A Generative Model for Time Series Discretization Based on Multiple Normal Distributions

Sunil Gandhi
sunilga1@umbc.edu

Tim Oates
oates@cs.umbc.edu

Arnold P. Boedihardjo
arnold.p.boedihardjo@usace.army.mil

Crystal Chen
crystal.chen@usace.army.mil

Jessica Lin
jessica@gmu.edu

Pavel Senin
senin@hawaii.edu

Susan Frankenstein
Susan.Frankenstein@erdc.dren.mil

Xing Wang
xwang24@masonlive.gmu.edu

ABSTRACT

Discretization is a crucial first step in several time series mining applications. Our research proposes a novel method to discretize time series data and develops a similarity score based on the discretized representation. The similarity score allows us to compare two time series sequences and enables us to perform pattern learning tasks such as clustering, classification, and anomaly detection. We propose a generative model for discretization based on multiple normal distributions and create an optimization technique to learn parameters of these normal distributions. To show the effectiveness of our approach, we perform comprehensive experiments in classifying datasets from the UCR time series repository.
Keywords: Discretization, Time series, Classification

1. INTRODUCTION

Time series data is ubiquitous with its volume expanding at a rapid rate. The data can be high dimensional [6], and processing multivariate data directly can be computationally expensive. Also, there is a number of useful algorithms like hashing, suffix trees, and grammar induction that are not defined for real values. This has led to the development of several time series representation models, one of which is Symbolic Aggregate approXimation (SAX) [14]. This representation has been shown to be effective in several data mining tasks including classification, clustering, anomaly detection[14] and motif discovery[17]. Although SAX is widely used for discretization of time series data, it assumes that time series data is normally distributed and consequently divides the normal distribution into equiprobable regions for symbol assignment. This assumption does not always hold in practice and can negatively impact the performance of SAX-based algorithms. We create a time series representation called MINIONS (MultIple Normal distributIONS) that relaxes this assumption and models the time series as a generative process from multiple normal distributions.

PIKM'15, October 19, 2015, Melbourne, Australia.
© 2015 ACM. ISBN 978-1-4503-3782-3/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2809890.2809892.

Within the last decade several time series distance measures have been proposed. Nevertheless, Euclidean distance and dynamic time warping (DTW) are still relevant and give high-performing baselines for classifying time series data using nearest-neighbor (1NN) classification [25]. We propose a similarity measure based on our discretization mechanism and compare its classification performance with Euclidean distance and DTW.

Following are our main contributions:

1. We propose MINIONS, novel discretization mechanism for time series data based on the more general assumption that it is generated from multiple normal distributions.
2. We give an effective and efficient similarity measure based on the discretization mechanism proposed. We evaluate our similarity measure by classifying data from the UCR time series repository and comparing it with state of the art methods.
3. We propose an approximate algorithm to significantly improve the processing time of discretization mechanism while retaining its effectiveness.

The remainder of the paper is organized as follows. Section 2 discusses related work, Section 3 explains the problem definition and notation, Section 4 describes MINIONS, our discretization mechanism, its approximate version and similarity measure. In Section 5 we evaluate the performance of our algorithms. We conclude and discuss future work in Section 6.

2. RELATED WORK

The time series representation closest to ours is SAX [14], a symbolic representation of time series data. SAX makes the assumption that time series data is generated from a single normal distribution. Although [14] suggested that z-normalized time series data tends to have normal distribution, consequent work reported that accounting for the specificity in data distribution improves accuracy [21] [16]. Also, there is loss of information in the process of going from numerical data to a SAX string as SAX does not retain direction and distribution information related to a particular symbol[16]. [16] augments SAX symbols with direction information and shows that it improves accuracy on downstream tasks like classification. We relax the gaussianity assumption and retain distribution information related to symbol by assuming time series data is generated from multiple normal distributions. This allows us to combine the best of both worlds, i.e. symbolic representations and numerical representations.

