BEAT-PD DREAM Challenge

Che-Pai Kung
University of Southern California
chepaiku@usc.edu

1 Abstract

Wearable devices enable us to track human behaviors continuously. Recognizing the connection between continuous sensor data and human actions may help the physicians better understand the conditions of patients. In this report, the Hidden Markov Model is utilized to extract the patterns of continuous accelerometer sequences and the extracted features are proved to be helpful in predicting Parkinson's Disease patients' medical states. I directly use the raw data provided and the accuracy rates are around 60% for predicting medication states and classifying severity of tremor and dyskinesia. Additional analysis and data preprocessing on the human sensory data may further recognize the human behavior patterns and increase the classification accuracy. Therefore, it may become feasible for doctors to use sensor data recorded by the portable devices as a support to prescribe the medicine.

2 Introduction

2.1 Project description

The Parkinson's Disease (PD) [1] affects the brain's motor functions and leads to other symptoms such as tremor, bradykinesia (slowness) and rigidity. Other than that, side effects such as dyskinesia (involuntary muscle movements) also shows from taking medication. In a medical setting, doctors' examinations and patients' observations are used to diagnose the severity of the symptoms, which are infrequent and may not fully reflect a patient's state.

Nonetheless, patients' sensor (accelerometers) data can be tracked more frequency through mobile devices, such as smartphones. Therefore, the main goal of this project is to provide more detailed information of patients so that doctors can make more well-informed decisions.

Sensory data (movement in direction X, Y and Z) is available for patients with known conditions (classified into severity labels). The model aims to predict subjects with unknown conditions and classify them into their corresponding severity labels, using their sensory data.

2.2 Data description

Data is provided by Clinician Input Study [1] (refers to CIS-PD for the later use) and the Parkinson@Home validation study [1]. In this report, the analysis will be focused on data from CIS-PD.

For CIS-PD, the study period was half a year. During the research, 16 patients were asked to wear the mobile equipment to monitor sensor data through the whole term. In addition, they needed to record the medication states and symptoms severity at every half an hour interval during the 48 hours before visiting the clinic. There are three kinds of labels and each of them is measured within four levels:

Figure 1

Severity	On/off	Dyskinesia	Tremor
0	Fully on	Normal	Normal
1	Partially on	Slight	Slight
2	Moderate	Mild	Mild
3	Partially off	Moderate	Moderate
4	Fully off	Severe	Severe

At each reported time, 20 minutes sensor (Accelerometer) data collected from the wearables is a time sequence data corresponding to three labels (Figure 1). Each 20 minutes accelerometer file was saved as a single csv file with columns shown as below:

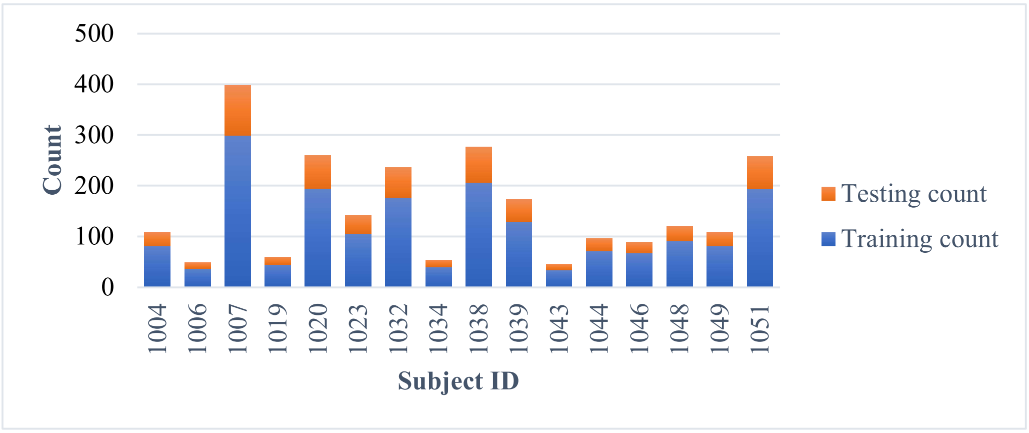
Figure 2

Column Header	**Description**
Timestamp	Time in seconds (0.02 sec/step)
X	X-direction acceleration (gravitational units)
Y	Y-direction acceleration (gravitational units)
Z	Z-direction acceleration (gravitational units)

Then around 25% of the data were used as testing datasets for each patient. The counts are presented below:

Figure 3:

Amount of data collected from each patient. There are 1858 training data and 618 testing data.



2.3 Imbalanced labels

As mentioned above, there are three kinds of labels and each of them is classified into five levels. Nonetheless, this is the skewed dataset as most of the data are of the normal or slight symptoms.

Figure 4: Overview of the label distribution.

