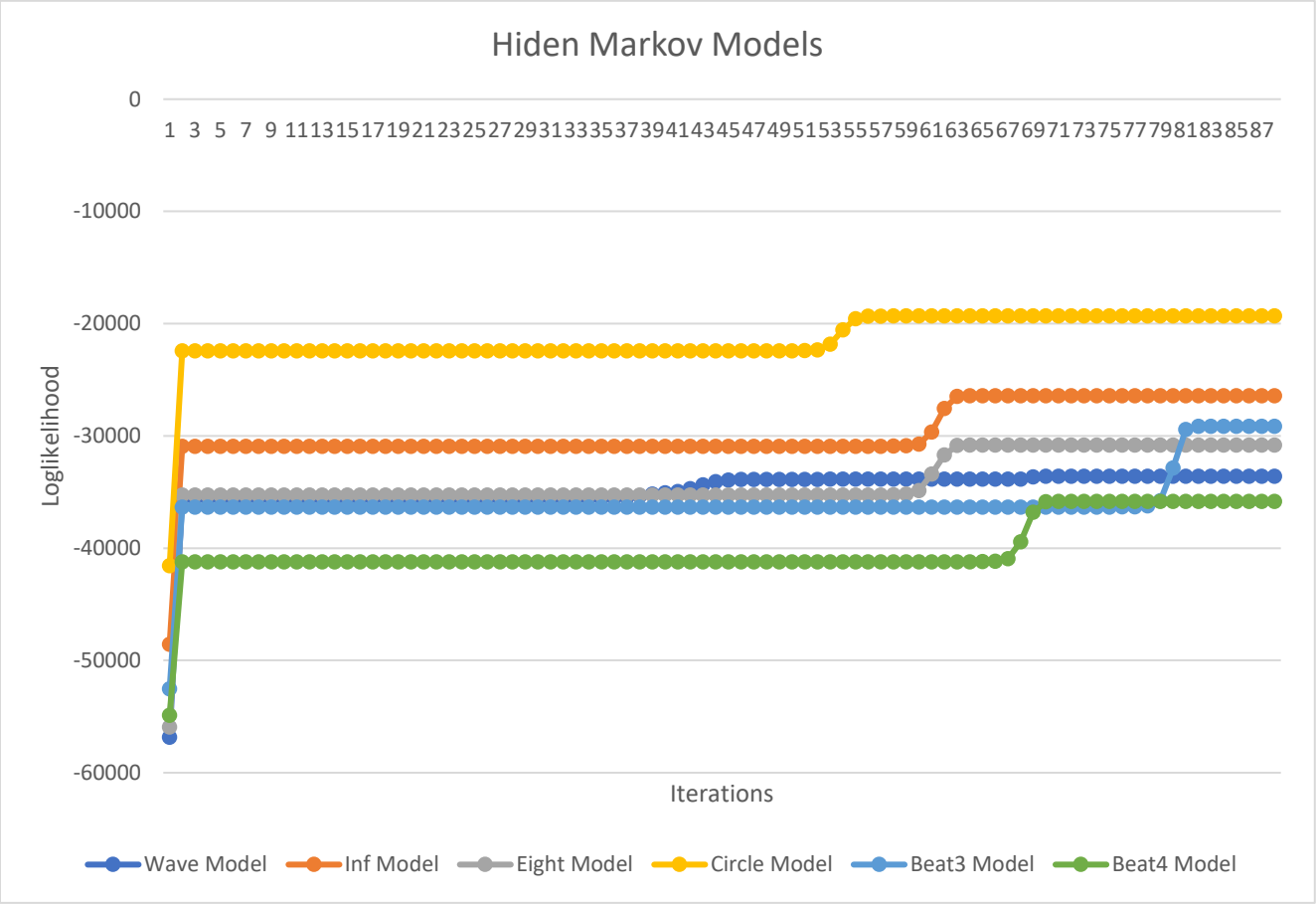

Figure 2: Loglikelihood of training HMMs

The above Figure 2 shows how the loglikelihood rises for the different HMMs as the Baum Welch algorithm runs for 88 iterations. From this figure it is clear that the models improve as the algorithm is run for more iterations. The loglikelihood stays nearly the same before there is a jump at some point for all the models and then staying the same again. Looking at the effect this improvement of the models has on the predictions made by the program it is observed that predictions stay nearly the same with rankings staying the same and small differences in the loglikelihood scores of the predictions. To improve performance of the program and maintain nearly the same performance a convergence threshold of 1 was set. The effect this had was that all the models reached convergence after 3 iterations producing the

below graph in Figure 3, which are the loglikelihood scores of the actual models used for predictions on the test data.

