nTreeClus: A MODEL-BASED PARTITIONING CLUSTERING OF SEQUENTIAL DATA

by

Hadi Jahanshahi

B.S., Industrial Engineering, Iran University of Science and Technology, 2011

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Industrial Engineering

Boğaziçi University

2018

ACKNOWLEDGEMENTS

First and foremost, I would first like to thank my thesis supervisor Dr. Mustafa Gökçe Baydoğan at Boğaziçi University. Without his help and dedicated involvement in all steps throughout the process, the thesis would have never been accomplished. The door of his office was always open and he answered all my questions truthfully and immediately. Thank you for your support, friendly advice and constructive criticism during the past year. Besides my advisor, I would like to thank my thesis committee: Prof. Dr. Necati Aras and Assist. Prof. Dr. Mehmet Gönen for their insightful comments.

Last but not least, I must express my very profound gratitude to my spouse for providing me with continuous encouragement and unfailing support throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without her.

ABSTRACT

nTreeClus: A MODEL-BASED PARTITIONING CLUSTERING OF SEQUENTIAL DATA

The overwhelming presence of categorical/sequential data in diverse domains emphasizes the importance of sequence mining. The challenging nature of sequences proves the need for a continuing research to find a more accurate and faster approach providing a better understanding of their (dis)similarities. The thesis proposes a new model-based approach for clustering sequence data, namely nTreeClus. Proposed approach is at the intersection of the conceptions behind existing approaches, including Decision Tree Learning, k-mers, and autoregressive models for categorical time series, culminating with a novel numerical representation of the categorical sequences. This new representation is used to perform clustering considering the inherent patterns in categorical time series. Furthermore, the only parameter of the method, the window size (n), is examined and the robustness of the method to its parameter has been shown. Under different simulated scenarios and using various internal and external cluster validation indices, the performance of the method is analyzed and reported. Finally, empirical evaluation using synthetic and real datasets, protein sequences and categorical time series, declares that nTreeClus is competitive or superior to most of the state-of-the-art methods.

ÖZET

nTreeClus: SIRALI VERİNİN MODEL TABANLI KÜMELENMESİ

Farklı alanlarda gözlemlenen kategorik/ardışık dataların sıklığı dizi madenciliğini önemini ortaya koymaktadır. Dizilerin çetrefilli yapıları nedeniyle kendi aralarındaki benzerliklerinin veya farklılıklarının daha iyi anlaşılabilmesi konusunda daha kesin sonuç veren ve daha hızlı çözüme ulaşan yaklaşımları bulmak için sürekli bir arayış içinde olmak gerekmektedir. Yapılan bu çalışmada ardışık verilerin kümelenmesi için nTreeClus ismi verilen model bazlı yeni bir yaklaşım önerilmiştir. Önerilen yaklaşım, kategorik dizilerin yeni sayısal temsili ile sonuçlanan, aralarında Karar Ağacı Öğrenmesi, k-mers ve kategorik zaman serileri için kullanılan otoregresif modellerin de bulunduğu mevcut çözümlerin arkasındaki kavramların kesişimi konumundadır. Bu yeni temsil kategorik zaman serilerindeki mevcut modelleri dikkate alarak kümeleme işlemini yapmakta kullanılmaktadır. Ayrıca, yöntemin tek parametresi olan pencere boyutu (n) incelenmekte ve yöntemin parametresine olan sağlamlığı gösterilmektedir. Farklı simülasyon senaryoları altında ve çeşitli iç ve dış kümeleme doğrulama indeksleri kullanılarak yöntemin performansı analiz edilmekte ve raporlanmaktadır. Son olarak, sentetik ve gerçek veri setleri, protein dizileri ve kategorik zaman serileri kullanılarak yapılan ampirik değerlendirme göstermektedir ki, nTreeClus yöntemi günümüzde kullanılan metotlar içinde rekabetçi bir konumda hatta en gelişmiş yöntemlerin çoğundan daha üstün bir seviyede yer almaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

a	The number of categorical values chosen from set $\mathcal{A}$
$\mathcal{A}$	Set of categorical values
C	The number of clusters
$\mathcal{L}$	The number of sequences in each batch
$\mathcal{M}_p$	The length of pattern
$\mathcal{M}$	The length of sequences
$\mathcal{N}$	The number of batches

LIST OF ACRONYMS/ABBREVIATIONS

1NN	One Nearest Neighbor
ASW	Average Silhouette Width
CH	Calinski-Harabasz
DI	Dunn Index
DT	Decision Tree
JW	Jaro-Winkler
LD	Levenshtein Distance
MHMM	Mixture of Hidden Markov Models
RF	Random Forest

1. INTRODUCTION

Sequences are among the most common types of data found in diverse domains such as bioinformatics, marketing, social science, text mining, web mining, security, health-care, or business. For example, short repeated DNA sequences are one of the most interesting features of prokaryotic and eukaryotic genomes [1] since they both encode the structure and mechanism and are associated with neurological disorders. Learning the structure of these repeated patterns also is applicable to the studies of genetic variation, gene tagging and linkage mapping [2]. Moreover, sequences can also capture some behavioral characteristics of human through their purchase history in a shop or their click-stream sequences in a website [3–5]. Analyzing the past transaction logs or historical data of an organization plays a crucial role for decision makers [6]. Assessing process flows in a hospital (as a sequence of past events) [7] or evaluating the past transaction data of the orders a company received [8] are some other typical examples of the ubiquity of sequences to be mined.

Numerous studies have been carried out in the field of sequence data mining, although clustering categorical sequences gained the least attention in the literature noting that clustering sequences is a major challenge to be addressed in the aforementioned domains. The task demands either accurately computing (dis)similarity between sequences or finding the underlying model generating sequences. Therefore, from the problem-solving approach, we categorize them into two folds: *Proximity-Based approaches*, including *Similarity-Based* and *Feature-Based*; and *Model-Based approaches* [9]. The main aim of Proximity-Based methods is to devise a similarity or distance measure between sequences, applied either on raw-data in *Similarity-Based* or on feature-extracted data in *Feature-Based*. On the other hand, in *Model-Based* methods, a new representation of the given data has been introduced through a model which best fits the data. Hence, to cluster categorical sequences, we need to define

a (dis)similarity measure or latent model appropriately depicting the behavior of sequences.

There has been a significant change in the approaches towards solving these types of problems during past decade. Previously, Similarity-Based approaches, especially alignment-based ones, was the most popular method trying to align sequences and find the best or longest common pattern which exists in a sequence. This attitude has been more prevalent in bioinformatics than other domains. The leading paper in this field tried to find a general method for sequence comparison [10]. Afterwards, many researches have been conducted in the same field which can be categorized as global, local, or glocal alignments based on dynamic programming [10–14], pairwise alignment which can be used for only two sequences at a time [15–17], multi-sequence alignment which is the extended version of pairwise alignment [18–23], and structural alignment which is applicable to the sequences whose structures are known [24, 25]. Accordingly, most of these methods are used to handle bioinformatics sequences and in most cases cannot be generalized to other domains. Additionally, most of the more accurate ones have many limiting factors; namely, number of sequences to be handled, computation time and memory, the position of the pattern in a sequence, and the length of sequences which should not be varying significantly.

Similarity-Based approaches considering distance among sequences are apt to disclose the (dis)similarity between given sequences. Most of the methods in this category use cost of dissimilarity to count mismatches; nevertheless, the definition of mismatch in each method differs. Hamming distance counts the number of elements to be substituted in order to change one string to another. Furthermore, in this method, both strings should have the same length. The Levenshtein distance, or edit distance, is the number of insertion, deletion, and replacement which is required to turn one string into another [26]. The drawback of this distance method is widely discussed; it is not a good choice as far as computational complexity is concerned, the median

string problem is not tractable in edit distance method [27], and finally, it fails in some important cases such as automated language classification tasks [28].

k-mer (n-gram) is another method which can be classified as a Feature-Based approach. The main superiority of n-grams over alignment methods is universality of them which can be generalized to many domains, including word prediction (speech recognition) [29, 30], text categorization and classification [31], detection of new malicious code [32], similarity and distance of strings [33], authorship identification [34], protein and genome sequences [35], file and data type classification [36], and information retrieval [37]. Nevertheless, there are some arguments about its performance such as finding the optimal k for k-mer [7]. Position Weight Matrix (PWM) is a method to find a pattern, especially a motif, in a sequence [38]. The method uses a matrix representation of the sequence based on the appearance of each element in a specific window. In order to have a better understanding of the sequence, this method needs pre-alignment for the sequences with different length to find the optimal window. Hence, again the aforementioned problem of finding the best window arises.

The last one is the Model-Based approach in which a model is trained on the sequence to find the logic of how the letters are ordered. Markovian models are more general methods which are founded on the assumption that sequences have parametric distribution. The method is heavily dependent on the conditional probability of occurrence of each element in the sequence while satisfying the so-called "Markov Property". There are many types of Markovian Models, the most famous ones of which include first-order Markov Chain, Higher order Markov Chain, Interpolated Markov Motif Models (IMM), Multivariate Markov Chain Model, and Hidden Markov Model (HMM). Unlike PWM that has no memory, Markov Chain Model considers previous elements of a sequence by introducing states [3]. Despite being utilized in diverse domains [39–42], Hidden Markov Models are hard to be tuned and in most cases are computationally intensive, thus, affecting its generality and scalability [7]. Autoregres-

sive models for categorical sequences (AR models) have been introduced in the past decade. These methods, which can be categorized as Model-Based approaches, investigate the correlative structure of sequences. Discrete Autoregressive Model (DAR) [43] and One-channel Autoregressive Model (AR) [44] were presented to cluster DNA sequences. Further researches on Autoregressive Models for sequential data are mostly applicable to bioinformatics and lack generality. They also require setting model order and window length. Finding the optimal parameters takes time and may lead to higher complexity of the model [45]. AR models also require numerical mapping while categorical data might not have a logical order and mapping them may lead to an irrational deduction.

Here we provide a new Model-Based clustering method, which is called nTreeClus, to capture the sequence features while ignoring the aforementioned restrictions. Moreover, the method offers a novel representation of categorical sequential data which can be used as a feed for (dis)similarity measures to cluster the sequential datasets. nTreeClus combines the notions behind k-mers, autoregressive models, decision trees, and similarity measures to generate a new comprehensive method to enrich the understanding of the sequences. nTreeClus tries to segment a given sequence to shorter sequences with the length of n while tracking down the relation among all elements of the sequence using autoregressive DT. The new numeric representation is the upshot of counting the number of repetition of each segmented subsequence in terminal nodes (leaf nodes) of Decision Trees. Based on the nTreeClus representation of the data, available (dis)similarity methods, such as Cosine and Manhattan, can be used. The trials reveal the robustness and accuracy of the method.

The remainder of the thesis is organized as follows. Section 2 summarizes a brief background. In section 4, the preliminaries are introduced and nTreeClus as a method to cluster sequential data is presented, including the novel approach for representing categorical data and similarity measurements. Furthermore, computational complexity

of the method briefly has been discussed. In section 5, the sensitivity of the method to the parameter n and the ability of the method to predict the number of clusters have been discussed. Moreover, results on the performance of the approach for different scenarios have been reported and the clustering ability of nTreeClus on the real data has been examined. Section 6 provides conclusions.

2. BACKGROUND

nTreeClus is a synthesis of the ideas from Decision Tree learners, k-mers, and autoregressive models for categorical time series. In this section, a brief elucidation of the concept behind these methods and models, in addition to their virtues and shortcomings, has been discussed. Thereafter, we introduce other approaches for sequence mining with which we are going to compare nTreeClus. Based on the definition, we selected a more common method from each of three available approaches, including Jaro–Winkler and Levenshtein from Similarity-Based approaches, k-mers from Feature-Based approaches, and finally Mixture of Hidden Markov Models from Model-Based approaches. Those methods have been introduced in the next part.