Existing approaches for learning parameters of multiple normal distributions generating time series data can be broadly categorized into ones based on Gaussian processes and others based on Gaussian mixture models (GMMs). [2] gives an approach for time series prediction based on Gaussian processes, but their approach requires cubic time in the length of the time series which is prohibitively expensive. Another approach used in [22] is based on GMMs, which uses the Expectation Maximization (EM) algorithm to learn parameters of Gaussian distributions. Unlike our approach, the computation time required for GMMs to converge is unbounded. Also, when using [22], the number of normal distributions generating time series has to be specified. This assumes user knowledge of the number of normal distributions beforehand and that the number of normal distributions is constant across all time series in the datasets. Both these assumptions are unrealistic in real life and limit the algorithm's capacity. Our algorithm does not make these assumptions.

3. NOTATION AND PROBLEM DEFINITION

In this section we precisely define terms and notation used throughout the paper.

Time series: A time series is a sequence of data points ordered in time. We denote a time series by T and individual observations by $t_1, t_2, \ldots, t_n$, where n is the number of observations in a time series. For example, consider the *'cylinder bell funnel'* dataset that contains 3 classes called *'Cylinder'*, *'Bell'* and *'Funnel'*. Figure 1 shows an example time series of cylinder class from *'cylinder bell funnel'* dataset .

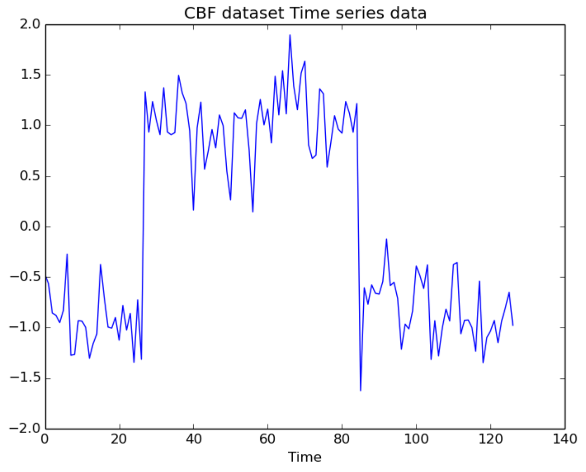

Figure 1: Cylinder time series from cylinder bell funnel dataset

Z-normalization is a process that brings the mean of a time series' values to zero and their standard deviation to one. Note that all time series from the UCR time series repository used in this paper are z-normalized as it is a common practice in the time series literature [20] and it gives better classification accuracy.

Distance/Similarity measure: Given two time series T_1 and T_2, both of length n, a distance/similarity measure is a function that gives how dissimilar/similar two time series are. Note that a similarity measure defined here may or may not satisfy the properties of a true distance metric.

Probability density function of the Normal Distribution is

$$N(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{1}$$

where parameter μ is the mean of the distribution and σ is the standard deviation.

Likelihood is the probability of an observed outcome given the parameter values of a model. The likelihood of observation x generated from a normal distribution with parameters μ and σ is denoted by $L(x|N)$ and is given by its probability density function, i.e., Equation 1.

Figure 2 shows the histogram of the time series data shown in Figure 1. On fitting the data using maximum likelihood method we get a normal distribution with $\mu = 0.009$ and $\sigma = 0.99$. Clearly, from Figure 2 we can observe that a single normal distribution does not fit the data well. We can increase the likelihood of the data by representing it using multiple normal distributions. To do so, we divide the data into subsets such that each subset is generated from a different normal distribution.

Let T be a time series of length n, then $t_{i:k}$ represents the subset of data between index i and k. Assume that $t_{i:k}$ comes from normal distribution $N(\mu_1, \sigma_1)$ and $t_{k:n}$ comes from normal distribution $N(\mu_2, \sigma_2)$. In this case, we define the likelihood of time series T as follows

$$L(T \mid N(\mu_1, \sigma_1), N(\mu_2, \sigma_2)) =$$
$$L(t_{i:k} \mid N(\mu_1, \sigma_1)) \times L(t_{i:k} \mid N(\mu_2, \sigma_2))$$

