Title Of Thesis:  A GENERATIVE MODEL FOR TIME SERIES BASED ON MULTIPLE NORMAL DISTRIBUTIONS

Name of Candidate:  Gandhi Sunil Rajkumar
M.S. in Computer Science,
May 2015

Thesis and Abstract Approved:  __________________________
Dr. Tim Oates
Professor
Department of Computer Science and Electrical Engineering

Date Approved:  __________________________
Discretization is a crucial first step in several time series mining applications. Our research proposes a novel method to discretize time series data and develop 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 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 and trajectory datasets. We also explore the usefulness of grammar induction technique in discriminating shifted time series and completely different time series.

*Keywords: Discretization, Time series, Classification*
A generative model for time series based on multiple normal distributions

by

Gandhi Sunil Rajkumar

Thesis submitted to the Faculty of the Graduate School of the University of Maryland in partial fulfillment of the requirements for the degree of M.S. Computer Science

2015
Dedicated to friends and family who made journey worthwhile and mentors who guided me to destination
# TABLE OF CONTENTS

DEDICATION ................................................................. ii

ACKNOWLEDGMENTS ......................................................... iii

LIST OF FIGURES ............................................................. vi

LIST OF TABLES ............................................................. viii

Chapter 1  INTRODUCTION .................................................. 1

Chapter 2  BACKGROUND AND RELATED WORK ......................... 3

  2.1  Preliminaries ......................................................... 3
  2.2  Hilbert Space Filling Curve ......................................... 6
  2.3  Related Work .......................................................... 9

Chapter 3  LEARNING MULTIPLE NORMAL DISTRIBUTIONS OVER
            TIME SERIES DATA ............................................. 12

  3.1  Time Series Similarity Measure ................................... 12
    3.1.1  Problem definition ............................................. 13
    3.1.2  Discretization .................................................. 14
    3.1.3  Approximation of Discretization Mechanism .................. 18
3.1.4 Similarity measure .................................................. 18

Chapter 4 EVALUATION AND EXPERIMENTAL RESULTS ........... 20

4.1 Classification of time series data ........................................ 20
4.2 Effect of approximation algorithm ..................................... 22
4.3 Spatial Trajectory Classification ....................................... 26

Chapter 5 GRAMMAR INDUCTION OVER TIME SERIES DATA ....... 34

5.1 Motivation ................................................................. 34
5.2 Background and Related work .......................................... 37
  5.2.1 Sequitur Algorithm ................................................ 37
  5.2.2 Minimal grammar parsing (MGP) ............................... 39
5.3 Grammar Induction over time series data ............................ 40
5.4 Results and Discussion ............................................... 42

Chapter 6 CONCLUSION AND FUTURE WORK ......................... 45

REFERENCES ...................................................................... 46
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Cylinder time series from cylinder bell funnel dataset</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Histogram of Cylinder time series data from cylinder bell funnel dataset</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>Hilbert space filling curve of (a)order=1, (b)order=2, (c)order=3, (d)order=4,(e)order=5, (f)order=6 in 2-dimensional space</td>
<td>7</td>
</tr>
<tr>
<td>2.4</td>
<td>Conversion of data from 2-dimensional space to 1-dimensional space using hilbert curve of order 2</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>Two overlapping normal distribution with mean = 50</td>
<td>16</td>
</tr>
<tr>
<td>4.1</td>
<td>Speedup with respect to change in radius</td>
<td>23</td>
</tr>
<tr>
<td>4.2</td>
<td>Classification Accuracy with change in radius</td>
<td>24</td>
</tr>
<tr>
<td>4.3</td>
<td>Effect of radius on clustering with change in min variance</td>
<td>25</td>
</tr>
<tr>
<td>4.4</td>
<td>Effect of radius on clustering with change in penalty</td>
<td>25</td>
</tr>
<tr>
<td>4.5</td>
<td>Histogram of length of animal tracking trajectories</td>
<td>27</td>
</tr>
<tr>
<td>4.6</td>
<td>Histogram of length of hurricane tracking trajectories</td>
<td>28</td>
</tr>
<tr>
<td>4.7</td>
<td>Features learnt for the animal tracking dataset for the RB-TB method</td>
<td>30</td>
</tr>
<tr>
<td>4.8</td>
<td>Features learnt for the hurricane tracking dataset for the RB-TB method</td>
<td>31</td>
</tr>
<tr>
<td>4.9</td>
<td>Effect of hilbert curve order parameter on accuracy on the animal tracking dataset</td>
<td>32</td>
</tr>
</tbody>
</table>
4.10 Effect of hilbert curve order parameter on accuracy on the hurricane tracking dataset ........................................ 33

5.1 Two identical but shifted time series ........................................ 35

5.2 Minimal grammar parsing (MGP) ........................................ 40
LIST OF TABLES

4.1 Classification Accuracy comparison for Euclidean distance, SAX, DTW and our algorithm ............................................. 21

4.2 Classification Accuracy comparison for TB-Only and RB-TB methods and our algorithm ............................................. 29

5.1 Grammar rules generated by sequitur for string ”a a a a b a a a a” ............................................. 39
Time series data is ubiquitous with its volume expanding at a rapid rate. Hence, time series data mining is an important area of research and has useful applications in a variety of domains including health care, shape recognition, and spatial trajectory data analysis. Consequently, time series data mining has been studied in diverse research areas.

Time series can be high dimensional (Han & Kamber 2006), and processing multivariate data directly can be computationally expensive. Also, there are several algorithms like hashing, suffix trees, and grammar induction that are not defined for numerical values. This has led to the development of several time series representation models. Symbolic aggregate approximation (SAX) is one such representation. This representation has been shown to be effective in several data mining tasks including classification, clustering, anomaly detection (Lin et al. 2007) and motif discovery (Oates et al. 2013). 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. This is evident from the results we give in Chapter 4. The time series from the ‘cylinder bell funnel’ dataset shown in Figure 2.1 and Figure 2.2 are good examples of data that are not normally distributed. We relax...
this assumption and model the time series as a generative process from multiple normal distributions.