2.1. k-mer (n-gram)

k-mers, a more specific version of n-grams, are all the possible substrings of fixed length k which are contained in the original string (read). In a given string of length $\mathcal{M}$, there are $\mathcal{M} - k + 1$ many possible substrings to be extracted. On the other hand, if a number of alphabets be available in a string ($a = 4$ for DNA, namely ACTG), then a^k number of k-spectrum can be generated. A simple example of k-mers for $k = 3$ has been shown in Figure 2.1 where the number of 3-mers is equal to $10 - 3 + 1 = 8$ and the number of 3-spectrums is $3^4 = 81$.

Read : $AGATCGAGTG$
3-mers : $AGA\ -GAT\ -ATC\ -TCG\ -CGA\ -GAG\ -AGT\ -GTG$
3-spectrum : $AAA\ -AAG\ -AAT\ -AAC\ -...\ -CCA\ -CCG\ -CCT$

Figure 2.1: An example of k-mer and k-spectrum for $k = 3$

Some methods perform sequence comparison using the frequency of all combinations of k-spectrums [46] while others directly use vectors of k-mer frequencies [47]. The more advanced version of k-mers (n-grams) is a k-gapped pair (or gapped residue couple) where a gap of fixed size k exists between the ordered pair (i, j) [48]. Here, its k-gapped occurrence frequency denoted by $F^k_{i,j}$ is defined by

$$F^k_{i,j} = \frac{1}{\mathcal{M} - k - 1} \sum_{m=1}^{\mathcal{M}-k-1} O_{i,j}\big(m, m + k + 1\big) \tag{2.1}$$

where $\mathcal{M}$ is the length of sequence X, $k \leq \mathcal{M} - 2$, $O_{i,j}\big(m, m+k+1\big) = 1$ if $x^m = i$ and $x^{m+k+1} = j$, and $O_{i,j}\big(m, m + k + 1\big) = 0$ otherwise. In the case $k = 0$, the k-gapped pair is called bigram (2-mer). For the purposes of illustration, in DNA sequence of Figure 2.2, $F^0_{G,A} = \frac{4}{29}$, $F^1_{G,A} = \frac{3}{28}$, and $F^2_{G,A} = \frac{3}{27}$.

$$GAATTCTCTGTAACCTCAGAGGTAGATAGA$$

Figure 2.2: A DNA sequence

Since the logic behind window size (n in n-gram and k in k-mer) is closely comparable to n in nTreeClus, we have used this method as one of the competitive approaches.

2.2. Decision Tree learners

Decision Tree learners are interpretable models with satisfactory accuracy used in a vast variety of domains. Tree structure classifier, or, more accurately, binary tree structured classifier, is a decision support tool which has a tree-like structure based on repeated splits of a subset into two descendant subsets, where a decision should be made in each decision (nonterminal) node and it should be culminated with either another decision node or a terminal node [49]. The general rule for the decisions inside

of the DT learners indicates that the data in each of the descendant subsets should be "purer" than that of the parent subset. In order to stop the growth of the tree, a heuristic rule is implemented in which if no significant reduction in the impurity is possible, then splitting will be terminated and a terminal node will be generated. In other words, DT can be simplified to a set of rules which are represented by the conditions along the path form a root to a leaf [50]. The rules have the form as

$$\textbf{if } \textit{condition } 1 \textbf{ and } \textit{condition } 2 \textbf{ and } \textit{so on } \textbf{then } \textit{class label } x. \tag{2.2}$$

Such a path from the first state (root) to the terminal node (leaf) includes few or many rules dependent on the length of the path (depth of the tree). Finally, each leaf node represents a class label, a decision which is taken after taking the path. The structure of Decision Tree has been depicted in Figure 2.3. Nevertheless, Univariate Decision Trees, such as CART [51] and C4.5 [52], regard only one attribute (feature) at each decision node leading to axis-aligned splits.

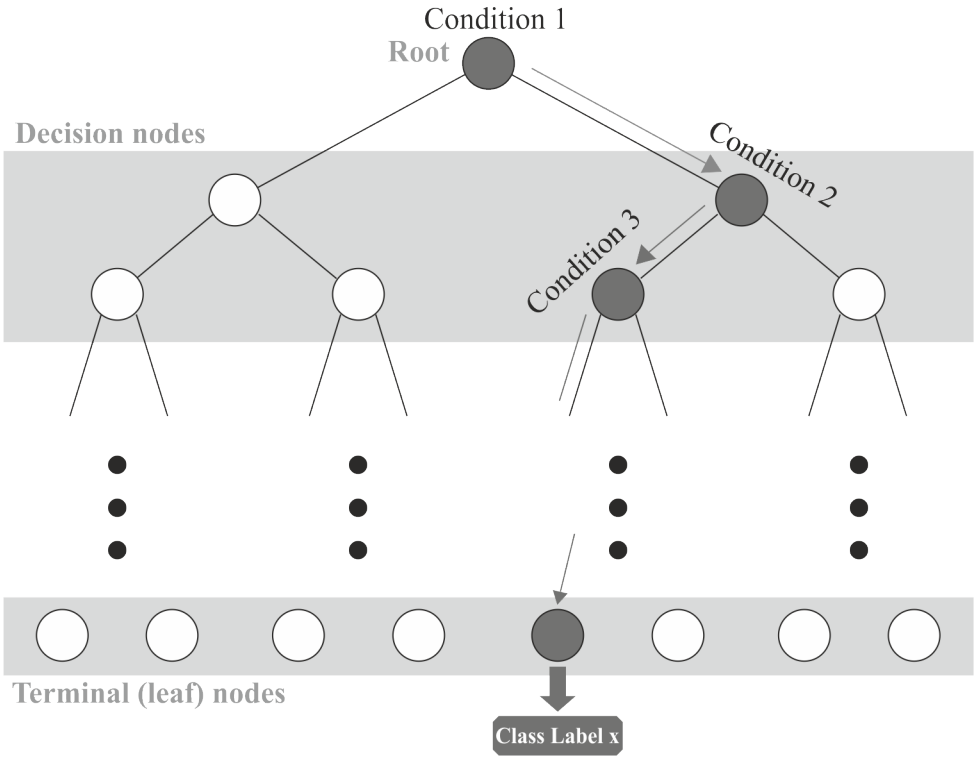

Figure 2.3: The structure of a typical Decision Tree

Unlike the greedy nature of univariate trees, tree ensembles, such as Random Forest (RF), are less likely to overfit on data since they are simply a collection of DTs whose results are less biased and less likely to overfit due to the random subset of features they pick. RF classifier is an ensemble of k decision trees, $h(x, \Theta_k)$, where the Θ_k is a random vector which is independent of past vectors but with the same distribution, each casting a unit vote for the most popular class at input x [53]. For choosing the number of features to be used for tree k, often $\sqrt{\nu}$ is considered where ν is the number of total available features. The random selection of features leads to the reduction of variance of the classifier and decreases computational complexity of DT from $O(\nu\mathcal{L}\log\mathcal{L})$ to $O(\sqrt{\nu}\mathcal{L}\log\mathcal{L})$ where $\mathcal{L}$ is the number of training instances (in-bag). Random Forests are simple to interpret and implement, robust to outliers, strong in handling non-linearity, fast in real cases, and friendly in parallel training to save more time, although there are ways to improve its suboptimal issues and greed nature of each subtree [54, 55].

Figure 2.4 clearly depicts the way in which both methods behave. DT finds the best class label using a single tree, thus increasing the chance of overfitting while RF exploits an ensemble of smaller decision trees from random subset of the data. Each of the trees returns a biased classifier since it has only considered a subset of the data. However, at the end, the majority voting delineates which class has the highest votes from the trees. This modification of original univariate trees leads to less biased and more accurate output. As the Figure 2.4 indicates, decision trees are a method for classifying subjects into known groups. It is also capable of doing regression but in our case, we have utilized DTs because of their Classification Capabilities.

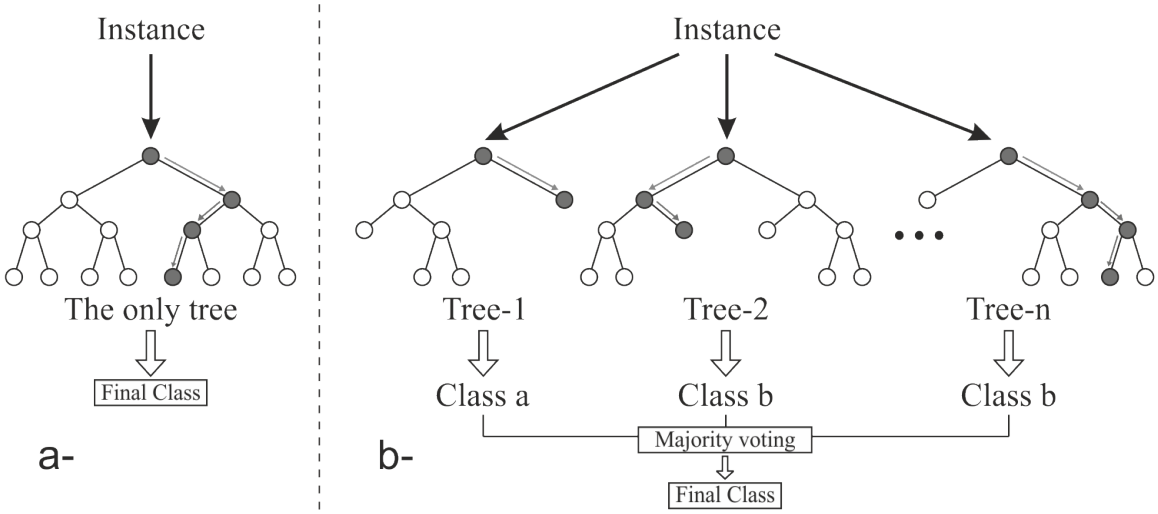

Figure 2.4: (a): A sample Decision Tree; (b): A sample Random Forest or Decision Tree ensembles

2.3. Autoregressive Models

Autoregressive Models (AR models) are widely employed in data mining domains especially time series, indicating the output variable depends on its own previous values and because of its stochastic nature, it takes advantage of the statistics of the data [56]. Not until recently have data scientists been engaged in applying AR models to categorical sequences. AR spectral analysis tools capture certain features of DNA sequences of coding and noncoding regions [44]. It claims that considering the type of the numerical mapping rule which is used, the Autoregressive Feature-Based searching techniques have a high accuracy in identifying the patterns and their locations in DNA sequences. Thereafter, many scholars verify that AR models for categorical time series perform well in analyzing the structure of sequences and finding patterns in them [45, 57–61]. However, AR models are new and mostly implemented in bioinformatics, thus lacking generality. They also have two parameters, model order and window length, to be set, leading a higher complexity of them.

AR models for categorical time series mainly include two steps: first, numerical mapping of sequence; then, implementing the AR model. However, categorical data might not have a logical order and this type of numerical mapping may lead to an irrational deduction. For instance, integer mapping rule such as converting DNA sequences in a way that $A = 1$, $G = 2$, $C = 3$, and $T = 4$ [57] (or $A = -1.5$, $T = 1.5$, $C = 0.5$, and $G = -0.5$ [44]) creates a false relation between categorical data, because the exact distance between A and G is not half the distance between A and C.

2.4. Jaro-Winkler and Levenshtein distance

Jaro metric [62], a metric widely used in the record-linkage community, has a variation due to Winkler [63], counting the common characters between strings even if their positions are slightly different. This common edit-distance method places more emphasis on the strings that match from the beginning (common prefix); therefore, it is unstable from one set to another [64].

Levenshtein distance (LD) [65], edit distance, is a promising measurement to compare two or more strings by various operations, including the insertion, deletion, and reversal (substitution) of symbols. It is defined as the minimum cost to transform one string into another through the aforementioned modifications [66]. The more edit operations are needed for transformation, the more distant two strings are. For instance, the LD between "Make" and "Mark" is 2 (we need to delete "e" and add "r") and the LD between "Make" and "Park" is 3 (Replacing "M" with "P", adding "r" and deleting "e"). Some [3, 67] argue that the edit distance is inefficient because of its inefficiency in calculation and capturing merely the optimal global alignment. For instance, the LD between "aaaabbb" and "abcdefg" is equal to the LD between "aaaabbb" and "bbbaaaa" (both are 6.) while the second comparison seems more similar. Furthermore, LD's computational complexity increases as the size of two strings is long.