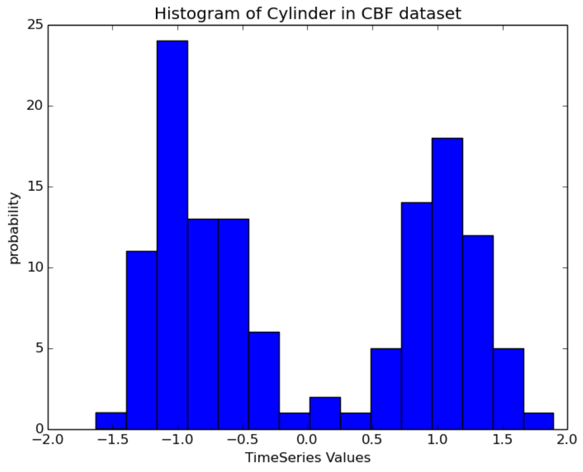

Figure 2: Histogram of Cylinder time series data from cylinder bell funnel dataset

3.1 Problem Definition

Given time series T_1 and T_2, we define a function that returns a similarity measure between two time series. To compare these time series, we divide the time series T_1 into subsets such that each subset comes from a separate normal distribution. Our goal is to learn the parameters of these normal distributions such that they maximize the likelihood of the data while reducing the number of normal distributions. We use the likelihood of time series T_2 with respect to the normal distributions' parameters learned for T_1 as the similarity measure.

There are two extremes when we learn parameters of normal distributions for time series T_1. One extreme is to have every point in time series T_1 in a separate subset and have a normal distribution with mean equal to its value and zero variance. This model will perfectly fit T_1 and the likelihood of this model will be maximum, but will require a large number of normal distributions. The other extreme will be to fit a single normal distribution that maximizes the likelihood of the complete time series. This model will not fit the data in the time series, similar to the example shown in Figure 2, and the likelihood of the data will be low. We optimize between these two extremes and try to find the minimum number of normal distributions that maximizes the likelihood of the data.

4. TIME SERIES SIMILARITY MEASURE

The following are some of the key tasks in time series data mining[14]. This list is not exhaustive, but highlights key areas.

1. Classification: Given a time series T, identify the class it belongs to using some training data [5].

2. Query by Content: Given a query time series T, and some (dis)similarity measure D(T,C) find the most similar time series in the database [23] [4] [8].

3. Clustering: Find natural groupings of the time series in the database under some similarity/dissimilarity measure D(T,C) [7] [11].

4. Anomaly Detection: Given a set of time series T, find time series that are anomalous compared to other time series in database. [3] [9] [24] .

All the above problems can be solved given a suitaible similarity measure. Thus, in this work our goal is to give a reasonable similarity measure.

To compute the similarity between time series we first discretize the time series and then calculate likelihood of other time series as similarity measure.

4.1 MINIONS: Discretization Based On Multiple Normal Distributions

Let $T = t_1, t_2, \ldots, t_n$ be the time series to be discretized. Our goal is to find the most likely model consisting of a minimum number of normal distributions with respect to the time series data. Let $M = N_1, N_2, \ldots, N_l$ be a model with l such normal distributions each representing a subset of time series T. Each subset of the time series is denoted by $S_1, S_2, \ldots, S_l$, where points in subset S_i are generated from normal distribution N_i. The likelihood of time series T with respect to the above model M is given by

$$L(T \mid M) = \prod_{i=1}^{l} L(S_i \mid N_i)$$

To avoid a scenario where each subset is a single point in the time series, we do not allow the variance of normal distributions to go below a certain threshold. We denote this threshold by a hyperparameter called $min\_variance$.

We want to find the most likely model for time series T, i.e we want to maximize $L(M \mid T)$

$$L(M|T) = (L(T|M) \times L(M))/L(T)$$

We ignore $L(T)$ because it remains constant and is independent of the model for time series T:

$$\begin{aligned} L(M \mid T) &\propto (L(T \mid M) \times L(M)) \\ &\propto L(M) \times \prod_{i=1}^{l} L(S_i \mid N_i) \end{aligned}$$

Since we would like our model to fit the data as accurately as possible, while using few normal distributions, we define $L(M)$ as $e^{-l \times \lambda}$. Here λ is a hyperparameter that represents the penalty for using extra normal distributions. It controls the granularity of the discretization mechanism. Larger λ will have fewer normal distributions, thus $L(T \mid M)$ will decrease. This formalization will prefer a simpler model with fewer normal distributions, thus avoiding overfitting. We use this prior because of its connections with L_1-Norm. [19]

$$L(M \mid T) \propto e^{-l \times \lambda} \times \prod_{i=1}^{l} L(S_i \mid N_i) \tag{2}$$