Within the last decade there are several time series distance measures that 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 algorithm (1NN) classification (Senin & Malinchik 2013). We propose a similarity measure based on our discretization mechanism and compare its classification performance with Euclidean distance and DTW.

To summarize, the following are our contributions:

1. We propose a 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 similarity measure based on the discretization mechanism proposed. We evaluate our similarity measure by classifying data from the UCR time series repository and comparing the performance with state of the art methods. We also evaluate our measure on trajectory datasets.

3. We propose an approximate algorithm to significantly improve the discretization processing time while retaining its effectiveness.

The remainder of the thesis is organized as follows, Chapter 2 discusses background and related work, Chapter 3 explains the problem definition, our discretization mechanism, its approximate version and similarity measure. In Chapter 4 we evaluate the performance of our algorithms. We explore the usefulness of grammar induction algorithms in Chapter 5. We conclude and discuss future work in Chapter 6.
Chapter 2

BACKGROUND AND RELATED WORK

2.1 Preliminaries

In this section we precisely define terms and notations used throughout the thesis.

**Time series**: A time series is a sequence of data points ordered in time. We denote 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 2.1 shows an example time series of cylinder class from 'cylinder bell funnel' dataset. Our goal in this thesis is to compare two time series.
**Z-normalization** is a process that brings the mean of a time series $T$ to zero and its standard deviation to one. Note that all time series from the UCR time series repository used in this thesis are z-normalized as it is a common practice in the time series literature (Patel et al. 2002) and it gives better classification accuracy.

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

\[
N(x, \mu, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}
\]

where parameter \( \mu \) is the mean of the distribution and \( \sigma \) is the standard deviation. In this thesis we make the assumption that time series are generated from a set of normal distributions, and we try to recover the parameters of these normal distributions.

**Likelihood** is the probability of an observed outcome given the parameter values of a model. The likelihood of observation \( x \) generated from normal distribution with parameters \( \mu \) and \( \sigma \) is denoted by \( L(x|N) \) and is given by its probability density function i.e. Equation 2.1.

Figure 2.2 shows the histogram of the time series data shown in Figure 2.1. On fitting the data using maximum likelihood method we get a normal distribution with \( \mu = 0.009 \) and \( \sigma = 0.99 \). Clearly, from Figure 2.2 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 data 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

\[
L(T \mid N(\mu_1, \sigma_1), N(\mu_2, \sigma_2)) = L(t_{i:k} \mid N(\mu_1, \sigma_1)) \times L(t_{i:k} \mid N(\mu_2, \sigma_2))
\]
2.2 Hilbert Space Filling Curve

To compare two trajectories we have to convert multi-dimensional trajectory data (time, latitude, longitude) into a sequence of scalars. We use Hilbert space filling curve (SFC) for this transformation. In this section we introduce hilbert space filling curve and how they can be used for transformation of 2-dimensional data to 1-dimension.

Space filling has been a topic of interest for mathematicians since the late 19th century when the first graphical representation was proposed by David Hilbert in 1891 (Hilbert 1891). Space filling curves provide mapping between d-dimensional space and...
1-dimensional space. This mapping can be thought of as dividing d-dimensional space into d-dimensional hypercubes with a line passing through each hypercube.

![Hilbert space filling curve](image)

**Fig. 2.3.** Hilbert space filling curve of (a) order=1, (b) order=2, (c) order=3, (d) order=4, (e) order=5, (f) order=6 in 2-dimensional space

Hilbert space filling curve are one of the most widely space filling curves because of their locality preserving property. According to this property, if two points are close in 1-dimensional space, then they are necessarily close in d-dimensional space. Although the converse of this property is not true, i.e. there can be points which are close in d-dimensional space, but are far apart in 1-dimensional space. Figure 2.3 shows Hilbert space filling curves of order = 1 to order = 6. The basic curve has Hilbert curve order 1 and is shown in Figure 2.3 (a). For generation of a Hilbert space filling curve of kth order...
in 2-dimensional space, each vertex of the Hilbert curve with k-1th order is replaced with the basic curve rotated and/or reflected to fit the curve. Thus, if we replace every vertex in Figure 2.3 (a) with the basic curve and rotate it in the appropriate direction Hilbert curve in Figure 2.3 (b) can be generated. Note that the direction of rotation and/or reflection remains the same irrespective of Hilbert curve order.

![Figure 2.4](image)

**Fig. 2.4.** Conversion of data from 2-dimensional space to 1-dimensional space using hilbert curve of order 2

The basic Hilbert space filling curve with order = 1 divides the square into 4 sub-squares as shown in Figure 2.4. For the Hilbert curve with order = 2, each sub-square of the Hilbert curve with order = 1 is further divided into 4 sub-squares. This process goes on as order of hilbert curve increases. Thus the number of sub-squares in 2-dimensional Hilbert space filling curve is $4^{order}$. To convert 2-dimensional data points to 1-dimensional points, each sub-square is integer numbered from 0 to $4^{order} - 1$ starting from the lower left corner as 0 and the lower right corner as $4^{order} - 1$. All other sub squares are numbered in order of occurrence of corresponding vertex of the Hilbert space
filling curve from lower left corner. This ordering is shown in Figure 2.4 for Hilbert curves of order $= 1$ and order $= 2$. It also shows conversion of series of 2D points to a sequence of scalars. The sequence of scalars generated after conversion to 1-dimensional points is $T = [0, 3, 2, 2, 2, 7, 7, 8, 11, 13, 13, 2, 1, 1]$

2.3 Related Work

In the past two decades, there has been an enormous interest in time series data mining. There have been several proposed representations for time series data. The representation closest to ours is SAX (Lin et al. 2007), a symbolic representation of time series data. SAX takes the raw time series data as input and converts it into a set of symbols. This representation is well received and has been used widely in tasks like classification, clustering (Lin et al. 2007), anomaly detection (Lin, Khade, & Li 2012) and motif discovery (Oates et al. 2013). But, SAX makes the assumption that time series data is generated from a single normal distribution (Lin et al. 2007). This assumption is not always true and might hurt the performance of SAX based algorithms. Also, there is loss of information in the process of going from numerical data to SAX string as SAX does not retain distribution information related to a particular symbol. We solve both these problems 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. We can treat the index to normal distributions in a time series as a sequence of symbols and use algorithms that work only for discrete data, including hashing, Markov models, and suffix trees. We also retain the distribution information of data for each symbol and use it to calculate similarity between time series.