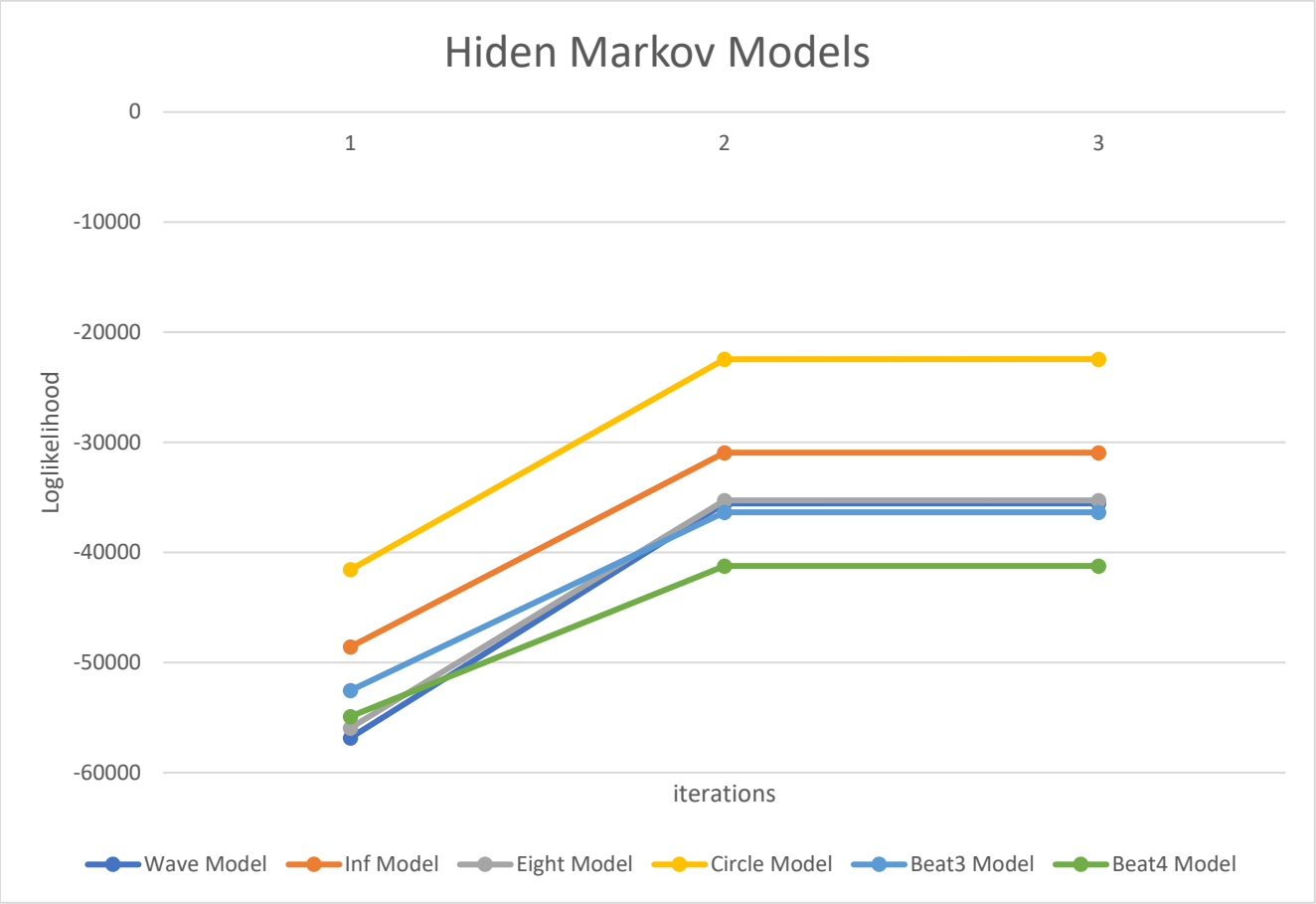

Figure 3: Loglikelihood of trained HMMs

Table 1: Loglikelihood scores of trained HMMs

Iteration	Wave Model	Inf Model	Eight Model	Circle Model	Beat3 Model	Beat4 Model
1	-56841.6943	-48575.59	-55945.841	-41573.0685	-52542.381	-54901.3309
2	-35545.9656	-30939.35	-35274.9552	-22443.7596	-36351.4996	-41231.3176
3	-35545.937	-30939.34	-35274.9494	-22443.7521	-36351.4974	-41231.3137

The above Table 1 shows the loglikelihood results obtained when training the HMMS using the Baum Welch algorithm.

Prediction Results

The trained models where used to predict the gestures preformed in the test data files. The predictions are ranked from top to bottom with prediction one being the best guess, prediction 2 being the second-best guess and so on. The predictions where ranked using the loglikelihood score obtained when using the data with the corresponding model.

Table 2: Test data loglikelihood scores

Model	Test1	Test2	Test3	Test4	Test5	Test6	Test7	Test8
Wave	-930	-14126	-12120	-10719	-14236	-14172	-10909	-13167
Inf	-6054	-15147	-2108	-10737	-14236	-2071	-7952	-14243
Eight	-5923	-18008	-9601	-16951	-14236	-11107	-2151	-17262
Circle	-7771	-18008	-17531	-16951	-1281	-18153	-15749	-17262
Beat3	-7249	-9037	-11352	-2576	-7342	-13138	-10258	-8630
Beat4	-7181	-2577	-12040	-9991	-6831	-13796	-10384	-2443

The above table shows the loglikelihoods gotten when running the different test data files with the different trained HMMs.

Table 3: Ranked predictions of test data

	Test1	Test2	Test3	Test4	Test5	Test6	Test7	Test8
Prediction 1	Wave	Beat4	Inf	Beat3	Circle	Inf	Eight	Beat4
Prediction 2	Eight	Beat3	Eight	Beat4	Beat4	Eight	Inf	Beat3
Prediction 3	inf	wave	Beat3	wave	Beat3	Beat3	Beat3	wave

The above table shows the top 3 predictions for each test data file.

Discussion

Looking at the above results as well as the results that where obtained when using the models to classify the single gesture training data files it appears that the model works quite well at distinguishing these gestures. There is large score differences between the first and second predictions meaning that there is quite a high level of certainty in the first prediction. The Gestures