Given a time series of length n, there are exponentially many combinations of subsets of a time series. So it will take exponential time to maximize Equation 2 by calculating the likelihood for each possible model. To simplify the problem we make the following assumption:

Given two data values in a time series t_i, t_k such that $t_i < t_k$ and t_i, t_k are generated from the same normal distribution N, all data values t_j such that $t_i < t_j < t_k$ are generated from normal distribution N.

Thus, all the points adjacent to each other in the sorted time series can only come from one normal distribution. We believe that this assumption is valid as contiguous points are likely to be generated from the same normal distribution. Consider a time series with points $[0, 100, 48, 50, 52]$, two out of many possible ways of dividing this time series are $S_1 = [[0, 100], [48, 50, 52]]$ and $S_2 = [[0], [48, 50, 52], [100]]$. Both normal distributions corresponding to S_1 have $\mu = 50$ and different variances. We are rejecting models similar to S_1 by making the above assumption because we want to minimize overlap between normal distributions. This will enable us to use this discretization mechanism for algorithms like grammar induction that can only work with discrete data. Of the two options specified above, we select model S_2 irrespective of the likelihood of both models.

Due to this assumption, we can maximize Equation 2 efficiently. We first sort the time series data and then try to maximize over all possible ways of dividing the sorted time series. Denote the sorted time series by $ST = st_1, st_2, \ldots, st_n$. We maximize Equation 2 over sorted time series using dynamic programming. Below is the recurrence relation of the dynamic programming algorithm:

$$L(ST_{i:j}) = max \begin{cases} L(ST_{i:j} \mid N) * e^{-\lambda}, \\ max_{i \le k \le j} L(ST_{i:k}) \times L(ST_{k:j}) \end{cases} \tag{3}$$

Here $L(ST_{i:j} \mid N)$ is the likelihood of data from the sorted time between index i and j coming from a single normal distribution. $max_{i \leq k \leq j} L(ST_{i:k}) \times L(ST_{k:j})$ is the likelihood if we divide the time series from index i to k and k to j, for all values of k.

On implementation of this recurrence, we get l normal distributions that maximize Equation 2. We store these normal distributions using two arrays, the *params_normals* array stores parameters of each normal distribution and the *index_sorted* array stores the index to normal distribution that generated the corresponding point in sorted time series. We unsort *index_sorted* and store them in *index_normals* to preserve ordering. Now, algorithms that work only on discrete data like grammar induction algorithms can use indices in *index_normals* for processing. Thus, these indices can be used in all algorithms that can work with SAX symbols generated using [14].

To understand the algorithm and how unsorting helps in maintaining temporal ordering of the time series data, consider the example of following time series t generated from a step function with Gaussian noise,

$$t = [0.011, 0.045, -0.037, 5.022, 5.027, 4.98, 0.007, 0.089, 0.026, 4.983, 5.087, 4.956, 0.047, 0.007, 0.031, 5.013, 4.919, 5.035]$$

On sorting this time series, we get

$$sorted\_t = [-0.037, 0.007, 0.007, 0.011, 0.026, 0.031, 0.045, 0.047, 0.089, 4.919, 4.956, 4.98, 4.983, 5.013, 5.022, 5.027, 5.035, 5.087]$$

The next step in the discretization process is to find subsets in the sorted time series where each subset is generated from a single normal distribution. This can be done by maximizing the recurrence relation given in Equation 3. On maximizing the recurrence relation with $min\_variance = 0.4$ and $penalty = 1$ we get,

$$params\_normals = [(0.02511112, 0.4), (5.00244454, 0.4)]$$

$$index\_sorted = [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]$$

where *params_normals* contains the parameters of the learned normal distributions and *index_sorted* contains an index to *params_normals* that generated the corresponding point in the sorted time series. Next, we unsort the time series and store the unsorted indexes in *index_normals*.

$$index\_normals = [0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1]$$

We perform any further computations only on unsorted indexes *index_normals* which helps us preserve temporal proximity information. Notice that *index_normals* can be directly plugged in with any algorithm instead of the SAX representation. For example, we can use [18] for periodicity detection using *index_normals* instead of the SAX representation. In addition to *index_normals*, we also provide information about parameters of the learned normal distribution which can be useful for tasks like classification as shown in Section 5.

