A Study of kNN using ICU Multivariate Time Series Data

Dan Li[1], **Admir Djulovic**[1], **and Jianfeng Xu**[2]
[1]Computer Science Department, Eastern Washington University, Cheney, WA, USA
[2]Software School, Nanchang University, Nanchang, Jiangxi, China

Abstract— *A time series is a sequence of data collected at successive time points. While most techniques for time series analysis have been focused on univariate time series data at fixed intervals, there are many applications where time series data are collected at irregular and uncertain time intervals across multiple input variables. The uncertainty in multivariate time series makes analysis difficult and challenging. In this research, we study kNN classification approach applied to ICU multivariate time series data for patient's mortality prediction. We propose three time series representation strategies to handle irregular multivariate time series data. The experiments show the performance of these three methods in different settings. We also discuss the impact of imbalanced class distribution and the effect of k in kNN classification.*

Keywords: Classification, kNN, Multivariate Time Series.

1. Introduction

Time series data have become available in many fields of study including signal processing, pattern recognition, weather forecasting, electroencephalography, scientific simulation, etc. Consequently, there have been increased interests in analyzing and predicting time series data using data mining techniques. Besides all the common features of data mining, the research on time series analysis has its unique challenges because of the high-dimensionality and multi-granularity features of time series data. Therefore, it is a non-trivial task to develop efficient and effective data mining solutions for time series analysis.

The research on time series analysis has been focused on two main areas [1]: (1) to find proper representation methods to reduce high dimensional time series data; and (2) to define effective similarity/distance measures to compare multiple time series sequences. Many dimensionality reduction techniques have been proposed and implemented to transform original raw time series data into lower dimensional representations. These techniques include Discrete Fourier Transformation (DFT) [2], Discrete Wavelet Transformation (DWT) [3], Piecewise Aggregate Approximation (PAA) [4], Principal Component Analysis (PCA) [5], Symbolic Aggregate approXimation (SAX) [6], Single Value Decomposition (SVD) [7], etc. Correspondingly, similarity/distance measures have been discussed in the literature focusing on the comparison of time series data. The commonly used measures include point-to-point distance measures such as Euclidean distance and Dynamic Time Warping (DTW) [8], and edit distance measures for strings such as the Longest Common Subsequences (LCS) [9].

A time series is a sequence of data typically measured at successive points over time at uniform time intervals. The time *granularity* determines the interval length between two adjacent time points [10]. While most time series representation methods have been focused on the analysis of time series data at fixed and stable time granularity, there are many applications where time series data are collected at irregular and uncertain time intervals. For instance, the Computing in Cardiology Challenge (2012) provides the data sets for predicting mortality of Intensive Care Unit (ICU) populations [11]. The input time series measurements are recorded in chronological order within each patient record. Some measurements are recorded at regular intervals ranging from hourly to daily, while others are recorded at irregular intervals as required [11]. This is an example of irregular time intervals caused by intended irregular data collection patterns. There are other cases when the uncertainty and the irregularity are caused by inherent imprecise data collection tools and privacy-preserving transformations [12].

There have been some studies on the analysis of imprecise and uncertain time series data [13], [14], [15], [16]. In these studies, various distance measures and probabilistic query models have been proposed to embrace the uncertainty and the incompleteness of time series data. However, most of these studies have been limited to the cases where the time series data are collected over **uniform time intervals**, and the time series itself is **univariate time series**. In this paper, we focus on the study of **multivariate time series** measurements being collected at **irregular time intervals**. We use the data collected from patients' ICU stays [11] as an example and the ultimate goal is to design proper analytical solutions to deal with multivariate time series data at irregular intervals for ICU patients' mortality prediction. The rest of this paper is organized as follows: Section 2 describes the notations and the concepts of the related work on multivariate time series representations; Section 3 discusses the methodologies we have proposed for representing irregular ICU multivariate time series; The experimental results and analysis are provided in Section 4; Finally, concluding remarks along with directions for future improvements are presented in Section 5.