2.5. Mixture Hidden Markov models

Mixture Hidden Markov models (MHMM), also called mixed Markov latent class models [68] or mixture latent Markov Model (MLMM) [69], contains five types of variables: response variables, time-constant explanatory variables, time-varying explanatory variables, time-constant discrete latent variables, and time-varying discrete latent variables. Here, for the simplicity of the model, we only consider time-constant covariates. Having a set of Hidden Markov Models $\mathcal{M} = \{\mathcal{M}^1, ..., \mathcal{M}^K\}$, where $\mathcal{M}^i = \{\pi^i, A^i, B^i_1, ..., B^i_C\}$ for submodels $i = 1, ..., I$. For each subject Y_i, $P(\mathcal{M}^i) = \omega_i$ is the prior probability that it follows the submodel $\mathcal{M}^i$. The log-likelihood of the parameters of the Mixture Hidden Markov Model can be formulated as [70]

$$\begin{aligned} \log L &= \sum_{i=1}^{N} \log P(Y_i|\mathcal{M}) \\ &= \sum_{i=1}^{N} \log \Big[\sum_{k=1}^{N} P\big(\mathcal{M}^k\big) \sum_{allz} P\big(Y_i|z, \mathcal{M}^k\big) P\big(z|\mathcal{M}^k\big) \Big] \end{aligned} \tag{2.3}$$

This method assumes that there exists a latent state resulting in the observed sequence. In all the experiments, we select the number of latent states (m) based on Bayesian Information Criterion (BIC),

$$BIC = -2l(\theta) + p \log m \tag{2.4}$$

for m in $[1, 5]$. Minimizing the BIC which corresponds to maximizing the posterior model probability is desirable [71].

3. CLUSTERING AND ITS METRICS

3.1. Unsupervised and supervised learning

Unsupervised learning, also called clustering, involves a process that automatically and without any priori information reveals the structure of data [72]. In this attitude of machine learning, since we lack the labeled training data, the learned model will never show the semantic meaning of the clusters which are found. Conversely, the paradigm of *supervised learning*, quite often referred to as a synonym for classification [73], involves either a classifier or a regression function working on a labeled examples in the training data set. Therefore, the output, acquired on the test set, is meaningful and is comparable to the past data. In fact, the task of classification is to find a function mapping data to the available set of labels [74].

Cluster analysis or simply clustering can be categorized as unsupervised learning process where a set of data (observations) is partitioned into subsets. In other word, it is a form of learning by observation rather than by examples. The clustering should be done in a way that objects in a cluster become similar to each other while dissimilar to those of other clusters. In order to evaluate the goodness of clustering, there are two important ways [73]:

- Measuring clustering quality
- Determining the number of clusters in a data set

For measuring clustering quality, some performance techniques have been introduced in the following section. Furthermore, at the experiment part, the number of clusters has been examined to see how well the new representation performs in clustering.

3.2. Performance metric

Clustering as an unsupervised learning approach strives to determine the innate pattern in a set of unlabeled data while considering intra-cluster quality and inter-cluster separation [75]. Two types of cluster validity, namely internal and external criteria [76], have been introduced. External validation is the one which is based on pre-specified structure or previous knowledge of data (knowing the true labels); whereas, Internal Validation evaluates the clustering result using features inherited in the dataset when no external criteria are available. In most real cases, using External Validation criteria is not possible; nevertheless, when synthetic data is generated with the known objective functions, associating cluster identifiers with the generated dataset is viable [77].

Here, External Validation indices are used whenever we are conversant with the structure of data and Internal Validation indices are implemented whenever we desire to determine the correct number of clusters.

3.2.1. Internal cluster validation indices

Calinski-Harabasz Index (CH) is referred to as two top performers for determining the number of clusters [78], representing a global criterion. Calinski and Harabasz [79] suggest "the best sum of squares split" of the dendrite is where not only is the within-group (cluster) sum of squares (WGSS) minimum, but also the between-group sum of square (BGSS) becomes maximum, which corresponds to the maximum value of $CH(k)$, which is defined as

$$CH(k) = \frac{BGSS}{k-1} \Big/ \frac{WGSS}{n-k} \tag{3.1}$$

where n is the number of clustered samples and k is the number of clusters. The average silhouette width (ASW) [80] has been computed whenever there is no need to establish a comparison with the true parameter. Given a partition ξ_S, the ASW is given by

$$ASW = \frac{1}{N}\sum_{i=1}^{N}(s(i)) \tag{3.2}$$

where $s(i)$, silhouette width, is obtained by

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{3.3}$$

with $a(i)$ being the average dissimilarity of object i to all other objects of A, the cluster to which it has been assigned; and if $d(i, C)$ be defined as the average dissimilarity of object i to all objects of C, any cluster which is different from A, then

$$b(i) = \underset{C \neq A}{minimum}\ d(i, C) \tag{3.4}$$

$s(i)$ takes values in $[-1, +1]$; when $s(i)$ is close to the value $+1$, object i is nearer to its own cluster than the other closest cluster and therefore is taken as "well-classified". On the other hand, when it is close to -1, the opposite relationship applies and i is taken as "misclassified". Finally, being about zero means it is unclear whether it should be assigned to its own cluster or the neighboring cluster.

As the last internal validity index, the Dunn Index [81] measures the ratio between the smallest intercluster and the greatest intra-cluster distance. Let $\delta(C_q, C_r)$ be the intercluster distance metric between clusters C_q and C_r, then for the number of clusters

k, the Dunn Index is defined as:

$$DI_k = \min_{1 \leqslant q \leqslant k} \left\{ \min_{\substack{1 \leqslant r \leqslant k \\ r \neq q}} \delta(C_q, C_r) \Big/ \max_{1 \leqslant p \leqslant k} \Delta(C_p) \right\} \tag{3.5}$$

where $\Delta(C_p)$ is the diameter of cluster C_p or the maximum intra-cluster distance. The larger the value of DI_k is, the more exact the clustering solution will be.

3.2.2. External cluster validation indices

When a priori information of the dataset is available, external cluster validation indices can be used. A recent study [82], considering most of the papers in the field of categorical time series, suggests that Clustering Accuracy/Purity, Rand Index (RI)/Adjusted Rand Index (ARI), and F-measure are the most common External Validation measures. We incorporate those methods in the simulation part wherever the true labels are known.

Given a set of n elements $S = \{X_1, ..., X_n\}$, and two clusterings of them, RI is the proportion of pairs of the two clusters which either belong to the same cluster or different clusters. RI [83] ranges from 0 when two clusterings have no similarity to 1 when the two clustering are identical. The Adjusted Rand Index (ARI) [84] is an index-corrected-for-chance version of RI which is bounded above by 1 meaning perfect agreement and takes on the value 0 when partitions are selected randomly. Cluster Purity, as one of the frequently used external measures [85], determines to which extent clusters contain a single class. Given $P_j = \frac{1}{n_j} Max_i \left(n_j^i \right)$ as the purity of cluster j with class label i, the overall Purity is defined as

$$Purity = \sum_{j=1}^{k} \frac{n_j}{n} Pj \tag{3.6}$$

which is as a weighted sum of the individual cluster's purities. Here, k is the number of clusters, n_j the size of cluster j, and n the total number of objects. High values of the Purity is desirable. Finally, F-measure which combines the precision and recall concepts can be used as an external cluster criterion which takes the values within [0,1], for which the higher value indicates a better clustering performance. F-measure is a useful method to evaluate clustering structure [86].

3.2.3. Comparing with Nearest Neighbor classifier

The K-NN classifier is based on the premise that classification of unknown instances can be done by associating the unknown object to the known using the given dis(similarity) function. The idea is to apply one nearest neighbor wherever the ground truth of a data set is available to measure the efficacy of the newly introduced method using "leaving-one-out" [87–89]. Therefore, We employed 1-NN to assess the effectiveness of nTreeClus and compare it with the other available methods.

4. nTreeClus: A NEW METHODOLOGY FOR CLUSTERING CATEGORICAL SEQUENTIAL DATA

The categorical sequential dataset, $X_{\mathcal{L}}^{\mathcal{M}}$, is comprised of $\mathcal{L}$ sequences of length $\mathcal{M}$ over an alphabet set of $\mathcal{A}$ where x_l^m is the mth element of sequence l. For the purposes of simplification, the sequences are assumed to be of the same length, $\mathcal{M}$, while the approach can handle sequences of different length. When the length of sequences is different, $\mathcal{M}$ is equal to $max\{\mathcal{M}_1, \mathcal{M}_2, .., \mathcal{M}_l\}$ and the shorter sequences will be treated as sequences with some missing values. In other words, nTreeClus can handle missing values no matter how they are distributed in the dataset. Therefore, x_i^j can either be selected of the alphabet set $\mathcal{A} = \{s_1, s_2, .., s_a\}$ or remain empty in the case that $\mathcal{M}_i < j$. X_i, the ith sequence of the dataset, can be represented by

$$X_i = x_i^1 x_i^2 .. x_i^{m-1} x_i^m \tag{4.1}$$

where it forms the ith row of the categorical dataset, $X_{\mathcal{L}}^{\mathcal{M}}$; thus, $X_{\mathcal{L}}^{\mathcal{M}}$ is illustrated by an $\mathcal{L} \times \mathcal{M}$ matrix as

$$X_{\mathcal{L}}^{\mathcal{M}} = \begin{bmatrix} x_1^1 & x_1^2 & \cdots & \cdots & x_1^{m-1} & x_1^m \\ x_2^1 & x_2^2 & \cdots & \cdots & x_2^{m-1} & x_2^m \\ \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ x_{l-1}^1 & x_{l-1}^2 & \cdots & \cdots & x_{l-1}^{m-1} & x_{l-1}^m \\ x_l^1 & x_l^2 & \cdots & \cdots & x_l^{m-1} & x_l^m \end{bmatrix} \tag{4.2}$$

4.1. Representation Learning

nTreeClus is founded based on two existing approaches, from both supervised and unsupervised learning. The primary aim of the algorithm is to find specific patterns in Categorical Sequence Data. The premise on which it is founded is extensible to any problem in which a pattern should be determined. In order to find a pattern in data, nTreeClus utilizes basic principles in available and commonly used approaches. In sequences, in which the definite number of alphabets is used to specify a distinctive characteristic of the instances, a pattern recognizer can be used to cluster them based on including or excluding particular pattern.

4.1.1. Phase I: segmented matrix representation

In the first step, nTreeClus tries to segment each sequence using a predefined length (n) in order to have a length-smoothed data instead of having sequences of different length. This step provides the algorithm an environment in which employing single Decision Tree or Decision Tree Ensembles is attainable. It also has the same logic as k-mers with the window size of n while trying to extract all possible substrings of length n that are contained in a given string. In the segmentation part, as shown in Figure 4.1, based on the predefined length n we divide each string into some substrings which have the length of $n + 1$. The last column is used as a class label while the first n elements play the role of features which result in the class label (in an autoregressive manner). In the given sequence in Figure 4.1, n is equal to 5. "a b c a a" are the five elements leading to the output "a" which is the next alphabet of the string. Based on both the length of each string and the predetermined "n", many or few new substrings are expected. In Algorithm 1, the loops to create the segmented matrix out of a given sequential Dataset have been demonstrated.

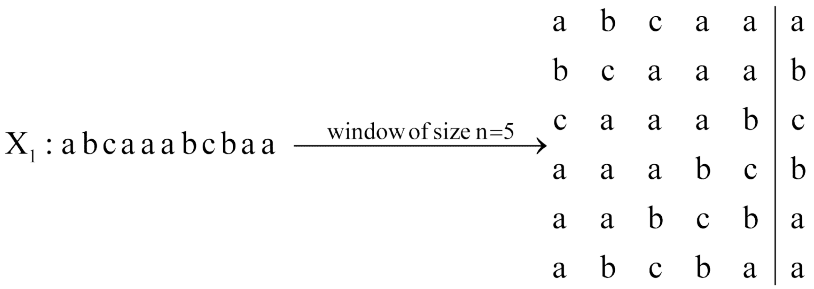

Figure 4.1: A sample of segmented sequence (the right column indicates labels while the first 5 columns are features. Since both labels and features come from the same data, it mimics autoregressive behavior.)

