Extracting Class-specific Sequential Pattern for Continuous Glucose Monitoring

Masaki Ono[1,a)] Takayuki Katsuk[1] Masaki Makino[3] Kyoichi Haida[2] Atsushi Suzuki[3]

Abstract: Continuous glucose monitoring (CGM) is temporal time-series data that has been available for approximately 10 years thanks to the invention of a device with low measurement error. Understanding the time series variation of glucose helps you treat specific patient groups better by understanding their lifestyles. Therefore, we propose a method of extracting characteristic sequential patterns from given pairs of labels and sequences. First, we apply time-series clustering to transform CGM value sequences into cluster id sequences. Next, we apply sequential pattern mining to extract frequently occurred sequences. Finally, we evaluate each frequent sequence based on their correlation to a specific class. We experimented with two datasets that is manually created and one real CGM dataset to prove our method is effective.

1. Introduction

Diabetes mellitus is a major health condition whose prevalence is rapidly increasing worldwide. One of the major causes of diabetes mellitus is patient lifestyles, medical service worker have to understand it to decide a therapeutic strategy. Thanks to technological advancements for medical devices, we can measure glucose by the minute for weeks using a sensor called the " continuous glucose monitoring (CGM) system. "CGM is time-series data and has been available since devices with low measurement errors appeared 10 years ago. CGM can be relied upon to help make treatment decisions.

A major issue regarding CGM is educating physicians to use it. Since many physicians are unfamiliar with CGM, they cannot use it to make treatment decisions. The review study about CGM [1] reported that there is an urgent need for a standardized interpretation of glucose data and patterns akin to automated electrocardiogram interpretation. We discuss current glucose management methods in Section 3, although they are not technical issues.

In this paper we propose a method of extracting characteristic sequential patterns from given pairs of labels and sequences. Figure 1 shows the overview of the method. This method can help us understand the typical CGM patterns of a specific patient group. We believe that the method can greatly contribute to CGM interpretation.

In this paper we propose the method for extracting characteristic sequential patterns from given pairs of labels and sequences. Every user has its label that is predefined based on age, sex and laboratory test result. Figure 1 shows the overview of the method. This method can help us understand the typical CGM patterns of a specific patient group. We believe that the method can greatly contribute to CGM interpretation.

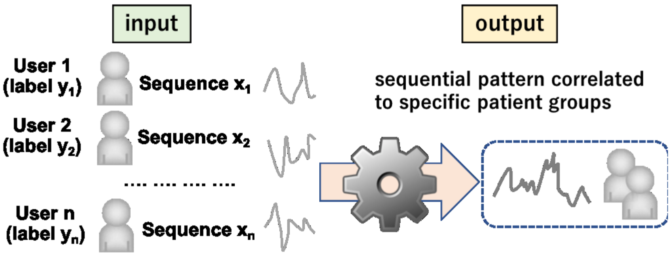

Fig. 1 Overview of the task

We deal with the following technical issues for handling time-series data to extract characteristic sequential patterns.

(1) *data normalization.* Generally data has noise and it make handling data more difficult.

(2) *shift-invariance.* Sometimes similar waves starts from different points. This is one of the major issues on handling time-series data.

(3) *method scalability.* The review study of time-series data processing [2] reports that time-series data tends to be huge due to its application's characteristic.

The method consists of 3 steps. At first we apply time-series clustering to transform CGM value sequences into cluster id sequences. By clustering the data we group similar elements of data to find patterns more effectively. We propose mini-batch top-n k-medoids for the clustering method that runs on part of data and assigns cluster centroids to multiple elements on a cluster. Secondly we apply sequential pattern mining to extract frequently occurred sequences. Finally we evaluate each frequent sequence based on its correlation to a specific class.

The research questions that we answer based on the result of

[1] IBM Research - Tokyo, 19-21, Nihonbashi Hakozaki-cho Chuo-ku, Tokyo 103-8510 Japan
[2] The Dai-ichi Life Insurance Company, Limited, 13-1, Yurakucho 1-chome, Chiyoda-ku, Tokyo 100-8411, Japan
[3] Fujita Health University, 1-98 Dengakugakubo, Kutsukake-cho, Toyoake, Aichi 470-1192, JAPAN
[a)] moono@jp.ibm.com

the experiment are the followings.

- Does mini-batch make the performance worse?
- Does mini-batch improve the scalability?
- Does multiple centroids work better than only one centroid?
- Does the method deal with shift-invariance?
- Does the method find class-specific sequential patterns?

We conducted two experiment to answer the research questions with manually created datasets. We design one experiment for evaluating mini-batch top-n k-medoids and the other one for evaluating class-specific sequential pattern extraction. And we applied the proposed method to the real CGM dataset. We conclude that the proposed method is effective for the technical issues.

2. Motivation

In this section, we discuss technical issues with the tasks and how we solved them. We proposed a method that receives time-series data and its labels and produces label-specific sequential patterns. In the field of CGM analysis, " labels " means the CGM user's attributes, such as age, sex, and laboratory test results.

2.1 Sequential normalization with subsequence time-series clustering

Generally, data has noise, and finding patterns in raw data is difficult. Although CGM data is one-dimensional data, we need to normalize the data for preprocessing. Therefore, we extracted a set of subsequences from a sequence and applied a clustering method to group similar subsequences in the same cluster. We also translated a given sequence into a cluster id sequence.