2. Multivariate Time Series Representation

Typically, a *multivariate time series* instance can be represented as an $m \times n$ two-dimensional matrix D:

$$D = \begin{pmatrix} d_{11} & d_{12} & \dots & d_{1j} & \dots & d_{1n} \\ d_{21} & d_{22} & \dots & d_{2j} & \dots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ d_{i1} & d_{i2} & \dots & d_{ij} & \dots & d_{in} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \dots & d_{mj} & \dots & d_{mn} \end{pmatrix} \quad (1)$$

where m is the number of input variables, n is the total number of time points, and d_{ij} is the data point measured on input variable i at time point t_j ($1 \leq i \leq m$ and $1 \leq j \leq n$). This representation assumes that all m input variables are measured along the same time sequence $(t_1, t_2, ..., t_n)$ and the time intervals between each adjacent pair are equal. Here *time interval* is a time unit measuring the sampling rate of a time series. One example of such representation is for weather forecasting where the weather-related variables (temperature, precipitation, wind, etc.) are collected over an even time interval, e.g., every ten minutes.

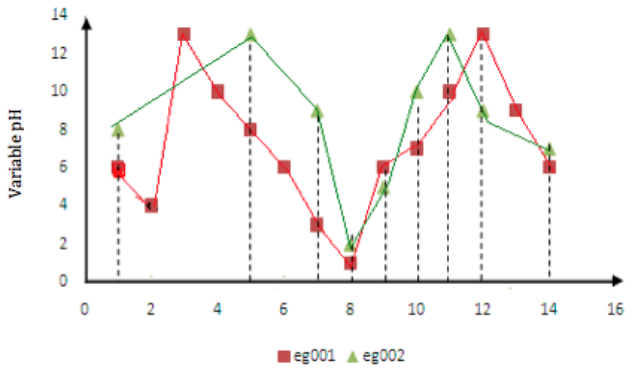

Fig. 1: Time Series at Regular and Irregular Interval

While the above matrix D is typically used to represent the multivariate time series at uniform time intervals, there are many real-world applications where the time series demonstrates various and irregular time intervals due to various data sampling rates. For instance, Figure 1 shows two time series sequences collected from two ICU patients (eg001 and eg002) on variable *pH* (a measure of the activity of a patient's hydrogen ion) [11]. The time series of patient eg001 has uniform intervals, while the time series of patient eg002 demonstrates various and irregular time intervals. Therefore, to be able to generally represent multivariate time series instances at either uniform or irregular time intervals, we modify the above matrix D into the following format:

$$D = \begin{pmatrix} (d_{11}, t_{11}) & \dots & (d_{1j}, t_{1j}) & \dots & (d_{1n_1}, t_{1n_1}) \\ (d_{21}, t_{21}) & \dots & (d_{2j}, t_{2j}) & \dots & (d_{2n_2}, t_{2n_2}) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ (d_{i1}, t_{i1}) & \dots & (d_{ij}, t_{ij}) & \dots & (d_{in_i}, t_{in_i}) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ (d_{m1}, t_{m1}) & \dots & (d_{mj}, t_{mj}) & \dots & (d_{mn_m}, t_{mn_m}) \end{pmatrix} \quad (2)$$

where m still denotes the number of input variables in a multivariate time series. Since the time series data are collected over uncertain intervals, the time series sequences from different input variables may end up with different number of data observations. To be more general, we use n_1, n_2, ..., and n_m to denote data observation numbers of each variable in an m-variate time series. Each pair (d_{ij}, t_{ij}) represents the data point of the i^{th} input variable measured at the j^{th} time stamp.

Note that D represents only one multivariate time series instance. If multivariate time series data are collected from multiple instances (e.g., data could be collected from multiple ICU patients, and the data set for each patient is a multivariate time series data set), we use $D_{all} = \{D_1, D_2, ..., D_p\}$ to represent a set of multivariate time series of p instances, where each D_i ($1 \leq i \leq p$) is an m-dimensional vector of the above structure.

Now let's introduce our problem definition: **Given an unlabeled multivariate time series Q, assign it to one of the two pre-defined classes {0, 1} by learning from a training set D_{all} of p multivariate time series instances**. Here, $D_{all} = \{(D_1, c_1), (D_2, c_2)...., (D_p, c_p)\}$ and $(c_1, c_2, ..., c_p) \in \{0, 1\}$ denote the known class labels of p training instances.