The assumption of time series data being generated from multiple normal distributions has been explored in previous research. The 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 (GMM). (Brahim-Belhouari & Bermak 2004) 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 expensive for large time series data. Another approach used in (Povinelli et al. 2004) is based on GMM, which uses the Expectation Maximization (EM) algorithm to learn parameters of Gaussian distributions. Unlike our approach, the computation time required for GMM to converge is unbounded. Also, when using (Povinelli et al. 2004), the number of normal distributions generating time series has to be specified. This assumes user’s 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 algorithms’ capacity. Our algorithm does not make these assumptions.

We calculate the similarity measure between two time series and use it for nearest neighbour classification of data. There are several techniques proposed in the literature for comparing time series and time series classification. To the best of our knowledge, SAX-VSM (Senin & Malinchik 2013) is the state of the art technique for time series classification, but it does not give a similarity measure and thus cannot be applied directly for other data mining tasks like clustering, query by content, etc. Euclidean distance and DTW are two similarity measures which are related and are used extensively. However, Euclidean distance based techniques are sensitive to noise and shift along the temporal dimension. One way to mitigate this is to use an elastic distance measure such as Dynamic Time Warping (DTW) (Keogh & Ratanamahatana 2005). However, while Dynamic Time Warping based techniques work well for short sequences, they consistently fail to produce satisfactory results when the sequence is long (Lin & Li 2009). We compare our distance with Euclidean distance, SAX based distance and DTW in the experimental results.
We evaluate our similarity measure on standard time series datasets used in (Lin et al. 2007) and trajectory classification datasets. We classify trajectories from an animal tracking dataset and a hurricane dataset using the same training and test dataset as used in (Lee et al. 2008). We show that our similarity measure gives competitive performance on the animal tracking dataset, but (Lee et al. 2008) outperforms our similarity measure on hurricane dataset. Although, (Lee et al. 2008) gives better accuracy on hurricane dataset, (Lee et al. 2008) does not gives a similarity measure and thus is unsuitable for data mining tasks other than classification like clustering and query by content.
LEARNING MULTIPLE NORMAL DISTRIBUTIONS
OVER TIME SERIES DATA

3.1 Time Series Similarity Measure

The following are some of the key tasks in time series data mining (Lin et al. 2007). 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 (Geurts 2001).

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 (Psaila & Wimmers Mohamed &It 1995) (Faloutsos, Ranganathan, & Manolopoulos 1994) (Keogh et al. 2001).


4. Anomaly Detection: Given a set of time series T, find time series that are anomalous compared to other time series in database. (Dasgupta & Forrest 1996) (Keogh,
Lonardi, & Chiu 2002).

All the above problems can be solved given a suitable 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.

3.1.1 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 time series $T_1$ into subsets such that each subset comes from a separate normal distribution. Our goal is to learn parameters of these normal distribution such that they maximize the likelihood of the data while reducing the number of normal distributions. We use likelihood of time series $T_2$ with respect to normal distributions parameters learned for $T_1$ as 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 2.2, 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.
3.1.2 Discretization

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

$$L(T \mid M) = \prod_{i=0}^{l} L(S_i \mid N_i)$$

To avoid a scenario where each subset is a single point in the time series, we do not allow 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 $L(M \mid T)$

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

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

$$L(M \mid T) = (L(T \mid M) \times L(M))$$

$$= L(M) \times \prod_{i=0}^{l} L(S_i \mid N_i)$$

Since, we would like our model to fit the data as accurately as possible, while using few normal distribution, we define $L(M)$ as $e^{-l \times \lambda}$. Here $\lambda$ is the hyperparameter that represents the penalty for using extra normal distributions. It controls the granularity of the
discretization mechanism. Higher $\lambda$ 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.

$$(3.1) \quad L(M \mid T) = e^{-\lambda \times \lambda} \times \prod_{i=0}^{t} L(S_i \mid N_i)$$

Given a time series of length $n$, there are exponentially many combinations of subsets of time series. So it will take exponential time to maximize Equation 3.1 by calculating likelihood for each possible model. To simplify the problem we make the following assumption:

*Given two points in time series $t_i, t_k$ such that $t_i < t_k$ and $t_i, t_k$ are generated from the same normal distribution $N_1$, all points $t_j, t_i < t_j < t_k$ are generated from normal distribution $N_1$*

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 be generated from the same normal distribution. Consider time series with points $[0, 100, 48, 50, 52]$, the two out of many possible ways of dividing this time series are $S_1 = [[0, 100], [48, 50, 52]]$ and $S_2 = [[0], [48, 50, 52], [100]]$. The two normal distributions corresponding to $S_1$ have $\mu = 50$ and are shown in figure 3.1. 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. Out of two options specified above, we select model $S_2$ irrespective of the likelihood of both models.
Fig. 3.1. Two overlapping normal distribution with mean = 50

Due to this assumption, we can maximize Equation 3.1 efficiently. We first sort the time series data and then try to maximize over all possible ways of dividing the sorted time series. Let’s denote the sorted time series by $ST = s_{t_{1}}, s_{t_{2}}, \ldots, s_{t_{n}}$. We maximize Equation 3.1 over the sorted time series using dynamic programming. Following is the recurrence relation of the dynamic programming algorithm:

$$L(ST_{i:j}) = \max \begin{cases} L(ST_{i:j} \mid N) \ast e^{-\lambda}, \\ \max_{i \leq k \leq j} L(ST_{i:k}) \ast L(ST_{k:j}) \end{cases}$$

(3.2)
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 3.1. We unsort indexes to the normal distributions and store them in $index\_normal$ to preserve ordering. Now, algorithms which work only on discrete data like grammar induction algorithms can use these indices for processing. Thus, these indices can be used in all algorithms that can work with SAX symbols generated using (Lin et al. 2007). Algorithm 1 gives the complete algorithm.