The time complexity of this algorithm is $O(n^2)$. We realize that the complexity of the algorithm is still high, especially for longer time series. Hence, in next section we give a faster approximate algorithm.

4.2 Approximation Of Discretization Mechanism

Time complexity of our discretization mechanism is $O(n^2)$. In many scenarios where time series length is large, $O(n^2)$ might be slow. However, we observe that points close to each other tend to be generated from the same normal distribution, we speed up the algorithm by grouping together points with similar numerical values. Here we introduce another parameter called the radius r that controls the precision of the approximation.

After sorting in the discretization algorithm, we start with the first point in ST and group all points within radius r of the first point. We then repeat this process with the next point outside radius r and so forth until all points in the series are processed. If this gives k divisions of the time series, we assume that each of the k divisions are generated from the same normal distribution. So either we merge two adjacent groups and represent them using the same normal distribution or we represent each group using individual normal distributions. We use the recurrence relation in Equation 3 to decide whether to merge groups or not. But instead of n points we are optimizing over k groups. Thus, the complexity comes down to $O(k^2 + n * \log n)$, where $k < n$. As we perform approximation step after the sorting, time complexity of the discretization is lower bounded by the time complexity of sorting time series data. We perform experiments with the approximation algorithm in section 5.2 and show its effectiveness.

4.3 Similarity Measure

In this section, we give a similarity measure between time series T_1 and T_2 based on the output of the discretization mechanism described in the previous section. We discretize T_1 and calculate $L(T_2 \mid T_1)$ as a similarity measure. We calculate $L(T_2 \mid T_1)$ as $\prod_{i=0}^{n} L(t_{2i} \mid N_i)$, where t_{2i} is the i^{th} point in time series T_2 and N_i is the normal distribution corresponding to the i^{th} point in time series T_1.This computation is fast and takes $O(n)$ time. Our similarity measure is not symmetric but it is trivial to make it symmetric by averaging $L(T_2 \mid T_1)$ and $L(T_1 \mid T_2)$. We use asymmetric version of similarity metric as making it symmetric increases the time taken to compute similarity.

This assumes that the two time series being compared have the same length. This holds true for all the datasets that we use for classification from the UCR time series repository. But this assumption might not hold true in practice. We solve this problem by using a window-based approach, where the window size is the size of the smaller of the two time series being compared. For example, if we want a similarity measure between time series T_1 and T_2 such that $size(T_1) > size(T_2)$, we compute $L(T_1(1 : size(T_2)) \mid T_2)$ as the distance measure. We then move the window by 1 to compute $L(T_1(2 : size(T_2) + 1) \mid T_2)$, and keep doing this until we are at the end of the time series T_1. We then return the average of the similarity computed for all windows. The average takes into account the similarity between all subsequeneces of T_1 and T_2. We do not take the minimum or maximum, thus not making the assumption that the best or the worst match between the subsequences of T1 and time series T2 is the discriminatory subsequence. This is particularly helpful in classification of trajectory data, as our own experiments confirmed.[12]

5. RESULTS

We proposed a time series discretization algorithm and a distance measure based on it in section 4. In this section we evaluate MINIONS and its more efficient approximation via classification accuracy on UCR time series datasets [10]. This approach was used in previous research for evaluating discretization mechanism [14] .

5.1 Classification Of Time Series Data

We evaluate our approach on 10 datasets taken from benchmark data at the UCR time series repository[10]. We compare accuracy with previously published results of competing classifiers based on Euclidean distance, DTW and SAX [14] [25]. To make the comparison fair, we use the 1-Nearest Neighbour classifier with exactly the same training and testing data as these classifiers. Note that [25] gives a state of the art classifier, but this classifier is not based on a similarity measure making it difficult to adapt directly to other data mining tasks. Also, [25] and [15] use SAX as the underlying discretization mechanism. These techniques can also be used with our discretization mechanism instead of SAX.