From the problem description, it is not hard to see that the problem itself is a typical classification problem. However, what makes this topic challenging is the needs of handling time series among multiple input variables and multiple instances. In other words, the number of available data observations varies among different input variables regarding each multivariate time series instance. Meanwhile, it also varies among different instances regarding the same input variable. Therefore, it is a non-trivial task to develop feasible solutions for the above classification problem.

3. Methodologies

Among many existing classification algorithms, we plan to use *k-Nearest-Neighbor* (kNN) approach because it is easy to implement and it handles numerical values conveniently and effectively. The key component of this research lies in how to define the distance measures among multivariate time series at various and irregular time intervals. The rest of this section discusses three approaches we have proposed and the basic idea of each approach is described as follows:

1) ***CaptureStatistics***: This approach captures the statistics of a time series using minimum, maximum, mean, and moving average and use these values to represent the time series.
2) ***DetectChanges***: This approach detects the key change-points in each time series, and uses these points to represent the entire time series.
3) ***AggregateSegments***: The main idea of this approach is to break a multivariate time series instance into multiple univariate time series, then each univariate time series is processed separately into disjoint segments and the aggregated distance is generated.

3.1 Capture Statistics

As shown in Figure 2, the *CaptureStatistics* algorithm is pretty straightforward. Step (1) is to normalize each time series using z-score normalization [17]. After normalization each time series has a mean of zero and a standard deviation of one. Step (2) is to capture the statistics including minimum, maximum, mean, and moving average for each variable in a multivariate time series. Remember that D_{all} denotes the entire training set of p time series data objects, and each object in D_{all} is a multivariate time series represented by Equation (2). Since each time series has variable number of observations at irregular time intervals, the mean and the moving average are calculated based on the existing observations in the time series. This process is repeatedly applied to all m variables in an m-variate time series. In Step (3), similar process is applied to the unlabeled test case Q. Step (4) compares each instance in D_{all} with Q and identifies the k-nearest neighbors using Euclidean distance (k is a pre-defined odd number). Step (5) uses the class label information from the k-nearest neighbors and applies majority vote to assign a class label to Q.

The *CaptureStatistics* algorithm transforms each variable-length time series into four numerical values. Thus, the variable number of observations from different time series instances is not a concern any more. We hope that these four statistical data values still capture the key features of a time series even though the temporal feature of the data is ignored.

3.2 Detect Changes

The main idea of the *DetectChanges* algorithm is to detect the key change-points in each time series and use these change-points to represent the time series. In other words, rather than focusing on the behavior of the entire time series, the trend of variations in the time series could be more informative than the time series itself. Even though the original multivariate time series has various numbers of data observations due to irregular measuring patterns, this approach uses the fixed number of change-points to represent each univariate time series. This makes the comparison between different data instances feasible.

Algorithm: `CaptureStatistics(`D_{all}`, Q, k)`

1) Apply z-score normalization to D_{all} and Q;
2) For each element in $D_{all} = \{D_1, D_2, ..., D_p\}$, find min, max, mean, and moving average;
3) Find min, max, mean, and moving average for Q;
4) Use kNN to find k-nearest neighbors of Q from D_{all};
5) Apply majority vote among k-nearest neighbors and assign a label to Q.

Fig. 2: ***CaptureStatistics*** **Algorithm.**

Figure 3 shows the *DetectChanges* algorithm. Comparing with *CaptureStatistics*, *DetectChanges* introduces one more parameter w which denotes the number of key points we plan to capture. The key of *DetectChanges* lies in Step (2) which detects w change-points in each time series by top-down piecewise segmentation approach [18]. These w representative data points are later used in Equation (3) to calculate the aggregated distance between each data object $D \in D_{all}$ and the test object Q, as shown in Step (3) of the algorithm. Here d_{ij} and q_{ij} denote the j^{th} change-point in the i^{th} univariate time series in D and Q, respectively.

$$ChangeDist(D, Q) = \sum_{i=1}^{m} \sum_{j=1}^{w} (d_{ij} - q_{ij}) \qquad (3)$$

The last two steps of *ChangeDetection* are similar to the last two steps of *CaptureStatistics*, which find the k-nearest neighbors of Q and assign a corresponding class label to it.