Label	*on_off*	*Dyskinesia*	*Tremor*
0	**828 (47%)**	**688 (58%)**	**513 (35%)**
1	405	252	**633 (43%)**
2	296	168	205
3	131	73	102
4	107	7	9
NaN	91	670	396
Total	1858	1858	1858

In addition, several labels are missing in the provided data, which further decreased the number of data we can use to train the model. In the following sections, all the data are used to train HMM and data without labels are excluded when training classification or regression models.

3 Models

As we were provided with the continuous collection of sensor data from wearable devices, there are more than hundred thousand timestamps in each individual file. Instead of training machine/deep learning on the dataset directly, it may be more practical and efficient to analyze data after extracting some patterns of data.

I utilized the Hidden Markov Model (HMM) to help discover the potential patterns of human behaviors. Then, embedded the HMM before running classification and regression model on the data.

In the following sections, the steps mentioned above will be illustrated.

3.1 Extracting Patterns

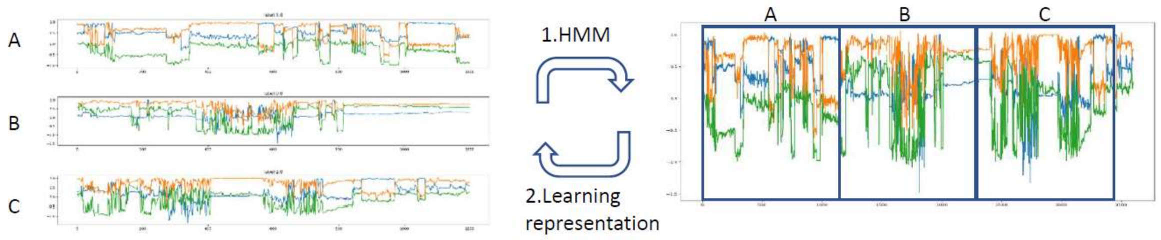
Two kinds of HMM are used to extract possible patterns, Beta Process Autoregressive HMM [3][4] in MATLAB and "hmmlearn" library in Python.

Each hidden state of Beta Process Autoregressive HMM (BPHMM) is modeled using an autoregressive (AR) model. In this project, I set lag, r, equal to one. In addition, unlike other HMM, it applies a beta process to determine the number of states so that it is not required to specify the number of states in advance. As the AR model is favorable when analyzing time series data, combining it with the hidden markov model may capture more complicated patterns. However, it takes a long time to run the model, especially when there are more than 200,000 timestamps for each participant. As a result, 500 iterations were made at this stage.

At the same time, I also tried another model, hmmlearn, to make the comparison of them and see the performance of it. In this model, it is required to specify the number of hidden states. I referred to the results from BPHMM and first set the value as 50.

To increase the model efficiency, some data preprocessing is needed. First, I grouped each data sequence by taking the average of each feature within every 0.2 seconds. In other words, the data size is reduced by ten times. Second, as BPHMM generates a transition matrix for each time series input and there are 2,476 data points, it will take a long time and much memory to run the model. To solve this problem, I concatenated the data sequence of each patient as a time series, as illustrated in Figure 5. Consequently, the data was transformed into 16 time series but with hundred thousand length. Note that not only the training data but also the testing data was used to learn HMM model so that we can do the predictions on the testing data based on its HMM models after building the final model for the project. Codes for this part are saved in "cis_pd_concatenate.py". The program reads all the data from "training_data" and "testing_data" folders, and then saves the concatenated files in "outputA_Total" folder. Other than that, "cis-pd-memo.pkl" including the information of the length of data, "cis-pd-y_train.pkl" and "cis-pd_testid.pkl" are saved for subsequent benefit.[1]

Figure 5: Outline of data preprocessing. In patterns extracted step, data sequence of each patient is concatenated into a time series. After having state sequences, the time sequences are split to original length.



3.2 Learning Representations

I referred to the paper [2] and applied the stationary representations to embed the HMM we got from 3.1. As mentioned in Figure 5, the data of the same patient were concatenated into a time series and thus, only a HMM for each patient. Therefore, it is vital to separate them so that we could do regression/classification in the later steps. After splitting them, I generate the transition matrix for each original time sequence according to its state sequence. For instance, suppose the state sequence is 123321, then the count matrix and the transition matrix are shown in Figure 6, where cell (i, j) means the total count of state i to state j. The transition matrix is derived after dividing each cell by the sum of its corresponding row.