The segmentation part will be done to all rows of matrix $X_{\mathcal{L}}^{\mathcal{M}}$ and the segmented matrix will be generated. Figure 4.1 demonstrates the segmentation algorithm only for X_1, the first row of the supposed $X_{\mathcal{L}}^{\mathcal{M}}$. Assuming that all rows of $X_{\mathcal{L}}^{\mathcal{M}}$ have the same length $\mathcal{M} = 11$ and the total number of sequences is $\mathcal{L} = 100$, then for the window size of $n = 5$, we will have the segmented matrix (S) with 6 columns and $6 \times 100 = 600$ rows. Generally speaking, the segmented output is represented by a $((\mathcal{M} - n)\mathcal{L}) \times (n + 1)$ matrix.

Algorithm 1 Matrix Segmentation

Data: sequential/categorical dataset (D) of size $\mathcal{L} \times \mathcal{M}$, the segmentation length (n) (*with default value of* $\sqrt{\mathcal{M}}$).

Result: Segmented matrix (length-smoothed sequences)

```
initialization;
  Segmented Matrix = ∅
for i ∈{1,..,ℒ} do
    for j ∈{1,..,(ℳ_i-n)} do
        Temporary Row ← Dataset [i,j:j+n]
        Temporary Row ['y']← i
        Append [Temporary Row ] to Segmented Matrix
    end
end
```

Figure 4.2: Algorithm: Matrix Segmentation

4.1.2. Phase II: providing the novel nTreeClus representation

The segmented matrix is an input for the next phase of the algorithm in which the existing classification method, Decision Tree (DT), will be used. nTreeClus utilizes either Decision Tree or Tree Ensembles to have the least error at the end. Nonetheless, in this stage, the set of terminal nodes (leaf nodes) to which a specific substring belongs generates the most valuable input for Clustering. In general, after implementing Tree Ensembles, each substring goes to a specific terminal node of each generated tree while nTreeClus traces the exact position of the substring – that is to say, each rule of DT describes a pattern in the sequence. Assuming that the total number of trees in Decision Tree Ensembles is 3, the first substring in Figure 4.1, "abcaa", is assigned to terminal nodes 5, 3, and 6 of tree 1, 2, and 3 respectively (Figure 4.3). The same case for the rest of substrings is held.

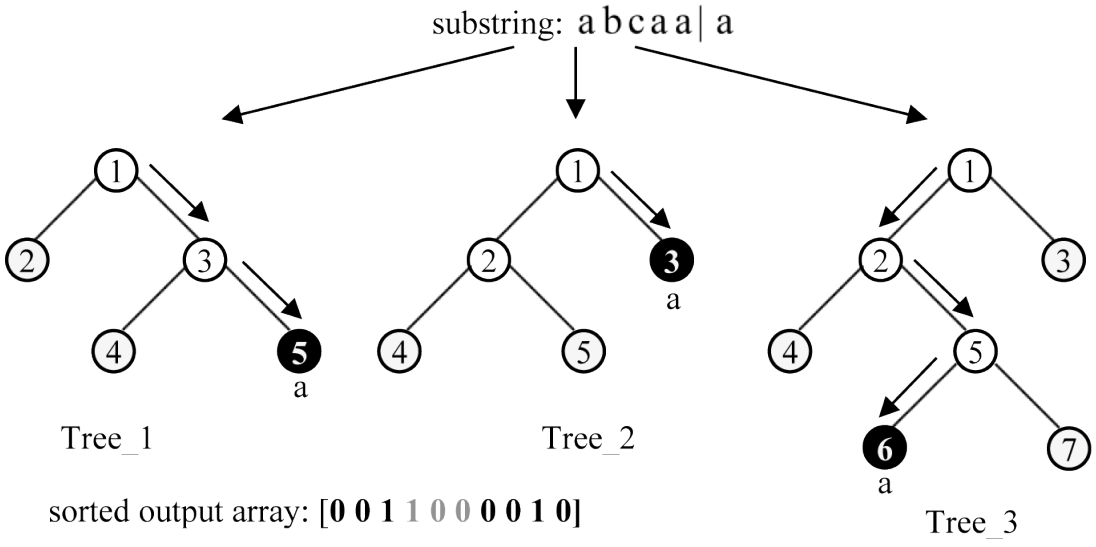

Figure 4.3: Introducing an instance to Decision Tree Ensembles

What nTreeClus is eager to record is a large, sparse matrix of terminal nodes to which each instance belongs. Therefore, the number of its columns is equal to the number of unique terminal nodes, here 10, and the number of its rows is equal to the number of strings (instances) given by the original matrix. Assume for the given sample in Figure 4.1, six substrings end in the terminal nodes as shown in Table 4.1.

The Table 4.1 can easily be shown as a binary matrix (η) whose columns are terminal nodes and whose rows are substrings. The equivalent matrix for the Table 4.1 will be as

Table 4.1: Terminal nodes for a sample instance.

substring's id	**tree 1**	**tree 2**	**tree 3**
1 (abcaa)	5	3	6
2 (bcaaa)	2	5	6
3 (caaab)	5	5	6
4 (aaabc)	5	5	7
5 (aabcb)	2	3	7
6 (abcba)	4	3	6

$$\eta_{(abcaaabcbaa)} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{4.3}$$

Table 4.1 holds the number of repetition for each instance which is defined by its substrings. It can be represented as Table 4.2 whose column names indicate the trees' ids and their terminal nodes; for instance, "2-4" specifies terminal node 4 of tree 2. The numerical values inside the Table 4.2 demonstrate how many times an instance is ended in a specific tree of Decision Tree Ensembles. "0" under column "2-4" declares none of the substrings of instance id 1 is ended in terminal node 4 of tree number 2 and so forth. (Table 4.2 has the number of rows equal to the total number of instances,

one of which is illustrated herein.)

Table 4.2: Decision Tree Ensembles' output matrix

instance id	**1-2**	**1-4**	**1-5**	**2-3**	**2-4**	**2-5**	**3-3**	**3-4**	**3-6**	**3-7**
1	2	1	3	3	0	3	0	0	4	2

that is to say the matrix 4.3 can be aggregated and generate the equivalent matrix form of Table 4.2 as

$$\eta_{aggregated} = \begin{pmatrix} 2 & 1 & 3 & 3 & 0 & 3 & 0 & 0 & 4 & 2 \end{pmatrix} \tag{4.4}$$

Algorithm 2 nTreeClus Representation

Data: Segmented Matrix of size $((\mathcal{L} \times (\mathcal{M} - n)) \times (n + 2))$, number of trees (t) *with default value of 10.*

Result: nTreeClus representation of dataset D (nTreeClus Representation)

initialization;
  xtrain ← Segmented Matrix [:,1:n]
  ytrain ← Segmented Matrix [:,n+1]

Train RandomForest (X=xtrain, Y=ytrain, ntree = t)

terminalRF ← trace the path of each row in Segmented Matrix to the terminal node for all t trees and store it.

nTreeClus Representation = An empty DataFrame whose number of columns is equal to terminalRF's one and whose number of rows is equal to Dataset's one.

for $i \in \{1,..,\mathcal{L}\}$ **do**

  nTreeClus Representation [i,:] = $\sum_{j=1}^{(\mathcal{L} \times (\mathcal{M}-n))}$ terminalRF$[j,:]$ iff Segmented Matrix [j,'y']==i

end

Figure 4.4: Algorithm: nTreeClus Representation

4.2. Similarity Measure

Using Algorithm 2, we can find a new representation for each sequence. Here, for the categorical sequence (abcaaabcbaa), we propose the numeric representation (2133030042) which can record the logic behind this set of alphabets using autoregressive DT ensembles. This numeric representation could be a proper feed to find the distance between each pair of strings. There are numerous ways to find (dis)similarity between these sequences, from which we have implemented Cosine and Manhattan distance to find out which one is the optimal method to be used. Having the distance between each sequence, we employ Hierarchical grouping using Ward's method to cluster data [90]. Ward's minimum variance method, outperform other linkage method except when the outliers exist [91].

Given two sets of numeric vectors of the same length (l), here nTreeClus representations, Cosine dissimilarity (θ) can be shown as

$$cos(\theta) = 1 - \frac{a.b}{||a||_2||b||_2} = 1 - \frac{\sum_{i=1}^{l} a_i \times b_i}{\sqrt{\sum_{i=1}^{l}(a_i)^2} \times \sqrt{\sum_{i=1}^{l}(b_i)^2}}, \tag{4.5}$$

where $||*||_2$ indicates 2-norm of its argument *, and $a.b$ is the dot product of a and b. Cosine similarity is within the range of 0 and +1 while +1 indicates close similarity and 0 declares strong dissimilarity. On the other hand, Manhattan distance places more emphasis on differences between elements of two different vectors. Manhattan distance, or the Minkowski distance with 1-norm, can be shown as

$$D_{man}(a, b) = ||a - b||_p = \Big(\sum_{i=1}^{l} (|a_i - b_i|)^p \Big)^{\frac{1}{p}}; p = 1 \tag{4.6}$$

The complexity of nTreeClus is comprised of the complexity of two phases, which is $\mathcal{L} \times (\mathcal{M} - n)$ for segmentation part and the complexity of Random Forest $O(\sqrt{(\mathcal{M} - n)}\mathcal{L} \log \mathcal{L})$; therefore, it is highly related to the length and the number of sequences. Since Decision Tree ensembles are fast in real cases, the method have a satisfactory running time. However, the computational complexity of k-mer is $O(\mathcal{L}^2\mathcal{M})$ and quadratically increases if the number of sequences increases [92, 93]. The same case exists for Jaro-Winkler where the time complexity is quadratic $O(\mathcal{M}_p + \mathcal{M}^2)$ [94]. Furthermore, in Levenshtein distance, the time complexity is $O(\mathcal{M}^2)$ [95] and similarly it has a quadratic form. Mixture of Hidden Markov Models have the quadratic relation to the number of hidden states while both the length and the number of sequences have direct impact on its complexity.

5. EXPERIMENTS AND EXAMINATIONS

In order to have a fair comparison between nTreeClus and other methods, different scenarios imitating the properties of categorical/nominal sequential data with varying levels of intricacy have been generated. The algorithm has been applied to the generated data with different amounts of n (window size).

A comparison with alternative approaches which are defined as *Similarity-Based*, *Feature-Based*, and *Model-Based Approach* has been conducted to make deductions about the competitiveness of nTreeClus.

5.1. Patterns to be identified

Sequential pattern discovery has been available in a vast variety of domains. Such an omnipresence leads to many suggested approaches processing data in a hope of finding a better understanding of the nature of different data types. In a given sequence, manifold forms of patterns may be available, some of which are illustrated in Figure 5.1. In Figure 5.1a, the first important type of the patterns, a shifted pattern, has depicted; Figure 5.1b is a modification of the previous type while a gap is seen between the pattern; Figure 5.1c is a closed lopped version of the first type of the pattern; finally Figure 5.1d represent a rare type of the sequence in which the same structure and order can be seen while different alphabet has been used. Because of scarcity of the last case, in the following trials, the first three patterns have been examined and a clear contrast between all the available methods has been drawn.

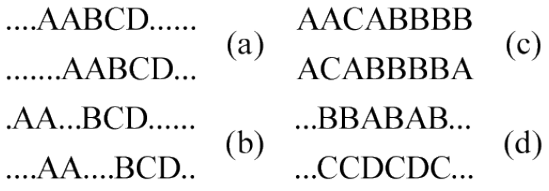

Figure 5.1: (a): repeated pattern in different positions; (b): repeated gapped pattern; (c): shifted pattern in a closed loop; (d): same pattern with different alphabets.

5.2. Tuning parameter n

The parameter n in nTreeClus is similar to the parameter k in k-mers. Using a large n has the advantage of covering a general relation among the elements; by contrast, the small n is superior wherever the pattern is short and the volume of noise in data is high. Apparently, if the size of the pattern is known and equal to τ, the window size (n) should be set as τ to capture whole substrings with the equal length; nonetheless, in most real cases, the size of the pattern is unknown. In order to analyze the sensitivity of the model to the parameter n, a contrast has been drawn between nTreeClus and k-mers (n-grams).