Some clustering methods do not work well for subsequence time-series clustering. Previous studies [3] [4] reported that k-means with an Euclidean distance metric for subsequence time-series clustering obtains non-intuitive results. Centroids in all clusters tend to be similar, which implies that the clustering method does not grasp the true nature of clusters. To resolve the issue, we applied a clustering method with a non-Euclidean distance metric and did not calculate the centroids of the clusters.

2.2 Shift invariance

Time-series data has issues with time-series invariances. We focus on shift invariances, one of the time-series invariances.

Shift-invariance is the phenomenon that two sequences are similar but differ in phase. In the field of CGM analysis this issue may occur when two CGM users are similar conditions and eat breakfast on the different time. We probably would observe two similar sequences starting at different times.

Figure 2 shows an example of the shift invariance that uses two sine waves. These two waves are the same and start from different positions. They have the same frequency and amplitude, and either one can have translation.

To deal with shift invariances, we used a sliding window for feature extraction from given time-series data. The sliding window repeated subsequence extraction that had the same length while changing the start position. Therefore, we could handle all the subsequences and find the patterns with translation.

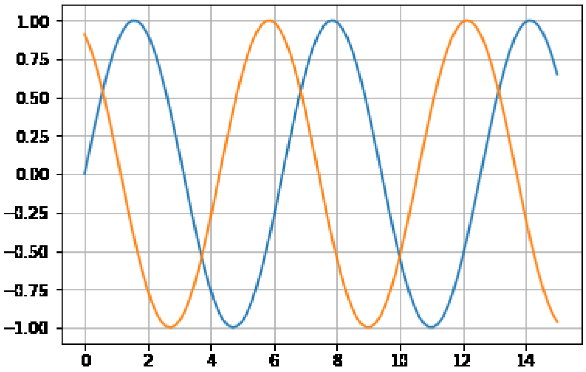

Fig. 2 Example of shift-invariance: The two waves have the sample amplitude and frequency and start from the different points.

2.3 Handling large scale data

Generally time-seres data processing needs large amount of calculation because of two reasons. The review study [2] reports that time-series data tends to be huge due to its application's characteristic. And we discuss about sliding window for feature extraction and its advantage in the section 2.2. One side effect of a sliding window is that it augments the data size. Moreover, if CGM becomes more common, the data size we will have to handle will increase dramatically.

Second reason is that similarity metrics used in time-seres data processing is often more complicated than euclidean distance. For example we briefly compare two similarity metrics, Euclid distance and Earth mover's distance. We found Earth mover's distance is 200 times slower than Euclid distance. In the comparison we used randomly produced data with 12 dimension and repeated calculation more than 1000 times.

To solve the issue we proposed new clustering method, we propose mini-batch top-n k-medoid. The method runs on part of data not whole data and it can reduce the amount of calculation. Although we used similarity metrics designed in time-seres data processing, the whole amount of calculation is not very large.

3. Related work

3.1 Glucose monitoring

Glucose management is essential, especially for diabetes mellitus patients. HbA1c is one of the laboratory tests for blood, and it is important for glucose management. Generally, HbA1c reflects the averaged glucose value over the course of three months. A limitation of HbA1c is that HbA1c does not provide a daily change of glucose or information about hyperglycemia and hypoglycemia [5].

Self-monitoring of blood glucose (SMBG) is when a patient draws his/her own blood and checks his/her current glucose level with a home glucose meter. SMBG has been widely used for decades, and the American Diabetes Association initially established guidelines for SMBG in 1987.

CGM provides real-time glucose data and uses a human implantation sensor that continuously operates for a few weeks. This information contains signs of hyperglycemia or hypoglycemia and aids glucose management. Currently, device makers provide analysis software for CGM logs[*1][*2]. The International Diabetes Center created an analysis schema called the “ Ambulatory Glucose Profile ”that consists of more than 10 metrics, including mean glucose [6]. This analysis schema handles one patient’s CGM logs and does not compare them with other patient’s logs.

3.2 Sequential pattern mining

In sequential pattern mining, we receive a set of sequences and a frequency threshold and find all the subsequences that occur more often than the threshold. Subsequences are not always evaluated by frequency, and they sometimes use criteria such as length and profit. Sequential pattern mining is a special kind of pattern mining, and the difference is that elements of data are in order. The order of elements is not important in other pattern mining tasks, such as frequent itemset mining.

A lot of research work has applied sequential pattern mining in many fields [7]. For example, we use it for term recognition in the field of natural language processing [8]. Other applications are market basket analysis and webpage click-stream analysis.

The approaches of sequential pattern mining are classified into two general groups: bread-first searches and depth-first searches [7]. PrefixSpan is a depth-first search proposed by [9]. PrefixSpan counts sequence event frequencies using tree structure called prefix tree or trie tree.

Some studies [10][11] applied a deep learning approach to understand the pattern of given time series data. They used autoencoder-based methods that are not literally interpretable. Thus we do not learn medical insights from the learning result.

3.3 Clustering

Clustering receives unlabeled data and produces multiple groups where similar elements belong to the same group. Clustering is a kind of unsupervised machine learning, and typical methods are hierarchical clustering and partitional clustering. Hierarchical clustering assigns a cluster to every element, then repeatedly measures the similarity between every cluster pair and merges one cluster pair with maximum similarity until the number of clusters becomes the given one. The algorithm has a time complexity of $O(n^3)$, where n is the number of elements. The memory complexity is $O(n^2)$ if we use the cache.

k-means is a kind of partitional clustering that receives the number of the clusters k and data X and produces k centers C that minimize the loss function ϕ.

$$\phi(X, C) = \sum_{x \in X} \min_{c \in C} ||x - c||^2 \tag{1}$$

A centers $c \in C$ is calculated from the mean of elements $X' \subset X$ whose nearest center are c. This is not suitable for temporal series data. The algorithm has a time complexity of $O(n \cdot k \cdot t)$, where n is the number of elements and t is the iteration number. The memory complexity is $O(n \times k)$.