Figure 6

states	1	2	3
1	0	1	0
2	1	0	1
3	0	1	1

$\Longrightarrow$

states	1	2	3
1	0	1	0
2	0.5	0	0.5
3	0	0.5	0.5

As the stationary distribution is unique for each HMM/ transition matrix, I use it as features for regression and classification tasks. The stationary distribution is an eigenvector of transition matrix with corresponding eigenvalue equal to one. Consequently, the numpy.linalg.eig function is used to get eigenvalues and eigenvectors in this part. The related code is saved in "cis_pd_HMM_to_trainingdata". The program takes the results (states sequence) from the HMM as inputs, then outputs two files after deriving the stationary

[1] Due to the memory issues, the programs relating to BPHMM were executed on HPC (Center for High-Performance Computing) server and the file extension of them is "py". Since, some issues occurred when installing hmmlearn library, so the codes relating to hmmlearn were implemented on Google Colab and the file extension of them is "ipynb".

distribution of each data sequence, one is the features of all training data and another one is the features of all testing data.

3.3 Classification/ Regression

At first, the competition was thought as a classification problem, so the classification model was applied to predict the labels for the testing data. Nonetheless, after carefully checking the evaluation metric of the competition, for each patient k, it is being assessed using the mean square error (MSE): (n_k is the number of testing data)

$$MSE_k = \frac{1}{n_k}\sum_{i=1}^{n_k}(y_{ik} - \hat{y}_{ik})^2$$

$$\text{Final score} = \frac{\sum_{k=1}^{N}\sqrt{n_k}MSE_k}{\sum_{k=1}^{N}\sqrt{n_k}}$$

As a result, a regression model is also applied to train the model and the results were compared.
It is noticed that the score is estimated on a patient basis. Thus, it is beneficial if the model can recognize which patient it is now when prediction labels. The method I used is by concatenating the mean value of features to the original features for each person. In other words, if there are n features for initial data, there will be double size （2*n） of features after concatenating. This part is done by executing “cis_pd_HMM_to_trainingdata” in the previous part. The results before and after concatenating were also examined.
3.3.1 SVM
The model I used for classification is sklearn.svm.SVC, which stands for Support Vector Classification. To find the suitable hyperparameters for the model, RandomizedSearchCV from sklearn.model_selection was utilized. Since there are multiple parameters to tune, instead of using GridSearchCV, I used RandomizedSearchCV to decrease the running time. As it is the skewed dataset, the setting of parameter “class_weight” might affect the results to some extent. The intention of this parameter is to adjust “C” among all the classes inversely proportional to the amount of labels, meaning that labels with small portions have larger “C” and the model is more strict for those labels and allows less misclassified points. Consequently, the parameters set included “kernel”, “degree”, “C”, “gamma” and “class_weight”. The results are presented below:

Figure 7: Best estimators of svc model using hmmlearn results (Other parameters are set as defaulted).

Labels\Parameters	*C*	*kernel*	*degree*	*gamma*	*Class_weight*
On_off	10	rbf	3	0.01	None
Dyskinesia	1	linear	3	1	None
Tremor	10	rbf	3	0.01	None

Figure 8: Best estimators of svc model using BPHMM results (Other parameters are set as defaulted).

Labels\Parameters	*C*	*kernel*	*degree*	*gamma*	*Class_weight*
On_off	1	linear	3	0.001	None
Dyskinesia	10	rbf	3	0.01	None
Tremor	1	linear	3	0.001	None

3.3.2 Regression

Three kinds of regression models are applied, linear, lasso and ridge regression. Lasso and Ridge regression are similar to linear regression but with L1 and L2 regularization, respectively. As only parameter “alpha” is tuned, GridSearchCV from sklearn.model_selection was utilized to find the best hyperparameter. The linear regression model led to the worst performance among three models. The reason I believe is that it is overfitting when training the model as it has no regularization terms. Thus, it was excluded from further analysis. The results of GridSearchCV are shown in Figure 9 and Figure 10.

Figure 9: Regression models of three labels using patterns generated by hmmlearn.

Labels\Parameters	*Lasso_alpha*	*Ridge_alpha*
On_off	0.01	10
Dyskinesia	0.1	10
Tremor	0.01	10

Figure 10: Regression models of three labels using patterns generated by BPHMM.

Labels\Parameters	*Lasso_alpha*	*Ridge_alpha*
On_off	0.005	1
Dyskinesia	0.01	10
Tremor	0.005	10

4 Results

To make the comparison between models more accurate, the scores are evaluated using cross-validated metrics and stratified shuffle is used to split data. Stratified shuffle creates the splits that have the same percentage of each target value as the whole dataset. As the dataset is skewed, the model is believed to have better prediction ability in rare labels using this shuffle technique. As described in section 3.3, I compared the outcomes of models trained before and after concatenating the mean value. The results after concatenating mean values of each patient are generally better, and therefore, only the results of those models are presented below.

4.1 Classification

For the classification models, the accuracy rate is used as an evaluation metric. The outcomes are shown in Figure 11 and Figure 12. They illustrate that the features generated by the BPHMM provide not only higher accuracy rate in label “on_off” and “dyskinesia” but also smaller standard deviation except for “on_off” when doing cross validation. Thus, it is believed that at this stage, BPHMM achieves better and more stable features for the sensory data.