A simulation including $\mathcal{N} = 40$ batches of sequences has been conducted. Each batch is formed from $\mathcal{L} = 180$ sequences with different characteristics. They may take $a = \{7, 20\}$ different categorical values in $\mathcal{A} = \{A, B, C, ..., S, T\}$ with the length of $\mathcal{M} = 40$. Eventually, a fixed random pattern with the length of $\mathcal{L}_P = \{8, 15\}$ chosen from the same set $\mathcal{A}$ has been inserted in a random position of half of the sequences. In order to eliminate any biased randomness, the simulation has been replicated 10 times.

Figure 5.2 clarifies the result of the experiment on 4 different datasets with the aforementioned properties. In Figure 5.2 (a) and (b), the length of the pattern ($\mathcal{M}_P$) is 7 and only the number of alphabets which is selected from set $\mathcal{A}$ is different. As

the number of alphabets rises (going from (a) to (b)), so does the Average Silhouette Width (ASW) in that the noise in the sequences increases and recognizing the pattern turns less challenging. Skipping the shift in ASW, the general form of both graphs are similar; as long as the n in nTreeClus is less or equal to the length of pattern (gray area), the ASW for nTreeClus is stable and afterward, it converges to a less but acceptable ASW. k-mers, by contrast, experiences no stable region, no convergence, and even no specific correlation with the pattern length. In Figure 5.2 (c) and (d), the length of the pattern ($\mathcal{M}_P$) is larger and, therefore, it is easier to be recognized. Again nTreeClus possesses the safe gray region of accuracy (for $3 \leqslant n \leqslant \mathcal{L}_P$), becomes stable, and then asymptotically approaches a reasonable value for ASW. Conversely, k-mer lacks the safe region and is not applicable to the generated dataset for the large values of k.

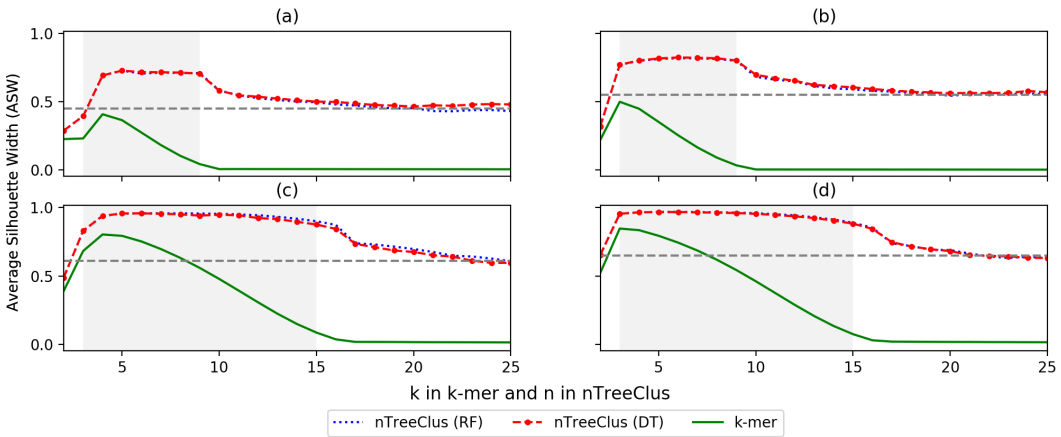

Figure 5.2: The output of 10 replications for different characteristics of data: (a): $a = 7$ & $\mathcal{L}_P = 8$; (b): $a = 20$ & $\mathcal{L}_P = 8$; (c): $a = 7$ & $\mathcal{L}_P = 15$; (d): $a = 20$ & $\mathcal{L}_P = 15$; three methods have been compared based on ASW for different values of n.

Figure 5.2 also illustrates that for the randomly simulated datasets, nTreeClus consistently outperforms its rival k-mers. In this simulation, cosine dissimilarity has been used and, as the Figure indicates, nTreeClus is robust to the classification methods since both Random Forest (RF) and Decision Tree (DT) lead to almost identical consequences.

As the simulation puts forward, if the ground truth is available, the best n is the value close to $\mathcal{M}_P$. For the cases where no knowledge about data is at hand, the second simulation has been carried out and a default value for parameter n is examined. Three different values for n are inspected, namely square root, one-half, and one-tenth of the length of sequences, $\mathcal{M}$.

Second simulation includes 90 batches of sequences with the structure of $\mathcal{L} = \{180, 450, 720\}$, $a = 7$ different categorical values in set of $\mathcal{A} = \{A, B, C, ..., G\}$, $\mathcal{L}_P = \{7, 15\}$, and finally $C = \{2, 5, 8\}$ number of clusters. Unlike the first simulation, the number of clusters varies in order to cover different circumstances that may occur in real cases. After five replications, 40500 sequences with the specified characteristics have been simulated. Figure 5.3 shows the average ranks for all different settings of n. The Friedman test shows a significant difference between the 3 settings at the both 0.05 and 0.1 level exists. Proceeding with the Nemenyi test, Critical Difference (CD) is computed, showing the performance of $n = \sqrt{\mathcal{M}}$ is significantly better than $n = 0.1\mathcal{M}$ and $n = 0.5\mathcal{M}$ at different α levels. Therefore, in the following experiments, n always has been set to the square root of $\mathcal{M}$.

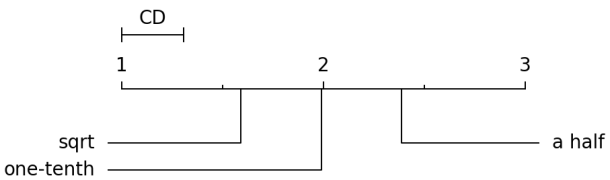

Figure 5.3: The average rank for three different setting of n. The critical difference at 0.05 and 0.1 levels are 0.349 and 0.306, respectively. The performance of $n = \sqrt{\mathcal{M}}$ is significantly better at different α levels.

5.3. Simulations

As it is described in section 5.1, in this part, we try to mimic different scenarios which may occur as a pattern in a sequence data. The simulated datasets are either with the normal shifted pattern inside or with a more complicated shifted-gapped

pattern. The length of the sequences and the number of alphabets differ case by case. The more detailed description of the scenarios is presented in the following sections.

5.3.1. Simulation 1 - pattern recognition

The first simulation is comprised of $\mathcal{N}$=1,080 batches of sequences, each of which is formed from $\mathcal{L}$= {40, 120, 200} sequences with different characteristics. They may take $a = \{4, 6, 10, 20\}$ different categorical values in $\mathcal{A}$={A, B, C, ..., S, T}. The length of the sequences ($\mathcal{M}$) oscillates between 20 to 90 while it is fixed for each batch. Eventually, a random pattern with the length of $\mathcal{M}_P$= {6, 10, 15} has been inserted in each half of the sequences of each batch, thus generating two clusters ($C = 2$). In order to eliminate any biased randomness, the simulation has been replicated 10 times. The ultimate output of these replications consists of $\mathcal{N}_S$=129,600 sequences.

Table 5.1 demonstrates the results of the six methods for the first scenario. It indicates nTreeClus, whether uses Decision Tree or Random Forest, leads to substantially higher indices than those of the rest of algorithms. The closest approach to nTreeClus is k-mers for $k = \sqrt{\mathcal{M}_i}$ which has the Average Silhouette Width (Internal Validation) only half as much as that of nTreeClus. Another somewhat surprising result is that the accuracy of nTreeClus is not affected by choosing either Decision Tree or Decision Tree Ensembles, indicating its robustness to the method selection. In MHMM, ranked third based on External indices, the number of latent states (m) is chosen based on Bayesian Information Criterion (BIC), $BIC = -2l(\theta) + p \log m$ for m in $[1, 5]$. The same setting for the following simulations has been done. Assuming no foreknowledge is available, the prior probability is selected randomly; hence, its accuracy changes constantly.

Table 5.1: Clustering goodness measures in Simulation 1; here, k in k-mer and n in nTreeClus is floating and equal to the square root of the length of sequences.

Methods	External Validation				Internal Validation	1NN
	Purity	RI	ARI	F-meas	ASW	
nTreeClus (DT)	0.988	0.980	0.959	0.987	0.655*	0.986
nTreeClus (RF)	0.992*	0.984*	0.969*	0.990*	0.651	0.989*
Levenshtein	0.892	0.830	0.660	0.868	0.343	0.932
Jaro–Winkler	0.654	0.497	-0.006	0.508	0.007	0.234
k-mers						
$k = 1$	0.793	0.688	0.377	0.762	0.385	0.740
$k = 2$	0.910	0.861	0.721	0.901	0.420	0.894
$k = 3$	0.967	0.945	0.891	0.963	0.442	0.958
$k = \sqrt{\mathcal{M}_i}$	0.971	0.936	0.873	0.939	0.346	0.937
MHMM	0.960	0.849	0.700	0.816	-	-

5.3.2. Simulation 2 - gapped pattern recognition

Unlike the rigid and simple nature of the first simulation, the second one consists of sequence patterns with gaps where the relative distance between positions is dynamic. Here, patterns x and y with the length of $\mathcal{L}_{(P_x,P_y)}$= {(2,3), (4,6), (6,9)} respectively have been inserted in each half of the data while the gap in between randomly changes. For x={A,C} and y={A,B,E}, it is expected to have them in sequences with different gap in between; for instance, ..**AC**AABDBA**ABE**.., ..**ACABE**.., or ..**AC**BE**ABE**.. . The other variables, including $\mathcal{L}$, a, $\mathcal{A}$, $\mathcal{M}$, and $\mathcal{N}_S$, are the same as the first simulation. The second simulation with dynamic distance and shorter pattern's length aims to examine the performance of each method in identifying intricate patterns in a given categorical sequence.

Table 5.2 indicates the general performance of the six methods has reduced as the structure of the data becomes more complex. Once again, it shows that nTreeClus results in considerably higher indices than those of the rest of algorithms. Here 3-mer and MHMM are the closest opponent of the current method. On account of the greater complexity of this new dataset, the ASW index for all the methods decreases while

the external indices still produce a satisfactory result. Again the worst performance corresponds to the Jaro–Winkler distance in that using edit distance is more rational whenever the length of the sequence is short; namely words and names [96].

Table 5.2: Clustering goodness measures in Simulation 2; here, k in k-mer and n in nTreeClus is floating and equal to the square root of the length of sequences.

Methods	External Validation				Internal Validation	1NN
	Purity	RI	ARI	F-meas	ASW	
nTreeClus (DT)	0.939	0.885	0.771	0.904	0.473*	0.909
nTreeClus (RF)	0.943*	0.899*	0.798*	0.916*	0.466	0.920*
Levenshtein	0.869	0.796	0.592	0.839	0.319	0.868
Jaro–Winkler	0.655	0.496	-0.006	0.507	0.007	0.241
k-mers						
$k = 1$	0.788	0.677	0.355	0.748	0.374	0.728
$k = 2$	0.888	0.825	0.650	0.870	0.368	0.864
$k = 3$	0.927	0.886	0.773	0.917*	0.335	0.907
$k = \sqrt{\mathcal{M}_i}$	0.902	0.774	0.549	0.777	0.195	0.808
MHMM	0.943*	0.783	0.571	0.732	-	-

5.3.3. Simulation 3 - length sensitivity analysis

In most real cases, the length of sequences to be compared with each other is varying. In such a case, the method which is robust to the length of sequences can yield the best result. Hence, the third simulation is comprised of $\mathcal{N}$=360 batches of sequences, each of which is formed from $\mathcal{L}$= {40, 120, 200} sequences with different characteristics. They may take $a = \{5, 7, 11, 16\}$ different categorical values in $\mathcal{A}$={A, B, C, ..., O, P}. Here, the length of the sequences ($\mathcal{M}$) oscillates between 80 to 120 and is not fixed even for each batch. Eventually, a random pattern with the length of $\mathcal{M}_P$= {6,10,20} chosen from set $\mathcal{A}$ has been inserted in each half of the sequences of each batch, thus generating two clusters ($C = 2$). To eliminate any biased randomness, the simulation has been repeated 10 times.