Some studies proposed extensions of k-means. k-means++ [12] improve cluster initialization step. Mini-batch k-means [13] uses random sample of given data to reduce time complexity and memory complexity.

k-medoids does not create centroids but uses an element of a cluster that minimizes the averaged distance to the other elements in the cluster as its centroid. The loss function is as follows. $dist(x, y)$ is the distance between x and y.

$$\phi(X) = arg \min_{x \in X} \sum_{y \in X, y \neq x} dist(x, y) \tag{2}$$

The algorithm has a time complexity of $O(n^2 \cdot k \cdot t)$, where n is the number of elements and t is the iteration number.

Time-series clustering is clustering technology for time-series and has three main topics: whole time-series clustering, subsequence time-series clustering, and time point clustering [2]. The survey [2] summarizes the three challenges of time-series clustering as large data sizes, high dimensions, and special distance metrics. Since we focused on subsequences of given time-series data, we did not handle high dimensional data. However, we did take care of large data sizes and special distance metrics. We proposed a new clustering method and used the existing distance metrics designed for time-series data.

Distance metrics used in time-series clustering are designed to deal with issues with a sequence. A dynamic time window handles the shift-invariance of a given pair of sequences by [comparing .... — through comparisons?] Short time series distance handles the shapes of a sequence. The earth mover’s distance (EMD) compares two value distributions of a given pair of sequences.

The following is the definition of EMD. Let $P = \{p_1, p_2, ..., p_m\}$ and $Q = \{q_1, q_2, ..., q_n\}$ two sequences, $D = [d_{i,j}]$ be the cost between p_i and q_j respectively. And let $F = [f_{i,j}]$ be the flow between p_i and q_j.

EMD calculation consists of two steps. Firstly we minimize the loss function ϕ under 4 constraints and find flow F that minimizes the cost for translating P into Q.

$$\phi(P, Q, D) = \min \sum_{i=1}^{m} \sum_{j=1}^{n} f_{i,j} \cdot d_{i,j} \tag{3}$$

$$\begin{cases} f_{i,j} \geq 0, 1 \leq i \leq m, 1 \leq j \leq n \\ \sum_{j=1}^{n} f_{i,j} \leq p_i, 1 \leq i \leq m \\ \sum_{i=1}^{m} f_{i,j} \leq q_j, 1 \leq j \leq n \\ \sum_{i=1}^{m} \sum_{j=1}^{n} f_{i,j} = min\{\sum_{i=1}^{m} p_i, \sum_{j=1}^{n} q_j, \} \end{cases}$$

Then we calculate the EMD $dist_{emd}$ using estimated flow F.

$$dist_{emd}(P, Q, D) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{i,j} \cdot d_{i,j}}{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{i,j}} \tag{4}$$

4. Method

First, we applied time-series clustering to transform CGM value sequences into cluster id sequences. Then, we extracted frequently occurred sequences with sequential pattern mining. Finally, we evaluated each frequent sequence based on correlations to specific classes.

Let $T = \{t_1, t_2, ..., t_{|T|}\}$ be a set of time-series data, $t_i =$

[*1] http://professional.medtronicdiabetes.com/ipro2-professional-cgm
[*2] https://www.dexcom.com/clarity

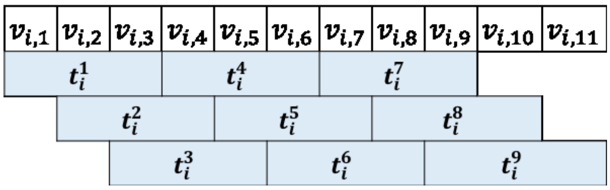

Fig. 3 Example of Sliding-window

$\{v_{i,1}, v_{i,2}, ..., v_{|t_i|}\}$ be a sequence of values, and $Y = \{y_1, y_2, ..., y_{|T|}\}$ be labels. The labels have m classes and we can also define labels as $Y = \{Y^{(1)}, Y^{(2)}, ..., Y^{(m)}\}$ where $Y^{(i)}$ is a set of indices of elements that have i-th class. Our task was to receive the time-series data T and labels Y above and produce subsequences with strong correlations with a specific label.

4.1 Extracting subsequence time-series with sliding-window

In the first step, we extracted a sequence of subsequences from one given sequence by using a sliding-window. Then, we applied the clustering method to a set of subsequences to group similar subsequences. This is the preprocess for extracting interesting patterns from given time series data. Although dealing with raw data is difficult, we abstract given data for a clearer interpretation.

Let n be the length of the subsequence. Then we can define a subsequence $t_i^j = \{v_{i,j}, v_{i,j+1}, ..., v_{i,j+n-1}\}$. This is a part of a given sequence.

We can extract a sequence of subsequences by repeating subsequence extraction. It means we transform t_i into $\{t_i^1, t_i^n, ..., , t_i^{|t_i|-n}\}$. Note that $t_i^1 = \{v_{i,1}, v_{i,2}, ..., v_{i,n}\}, t_i^{n+1} = \{v_{i,n+1}, v_{i,n+2}, ..., v_{i,2n}\}$.