We use our dynamic programming algorithm given by recurrence relation in Equation 3 over complete time series data for discretization. As we do not use approximation algorithm for results in Table 1, we do not need the radius r hyperparameter. Our algorithm has two hyperparameters, λ and $min\_variance$. For choosing parameters, we use 60 percent of the training set for training and the remaining 40 percent for validation. We use the python hyperparameter optimization library hyperopt [1] for optimizing over hyperparameters on training data. We use a tree of parzen estimators (TPE) algorithm in hyperopt with λ generated from a uniform distribution with $low = 0.1$ and $high = 50$ and $min\_variance$ generated from a lognormal distribution with $\mu = -0.9$ and $\sigma = 1.39$. The number of maximum iterations for optimization of hyperparameters is 40. In case of a tie, we randomly pick one of the hyperparameter values. We then use optimized hyperparameters on the test data.

Table 1 shows the accuracy of SAX-based distance, Euclidean distance, dynamic time warping and our distance metric. Accuracy for SAX, EU and DTW are reported from previously published results from [14] and [25]. We were unable to find results for SAX on the 'ECGFiveDays' dataset, hence we have left it blank. As shown in Table 1, out of 10 datasets, SAX outperforms MINIONS on only two datasets. Euclidean distance outperform MINIONS on two datasets and ties on three datasets and Dynamic Time Warping outperforms MINIONS on five datasets. Although the improvements in classification accuracy over Eucledian distance and Dynamic Time Warping achieved by MINIONS are not significant, but both these methods do not give discretized representations. Computing such a representation is essential first step before using algorithms like grammar induction [17], motif discovery [13] and periodicity detection [18]. SAX gives such a representation but our method outperforms SAX in time series classification on eight out of ten datasets.

5.2 Effect Of Approximation Algorithm

In this section we perform experiments to show the effectiveness of the approximation algorithm and give some intuition on choosing the radius. To show effectiveness of approximation, we look at change in accuracy of the classification algorithm and speedup with respect to radius. We also try to understand the relationship between $min\_variance$, λ and radius.

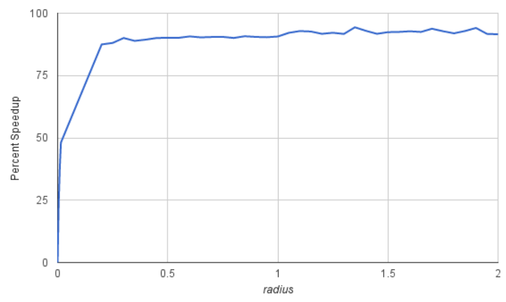

Figure 3: Speedup with respect to change in radius

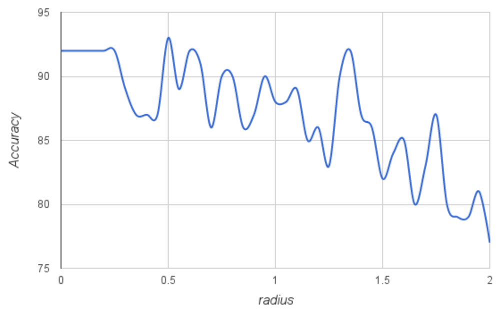

Figure 4: Classification Accuracy with change in radius

Figure 3 shows the speedup achieved by the approximation algorithm on the 'ECG200' dataset from the UCR time series repository with respect to radius. Figure 4 shows classification accuracy on the 'ECG200' dataset with respect to radius. The hyperparameter values in these experiments were $min\_variance = 19.1$ and $\lambda = 0.17$. Up to radius $r = 0.25$, accuracy does not change and is 92%, but the speedup increases to 88%. As we increase radius, speedup increases and there is a general trend of decrease in accuracy. Thus, we can achieve significant speedup (88%) without decrease in accuracy. This shows the effectiveness of our approximation algorithm.