**Algorithm 1: Discretization algorithm**

**Input**: TimeSeries $T$, $\text{min\_variance}$, $\lambda$

$ST = \text{sort}(T)$

Maximize following recurrence relation :

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

$L(ST_{i:j} \mid N)$ is likelihood of data from sorted time between index $i$ and $j$ coming from single normal distribution and $e^{-\lambda}$ is penalty because of each normal distribution.

$normals = \text{array of maximum likelihood fit of normal on subsets that maximize above recurrence relation}$

$indexes = \text{index in normals array for all points in sorted time series}$

$normals, indexes = \text{unsort}(indexes)$
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 the next section we give faster approximate algorithm.

### 3.1.3 Approximation of Discretization Mechanism

The time complexity of algorithm 1 is \( O(n^2) \). In many scenarios where time series length is large, \( O(n^2) \) might be slow. However, points close to each other tend to be generated from the same normal distribution. Thus, we speed up the algorithm by grouping together points with similar numerical values. Here we introduce another parameter called the radius \( r \) that will control the precision of approximation.

After sorting in Algorithm 1, we start with the first point in \( ST \) and group all points within radius \( r \) of first point. We then repeat the same 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 time series, we assume that each \( k \) 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 distribution. We use recurrence relation given in Equation 3.2 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 \( O(k^2) \), where \( k < n \).

We perform experiments with the approximation algorithm in section 4.2 and show its effectiveness. In Section 4.2 we show the effect of the radius hyperparameter on accuracy and the trade-off between accuracy and precision.

### 3.1.4 Similarity measure

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 of 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.

Notice the assumption for the calculation of the above mentioned likelihood that the two time series being compared must be of 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. For example, trajectories from the animal tracking dataset and hurricane dataset are of different length. We solve this problem by using a window based approach, where window size is the size of the smaller time series between two time series to be compared. For example, if want a similarity measure between time series $T_1$ and $T_2$ such that $\text{size}(T_1) > \text{size}(T_2)$, we compute $L(T_1(1 : \text{size}(T_2)) \mid T_2)$ as the distance measure. We then move the window by 1, to compute $L(T_1(2 : \text{size}(T_2)+1) \mid T_2)$ and keep doing this until we are at end of time series $T_1$. We then return the average of computed similarity for all windows. We take average because it takes into account similarity between all subsequences of $T_1$ and $T_2$. We do not take minimum or maximum, thus not making the assumption that the best or the worst match between the subsequences of $T_1$ and time series $T_2$ is the discriminatory subsequence. We have used this method for comparing the trajectories of unequal length in the animal tracking and hurricane dataset.
Chapter 4

EVALUATION AND EXPERIMENTAL RESULTS

We have proposed a time series discretization algorithm and a distance measure based on it in section 3.1. In this section we evaluate our time series algorithm and the approximation to it. We use classification accuracy on UCR time series datasets (Keogh et al. 2006) to evaluate the discretization mechanism. Classification accuracy has been used previously in (Lin et al. 2007) for evaluating discretization mechanisms. Note that our discretization mechanism can be used for other data mining tasks like clustering and query by content as well.

4.1 Classification of time series data

We evaluate our approach on 10 datasets taken from benchmark data at the UCR time series repository (Keogh et al. 2006). We compare accuracy with previously published performance results of competing classifiers based on Euclidean distance, DTW and SAX (Lin et al. 2007) (Senin & Malinchik 2013). To make comparison fair, all classifiers are 1-Nearest Neighbour classifier and with exactly the same training and testing data. Note that (Senin & Malinchik 2013) gives state of art classifier, but this classifier is not based on a similarity measure making it difficult to adapt directly to other data mining tasks. Also, (Senin & Malinchik 2013) and (Lin, Khade, & Li 2012) use SAX as underlying discretiza-
tion mechanism. These techniques can also be used with our discretization mechanism instead of SAX.

Our algorithm has two hyperparameters, $\lambda$ and $\min\_\text{variance}$. 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 (Bergstra et al. 2011) for optimizing over hyperparameters on training data. We use a tree of parzen estimators (TPE) algorithm in hyperopt with $\lambda$ generated from a uniform distribution with low $= 0.1$ and high $= 50$ and $\min\_\text{variance}$ 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 4.1 shows the accuracy of SAX based distance, Euclidean distance, dynamic time warping and our distance metric. In brackets it also shows binomial confidence interval which is interval estimate of population parameter and it includes parameter of interest if experiment is repeated several number of times. We use the Normal Approximation
method for calculation of confidence intervals. We say that performance of a method 1 is statistically significantly better than method 2, if the accuracy of method 1 is greater than the accuracy of method 2 and their confidence intervals don’t overlap. Accuracy for SAX, EU and DTW are reported from previously published results from (Lin et al. 2007) and (Senin & Malinchik 2013). We were unable to find results for SAX on the ’ECGFiveDays’ dataset, hence we have left it blank.

On average we perform better than SAX and Euclidean distance and are comparable to dynamic time warping. Our accuracy is statistically significantly better than the SAX on two datasets, CBF and Coffee dataset. On all other datasets except synthetic control and Lightning2 dataset, we perform better than the SAX but difference in accuracy is not statistically significant. Similarly, for Euclidean distance we perform statistically significantly better on CBF dataset and statistically significantly worse on ECGFiveDays. On remaining all datasets difference in accuracy is statistically insignificant. In comparison with DTW, on two datasets, ECGFiveDays and Gun point we perform statistically significantly better and on the datasets synthetic control and FaceAll we perform statistically significantly worse. Overall, classification accuracy of our method is atleast as good as SAX, Euclidean distance and DTW.