The sliding-window repeats the previous procedure with changing the first index and produces n sequences of subsequences. Figure 3 shows an example of a sliding-window. Here, the sequence has 12 values, and the length of the subsequence is 3. We obtain 3 sequences of subsequences: $\{t_i^1, t_i^4, t_i^7\}, \{t_i^2, t_i^5, t_i^8\}, and\{t_i^3, t_i^6, t_i^9\}$.

We use the sliding-window to deal with shift invariance where similar waves start from different points. With the sliding-window we can extract subsequences from different points and it is one way to solve shift invariance. The side-effect of a sliding-window is data augmentation. If we use n as the length of the subsequence, then we have to handle n times data size. For example, we set $n = 3$ in the example used on the Figure 3. Here we obtain 3 sequences of subsequences and this is triple size of the given one. We discuss the scalable clustering method to deal with large size data.

4.2 Mini-batch top-n k-medoids

This step receives sequences of subsequences created in the previous step, and applies the clustering method to all subsequences to group similar subsequences. Moreover, we transform sequences of subsequences into cluster id sequences to extract interesting sequential patterns more effectively.

We proposed mini-batch top-n k-medoids for the clustering method. Thisis a partitional clustering method that uses a non-Euclidean distance metric and runs faster with large scale data. We do not define the centroid of a cluster, but use some elements of the cluster to define the distance between the cluster and a given element.

Some studies reported that k-means with an Euclidean distance metric for subsequence time-series clustering is useless because the centroid of a cluster is similar to a sine wave. To deal with the issue, we use a non-Euclidean distance metric and do not make new centroids. We assign multiple centroids to some elements in the cluster. This strategy is based on k-medoids.

To define the steps of mini-batch top-n k-medoids we define some variables and functions. Let $X = \{x_1, ..., x_n\}$ be given data where each elements x_i has the same dimension. The $dist_{elem}$ calculates the distance between an element of the cluster $x_i \in X^{(j)}$ and the cluster $X^{(j)}$. The $dist_{clus}$ calculates the distance between two clusters $X^{(i)}, X^{(j)}$. We use $dist_{EMD}$ as a non-Euclidean distance metric in $dist_{elem}$ and $dist_{clus}(C_i, C_j)$, however we can use other distance metric.

$$dist_{elem}(x_i, X^{(j)}) = \frac{1}{|X^{(j)}|} \sum_{x_k \in X^{(j)}, x_k \neq x_i} dist_{EMD}(x_i, x_k) \qquad (5)$$

$$dist_{clus}(C^{(i)}, C^{(j)}) = \frac{1}{|C^{(i)}| \cdot |C^{(j)}|} \sum_{x_i \in C^{(i)}} \sum_{x_j \in C^{(j)}} dist_{EMD}(x_i, x_j) \qquad (6)$$

Let b be the batch size and a part of the given data $X' \subseteq X$ be a batch, respectively. Obviously $|X'| = b$. Let k be the number of cluster and $C = \{C^{(1)}, C^{(2)}, ..., C^{(k)}\}$ be the centers of the clusters, respectively. Let m be the number of centers in one cluster $C^{(i)}$ be centers of i-th cluster. Obviously $C^{(i)} = m$.

The following is the steps mini-batch top-n k-medoids.

(1) *Batch Initialization.* We make batch X' by randomly selecting b elements from X.

(2) *Cluster initialization.* We execute this procedure only one time. We choose one element from X' and assign it as the initial center for a cluster. Only on the first time each cluster has one center, not m centers.

(3) *Assign the new cluster.* We search the closest cluster of an element x_i by the following loss function and assign it as new cluster of the element x_i. Here we use the $dist_{EMD}$ function to calculate the distance between two elements.

$$\phi(x_i, C) = arg \min_{C^{(i)} \in C} \frac{1}{|C^{(i)}|} \sum_{x_j \in C^{(i)}} dist_{EMD}(x_i, x_j) \qquad (7)$$

(4) *Assign the new centers.* We assign new centers $C^{(i)}$ to k elements of the cluster $X^{(i)}$. We use $dist_{elem}$ function and calculate the averaged distance to each element in the same cluster. And we choose k-minimum elements or select elements based on the k-means++ initialization method.

(5) *Check the stop criteria.* We chek the distance between the current centers C and the previous centers C' with the following formula. If the distance is less than threshold delta, we can get results from the procedure. If not, we go back to the first step.

$$dist_{cent}(C, C') = \sum_{C^{(i)} \in C} \min_{C^{(j)} \in C'} dist_{clus}(C^{(i)}, C^{(j)}) \qquad (8)$$

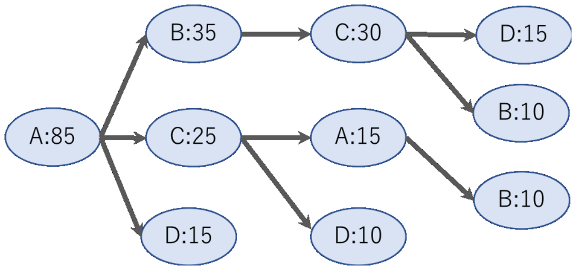

Fig. 4 Example of PrefixSpan: Each node shows character and its frequency. The root of the tree is A:85 and it means A is observed 85 times.

4.3 Sequential pattern mining

This step receives a set of cluster id sequences and produces frequently occurred sequential patterns by applying the sequential pattern mining method. Next, we evaluate the patterns to filter out the interesting ones.

We use the PrefixSpan extension that both extracts frequently occurred sequential patterns and indexes where the sequences are observed. The aim of the entire method is to extract label-specific patterns, and we have to know where a subsequence is observed.