Because of ever-changing nature of sequences, parameter n in nTreeClus has been set to 10 which is equal to the square root of average length 100 ($n = \sqrt{average(\mathcal{M})} = 10$).

Table 5.3: Clustering goodness measures in Simulation 3; here, $n = 10$ in nTreeClus is equal to the square root of the average length of sequences.

Methods	External Validation				Internal Validation	1NN
	Purity	RI	ARI	F-meas	ASW	
nTreeClus (DT)	0.971	0.951	0.901	0.969	0.367*	0.966
nTreeClus (RF)	0.983*	0.968*	0.936*	0.981*	0.360*	0.981*
Levenshtein	0.815	0.748	0.496	0.787	0.146	0.898
Jaro–Winkler	0.648	0.496	-0.008	0.503	0.007	0.276
k-mers						
$k = 1$	0.689	0.565	0.129	0.636	0.228	0.706
$k = 2$	0.898	0.830	0.660	0.888	0.201	0.884
$k = 3$	0.969	0.946	0.892	0.967	0.228	0.961
$k = \sqrt{\overline{\mathcal{M}}}$	0.988*	0.977*	0.953*	0.988*	0.087	0.983*
MHMM	0.944	0.781	0.566	0.741	-	-

Table 5.3 clearly shows that, based on External Validation indices, nTreeClus and k-mer exceed the others in performance. Notwithstanding the good performance of k-mer in External Validation, it is nTreeClus that outperforms all other methods regarding Average Silhouette Width index. Once more, the worst performance corresponds to Jaro–Winkler and Levenshtein method.

5.4. Estimating the number of clusters

In this experiment, the number of clusters has been estimated to find out how consistent the nTreeClus representation of the sequential data is with the internal indices. Average Silhouette Width, Dunn Index, and Calinski and Harabasz have been considered as the functions examining inter- and intra-clusters sum of squares. The data to measure the efficacy of the method in estimating the number of clusters consists of $\mathcal{N}$=216 batches of sequences; based on definition, for them $\mathcal{L}$= {30,100},

a={4, 7, 20}, $\mathcal{M}$={50,100}, and $\mathcal{M}_P$= {7, 18}. This time, the number of clusters (C) also has changed and has been taken from the values of 3, 6, and 10. The trials have been repeated 3 times which provide us with the set of 72 batches with different characteristics.

All the three methods have been utilized to estimate the number of clusters in each batch. Distribution of the estimated number of clusters using internal cluster validation indices is shown in Table 5.4. The surprising result is that ASW, Dunn, and CH show a robustness under different scenarios. ASW with the accuracy of 93.5 percent has the highest performance while CH and Dunn index comes second and third with 77.7 and 72.2 percent respectively. The last two methods underestimate the number of clusters whenever there is a high level of noise. The experiment indicates that no matter what the characteristic of the data is, nTreeClus has a high intra-cluster and low inter-cluster similarity which enable it to cluster the sequential data with robust, acceptable internal evaluation score.

Table 5.4: Distribution of the estimated number of clusters ($\hat{C}$) using internal cluster validation indices for 24 different scenarios, each replicated 3 times

Index	Number of clusters, $\hat{C}$											median
	2	3	4	5	6	7	8	9	10	11-15	16-20	
	Sim. 1 ($C = 3$)											
ASW		69*									3	3
Dunn	10	59*	3									3
CH	12	60*										3
	Sim. 2 ($C = 6$)											
ASW					66*					1	5	6
Dunn	17				51*	4						6
CH	15	1			56*							6
	Sim. 3 ($C = 10$)											
ASW							1	1	67*	1	2	10
Dunn	23								46*	3		10
CH	17	1		1				1	52*			10

Figure 5.4 shows how the performance of the methods changes for different number of clusters (C). The graph clearly indicates that Random Forest modification of nTreeClus, is robust for different amounts of C while the clustering goodness measure of all other methods decreases as the number of clusters increases.

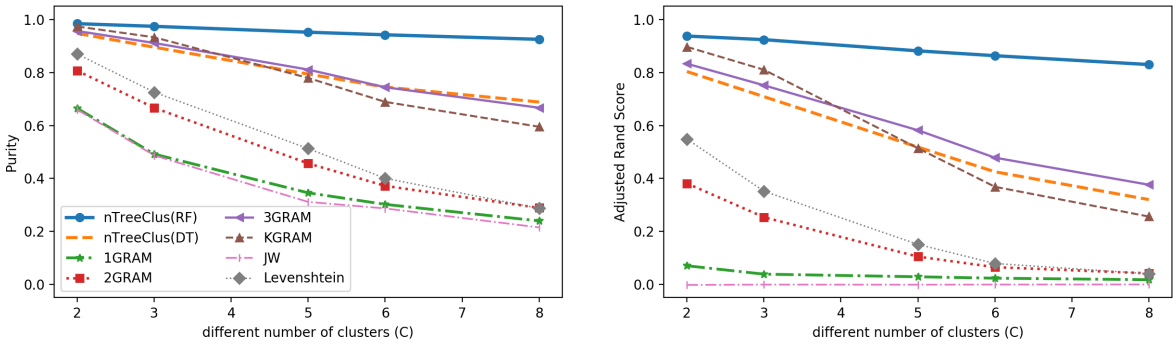

Figure 5.4: Analyzing robustness / ruggedness of the methods under different amounts of C

5.5. Real data application

5.5.1. Austrian Wage Mobility Data

The Austrian Wage Mobility dataset, first used by Pamminger and Fruhwirth-Schnatter [97], reports the wage category in successive years for the young men entering labour market from 1975 to 1980. It consists of $\mathcal{L} = 9402$ of such workers whose gross monthly wages of successive years (ranging from $\mathcal{M} = 2$ to $\mathcal{M} = 32$ years) have been examined. The gross monthly wages have been divided to 6 categories, from 0 to 5, in which zero corresponds to unemployment and categories one to five correspond to the quintiles of the income distribution [98].

Pamminger and Fruhwirth-Schnatter use Dirichlet multinomial clustering and Markov Chain clustering, suggesting that the number of cluster is between four and

six. We have tried nTreeClus on the data and for the Internal Validations, ASW and CH, it recommends 5 and 6 clusters. Considering the weighted average score of both validation methods, the number of clusters ($\hat{C}$) is more likely to be 5 (Figure 5.5).

Since the same dataset has been examined by García-Magariños and Vilar [99] with special attention to $\hat{C} = 4$ and $\hat{C} = 5$, and by Pamminger and Fruhwirth-Schnatter [97] with special attention to $\hat{C} = 4$, nTreeClus has been applied for the both cases to contrast it with the result of the previous studies.

Parameter n in nTreeClus has been set to 1, since the shortest sequence in the given dataset has the length of 2 years. In this case, where n is too small, nTreeClus cannot capture the relation between the categorical time series for the upcoming years. If n increases, inevitably short sequences should be removed from the dataset and in order to keep the consistency between nTreeClus and the other two methods, we decided not to augment the value of n. In spite of the compulsion to choose the least possible value for n, the output seems easily interpretable. Previously, the dataset has been categorized as *unemployed*, *climbers*, *low-wage*, and *flexible*, while nTreeClus suggests a new behavior in the dataset (shown in the first row of Figure 5.6), in which flexible (*temporary workers*), climbers to the level 4 of wage (*middle-level employees*), top-level employees (*top managers*), and finally *the low-paid workers* can be seen. Making inferences clearer, the first mixed behavior can be considered as the ordinary workers who have no opportunity to take middle or high rank in the organization and after a while they become unemployed; the second group are those who are the middle-level employees with the chance of attaining to a better paying position but not the highest; the third cluster mostly contains top-level employees who move mainly between the fourth and fifth income level; and finally the last cluster can be considered as the ones who work and work without any improvement in their income, keeping them as a low-wage employees during their whole service to the organization. Comparing to Fig. 6 in Pamminger and Fruhwirth-Schnatter [97] and to Fig. 11 in García-Magariños

and Vilar [99], nTreeClus includes the *low-wage* and *climbers* while defines new easily interpretable groups of *top managers* and *temporary workers*.

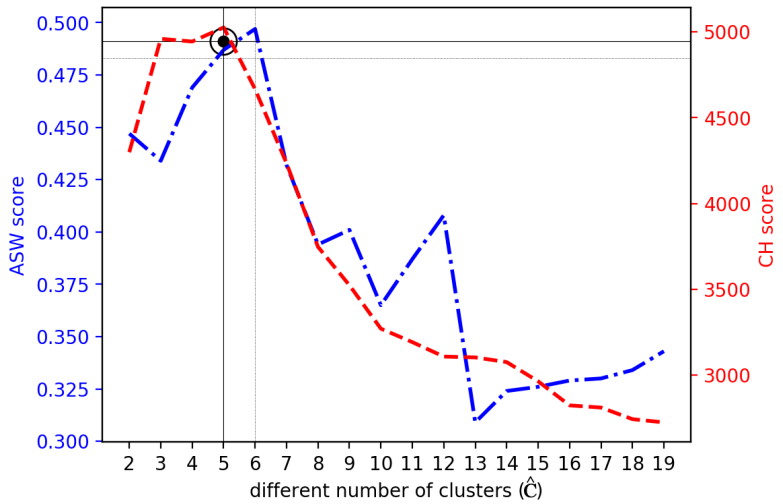

Figure 5.5: Finding the best number of clusters ($\hat{C}$) using Average Silhouette Width and Calinski-Harabasz Index

For $\hat{C} = 5$ (Figure 5.7), which is mainly discussed by García-Magariños and Vilar [99] and is considered as the optimal number of clusters by nTreeClus, flexible (*temporary workers*) cluster has been split up to two groups; first, the ones who can maintain their position in the organization without quitting the job and even with a limited prospect for promotions; second, the ones who resign from the organization after a short period and remain unemployed. The transition diagram in the second row of Figure 5.7 clearly indicates how each cluster moves among different wage levels. For cluster 1 (the most left Figure), there is a chance of promotion from level 1 to level 3, while for the cluster 2, the probability of quitting the job is more than any other states. For cluster 3, going from state three to four, for cluster 4, going from state four to five, and for the last cluster, remaining as a low-paid worker (keeping in a constant level 1) have the highest likelihood.

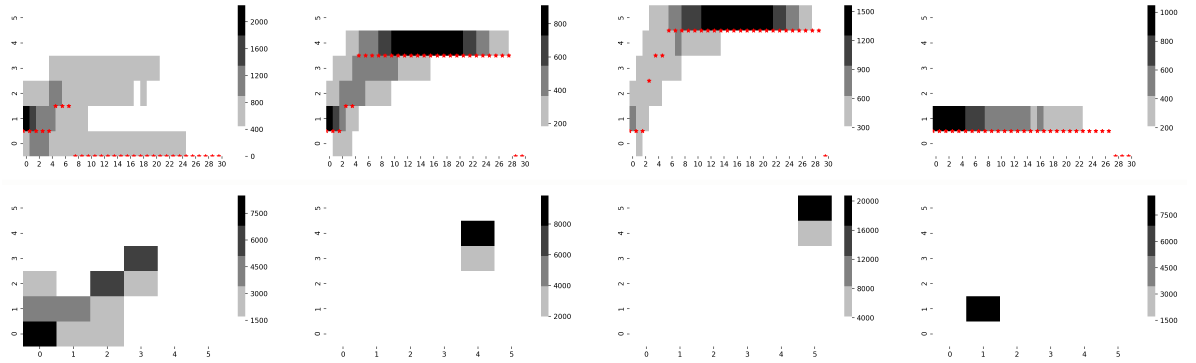

Figure 5.6: Heat-map of the wage clusters for $\hat{C} = 4$ (first row) and the transition between different wage groups for $\hat{C} = 4$ (second row)

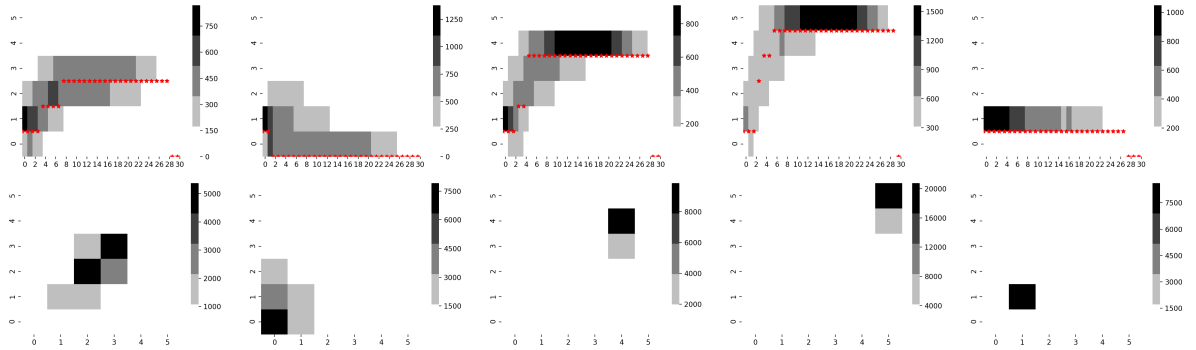

Figure 5.7: Heat-map of the wage clusters for $\hat{C} = 5$ (first row) and the transition between different wage groups for $\hat{C} = 5$ (second row)

5.5.2. Protein Data

As the second real dataset, a protein dataset has been taken from [7], containing $\mathcal{L} = 2112$ sequences with the length $\mathcal{M}$ between 80 and 127. The protein dataset has $a = 20$ alphabets (amino acids). Each sequence has one of two tasks, viz. "Might get involved in a Signal Recognition Particle (SRP) pathway" or "Bind to DNA and changes its conformation". Sequences are labeled based on these two functions and have a reasonably balanced distribution (979 of the first task and 1133 of the second task).