### 4.2 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 the 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 \( \min_{\text{variance}} \), \( \lambda \) and radius.
FIG. 4.1. Speedup with respect to change in radius
Figure 4.1 shows the speedup achieved by the approximation algorithm on the 'ECG200' dataset from the UCR time series repository with respect to radius. Figure 4.2 shows classification accuracy on the 'ECG200' dataset with respect to radius. The hyper-parameter values in these experiment were $\text{min}\_\text{variance} = 19.1$ and $\lambda = 0.17$. As we can observe towards radius $r = 0.25$, accuracy does not change and stays 92%, but 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.
We randomly picked a time series from the ECG200 dataset and compared the output of the approximation algorithm and the original discretization mechanism. To compare both algorithms we compare partitions generated by both algorithms using adjusted mutual information (AMI) (Vinh, Epps, & Bailey 2009). Figure 4.3 shows the effect of varying...
radius on AMI for different values of $\text{min\_variance}$ with $\lambda$ being same. Figure 4.4 shows effect of radius on $\lambda$ with $\text{min\_variance}$ constant. We can observe from Figure 4.3 that until a certain threshold of radius, output from exact algorithm and approximation are exactly same. But there is significant speedup. For example, for $\text{min\_variance} = 10$, threshold value is 0.5, where we get a speedup of 99%. As the $\text{min\_variance}$ increases threshold value decreases. We see similar behaviour when we vary $\lambda$, but threshold values of radius are much smaller than Figure 4.3. To compute an appropriate value of radius, we recommend computing $\text{min\_variance}$ and penalty using cross validation and choosing radius equal to the threshold value. This will ensure that we get maximum speedup without much difference in results from exact and approximate algorithms.

We show the effectiveness of our approximation algorithm on CBF and Gun\_Point dataset. We select maximum radius that does not change discretization output for the time series and compute 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 have fixed $\lambda = 0.1$ and $\text{min\_variance} = 0.1$. The average speedup for CBF and Gun\_Point dataset was 53.43% and 62.04%. Thus we can get significant speedup without change in discretization output using our approximation algorithm.

### 4.3 Spatial Trajectory Classification

In the previous section we tested our similarity measure on datasets as diverse as shape, medicine, surveillance, and industry from the UCR time series repository. But, all these datasets have time series of equal length. To understand the effectiveness of our similarity measure on datasets with series of unequal length, in this section we classify spatial trajectory data.
The trajectory data is intrinsically complex to explore since patterns of movement are often driven by unperceived goals and constrained by unknown environmental settings. We use two datasets: animal tracking dataset and hurricane dataset for evaluation of our distance measure. Both these datasets have trajectories of unequal length. Figure 4.5 and Figure 4.6 shows the histogram of the length of trajectories in the animal tracking dataset and the hurricane dataset. From Figure 4.5 and Figure 4.6 we can observe that both these datasets have trajectories with length which varies over a large range.
The animal movement data set was generated by the Starkey project (Preisler et al. 2004). We use the animal movements observed in June 1995 as it is used in previous research by (Lee et al. 2008). This data set is divided into three classes by species: elk, deer, and cattle. The numbers of trajectories (points) are 38 (7117), 30 (4333), and 34 (3540), respectively.

The hurricane track data set has the Atlantic hurricanes for the years 1950 through 2006. The Saffir-Simpson scale classifies hurricanes into categories 1-5 by intensity. A high category number indicates a high intensity. Categories 2 and 3 are chosen for two classes. The numbers of trajectories (points) are 61 (2459) and 72 (3126) respectively.

Both these datasets have been used in previous research by (Lee et al. 2008) who give two algorithms, TB-only and RB-TB based on features used for classification of these datasets. We used the same training and test set as used by (Lee et al. 2008) and compare classification accuracy. Note that although we show the accuracy of (Lee et al. 2008) in
Table 4.2, these accuracies are not directly comparable as (Lee et al. 2008) does not give distance metric. This makes method of (Lee et al. 2008) unsuitable for other data mining tasks mentioned in Section 3.1 like clustering and query by content.

To find the distance between two trajectories, the multi-dimensional trajectory data (time, latitude, longitude) has to be transformed into a sequence of scalars. To achieve this, the trajectory points were mapped to the visit order of a Hilbert space filling curve (SFC) embedded in the trajectory manifold space and indexed by the recorded times in the visit order. The Hilbert SFC-transformed trajectory produces a time series, which can be used for classification using our algorithm. This adds another hyperparameter called the Hilbert curve order which decides the granularity of the space filling curve.

We classify trajectories using a 1-nearest neighbour classifier. For choosing parameters for classification, we use 40% of the training set as a validation set. We decide the parameters using validation data and report the accuracy with optimized parameters on test data. We use the python hyperparameter optimization library hyperopt (Bergstra et al. 2011) for optimizing over hyperparameters on the training data. We use a tree of parzen estimators (TPE) algorithm in hyperopt with $\lambda$ generated from uniform distribution with $\text{low} = 0.1$ and $\text{high} = 50$, $\text{min\_variance}$ generated from a lognormal distribution with $\mu = -0.9$ and $\sigma = 1.39$ and $\text{order}$ can take any value between 2 and 10 . The number of maximum iterations for optimization of hyperparameters is 40. In case of a tie, we randomly pick one of the hyperparameter values.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Classes</th>
<th>Training Set</th>
<th>Testing Set</th>
<th>TB-Only</th>
<th>RB-TB</th>
<th>NDist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Animal Tracking Data</td>
<td>3</td>
<td>80</td>
<td>18</td>
<td>50 (26.9,73.1)</td>
<td>83.3 (66.07,1)</td>
<td>83.3 (66.07,1)</td>
</tr>
<tr>
<td>Hurricane Dataset</td>
<td>2</td>
<td>112</td>
<td>21</td>
<td>65.4 (45.05,85.75)</td>
<td>73.1 (54.13,92.07)</td>
<td>52.3 (30.94,73.66)</td>
</tr>
</tbody>
</table>

Table 4.2. Classification Accuracy comparison for TB-Only and RB-TB methods and our algorithm
Table 4.2 shows a comparison of the accuracy of the TB-only and RB-TB methods from (Lee et al. 2008) and our distance metric. We perform better than the TB-Only method and as good as the RB-TB method on the animal tracking dataset. But, both methods in (Lee et al. 2008) outperforms our method on the hurricane dataset. Although the accuracy is less for the hurricane dataset, the method by (Lee et al. 2008) cannot be used for other data mining tasks like clustering and query by content as they don’t give a distance measure.