PrefixSpan is a sequential pattern mining method where we receive a set of sequences and frequency thresholds and find all subsequences that occur more than thresholds. Tracing all subsequences of multiple sequences is time-consuming, so PrefixSpan devises a data structure for tracing and uses the tree structure that is called trie tree or prefix tree to quickly extract frequent sequence patterns.

Figure 4 shows an example tree structure used on PrefixSpan. The node of the tree has character, and its frequency and parent-child relationship shows the occurrence order of the characters. For example, $\{A\}$ occurs 85 times and the $\{A, B\}$ sequence occurs 35 times. Since the frequency of a parent must be greater than its child 's, we can easily stop tracing trees to extract sequences that occur more often than the threshold.

We extend PrefixSpan to save a subsequence's index in given cluster id sequences. With this index, we can easily understand labels of the subsequences and can evaluate frequent patterns with the given labels in the next step.

4.4 Sequence evaluation to find label-specific patterns

This step receives frequent sequential patterns that consist of cluster ids and labels, and evaluate patterns to find label-specific patterns. We design a score that assign a high value to a patter that frequently occurs and is correlated with specific label. The score is calculated from filling ratio of a label's elements and label frequency ratio.

The scoring function receives a pattern and returns real number. Since we mostly find multiple patterns by sequential pattern mining, we have to evaluate the score distribution, not a score for one pattern. We evaluate a set of patterns by the median of the scores that is calculated from them.

Let $S = \{s_1, s_2, ..., s_{|S|}\}$ be a set of sequential patterns and $L(s_i)$ be a set of indices where pattern s_i occurs. As preprocess of the score calculation we define two probability distributions $df(s_i)$ and $dc(s_i)$ as follows.

$$df(s_i) = \left\{ \frac{|Y^{(1)} \cap l(s_i)|}{|Y^{(1)}|}, \frac{|Y^{(2)} \cap l(s_i)|}{|Y^{(2)}|}, ..., \frac{|Y^{(m)} \cap l(s_i)|}{|Y^{(m)}|} \right\} \quad (9)$$

$$dc(s_i) = \left\{ \frac{|Y^{(1)} \cap l(s_i)|}{|l(s_i)|}, \frac{|Y^{(2)} \cap l(s_i)|}{|l(s_i)|}, ..., \frac{|Y^{(m)} \cap l(s_i)|}{|l(s_i)|} \right\} \quad (10)$$

$df(s_i)$ consists of filling ratio of a label's elements and $dc(s_i)$ shows label frequency ratio. Obviously $df(s_i)$ and $dc(s_i)$ have the same dimension. Note that the sum of $df(s_i)$ does not equal to 1 and the sum of $dc(s_i)$ equals to 1.

The score is calculated as follows. The formula is consists of two parts, f_{pos} and f_{neg}. The first pat is positive one and the second one is a penalty.

$$f(s_i) = f_{pos}(dc(s_i)) \cdot \frac{1}{f_{neg}(df(s_i))} \quad (11)$$

In the first part we calculate relative entropy between label frequency ratio and uniform distribution to evaluate correlation to a specific label. Relative entropy is known as Kullback-Leibler divergence and a measurement for distance between two probability distributions. Let $P = \{p_1, p_2, ..., p_n\}$ and $Q = \{q_1, q_2, ..., q_n\}$ be probability distributions. Then relative entropy is defined as follows.

$$KL(P||Q) = \sum_{i=1}^{n} q_i \cdot log\frac{q_i}{p_i} \quad (12)$$

We normalize positive part to change the range of the return. It makes the score more flexible. du is the uniform distribution. dd is the distribution where one element is 1.0 and the other elements is 0.0. The distribution dd shows that a pattern is only observed in the data with a specific label. We define the value $kl_{max} = KL(dd||du)$ for normalization. The following is the positive part $f_{pos}(X)$ that returns $x \in [0, 10]$.

$$f_{pos}(X) = \frac{10}{kl_{max}} \cdot KL(X||du) \quad (13)$$

The second part is a penalty to assign low value to a pattern that does not frequently occur and calculated from filling ratio of a label's elements. We use the function $f(x) = 1 - log(x)$ in the penalty calculation. When all samples with the specific label have the patterns, $x = 1$ and $f(x) = 1$. Therefore we do not have any penalty for the pattern.

$$f_{neg}(X) = \frac{1}{|X|} \sum_{x \in X} 1 - log(x) \quad (14)$$

5. Experimental Results and Discussion

In this section, we describe how we evaluated our method with two experiments with manually created datasets. We design one experiment for evaluating mini-batch top-n k-medoids and the other one for eval- uating class-specific sequential pattern extraction. And we applied the proposed method to the real CGM dataset.

5.1 Mini-batch top-n k-medoid

In this section, we discuss how we manually created a dataset for the mini-batch top-n k-medoids evaluation and the results we achieved. The aim of the evaluation is to answer the following questions:

Table 1 AMI with different batch size and the number of centroids: N on the header is the number of the centroids.

	bach size						
N	100	300	500	700	1000	2000	4000
1	0.305	0.271	0.216	0.354	0.244	0.301	0.182
10	0.442	0.419	0.425	0.389	0.301	0.474	0.468
20	0.386	0.496	0.389	0.413	0.397	0.374	0.448
30	0.396	0.496	0.426	0.402	0.359	0.526	0.528
50	0.394	0.443	0.277	0.441	0.422	0.326	0.414

- Does mini-batch make the performance worse?
- Does multiple centroids work better than only 1 centroid?
- Does mini-batch improve the scalability?