Algorithm: `DetectChanges(`D_{all}`, Q, k, w)`

1) Apply z-score normalization to D_{all} and Q;
2) Detect w change-points in D_{all} and Q;
3) Calculate the aggregated distance between Q and each $D \in D_{all}$ using Equations (3);
4) Use kNN to find k-nearest neighbors of Q from D_{all}.
5) Apply majority vote among k-nearest neighbors and assign a label to Q.

Fig. 3: ***DetectChanges*** **Algorithm.**

3.3 Aggregate Segments

Figure 4 shows the *AggregateSegments* algorithm. The main idea of this algorithm is to preprocess each individual time series into a set of equal-width segments. This way, the temporal feature of a time series is kept in the data set. Meanwhile, the time series sequences with different number of observations are transformed into the same number of segments through dimensionality reduction.

Among many time series representation techniques introduced in Section 1, one of the most commonly used dimensionality reduction approach is *Piecewise Aggregate Approximation* (PAA) [4], because PAA is intuitive and easy to implement. At the same time, it provides competitive performance comparing to other sophisticated dimensionality reduction techniques [6]. In PAA, an n-dimensional time series $d = (d_1, d_2, ..., d_n)$ is transformed into w disjoint equal-width segments $\bar{d} = (\bar{d_1}, \bar{d_2}, ..., \bar{d_w})$ $(w < n)$, where the i^{th} segment $\bar{d_i}$ is represented by finding the mean value of those data points falling into the i^{th} segment [6].

Besides PAA, we employ another dimensionality reduction technique SAX (Symbolic Aggregate approXimation) [6], which transforms a numeric time series into a symbolic representation. The idea of SAX is to identify the breakpoints from a highly Gaussian distributed time series, and use ordinal symbols to represent the segments between each pair of breakpoints. The authors in [6] have demonstrated that SAX is an effective representation on the classic data mining tasks of clustering, classification, anomaly detection, etc.

Algorithm: `AggregateSegments(`D_{all}`, Q, k, w)`

1) Apply z-score normalization to D_{all} and Q;
2) Transform D_{all} and Q into w equal-width segments using PAA and SAX, respectively;
3) Apply linear interpolation to fill in the segments without data observations;
4) Calculate the aggregated distance between Q and each $D \in D_{all}$ using Equations (4) and (5) separately;
5) Use kNN to find k-nearest neighbors of Q from D_{all}.
6) Apply majority vote among k-nearest neighbors and assign a label to Q.

Fig. 4: ***AggregateSegments* Algorithm.**

Unlike the original raw data set D_{all} where the multivariate time series instances have various number of data observations recorded at various time intervals, both PAA and SAX methods transform the instances in D_{all} into a set of $m \times w$ fixed-size matrices. Here m is the number of univariate time series and w denotes the number of segments in each time series. Note that w is usually much smaller than the number of original data observations in a time series. Similarly, the unlabeled instance Q is also transformed into an $m \times w$ matrix using PAA and SAX respectively, as shown in Step (2) of the *AggregateSegments* algorithm.

Now all the time series sequences have the same number of segments, but some segments may not have any valid data due to the unavailability of data observations in the corresponding segments. Therefore, in Step (3) of the algorithm, linear interpolation is employed to fill in these missing segments.

After Step (3), we are ready to use kNN classification to evaluate the distance between Q and each element $D \in D_{all}$. Since both D and Q have been transformed into m-variate time series with w segments in each time series, the distance between D and Q is calculated as the aggregated distance between each pair of segments in each univariate time series from D and Q. Equation (4) shows the distance function when both D and Q are represented as PAA sequences. Here $\overline{d_{ij}}$ and $\overline{q_{ij}}$ denote the piecewise aggregated average of the j^{th} segment in the i^{th} univariate time series in D and Q, respectively.

$$PAADist(D, Q) = \sum_{i=1}^{m} \sum_{j=1}^{w} (\overline{d_{ij}} - \overline{q_{ij}}) \quad (4)$$

Equation (5) shows the distance function when both D and Q are represented as SAX symbolic sequences. Here sym_d_{ij} and sym_q_{ij} denote the symbolic representations of the j^{th} segment in the i^{th} univariate time series in D and Q respectively, and $dist(sym\_d_{ij}, sym\_q_{ij})$ is the sub-distance function between two ordinal symbols, as illustrated in [6].

$$SAXDist(D, Q) = \sum_{i=1}^{m} \sum_{j=1}^{w} dist(sym\_d_{ij}, sym\_q_{ij}) \quad (5)$$

Again, the last two steps of *AggregateSegments* are similar to the last two steps of *DetectChanges*, which find the k-nearest neighbors of Q and assign a corresponding class label to it.