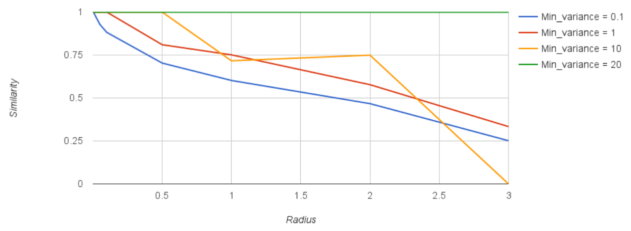

Figure 5: Effect of radius on clustering with change in min variance

We randomly picked a time series from the ECG200 dataset and compared the output of the approximation algorithm

Dataset	Classes	Training Set	Testing Set	Length	SAX	EU	DTW	MINIONS
CBF	3	30	900	128	89.6	85.2	**99.7**	94.2
ECG200	2	100	100	96	88	88	90.7	**92**
ECGFiveDays	2	23	861	136		**93.5**	76.8	88.7
Gun_point	2	50	150	150	82	91.3	77	**91.3**
50Words	50	450	455	270	65.9	63.1	**69**	66.4
synthetic_control	6	300	300	60	98	88	**99.3**	88.7
Coffee	2	28	28	286	53.6	75	82.1	**89.3**
Beef	5	30	30	470	43.3	53.3	50	**53.3**
FaceAll	14	560	1690	131	67	71.4	**80.8**	71.2
Lighting2	2	60	61	637	78.7	75.4	**86.9**	75.4
Average					74.01	78.42	81.23	81.1

Table 1: Classification Accuracy comparison for Euclidean distance, SAX, DTW and MINIONS

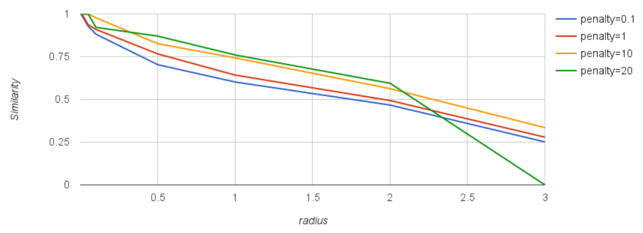

Figure 6: Effect of radius on clustering with change in penalty

and the original discretization mechanism. For comparing both the algorithms, we compare partitions generated by both algorithms using adjusted mutual information(AMI) [26]. Figure 5 shows the effect of varying radius on AMI for different values of *min_variance* with λ being same. Figure 6 shows effect of radius on λ with *min_variance* constant. We can observe from Figure 5 that until certain threshold of radius, output from the exact algorithm and the approximation are exactly same. But, there is significant speedup. For example, for $min\_variance = 10$, the threshold value is 0.5, where we get speedup of 99%. As the *min_variance* increases threshold value decreases. We see similar behaviour when we vary λ, but threshold values of radius are much smaller than Figure 5. To compute an appropriate value of radius, we recommend computing *min_variance* and *penalty* using cross validation and choosing radius equal to threshold value. This will ensure that we get maximum speedup without much difference in results from exact and approximate algorithm.

We show the effectiveness of our approximation algorithm on the *CBF* and *Gun_Point* datasets. We select the maximum radius that does not change discretization output for the time series and compute the speedup corresponding to it. We compute this speedup for 5 randomly selected time series in a dataset and average them. For all these experiments, we used $\lambda = 0.1$ and $min\_varaince = 0.1$. The average speedup for *CBF* and *Gun_Point* were 53.43% and 62.04%, respectively. Thus we can get significant speedup without change in discretization output using our approximation algorithm.

6. CONCLUSION AND FUTURE WORK

We proposed a MINIONS, novel discretization technique for time series data and a distance measure based on discretized data. We demonstrated that our approach is competitive with the existing distance metrics and can work with existing algorithms that require discretized data. We also provided an approximate discretization algorithm and showed its effectiveness.

There are several directions for future work based on MINIONS. One direction is to use grammar induction algorithms on top of MINIONS to understand structure of time series and use it for data mining tasks. This would help us create grammar based time series representation that is rotation invariant and performs well on shifted time series data. MINIONS lays foundation for grammar based representation as grammar induction algorithms cannot be implemented directly on real values. Also, grammar based representation with normal distribtutions as terminals will preserve structural as well as numerical information in the time series data.