The last question seems to be nonsense because you cannot get non-intuitive results with methods that do not create a centroid and assign a cluster to an element. However, we empirically evaluated the issue; we used a manually created dataset rather than a real CGM dataset or a public time-series dataset.

We created a dataset through the following steps. The dataset contained 4 types of waves with 12 dimensions, and each type has 1,000 samples. Thus, the dataset consisted of 4,000 waves with 12 dimensions.

(1) We created 4 base waves in the first step. We randomly created 3 waves with 4 dimensions and arranged them randomly to create 1 wave with 12 dimensions. An element of the waves is a random real number in the half-open interval $[0.0, 1.0)$. By repeatedly arranging the waves, we obtained 4 base waves. We used the Python library[*3] to create the initial 3 waves.

(2) We created a sample for one base wave by adding noise, a random real number in the half-open interval $[-0.3, 0.3)$. By repeating the procedure, we created 1,000 samples for 1 base wave and finally obtained 4,000 waves.

To answer the research questions, we applied the mini-batch top-n k-medoids to the dataset by changing the size of the mini-batch and the number of centroids n. The method is a randomized algorithm, and we repeated the same setting 5 times and used the averaged values. We used $k = 4$ in the experiment.

We use adjusted the mutual information (AMI) [14] for the performance metric of clustering. AMI is an extension of the mutual information. AMI handles the fact that the mutual information is generally higher for two clusterings that have many clusters.

Table 1 and Figure 5 shows the result of the experiment. When $N = 1$, we obtain the worst performance if we change the batch size. The number of the centroids is not always positively correlate with the performance. When batch size = $\{100, 300, 500, 2000, 4000\}$, we do not obtain the best performance on the setting where $N = 50$. The batch size is not correlate with the performance. When the batch size is 4,000, we obtained the best performance. However the performance does not always decrease when we make the batch size smaller. We answer the research questions in the rest of this subsection.

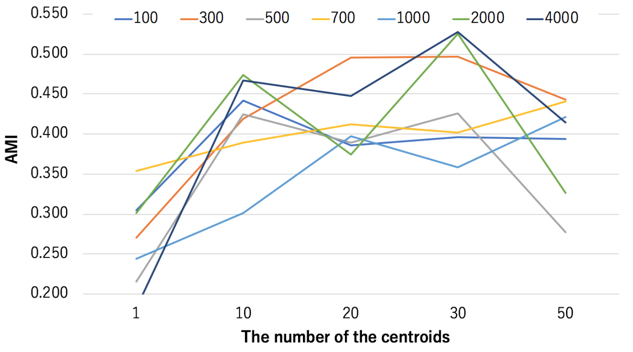

Fig. 5 **Influence of the batch size and the number of centroids on AMI.**

Does mini-batch make the performance worse? No. When the batch size is 4,000, we obtained the best performance. However the performance do not always decrease when we make the batch size smaller. We can conclude that the batch size is not correlate with the performance based on the result.

One of the positive aspects is that we can process the analysis with the part of given data. It can improve the scalability and it is strong advantage especially in time-series data processing. Because time-series data processing mostly needs much computation resource.

The result also provide one negative aspect that we have to think about making the method more stable. One of the strategies is that we evaluate the performance with the validation part of given data. However it sometimes takes extra cost.

Does multiple centroids work better than only 1 centroid? Yes. When $N = 1$, we obtain the worst performance if we change the batch size. Thus the multiple centroids always improve the performance. However it is not stable. The fact shows that we have to carefully consider the number of the centroids before applying the method.

Does mini-batch improve the scalability? Yes. The performance do not always decrease when we make the batch size smaller. The fact ensures that we can process the analysis with the part of given data and the method improves the scalability. We need the method to estimate the best batch size. If we apply the method with a specific application, then experimentally estimating the batch size with the validation dataset may run well.

5.2 Class-specific sequential pattern extraction

The aim of the evaluation is to answer the following questions. We do not apply clustering method because it is already evaluated in the previous section. Here we focus on our sequential pattern extraction method.

- Does the method deal with shift-invariance?
- Does the method find class-specific sequential patterns?

We created a dataset through the following steps.

(1) We randomly created base data $D = \{1, 2\}^{100\times 20}$ where the number of samples is 100 and the dimension of an element is 20. Since the range is limited, we can observe many sequences that occur frequently.

(2) We also created two queries $q_1, q_2 \in \{i \in \mathbb{N} | -10 \leq i \leq 0\}^6$ where the dimension of a query is 6 and the range of the query is different from that of base data. We can easily understand that a given sequence contains the query or not.

(3) We randomly replace the part of base data D with either query q_1 or q_2 and finally obtain dataset D'. We set the frequency of query q_1 is the same with that of query q_2. In other

[*3] `https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.random.random.html`

Table 2 The result of sequence pattern mining and pattern evaluation: $median_{all}$ is the median of the scores for all patterns. $median_{with}$ is the median of the scores for the patterns with the query. $median_{without}$ is the median of the scores for the patterns without the query.

threshold = 4

Len. of Subsec.	2	3	4	5	6	7
Num. of Patterns	238	198	73	31	134	108
$median_{all}$	2.173	2.173	2.335	2.173	0.325	2.173
$median_{with}$	2.410	2.410	3.169	2.254	10.090	4.832
$median_{without}$	2.173	0.445	2.173	2.173	0.325	2.173

threshold = 8

Len. of Subsec.	2	3	4	5	6	7
Num. of Patterns	87	69	26	0	61	31
$median_{all}$	2.905	2.769	3.103	0.000	0.393	3.169
$median_{with}$	3.169	2.769	3.169	0.000	10.090	4.832
$median_{without}$	0.393	0.393	2.769	0.000	0.393	3.038

threshold = 10

Len. of Subsec.	2	3	4	5	6	7
Num. of Patterns	59	41	19	0	40	21
$median_{all}$	3.169	3.038	3.169	0.000	3.038	3.428
$median_{with}$	3.299	10.090	3.234	0.000	10.090	4.832
$median_{without}$	3.038	3.038	3.038	0.000	3.038	3.169

words, the half of the samples contains query q_1 and the rest of the samples contains query q_2.