Table 5.5: Clustering goodness measures in Protein Dataset; here, k in k-mer and n in nTreeClus is equal to the square root of the average length of sequences (11).

Methods	External Validation				Internal Validation
	Purity	RI	ARI	F-meas	ASW
nTreeClus (DT)	1.000	1.000	1.000	1.000	0.815
nTreeClus (RF)	1.000	1.000	1.000	1.000	0.832
Levenshtein	1.000	1.000	1.000	1.000	0.950
Jaro–Winkler	0.750	0.500	-0.001	0.480	-0.003
k-mers	0.949	0.903	0.805	0.948	0.444
MHMM	1.000	1.000	1.000	1.000	-

The result shown in Table 5.5 demonstrates that nTreeClus has the pure result based on External Validation criteria, while for Internal Validation, it is Levenshtein that has the highest Average Silhouette Width. In previous datasets, Levenshtein obtained a low score indicating although there is a probability to get a proper result through the method, this result is highly correlated with the type of the pattern in the sequence. Again, nTreeClus is among the best methods based on the quality of clustering (Internal or External Validation indices).

6. CONCLUSION

In the thesis, we argued that sequence mining is a paramount need to be addressed by data scientists and that the state-of-the-art algorithms face major hurdles including universal applicability, computational complexity, sensitivity to the position of a pattern and the length of sequences, and vulnerability to parameter setting. We further showed empirically that based on high external cluster validation indices and low internal cluster validation indices, those methods overfit on sequences in the case that we are inconversant with the structure of the dataset.

As a step towards mitigating these potential problems, we introduced a new framework for clustering categorical time series. nTreeClus exploits the strength of the current common methods, including Decision Tree Ensembles, k-mers, and autoregressive models for categorical sequences, and introduces a comprehensive approach to answer the aforementioned problems of earlier works. The study proposes a numeric representation for categorical sequences established upon the autoregressive behavior of the data. At the first phase, sequences are segmented to length-smoothed substrings based on the predefined window size. This segmented matrix is a feed for autoregressive Decision Tree ensembles where each rule of DT describes a pattern in the sequence. In the second phase, nTreeClus representation of the sequences is introduced where the number of observation in each terminal node is counted. It indicates the behavior of sequences on the foundation of the correlative structure of the DT's rules. Using the common similarity methods, the new representation can be clustered.

Our experimental results show that nTreeClus is robust towards parameter setting and provides computationally efficient and superior results on benchmark real and synthetic datasets with different characteristics. Nevertheless, There are many directions in which the method may be extended. The performance of the method

for classification should be explored. Furthermore, nTreeClus should be extended to empirically and theoretically investigate categorical sequences which are continuous series of elements and where the time between their elements is of importance. This type of dataset can be found in a click-stream mining domain, although discretizing the time and repeating the element is an option to use the current version of nTreeClus for them.

REFERENCES

1. Karaca, M., M. Bilgen, A. N. Onus, A. G. Ince and S. Y. Elmasulu, "Exact tandem repeats analyzer (E-TRA): A new program for DNA sequence mining", *Journal of Genetics*, Vol. 84, No. 1, pp. 49–54, April 2005.

2. Kantety, R. V., M. La Rota, D. E. Matthews and M. E. Sorrells, "Data mining for simple sequence repeats in expressed sequence tags from barley, maize, rice, sorghum and wheat", *Plant Molecular Biology*, Vol. 48, No. 5, pp. 501–510, March 2002.

3. Dong, G. and J. Pei, *Sequence data mining*, Vol. 33, Springer Science & Business Media, 2007.

4. Garofalakis, M. N., R. Rastogi and K. Shim, "SPIRIT: Sequential pattern mining with regular expression constraints", *VLDB*, Vol. 99, pp. 7–10, 1999.

5. Montgomery, A. L., S. Li, sKannan Srinivasan and J. C. Liechty, "Modeling Online Browsing and Path Analysis Using Clickstream Data", *Marketing Science*, Vol. 23, No. 4, pp. 579–595, 2004.

6. Masseglia, F., M. Teisseire and P. Poncelet, "Sequential pattern mining", *Encyclopedia of Data Warehousing and Mining*, pp. 1028–1032, IGI Global, 2005.

7. Ranjan, C., S. Ebrahimi and K. Paynabar, "Sequence Graph Transform (SGT): A Feature Extraction Function for Sequence Data Mining", *arXiv preprint arXiv:1608.03533*, 2016.

8. Chen, M.-S., J. S. Park and P. S. Yu, "Efficient data mining for path traversal patterns", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10, No. 2,

pp. 209–221, March 1998.

9. Bicego, M., V. Murino and M. A. T. Figueiredo, "Similarity-Based Clustering of Sequences Using Hidden Markov Models", *Machine Learning and Data Mining in Pattern Recognition*, pp. 86–95, 2003.

10. Needleman, S. B. and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins", *Journal of Molecular Biology*, Vol. 48, No. 3, pp. 443 – 453, 1970.

11. Smith, T. and M. Waterman, "Identification of common molecular subsequences", *Journal of Molecular Biology*, Vol. 147, No. 1, pp. 195 – 197, 1981.

12. Bunke, H., *Hybrid Methods in Pattern Recognition*, pp. 367–382, Springer Berlin Heidelberg, Berlin, Heidelberg, 1987.

13. Brudno, M., S. Malde, A. Poliakov, C. B. Do, O. Couronne, I. Dubchak and S. Batzoglou, "Glocal alignment: finding rearrangements during alignment", *Bioinformatics*, Vol. 19, No. 1, pp. 54–62, 2003.

14. Bray, N., I. Dubchak and L. Pachter, "AVID: A global alignment program", *Genome research*, Vol. 13, No. 1, pp. 97–102, 2003.

15. Sonnhammer, E. L. and R. Durbin, "A dot-matrix program with dynamic threshold control suited for genomic DNA and protein sequence analysis", *Gene*, Vol. 167, No. 1, pp. GC1 – GC10, 1995.

16. Altschul, S. F., T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller and D. J. Lipman, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", *Nucleic Acids Research*, Vol. 25, No. 17, pp. 3389–3402, 1997.

17. Pearson, W. R., "Rapid and sensitive sequence comparison with FASTP and FASTA", *Methods in Enzymology*, 1990.

18. Edgar, R. C., "MUSCLE: multiple sequence alignment with high accuracy and high throughput", *Nucleic Acids Research*, Vol. 32, No. 5, pp. 1792–1797, 2004.

19. Thompson, J. D., D. G. Higgins and T. J. Gibson, "CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice", *Nucleic Acids Research*, Vol. 22, No. 22, pp. 4673–4680, 1994.

20. Notredame, C., D. G. Higgins and J. Heringa, "T-coffee: a novel method for fast and accurate multiple sequence alignment11Edited by J. Thornton", *Journal of Molecular Biology*, Vol. 302, No. 1, pp. 205 – 217, 2000.

21. Katoh, K., K. Misawa, K.-i. Kuma and T. Miyata, "MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform", *Nucleic acids research*, Vol. 30, No. 14, pp. 3059–3066, 2002.

22. Do, C. B., M. S. Mahabhashyam, M. Brudno and S. Batzoglou, "ProbCons: Probabilistic consistency-based multiple sequence alignment", *Genome research*, Vol. 15, No. 2, pp. 330–340, 2005.

23. Edgar, R. C. and S. Batzoglou, "Multiple sequence alignment", *Current Opinion in Structural Biology*, Vol. 16, No. 3, pp. 368 – 373, 2006, nucleic acids/Sequences and topology.

24. Holm, L. and C. Sander, "Dali: a network tool for protein structure comparison", *Trends in Biochemical Sciences*, Vol. 20, No. 11, pp. 478 – 480, 1995.

25. Holm, L. and P. Rosenstrom, "Dali server: conservation mapping in 3D", *Nucleic*

acids research, Vol. 38, No. suppl_2, pp. W545–W549, 2010.

26. Burkhardt, S. and J. Kärkkäinen, "One-gapped q-gram filters for Levenshtein distance", *Annual Symposium on Combinatorial Pattern Matching*, pp. 225–234, Springer, 2002.

27. Dinu, L. P. and A. Sgarro, "A low-complexity distance for DNA strings", *Fundamenta Informaticae*, Vol. 73, No. 3, pp. 361–372, 2006.

28. Greenhill, S. J., "Levenshtein Distances Fail to Identify Language Relationships Accurately", *Comput. Linguist.*, Vol. 37, No. 4, pp. 689–698, Dec. 2011.

29. Brown, P. F., P. V. deSouza, R. L. Mercer, V. J. D. Pietra and J. C. Lai, "Class-based N-gram Models of Natural Language", *Comput. Linguist.*, Vol. 18, No. 4, pp. 467–479, Dec. 1992.

30. Zissman, M. A. and E. Singer, "Automatic language identification of telephone speech messages using phoneme recognition and N-gram modeling", *Acoustics, Speech, and Signal Processing, 1994. ICASSP-94., 1994 IEEE International Conference on*, Vol. i, pp. I/305–I/308 vol.1, April 1994.

31. Cavnar, W. B. and J. M. Trenkle, "Ngram-based text categorization", *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, Citeseer, 1994.

32. Abou-Assaleh, T., N. Cercone, V. Keselj and R. Sweidan, "N-gram-based detection of new malicious code", *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.*, Vol. 2, pp. 41–42 vol.2, September 2004.

33. Kondrak, G., "N-gram similarity and distance", *String processing and information*

retrieval, pp. 115–126, Springer, 2005.

34. Houvardas, J. and E. Stamatatos, *N-Gram Feature Selection for Authorship Identification*, pp. 77–86, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

35. Tomović, A., P. Janičić and V. Kešelj, "n-Gram-based classification and unsupervised hierarchical clustering of genome sequences", *Computer Methods and Programs in Biomedicine*, Vol. 81, No. 2, pp. 137 – 153, 2006.

36. Beebe, N. L., L. A. Maddox, L. Liu and M. Sun, "Sceadan: Using Concatenated N-Gram Vectors for Improved File and Data Type Classification", *IEEE Transactions on Information Forensics and Security*, Vol. 8, No. 9, pp. 1519–1530, September 2013.

37. Millar, E., D. Shen, J. Liu and C. Nicholas, "Performance and Scalability of a Large-Scale N-gram Based Information Retrieval System", *Journal of Digital Information*, Vol. 1, No. 5, 2006.