4. Experimental Results

We use the data sets for Computing in Cardiology (CinC) Challenge 2012 [11] to evaluate our algorithms. The focus of the challenge is to develop classification methods for patient-specific prediction of in-hospital mortality [11]. The training set for the challenge is a multivariate time series data set consisting of records from 4,000 ICU patients and there are 42 variables recorded at least once during the first 48 hours of a patient's ICU stay. However, not all 42 variables are

available all the times. Six of these variables are general descriptors (e.g., ID, age, gender, etc.), and the remainder are time series, for which variable number of observations may be available [11].

4.1 Experiments with Imbalanced Data Set

In the original training set of 4,000 ICU records, there are 554 positive cases. In other words, about 14% patients did NOT survive their hospitalization. Our experiments are initially designed using this imbalanced data set and 10-cross validation with stratified sampling is use to evaluate the algorithms. Tables 1-3 show the experimental results from Algorithms *CaptureStatistics*, *DetectChanges*, and *AggregateSegments* with PAA5 (5 is the number of segments in PAA).

The tables show both the precision and the recall on negative and positive classes separately, and the overall system accuracy. We can see that the precision on negative class is as high as 88% and the recall on negative class is as high as 96% when 3NN or 5NN is used. However, when we look at the prediction for positive instances, the results are not promising. In the best case, the precision on positive class is only 38% (5NN with *DetectChanges* approach) while the recall is only 12% in that case. This means the false negative rate is very high. This is not surprising though, because for such an imbalanced data set, the nearest-neighbor method is hard to identify the instances with fewer number of samples in the training set, i.e., positive instances in our case.

Table 1: Results from *CaptureStatistics* using Imbalanced Data.

	Class 0		Class 1		Accuracy
	precision	recall	precision	recall	
k=1	0.88	0.89	0.24	0.21	0.80
k=3	0.87	0.95	0.30	0.13	0.84
k=5	0.87	0.97	0.37	0.10	0.85

Table 2: Results from *DetectChanges* using Imbalanced Data.

	Class 0		Class 1		Accuracy
	precision	recall	precision	recall	
k=1	0.87	0.88	0.21	0.19	0.79
k=3	0.87	0.95	0.28	0.12	0.84
k=5	0.87	0.96	0.38	0.12	0.85

Table 3: Results from *AggregateSegments* with PAA5 using Imbalanced Data.

	Class 0		Class 1		Accuracy
	precision	recall	precision	recall	
k=1	0.87	0.88	0.22	0.20	0.79
k=3	0.87	0.95	0.25	0.11	0.83
k=5	0.87	0.96	0.29	0.08	0.85

4.2 Varying Data Distributions

To deal with imbalanced data distribution and improve the performance for positive instance prediction, we vary the data distributions by undersampling negative instances in the original data set. Figure 5 shows the performance of Algorithm *CaptureStatistics* when the distributions between negative and positive class instances are 1:1, 2:1, 3:1, and 7:1, respectively.

We can see that the precision and recall on negative class and the overall accuracy increase steadily as the percentage of negative instances increases, but the performance on positive class prediction decreases dramatically as the percentage of positive instances decreases. This experimental result indicates that the class distribution in the training set is an important factor for system performance. We should choose an appropriate distribution based on the goal of the system. If the goal is to improve the overall prediction accuracy, or to improve the prediction on negative cases, we should keep more negative samples in the training set. If the goal is targeted at true positive rate, then the undersampling should be done on negative class samples. To balance between positive and negative classes, we determine to use the training set of 3:1 distribution rate between negative and positive samples. The rest experimental results are based on this setting.