We extracted subsequences from the dataset D' and translated them into cluster ids. As a result we obtain a set of cluster id sequences. Although we do not apply clustering method here, the number of cluster ids is not big due to the limited range of the base data D.

Then we applied sequential pattern mining method and evaluated the patterns that we found. Since the range of the queries is different from that of the rest of the dataset D', we can easily understand that a pattern contains the query. We change the length of subsequence and the threshold of the frequency a.k.a support and conducted evaluation.

Table 2 and Figure 6, 7 show the result of the evaluation. Since the number of the patterns and the median of the scores for the patterns is important, we show them by Figure 6, 7.

It is natural that the number of the patterns is negatively correlated with the threshold of the frequency as we show the replationship in Figure 6. And based on Figure 7 we conclude that the median of the scores for the patterns is not correlated with the threshold. We answer the research questions in the rest of this subsection.

Does the method deal with shift-invariance? Yes. We found the query that appears randomly in the dataset in the most cases. The discoverability depends on the length of subsequences. Therefore the length of the subsequences is very important to find sequential patterns. To deal with the issue, we proposed the evaluation method of the patterns we found.

Does the method find class-specific sequential patterns? Yes. We found the query that appears in one class. We also confirm that the patterns that contains the query has higher scores than other patterns.

The dataset we used in the experiment is small. It has only two classes and only two queries. We have to evaluate the method with large dataset.

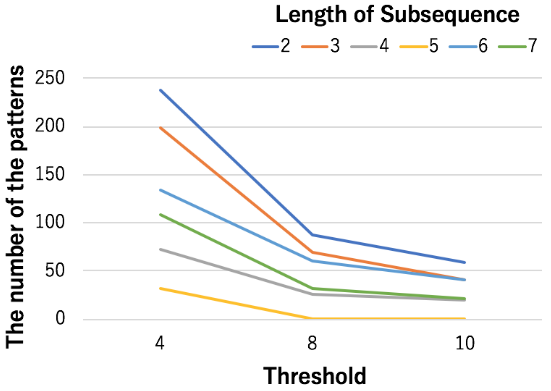

Fig. 6 Influence of the threshold and the length of subsequence on the number of the patterns.

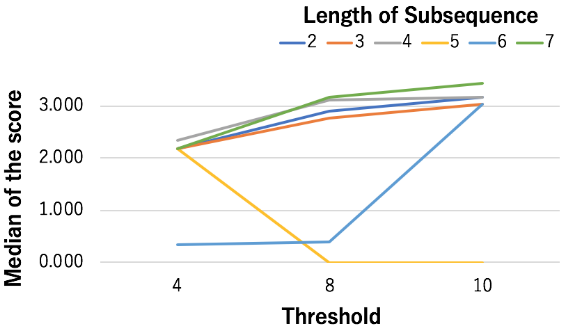

Fig. 7 Influence of the threshold and the length of subsequence on the median of the scores for the patterns

5.3 Experiment with the real CGM dataset

We applied the proposed method to the real CGM dataset and show an example of class-specific sequential patterns we found. The real CGM dataset is created through the clinical trial and contains 16 waves that is created 16 CGM users durning 2 weeks. We created the labels based on the users' information.

Figure 8 shows the example of class-specific sequential patterns. We select the pattern that has the maximum score. We decide the cluster representative by selecting one element that belongs to the cluster and has the minimum averaged distance to the other elements.

We do not discuss about medical issues here. And we do not provide the quantitative assessment of the result because we do not know the all class-specific sequential patterns in the dataset. However we confirm that the method can extract sequential patterns from the real dataset.

5.4 Discussion

Scalability: We propose mini-batch top-n k-medoids for the clustering method that runs on part of data and assigns cluster centroids to multiple elements on a cluster. We found the scalability of the method through the experiment with the dataset we manually created. However since the performance is not stable, we have to discuss about the strategy to make it more stable. One of the unsupervised methods to evaluate clustering results is calculating the averaged distance between clusters. By selecting the setting that minimize the averaged distance, we may obtain stable performance.

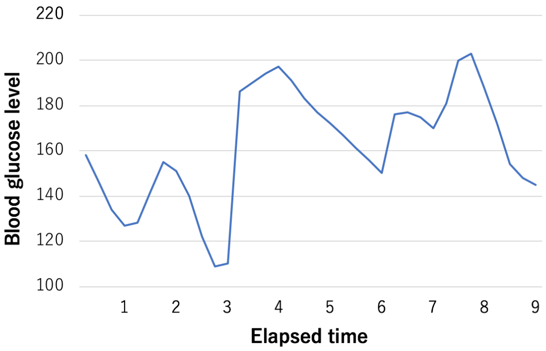

Fig. 8 Example of class-specific sequential pattern that is produced from the real CGM dataset.