![Image of features learnt for the animal tracking dataset for the RB-TB method]

**Fig. 4.7.** Features learnt for the animal tracking dataset for the RB-TB method
Figure 4.7 and Figure 4.8 show features learned by (Lee et al. 2008) for the animal tracking and hurricane datasets respectively. These methods use two types of features: region based and direction based shown by colored boxes and colored lines respectively. In the animal tracking dataset from Figure 4.7, both kinds of features are useful for classification. But for the hurricane dataset, direction based features are more useful than region based features. This can be seen in Figure 4.8 as number of trajectories classified using the region based feature are few. Direction based features are difficult to capture using our distance measure as we are calculating the likelihood of each point in the trajectory data. Thus the RB-TB and TB-only methods outperform our method on the hurricane dataset.
FIG. 4.9. Effect of hilbert curve order parameter on accuracy on the animal tracking dataset
Figure 4.10. Effect of hilbert curve order parameter on accuracy on the hurricane tracking dataset

Figure 4.9 and Figure 4.10 show the effect of Hilbert curve order on classification accuracy for the animal tracking and hurricane datasets respectively. From both figures we can notice that accuracy is low for very high as well as for very low Hilbert curve values. For very low Hilbert curve orders, even the points far apart in two dimensional space get the same mapping in the Hilbert space filling curve. And for high hilbert curve order, points closer in two dimensional space can be mapped to a value in the Hilbert space filling curve which is far apart. In both these cases the actual distance between two points is not accurately approximated by the Hilbert space filling curve, thus affecting classification accuracy negatively. Thus it is an important parameter and we determine it using hyperparameter optimization.
Chapter 5

GRAMMAR INDUCTION OVER TIME SERIES DATA

5.1 Motivation

In the previous chapter we gave a discretization mechanism and distance measure based on it for time series data. Although this distance measure is effective on datasets in the UCR time series repository and trajectory datasets, it is not rotation invariant and might not perform well for shifted time series data. Having a distance measure that is robust to shifted time series is an attractive property as it is unrealistic to assume that all time series to be compared will be aligned. In this chapter we explore the usage of grammar induction technique for comparison of shifted time series.

Consider two time series shown in Figure 5.1. Both time series are the same, except that one is a shifted version of the other. On discretization of both time series using the method given in section 3.1.2, time series 1 ($T_1$) and times series 2 ($T_2$) are discretized as shown below.

$T_1 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$

$T_2 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$

34
Here index 0 refers to normal distribution \((N_0)\) with \(\mu = -0.9\) and \(\sigma = 0.48\) and index 1 refers to normal distribution \((N_1)\) with \(\mu = 1.03\) and \(\sigma = 0.48\). Notice that normal distribution inferred are same as both time series and discretization parameters are same, just order in which they occur is different.

![Time Series 1](image1)

![Time Series 2](image2)

**FIG. 5.1.** Two identical but shifted time series

If we compare time series \(T_1\) and \(T_2\) using the similarity measure given in Section 3.1.4, we would get a very low similarity score even though one time series is a shifted version of the other. This is because we calculate \(L(T_2 \mid T_1) = \prod_{i=0}^{n} L(t_{2i} \mid N_i)\), 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 \(T_1\). In this case the similarity measure in Section 3.1.4 will return \(\prod_{i=0}^{n/2} L(t_{2i} \mid N_0) \times \prod_{i=n/2}^{n} L(t_{2i} \mid N_1)\). As the time series is shifted, both terms above will have low values and so will the similarity score. To solve this problem, we divide time series \(T_1\) into subsequences and compare each subsequence with every subsequence in time series \(T_2\). Ideally, in the above scenario the first half of time series \(T_1\) is compared with the second half of time series \(T_2\) and the second half of time series \(T_1\) is compared with the first half of time series \(T_2\). We use a grammar induction algorithm called Sequitur
(Nevill-Manning & Witten 1997a) for division of time series into subsequences and a subroutine of grammar induction algorithm IRRMGP (Carrascosa et al. 2010) to calculate a similarity measure. We give details of both these algorithms in the next section.

The use of grammar induction techniques for division of time series data is inspired by recent success of grammar induction for motif discovery (Oates et al. 2013) and anomaly detection (Senin et al.) on time series data. Both these problems use grammar induction for dividing time series into subsequences.

Apart from the advantage of being robust to shifts in time series data, the use of grammar induction is also theoretically motivated. Firstly, context free grammars are strictly more expressive than finite state machines. This helps recover context free structure from time series data and thus helps understanding long distance dependencies and hierarchical structure. Secondly, the computation of distance is based on a notion of Kolmogorov Complexity, size of the smallest Turing machine that can describe a single string. Kolmogorov Complexity is an uncomputable measure due to the non-computability of Halting Problem. Our distance measure can be considered as an approximation of conditional Kolmogorov Complexity.

(Li et al. 2004) proposes a practical application of Kolmogorov complexity and define a universal similarity metric between two sequences as

\[ d(x, y) = \frac{\max\{K(x|y), K(y|x)\}}{\max\{K(x), K(y)\}} \]

As K is uncomputable, any compression algorithm can be used in its place as an approximation. Since grammar induction has been used effectively as compressor (Nevill-Manning & Witten 1997b), we can use it for approximating Kolmogorov complexity. Here for computation of conditional Kolmogorov complexity for comparison of two time series \( T_1 \) and \( T_2 \), we first compress \( T_1 \) using grammar induction and then try to find the smallest parse
tree for \( T_2 \) based on the grammar learned from \( T_1 \).

To summarize, in this chapter we introduce a distance measure to compare time series data based on grammar induction algorithms and we evaluate our distance measure on trajectory datasets.

5.2 Background and Related work

Our algorithm for time series comparison is based on two existing algorithms called Sequitur (Nevill-Manning & Witten 1997a) and minimal grammar parsing (MGP) (Gallé 2011). In this section we introduce both these algorithms.