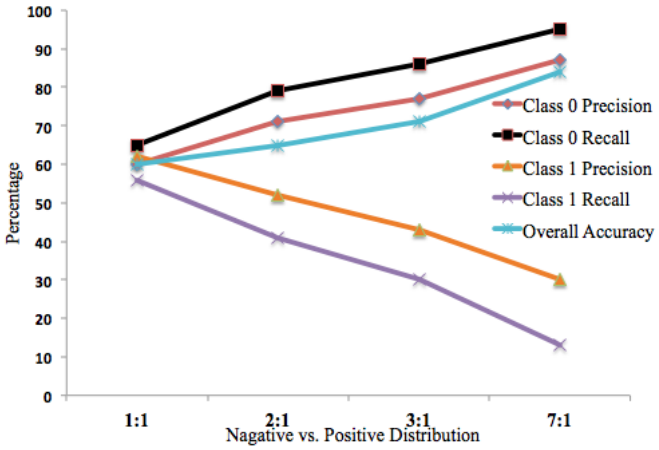

Fig. 5: Varying Data Distribution for *CaptureStatistics* Algorithm.

4.3 The Effect of k and the Comparison of All Algorithms

Table 4 demonstrates the effect of k in kNN classification. We can see that as k increases, the performance improves gradually until a certain point where the performance gets stable or slowly goes down. In particular, for Algorithms *CaptureStatistics* and *DetectChanges*, the overall results are best when k is set to 5. Under this setting, we have the highest recall on negative class, the highest precision on

Table 4: The effect of k and the comparison of all algorithms.

Method	k	Class 0		Class 1		Accuracy
		precision	recall	precision	recall	
Statistics	k=1	0.778	0.84	0.46	0.38	0.71
	k=3	0.78	0.88	0.52	0.34	0.74
	k=5	**0.77**	**0.97**	**0.63**	**0.24**	**0.76**
	k=7	0.78	0.92	0.59	0.31	0.75
Changes	k=1	0.77	0.82	0.42	0.35	0.70
	k=3	0.79	0.90	0.56	0.34	0.75
	k=5	**0.76**	**0.96**	**0.64**	**0.20**	**0.76**
	k=7	0.77	0.94	0.62	0.27	0.76
PAA10	k=1	0.76	0.80	0.38	0.34	0.68
	k=3	0.76	0.85	0.40	0.27	0.69
	k=5	0.76	0.88	0.45	0.26	0.72
	k=7	**0.76**	**0.90**	**0.47**	**0.22**	**0.72**
SAX10	k=1	0.77	0.80	0.39	0.35	0.68
	k=3	0.77	0.84	0.41	0.30	0.69
	k=5	0.76	0.87	0.42	0.25	0.70
	k=7	**0.76**	**0.89**	**0.43**	**0.23**	**0.71**

positive class, and the highest overall accuracy. For Algorithm *AggregateSegments* we test both PAA10 and SAX10 (10 denotes the number of segments), and the results show that setting k to 7 issues the best performance for both approaches while PAA slightly outperforms SAX.

Comparing all four approaches, the change-point detection and statistical approach have similar performance and both outperform PAA and SAX, the two piecewise segmentation approaches. This indicates that the statistical information and the change-points capture the key features of a time series well and the temporal features maintained through PAA and SAX segmentation approaches do not provide any additional useful information about the time series. In addition, the processing of segments in PAA and SAX could even cause the loss of meaningful time series information. Even though this finding is not encouraging, we have not found any research articles discussing this potential issue.

5. Conclusions

Time series analysis is different from traditional data mining tasks because of its high-dimensionality and multi-granularity features. Even though there is a large amount of research focusing on dimensionality reduction and representation of time series, there are a limited number of research papers discussing multivariate time series analysis, especially the time series at irregular and uncertain intervals.

This paper discusses the representations of irregular multivariate time series data and introduces a non-trivial classification problem using multivariate time series. We develop three kNN-based classification methods aiming at different time series representation strategies. ***CaptureStatistics*** uses minimum, maximum, mean, and moving average to capture the key features of a time series. ***DetectChanges*** uses the top-down segmentation approach to identify key change-points of a time series, and use these change-points to represent the entire time series. ***AggregateSegments*** is based on the piecewise aggregation approach and transforms each univariate time series into a fixed number of equal-width segments. We adopt both Piecewise Aggregate Approximation (PAA) and Symbolic Aggregate approXimation (SAX) approaches to segmenting the time series.