We would like to perform classification on more datasets from UCR time series repository. We would also want to explore usage of MINIONS in a different and more complex domain like classification of trajectory data. Extension of MINIONS for multidimensional time series data would also be a interesting future work.

7. REFERENCES

[1] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[2] S. Brahim-Belhouari and A. Bermak. Gaussian process for nonstationary time series prediction. *Computational Statistics & Data Analysis*, 47(4):705–712, 2004.

[3] D. Dasgupta and S. Forrest. Novelty detection in time series data using ideas from immunology. In *Proceedings of the international conference on intelligent systems*, pages 82–87, 1996.

[4] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. *SIGMOD Rec.*, 23(2):419–429, 1994.

[5] P. Geurts. Pattern extraction for time series classification. In *Principles of Data Mining and Knowledge Discovery*, pages 115–127. Springer, 2001.

[6] J. Han and M. Kamber. *Data Mining, Southeast Asia Edition: Concepts and Techniques.* Morgan kaufmann, 2006.

[7] K. Kalpakis, D. Gada, and V. Puttagunta. Distance measures for effective clustering of arima time-series. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 273–280. IEEE, 2001.

[8] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM SIGMOD Record*, 30(2):151–162, 2001.

[9] E. Keogh, S. Lonardi, and B.-c. Chiu. Finding surprising patterns in a time series database in linear time and space. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 550–556. ACM, 2002.

[10] E. Keogh, X. Xi, L. Wei, and C. A. Ratanamahatana. The ucr time series classification/clustering homepage. *URL= http://www. cs. ucr. edu/˜ eamonn/time_series_data*, 2006.

[11] E. J. Keogh and M. J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *KDD*, volume 98, pages 239–243, 1998.

[12] J.-G. Lee, J. Han, X. Li, and H. Gonzalez. Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering. *Proceedings of the VLDB Endowment*, 1(1):1081–1094, 2008.

[13] Y. Li, J. Lin, and T. Oates. Visualizing variable-length time series motifs. In *SDM*, pages 895–906, 2012.

[14] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing sax: a novel symbolic representation of time series. In *Journal Data Mining and Knowledge Discovery*, 2007.

[15] J. Lin, R. Khade, and Y. Li. Rotation-invariant similarity in time series using bag-of-patterns representation. *Journal of Intelligent Information Systems*, 39(2):287–315, 2012.

[16] S. Malinowski, T. Guyet, R. Quiniou, and R. Tavenard. 1d-sax: A novel symbolic representation for time series. In *Advances in Intelligent Data Analysis XII*, pages 273–284. Springer, 2013.

[17] T. Oates, A. P. Boedihardjo, J. Lin, C. Chen, S. Frankenstein, and S. Gandhi. Motif discovery in spatial trajectories using grammar inference. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1465–1468. ACM, 2013.

[18] R. Otunba, J. Lin, and P. Senin. Mbpd: Motif-based period detection. In *Trends and Applications in Knowledge Discovery and Data Mining*, pages 793–804. Springer, 2014.

[19] T. Park and G. Casella. The bayesian lasso. *Journal of the American Statistical Association*, 103(482):681–686, 2008.

[20] P. Patel, E. Keogh, J. Lin, and S. Lonardi. Mining motifs in massive time series databases. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 370–377. IEEE, 2002.

[21] N. D. Pham, Q. L. Le, and T. K. Dang. Two novel adaptive symbolic representations for similarity search in time series databases. In *Web Conference (APWEB), 2010 12th International Asia-Pacific*, pages 181–187. IEEE, 2010.

[22] R. J. Povinelli, M. T. Johnson, A. C. Lindgren, and J. Ye. Time series classification using gaussian mixture models of reconstructed phase spaces. *Knowledge and Data Engineering, IEEE Transactions on*, 16(6):779–783, 2004.

[23] R. A. G. Psaila and E. L. Wimmers Mohamed &It. Querying shapes of histories. 1995.

[24] P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, and S. Frankenstein. Time series anomaly discovery with grammar-based compression.

[25] P. Senin and S. Malinchik. Sax-vsm: Interpretable time series classification using sax and vector space model. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 1175–1180. IEEE, 2013.

[26] N. X. Vinh, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1073–1080. ACM, 2009.