5.2.1 Sequitur Algorithm

(Nevill-Manning & Witten 1997a) proposes a linear time grammar induction algorithm called Sequitur for finding hierarchical patterns in textual or symbolic data. Sequitur is a greedy algorithm which discovers structure in data by identifying repetitions of symbols and then storing them in the form of a context-free grammar. Sequitur generates grammar rules based on two properties:

1. Bigram uniqueness: No pair of adjacent symbols appears more than once in the grammar. While processing symbols, if we encounter a duplicate bigram, then we either create a new rule or substitute the bigram with a non-terminal if the rule for that bigram exists already.

2. Rule utility: Every rule is used more than once. This constraint helps in creating longer rules of size greater than 2.
Algorithm 2: Sequitur Algorithm

while new input symbol do
    append it to rule S
while Two symbols are linked do
    if bigram is repeated and the repetitions do not overlap then
        if other occurrence is a complete rule then
            replace new bigram with the non-terminal symbol that heads the rule
        else
            form a new rule and replace both bigrams with the new non-terminal symbol
    else
        insert the bigram into the index

while bigram is replaced by a non terminal do
    if either symbol is a non terminal symbol that only occurs once elsewhere then
        remove the rule, substituting its contents in place of the other non-terminal symbol

Algorithm 2 gives the Sequitur algorithm for inferring context free grammar from data as given by (Nevill-Manning & Witten 1997a). We use the Sequitur algorithm due to its simplicity and efficient linear time computational cost. Table 5.1 shows rules generated by Sequitur algorithm for input string $S = "aaaabaaaa"$. It also shows expanded grammar rule associated with each non terminal. These expanded grammar rules are also called constituent associated with the respective non-terminal. As we can notice in Table 5.1, Sequitur effectively compresses an input string and represents it as grammar rules.
### Grammar Rule

Table 5.1. Grammar rules generated by sequitur for string ”a a a b b a a a a”

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>Expanded Grammar Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1 ( \rightarrow ) R3 b b R3</td>
<td>a a a a b b a a a a</td>
</tr>
<tr>
<td>R2 ( \rightarrow ) a a</td>
<td>a a</td>
</tr>
<tr>
<td>R3 ( \rightarrow ) R2 R2</td>
<td>a a a a</td>
</tr>
</tbody>
</table>

#### 5.2.2 Minimal grammar parsing (MGP)

(Gallé 2011) introduces the problem of minimal grammar parsing (MGP) for straight line grammars, i.e., grammar which can parse only one string. The problem can be stated as follows:

*Given a string \( S \), set of non-terminals \( N = \{N_0, N_1, \ldots N_m\} \) and corresponding expansions/constituents for each non-terminal \( w = \{w_0, w_1, \ldots w_m\} \), the goal of the MGP algorithm is to find a grammar of minimal size using only non-terminals in \( N \) that can parse string \( S \).*

Minimal grammar parsing (MGP) can be solved in polynomial time. To solve MGP, first a path graph \( G \) is constructed with every element in string \( S \) as edge label in path graph. Thus a path graph \( G \) has \( n + 1 \) vertices, where \( n \) is the number of tokens in string \( S \). Then we iterate through each constituent and search for constituents in graph \( G \) by matching each edge label with each token in the constituents. This can be thought of as searching all sub paths in path graph \( G \). When a constituent is matched, an edge is created from the node where the match started to the node where the match ends. Notice that there can be several overlapping edges because of the same or different constituents. We repeat this process with each constituent as a string. The shortest path across this graph corresponds to the smallest parse tree using only non terminals given by \( N \).

For example consider string \( S = ababbababbabaababaa \) and suppose the constituents
are $\Omega = \{S, abbaba, bab\}$, then the minimal grammar parsing is $N_0 \rightarrow aN_2N_2N_1N_1a$, and $N_1 \rightarrow abN_2a$, $N_2 \rightarrow bab$. Figure 5.2 shows computation of a minimal grammar parsing of string $S$ with constituents $\Omega = \{S, abbaba, bab\}$.

![Diagram of minimal grammar parsing](image)

**Fig. 5.2.** Minimal grammar parsing (MGP)

### 5.3 Grammar Induction over time series data

In this section, we describe our similarity measure based on grammar induction for time series data. This measure will be based on comparison of subsequences instead of complete time series and hence will be more robust to shift in time series data. This similarity measure is based on a notion of conditional Kolmogorov complexity as described in Section 5.1. Intuitively, we compute the extent to which we can compress a time series given that we know grammar from which another time series is generated and use it as
similarity measure.

Let \( T_1 = \{ t_1, t_2, \ldots, t_m \} \) and \( T_2 = \{ t_1, t_2, \ldots, t_n \} \) be two time series to be compared, where \( m \) and \( n \) are the length of time series \( T_1 \) and \( T_2 \) respectively. We want to compute the likelihood of \( T_2 \) given that we know the grammar generating time series \( T_1 \). As grammar induction algorithms only work on discretized data, we first discretize time series \( T_1 \) into set of normal distribution using the technique described in Chapter 3. On discretization we get normal distributions \( N = \{ N_1, N_2, \ldots N_p \} \) generating time series \( T_1 \) and an array of indexes \( I = \{ I_1, I_2, \ldots I_m \} \) where every index \( I_i \) refers to normal distribution in \( N \) that generated corresponding point in \( T_1 \). We run the grammar induction algorithm Sequitur described in Section 5.2.1 on the array of indexes generated by the discretization algorithm. This gives us grammar \( G \) where every rule in the grammar corresponds to a repeated subsequence in time series \( T_1 \). Here we choose Sequitur as the grammar induction algorithm because it is online and efficient. The subsequences related to rules learned by Sequitur correspond to recurrent patterns in time series data.