The experiments are conducted using ICU multivariate time series data for patient's mortality prediction. The original ICU data set is an imbalanced data set because most training instances have negative outcomes, i.e., most patients survive their ICU stays. This imbalanced data set has strong impact on the performance because kNN approach heavily depends on the class distribution of the data. With a small number of positive instances, the false negative rate would be inevitably high because kNN could not easily identify nearest neighbors with positive outcomes for a test case unless they present very strong common features of positive behavior. To deal with imbalanced data, undersampling method is used to change the distributions of positive and negative samples. The experiments show that as the distribution varies, the performance on positive and negative classes varies as well. The class with more samples usually has higher precision and higher recall than the class with fewer samples.

In addition, we conduct extensive experiments in different settings using our three time series handling algorithms. The experiments show the effect of k in kNN classification for each algorithm. Based on the results, we conclude that *CaptureStatistics* and *DetectChanges* outperform *AggregateSegments* in general. This indicates that the statistics and the change-points provide sufficient information to represent the time series and additional processing of time series could even downgrade the performance.

In future, we plan to develop weighted kNN approaches to handle imbalanced data distribution. We also plan to further investigate other sophisticated segmentation approaches and evaluate their effects on multivariate time series analysis.

References

[1] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: experimental comparison of representations and distance measures," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1542–1552, 2008.

[2] J. W. Cooley and J. W. Tukey, "An algorithm for the macine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, pp. 297–301, 1965.

[3] K. Chan and A. W. Fu, "Efficient time series matcing by wavelets," in *Proceedings of the 15th International Conference on Data Engineering [ICDE'99]*, March 1999, pp. 126–133.

[4] C. Guo, H. Li, and D. Pan, "An improved piecewise aggregate approximation based on statistical features for time series mining," in *Proceedings of the 4th international conference on Knowledge science, engineering and management [KSEM'10]*, Northern Ireland, UK, September 2010, pp. 234–244.

[5] I. D. Var, "Multivariate data analysis," *vectors*, vol. 8, p. 6, 1998.

[6] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery [DMKD'03]*, San Diego, CA, June 2003, pp. 2–11.

[7] J. A. Cadzow, B. Baseghi, and T. Hsu, "Singular-value decomposition approach to time series modelling," *Communications, Radar and Signal Processing, IEE Proceedings F*, vol. 130, no. 3, pp. 202–210, 1983.

[8] E. Keogh and C. A. Ratanamahatana, "Exact indexing of dynamic time warping," *Knowledge and Information Systems*, vol. 7, no. 3, pp. 358–386, 2005.

[9] L. Bergroth, H. Hakonen, and T. Raita, "A survey of longest common subsequence algorithms," in *Proceedings of the 7th International Symposium on String Processing and Information Retrieval*, 2000, pp. 39–48.

[10] C. Bettini, C. E. Dyreson, W. S. Evans, R. T. Snodgrass, and X. S. Wang, "A glossary of time granularity concepts," *Temporal Databases: Research and Practice. Lecture Notes in Computer Science*, vol. 1399, pp. 406–413, 1998.

[11] CinC2012, "Predicting mortality of icu patients: the physionet/computing in cardiology challenge 2012," [Online] http://physionet.org/challenge/2012/.

[12] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu, "Privacy-preserving data publishing: A survey of recent developments," *ACM Computing Surveys*, vol. 42, no. 4, pp. 14:1–14:53, June 2010.

[13] M. Dallachiesa, B. Nushi, K. Mirylenka, and T. Palpanas, "Uncertain time-series similarity: Return to the basics," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1662–1673, 2012.

[14] D. Suciu, A. Connolly, and B. Howe, "Embracing uncertainty in large-scale computational astrophysics," in *MUD Workshop*, 2009.

[15] T. T. Tran, L. Peng, B. Li, Y. Diao, and A. Liu, "Pods: a new model and processing algorithms for uncertain data streams," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, ser. SIGMOD'10, Indianapolis, Indiana, 2010, pp. 159–170.

[16] M.-Y. Yeh, K.-L. Wu, P. S. Yu, and M.-S. Chen, "Proud: a probabilistic approach to processing similarity queries over uncertain data streams," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, ser. EDBT '09, Saint Petersburg, Russia, 2009, pp. 684–695.

[17] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2006, ch. 6.

[18] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "Segmenting time series: A survey and novel approach," *an Edited Volume, Data mining in Time Series Databases*, vol. 57, pp. 1–22, 2004.