As we discussed in the section 2.3, EMD that is one of the non-Euclid distance is 200 times slower than Euclid distance. We confirm that the proposed clustering method runs effectively when batch size is 10 % of the whole data. However we do not try 1% or 0.1 % because the dataset is not so large. Thus we have to confirm the scalability with much larger dataset.

One strategy to improve scalability is to filter out unimportant elements from the dataset before we select batch. We can design the filtering method based on machine learning technology or heuristics method. For example we focus on the part of the data that is produced on specific duration. Another strategy is to transform the problem into different one where we can use Euclid distance.

Scoring a pattern: We proposed the score for evaluating patterns to find label-specific patterns in section 4.4 and evaluate it in the previous section. We assign score to each patterns and see the median of the all scores. This is one way to handle the score distribution and we found its effectiveness through the experiment. Since the dataset that we used in the experiment contains two classes and is not large size, we have to evaluate the method with more complicated dataset. One advantage of the current dataset is that we easily create it due to the simple setting.

Practically we sometimes focus on patterns with high score and do not investigate all patterns. In this case it is enough to decide a threshold for score and investigate the number of patterns whose score are more than threshold. However this strategy provide us another parameter and makes tuning more difficult. Thus we have to discuss it carefully.

Cluster representative: We do not discuss about methods to decide cluster representative. We apply time-series clustering to transform given data sequences into cluster id sequences for grouping similar elements of data and finding patterns more effectively. Thus cluster representative is an important issue for us. Other studies discusses about the issue and call it motif discovery. It is one of the future work to discuss about method to decide cluster representative for CGM data analysis.

6. Conclusion

In this paper we propose a method of extracting characteristic sequential patterns from given pairs of labels and sequences. We deal with multiple technical issues for handling time-series data to extract characteristic sequential patterns: data normalization, shift-invariance and method scalability.

The method consists of 3 steps. At first we apply time-series clustering to transform CGM value sequences into cluster id sequences. Secondly we apply sequential pattern mining to extract frequently occurred sequences. Finally we evaluate each frequent sequence based on its correlation to a specific class. By clustering the data we group similar elements of data to find patterns more effectively. We propose mini-batch top-n k-medoids for the clustering method that runs on part of data and assigns cluster centroids to multiple elements on a cluster.

We answered the 5 research questions through the experiment with the dataset we manually created. We found the method runs well based on the results of the multiple experiments. We also applied the proposed method to the real CGM dataset and showed one example of the patterns. One of the future work is to discuss about the strategy to make it more stable because the performance is not stable.

References

[1] Rodbard, D.: Continuous glucose monitoring: a review of successes, challenges, and opportunities, *Diabetes technology & therapeutics*, Vol. 18, No. S2, pp. S2–3 (2016).

[2] Aghabozorgi, S., Shirkhorshidi, A. S. and Wah, T. Y.: Time-series clustering–A decade review, *Information Systems*, Vol. 53, pp. 16–38 (2015).

[3] Lin, J., Keogh, E. and Truppel, W.: Clustering of streaming time series is meaningless, *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, ACM, pp. 56–65 (2003).

[4] Chen, J. R.: Making subsequence time series clustering meaningful, *Data mining, fifth IEEE international conference on*, IEEE, pp. 8–pp (2005).

[5] Danne, T., Nimri, R., Battelino, T., Bergenstal, R. M., Close, K. L., DeVries, J. H., Garg, S., Heinemann, L., Hirsch, I., Amiel, S. A. et al.: International consensus on use of continuous glucose monitoring, *Diabetes Care*, Vol. 40, No. 12, pp. 1631–1640 (2017).

[6] Bergenstal, R. M., Ahmann, A. J., Bailey, T., Beck, R. W., Bissen, J., Buckingham, B., Deeb, L., Dolin, R. H., Garg, S. K., Goland, R. et al.: Recommendations for standardizing glucose reporting and analysis to optimize clinical decision making in diabetes: the Ambulatory Glucose Profile (AGP) (2013).

[7] Fournier-Viger, P., Lin, J. C.-W., Kiran, R. U., Koh, Y. S. and Thomas, R.: A survey of sequential pattern mining, *Data Science and Pattern Recognition*, Vol. 1, No. 1, pp. 54–77 (2017).

[8] Frantzi, K., Ananiadou, S. and Mima, H.: Automatic recognition of multi-word terms:. the c-value/nc-value method, *International journal on digital libraries*, Vol. 3, No. 2, pp. 115–130 (2000).

[9] Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U. and Hsu, M.-C.: Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth, *icccn*, IEEE, p. 0215 (2001).

[10] Gensler, A., Henze, J., Sick, B. and Raabe, N.: Deep Learning for solar power forecasting—An approach using AutoEncoder and LSTM Neural Networks, *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on*, IEEE, pp. 002858–002865 (2016).

[11] Patraucean, V., Handa, A. and Cipolla, R.: Spatio-temporal video autoencoder with differentiable memory, *arXiv preprint arXiv:1511.06309* (2015).

[12] Arthur, D. and Vassilvitskii, S.: K-means++: The Advantages of Careful Seeding, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 1027–1035 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1283383.1283494⟩ (2007).

[13] Sculley, D.: Web-scale K-means Clustering, *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, New York, NY, USA, ACM, pp. 1177–1178 (online), DOI: 10.1145/1772690.1772862 (2010).

[14] Vinh, N. X., Epps, J. and Bailey, J.: Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance, *Journal of Machine Learning Research*, Vol. 11, No. Oct, pp. 2837–2854 (2010).