On computation of grammar \( G \) corresponding to time series \( T_1 \), we try to compress time series \( T_2 \) with respect to \( G \) using a variation of minimal grammar parsing (MGP). First, we create a path graph of length of time series \( T_2 \) such that edge labels are values in time series \( T_2 \) just like in the MGP. We use all subsequences generated by sequitur from \( T_1 \) as constituents in MGP algorithm. If \( I_p..I_q \) is a subsequence of indexes corresponding to a rule in \( G \). We say that symbol \( I_p \) is matched to edge label \( v \) if \( L(v \mid N_r) > \text{threshold} \), where \( N_r \) is the normal distribution pointed by \( I_p \) and \( \text{threshold} \) is a user defined parameter. If all indexes \( I_p..I_q \) are matched to all adjacent edge labels in the path graph, then there is a subsequence match. Subsequence matches correspond to matches of constituents in MGP.

In case of a match, an edge is added to the path graph from the node where the match started to the node where the match ends. We then find the shortest path from the start node to the end node and return the size of the shortest path as a distance measure. If the number of
rules matched is high, then both the time series are similar and returned size of the shortest path is small. If none of the subsequences are matched, the returned size is equal to the size of time series $T_2$.

### 5.4 Results and Discussion

We evaluate our grammar induction based distance measure on shifted time series and the trajectory dataset. The goal of experiments on shifted time series is to understand whether a grammar induction based distance measure is able to distinguish between shifted versions of time series and different time series. For this experiment, we take 3 time series from the CBF dataset, each belonging to cylinder, bell and funnel classes and use them as the training set. We create 5 time series for each time series in the training set by circularly shifting it from random time point. For example, if the train set time series is $T = [1, 2, 3, 4, 5]$ and random integer 2 is generated then the shifted time series would be $T = [4, 5, 1, 2, 3]$. By this process we get 15 time series in the test set. We classify this shifted time series dataset using grammar based distance measure with the same parameters as used in experiments with the CBF dataset in Table 4.1 and randomly picking a threshold parameter between 0.1 and 0.4. Not surprisingly, we get 100% accuracy every time on this dataset. As opposed to Euclidean distance and distance measure proposed in Chapter 3 which gave accuracy of 20% and 26.6% respectively. (The parameters used in the grammar induction based measure and distance measure in Chapter 3 were same) This shows the grammar induction based distance measure’s ability to distinguish between shifted versions of the same time series and different time series.

Motivated by results on shifted time series dataset we evaluated our grammar induction based distance measure on trajectory datasets. The experimental setup used was identical to one used in the previous section in Table 4.2. We evaluate distance measure on two
datasets: Animal tracking and Hurricane datasets. We decide parameter values using the hyperparameter optimization library hyperopt. The only difference is that there is an extra parameter for threshold in the grammar induction based method. We decide value of this parameter using hyperopt, and it is generated from uniform distribution with low = 0.2 and high = 0.4.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Classes</th>
<th>Training Set</th>
<th>Testing Set</th>
<th>TB-Only</th>
<th>RB-TB</th>
<th>NDist</th>
<th>rSequitur</th>
</tr>
</thead>
<tbody>
<tr>
<td>Animal Tracking Data</td>
<td>3</td>
<td>80</td>
<td>18</td>
<td>50</td>
<td>83.3 (66.07,1)</td>
<td>83.3 (66.07,1)</td>
<td>55.55 (32.59,78.51)</td>
</tr>
<tr>
<td>Hurricane Dataset</td>
<td>2</td>
<td>112</td>
<td>21</td>
<td>65.4 (45.05,85.75)</td>
<td>73.1 (54.13,92.07)</td>
<td>52.3 (30.94,73.66)</td>
<td>52.38 (30.94,73.66)</td>
</tr>
</tbody>
</table>

Table 5.4 shows comparison of accuracy of TB-only, RB-TB methods from (Lee et al. 2008), distance measure based on normal distributions and grammar induction based distance measure. Although grammar induction based distance measure is able to recognize shifted time series, it did not help in classification of trajectory datasets.

Surprisingly, classification accuracy of the distance measure given in Section 3.1.4 is better than the accuracy using grammar induction based distance measure. This might be because the distance measure described in Section 5.3 does not compute differences in uncompressed part of time series. For example, consider 3 time series $T_1 = [1, 1.1, 0.9, 2, 2.1, 1.9]$, $T_2 = [1, 1.1, 0.9, 3, 3.1, 2.9]$ and $T_3 = [1, 1.1, 0.9, 10, 10.1, 10.9]$. If we calculate the $L(T_2 \mid T_1)$ and $L(T_3 \mid T_1)$, clearly $L(T_2 \mid T_1) > L(T_3 \mid T_1)$. Thus using the distance measure given in Section 3.1.4, we will conclude that $T_2$ and $T_1$ are more similar to each other than $T_3$ and $T_1$. But, if we use distance measure given in Section 5.3, we will learn Sequitur grammar on the discretized version of $T_1$ and then try to parse time series $T_2$ and $T_3$. In both cases we are just able to parse first 3 points in time series $T_2$ and $T_3$. Thus the size of smallest parse tree is 4 for both comparison between $T_2$ against $T_1$ and $T_3$ against $T_1$. As we are not comparing uncompressed part of time series, we come to false conclusion that $T_3$ and $T_2$ are equally similar to $T_1$. This property of the grammar induction
based distance measure might be the reason of poor classification accuracy in Table 5.4.
Chapter 6

CONCLUSION AND FUTURE WORK

This thesis provides a novel discretization mechanism for time series data based on the more general assumption that it is generated from multiple normal distributions. An effective similarity measure based on the discretization mechanism is proposed and evaluated by classifying data from the UCR time series repository and comparing it with the state of the art methods. The similarity measure was evaluated on trajectory datasets to show its effectiveness in diverse domains. To improve efficiency we have also provided an approximate algorithm that significantly improves the discretization time while retaining its effectiveness. We have also demonstrated a grammar induction technique that discriminates shifted time series against completely different time series.

In future, we would like to evaluate the grammar induction based distance measure on UCR time series datasets. We would also like to evaluate our distance measures on other tasks like clustering, query by content and motif discovery.

The problem with current grammar based distance measure is that it does not compare uncompressed subsequences of times series. This negatively impacts classification accuracy as shown in Section 5.4. In future we would like to improve the grammar based distance measure by comparing uncompressed subsequences of time series data. This might improve classification accuracy on the animal tracking and hurricane datasets.
REFERENCES