Figure 11: Summary of mean and standard deviation of accuracy rates using hmmlearn features.

Labels	*On_off*	*dyskinesia*	*tremor*
cv_mean	0.548	0.660	0.627
cv_std	0.012	0.027	0.014

Figure 12: Summary of mean and standard deviation of accuracy rates using BPHMM features.

Labels	*On_off*	*dyskinesia*	*tremor*
cv_mean	0.561	0.664	0.612
cv_std	0.013	0.011	0.011

4.2 Regression

For the regression models, mean squared error is used as the evaluation metric. Figure 13 and Figure 14 show that Lasso regression is more appropriate for the dataset. In addition, as when building classification models, BPHMM features lead to better and reliable predictions.

Figure 13: Summary of mean and standard deviation of accuracy rates using hmmlearn features.

Labels	*On_off*	*dyskinesia*	*tremor*
Lasso_cv_mean	1.515	0.576	0.488
Lasso_cv_std	0.072	0.063	0.055
Ridge_cv_mean	15.36	1.222	4.226
Ridge_cv_mean	25.45	0.822	4.456

Figure 14: Summary of mean and standard deviation of accuracy rates using BPHMM features.

Labels	*On_off*	*dyskinesia*	*tremor*
Lasso_cv_mean	1.379	0.534	0.430
Lasso_cv_std	0.046	0.085	0.014
Ridge_cv_mean	1.457	0.552	0.505
Ridge_cv_mean	0.149	0.075	0.074

4.3 Summary

As the competition is using MSE as evaluation metrics, after deciding which features to use[2], I used the parameters in Figure 8 to train the SVC model using the BPHMM representations and parameters in figure 10 to train the Lasso model. The data provided with labels are split into training set and testing set using train_test_split from sklearn.model_selection with split ratio equal to 0.25 which is the same as the percentage of data provided with no labels (mentioned in section 2.2). Then training sets are used to train both models and testing set are used to estimate their score. The results are shown in figure 15. Lasso models provide lower MSE among three labels. Therefore, the results generated from the Lasso model will be submitted to the competition.

Figure 15: MSE of SVC and Lasso models.

Labels	*On_off*	*dyskinesia*	*tremor*
SVC	1.756	1.067	0.579
Lasso	1.482	0.542	0.416

[2] Though the representations of BPHMM result in better outcome in this report, it may due to the less iterations in hmmlearn model.

5 Conclusions

In this report, Hidden Markov Models are demonstrated to be conducive for predicting Parkinson Disease's patients' physical conditions. Initially, Hidden Markov Models are applied to learn the hidden states of human behavior captured by wearable devices. Then the Support Vector Classification model is trained using the representations, which are the stationary distributions of each transition matrix. After the comparison, Beta Process Autoregressive HMM (BPHMM) can deal with the time series with different length and discover the number of hidden states automatically. Moreover, it provides better and more concentrated predictions for the dataset.

As the dataset is skewed, there are more data misclassified in the labels with little portion. Some approaches are figured out to possibly eliminate such conditions and may further increase the accuracy rates (lower the mean square error). First, normalizing data either within the whole dataset or within each patient before training HMM. Second, as suggested in paper [2], utilizing "Spectral Representation" to do the embeddings after calculating the distance matrix of each transition matrix may generate better embeddings. As there are still some days before the deadline of the competition, the above methods will be implemented if possible.

6 Acknowledgement

I would like to express my sincere gratitude to Nazgol Tavabi who is the PhD student in the Information Sciences Institute. She shared the research [2] she has done with me and provided guidance on how to deal with the time series data. She also helped me apply for the HPC (Center for High-Performance Computing) account so that the BPHMM [5] can be executed with more than 5 GB file size successfully.

7 References

[1] Website of the competition: https://www.synapse.org/#!Synapse:syn20825169/wiki/

[2] Nazgol Tavabi, Homa Hosseinmardi, Jennifer L Villatte, Andrés Abeliuk, Shrikanth Narayanan, Emilio Ferrara, and Kristina Lerman. 2019. Learning Behavioral Representations from Wearable Sensors. arXiv preprint arXiv:1911.06959 (2019)

[3] "Joint Modeling of Multiple Time Series via the Beta Process with Application to Motion Capture Segmentation." Emily Fox, Michael C. Hughes, Erik B. Sudderth, Michael I. Jordan Annals of Applied Statistics, Vol. 8(3), 2014.

[4] BPHMM matlab code: https://github.com/michaelchughes/NPBayesHMM

[5] Cross-validation: https://scikit-learn.org/stable/modules/cross_validation.html