with closest scores are the 'beat3' and 'beat4' gestures which scores aren't always that far apart. Also, whenever one of these two gestures are identified the other one is always the next prediction. But this was too be expected considering how similar the two gestures are too each other. Even better models could be trained given more training data and more training data files consisting of single gestures that could be used for testing.

Conclusion

The implementation of HMMs where successful in the classification of a set of gestures. The approach used the K-means algorithm to identify features then the KNN algorithm to match data to these features. The resulting data was then used in the Baum Welch algorithm to create the HMMs that where used for predictions. The accuracy of the model could be increase by increasing the amount of training data and having more labeled datasets that could be used for testing which would allow fine tuning of the model. The performance of the model could also be further increased by making use of more vectorization in the calculation of the Baum Welch algorithm. The final success of the model still remains to be calculated when the final testing data labels gets released but given the training data files that simulate testing data on which the approach obtained a 100% present accuracy it is safe to assume that the model is quite accurate. Given this evidence it shows the effectiveness and power of HMMs to make predictions on sequential datasets where there are hidden states that little is known about or where there are

observation classes representing features that's precise nature is unknown to us. The number of iterations in which the algorithm could achieve reasonable convergence and the speed at which predictions could be made shows the applicability of the model in real world applications. The model could easily be deployed and used in applications such as speech and handwriting recognition given its responsiveness.

References

[1] M. Stamp, A Revealing Introduction to Hidden Markov Models, San Jose: Department of Computer Science, 2018.

[2] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE,* vol. 77, no. 2, pp. 257-286, 1989.