38. Stormo, G. D., T. D. Schneider, L. Gold and A. Ehrenfeucht, "Use of the 'Perceptron'algorithm to distinguish translational initiation sites in E. coli", *Nucleic acids research*, Vol. 10, No. 9, pp. 2997–3011, 1982.

39. Rabiner, L. R., "A tutorial on hidden Markov models and selected applications in speech recognition", *Proceedings of the IEEE*, Vol. 77, No. 2, pp. 257–286, February 1989.

40. Zucchini, W., I. L. MacDonald and R. Langrock, *Hidden Markov models for time series: an introduction using R*, CRC press, 2017.

41. Annachhatre, C., T. H. Austin and M. Stamp, "Hidden Markov models for malware classification", *Journal of Computer Virology and Hacking Techniques*, Vol. 11,

No. 2, pp. 59–73, May 2015.

42. Eddy, S. R., "Profile hidden Markov models.", *Bioinformatics*, Vol. 14, No. 9, pp. 755–763, 1998.

43. Dehnert, M., W. Helm and M.-T. Hütt, "A discrete autoregressive process as a model for short-range correlations in DNA sequences", *Physica A: Statistical Mechanics and its Applications*, Vol. 327, No. 3, pp. 535 – 553, 2003.

44. Chakravarthy, N., A. Spanias, L. D. Iasemidis and K. Tsakalis, "Autoregressive Modeling and Feature Analysis of DNA Sequences", *EURASIP J. Appl. Signal Process.*, Vol. 2004, pp. 13–28, Jan. 2004.

45. Akhtar, M., E. Ambikairajah and J. Epps, "Comprehensive autoregressive modeling for classification of genomic sequences", *6th International Conference on Information, Communications Signal Processing*, pp. 1–5, December 2007.

46. Vinga, S. and J. Almeida, "Alignment-free sequence comparison—a review", *Bioinformatics*, Vol. 19, No. 4, pp. 513–523, 2003.

47. Wu, C., M. Berry, S. Shivakumar and J. McLarty, "Neural networks for full-scale protein sequence classification: Sequence encoding with singular value decomposition", *Machine Learning*, Vol. 21, No. 1, pp. 177–193, October 1995.

48. Huang, S.-H., R.-S. Liu, C.-Y. Chen, Y.-T. Chao and S.-Y. Chen, "Prediction of outer membrane proteins by support vector machines using combinations of gapped amino acid pair compositions", *Fifth IEEE Symposium on Bioinformatics and Bioengineering (BIBE'05)*, pp. 113–120, October 2005.

49. Breiman, L., *Classification and regression trees*, Routledge, 1984.

50. Breslow, L. A. and D. W. Aha, "Simplifying Decision Trees: A Survey", *Knowl. Eng. Rev.*, Vol. 12, No. 1, pp. 1–40, Jan. 1997.

51. Breiman, L., J. Friedman, R. Olshen and C. Stone, "Classification and Regression Trees (CART) Wadsworth", *Pacific Grove, CA*, 1984.

52. Quinlan, J. R., "C4. 5: Programming for machine learning", *Morgan Kauffmann*, Vol. 38, p. 48, 1993.

53. Breiman, L., "Random Forests", *Machine Learning*, Vol. 45, No. 1, pp. 5–32, October 2001.

54. Ren, S., X. Cao, Y. Wei and J. Sun, "Global refinement of random forest", *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 723–730, June 2015.

55. Biau, G. and E. Scornet, "A random forest guided tour", *TEST*, Vol. 25, No. 2, pp. 197–227, June 2016.

56. Jandhyala, V., E. Michielssen and R. Mittra, "FDTD signal extrapolation using the forward-backward autoregressive (AR) model", *IEEE Microwave and Guided Wave Letters*, Vol. 4, No. 6, pp. 163–165, June 1994.

57. Zhou, H. and H. Yan, "Autoregressive Models for Spectral Analysis of Short Tandem Repeats in DNA Sequences", *2006 IEEE International Conference on Systems, Man and Cybernetics*, Vol. 2, pp. 1286–1290, October 2006.

58. Rosen, G., "Comparison of Autoregressive Measures for DNA Sequence Similarity", *IEEE International Workshop on Genomic Signal Processing and Statistics*, pp. 1–4, June 2007.

59. Blinowska, K., B. Trzaskowski, M. Kaminski and R. Kus, "Multivariate autoregressive model for a study of phylogenetic diversity", *Gene*, Vol. 435, No. 1, pp. 104 – 118, 2009.

60. Choong, M. K., D. Levy and H. Yan, "Clustering of DNA microarray temporal data based on the autoregressive model", *IEEE International Conference on Systems, Man and Cybernetics*, pp. 71–75, October 2008.

61. Song, N. Y. and H. Yan, "Short Exon Detection in DNA Sequences Based on Multifeature Spectral Analysis", *EURASIP J. Adv. Signal Process*, Vol. 2011, pp. 2:1–2:8, January 2011.

62. Jaro, M. A., "Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida", *Journal of the American Statistical Association*, Vol. 84, No. 406, pp. 414–420, 1989.

63. Winkler, W. E., *The State of Record Linkage and Current Research Problems*, Tech. rep., Statistical Research Division, U.S. Census Bureau, 1999.

64. Stoilos, G., G. Stamou and S. Kollias, "A String Metric for Ontology Alignment", Y. Gil, E. Motta, V. R. Benjamins and M. A. Musen (Editors), *The Semantic Web – ISWC 2005*, pp. 624–637, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

65. Levenshtein, V. I., "Binary codes capable of correcting deletions, insertions, and reversals", *Soviet physics doklady*, Vol. 10, pp. 707–710, 1966.

66. Gusfield, D., *Algorithms on strings, trees and sequences: computer science and computational biology*, Cambridge university press, 1997.

67. Yang, J. and W. Wang, "CLUSEQ: efficient and effective sequence clustering", *Proceedings 19th International Conference on Data Engineering (Cat.*

No.03CH37405), pp. 101–112, March 2003.

68. van de Pol, F. and R. Langeheine, "Mixed Markov Latent Class Models", *Sociological Methodology*, Vol. 20, pp. 213–247, 1990.

69. Vermunt, J. K., B. Tran and J. Magidson, "Latent class models in longitudinal research", *Handbook of longitudinal research: Design, measurement, and analysis*, pp. 373–385, 2008.

70. Helske, S. and J. Helske, "Mixture hidden markov models for sequence data: the seqhmm package in R", *arXiv preprint arXiv:1704.00543*, 2017.

71. Ernst, W., H. E. van den and R. Jan-Willem, "'All models are wrong...': an introduction to model uncertainty", *Statistica Neerlandica*, Vol. 66, No. 3, pp. 217–236.

72. Cios, K. J., W. Pedrycz, R. W. Swiniarski and L. A. Kurgan, *Data mining: a knowledge discovery approach*, Springer Science & Business Media, 2007.

73. Han, J., J. Pei and M. Kamber, *Data mining: concepts and techniques*, Elsevier, 2011.

74. Taniar, D., *Data mining and knowledge discovery technologies*, IGI Global, 2008.

75. Rendón, E., I. Abundez, A. Arizmendi and E. M. Quiroz, "Internal versus external cluster validation indexes", *International Journal of computers and communications*, Vol. 5, No. 1, pp. 27–34, 2011.

76. Halkidi, M., Y. Batistakis and M. Vazirgiannis, "Cluster validity methods: part I", *ACM Sigmod Record*, Vol. 31, No. 2, pp. 40–45, 2002.

77. Aggarwal, C. C., *Data mining: the textbook*, Springer, 2015.

78. Everitt, B. S., S. Landau, M. Leese and D. Stahl, *Hierarchical Clustering. In Cluster Analysis*, Wiley-Blackwell, 2011.

79. Caliński, T. and J. Harabasz, "A dendrite method for cluster analysis", *Communications in Statistics*, Vol. 3, No. 1, pp. 1–27, 1974.

80. Rousseeuw, P. J., "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis", *Journal of Computational and Applied Mathematics*, Vol. 20, pp. 53 – 65, 1987.

81. Dunn†, J. C., "Well-Separated Clusters and Optimal Fuzzy Partitions", *Journal of Cybernetics*, Vol. 4, No. 1, pp. 95–104, 1974.

82. Sowmiya, N. and B. Valarmathi, "A review of categorical data clustering methodologies based on recent studies", *The IIOAB Journal*, Vol. 8, p. 362.

83. Rand, W. M., "Objective Criteria for the Evaluation of Clustering Methods", *Journal of the American Statistical Association*, Vol. 66, No. 336, pp. 846–850, 1971.

84. Hubert, L. and P. Arabie, "Comparing partitions", *Journal of Classification*, Vol. 2, No. 1, pp. 193–218, December 1985.

85. Aggarwal, C. C. and C. K. Reddy, *Data Clustering: Algorithms and Applications*, Chapman & Hall/CRC, 1st edn., 2013.

86. Rokach, L. and O. Maimon, *Clustering Methods*, p. 330, Springer US, Boston, MA, 2005.

87. Wang, X., A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann and E. Keogh, "Experimental comparison of representation methods and distance measures for time series data", *Data Mining and Knowledge Discovery*, Vol. 26, No. 2, pp. 275–

309, March 2013.

88. Baydogan, M. G. and G. Runger, "Time series representation and similarity based on local autopatterns", *Data Mining and Knowledge Discovery*, Vol. 30, No. 2, pp. 476–509, March 2016.

89. Keogh, E. and S. Kasetty, "On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration", *Data Mining and Knowledge Discovery*, Vol. 7, No. 4, pp. 349–371, October 2003.

90. Jr., J. H. W., "Hierarchical Grouping to Optimize an Objective Function", *Journal of the American Statistical Association*, Vol. 58, No. 301, pp. 236–244, 1963.

91. Punj, G. and D. W. Stewart, "Cluster Analysis in Marketing Research: Review and Suggestions for Application", *Journal of Marketing Research*, Vol. 20, No. 2, pp. 134–148, 1983.

92. Li, Y. *et al.*, "MSPKmerCounter: a fast and memory efficient approach for k-mer counting", *arXiv preprint arXiv:1505.06550*, 2015.

93. Edgar, R. C., "MUSCLE: a multiple sequence alignment method with reduced time and space complexity", *BMC bioinformatics*, Vol. 5, No. 1, p. 113, 2004.

94. Leonardo, B. and S. Hansun, "Text Documents Plagiarism Detection using Rabin-Karp and Jaro-Winkler Distance Algorithms", *Indonesian Journal of Electrical Engineering and Computer Science*, Vol. 5, No. 2, pp. 462–471, 2017.

95. Kim, S.-R. and K. Park, "A dynamic edit distance table", *Journal of Discrete Algorithms*, Vol. 2, No. 2, pp. 303 – 312, 2004, combinatiorial Pattern Matching.

96. Christen, P., "A comparison of personal name matching: Techniques and practi-

cal issues", *Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on*, pp. 290–294, IEEE, 2006.

97. Pamminger, C. and S. Frühwirth-Schnatter, "Model-based clustering of categorical time series", *Bayesian Analysis*, Vol. 5, No. 2, pp. 345–368, 06 2010.

98. Pamminger, C., *R Package bayesMCClust: Mixtures-of-Experts Markov Chain Clustering and Dirichlet Multinomial Clustering*, R Foundation for Statistical Computing, 2012.

99. García-Magariños, M. and J. A. Vilar, "A framework for dissimilarity-based partitioning clustering of categorical time series", *Data Mining and Knowledge Discovery*, Vol. 29, No. 2, pp. 466–502, March 2015.

APPENDIX A: GITHUB PAGE

The open source code repository for nTreeClus is available at GitHub `https://github.com/HadiJahanshahi/nTreeClus`. This is the Python implementation of the method and a simple example.

APPENDIX B: PARAMETER SETTING

Table B.1 shows the amount of the parameters for each dataset which is used in the thesis.

Table B.1: Parameters' Setting

Dataset	R	$\mathcal{N}$	$\mathcal{L}$	$\mathcal{M}$	a	$\mathcal{M}_p$	C
Parameter Tuning I	10	40	180	40	{7,20}	{8,15}	2
Parameter Tuning II	5	90	{180,450,720}	{17,40,65}	7	{7,15}	{2,5,8}
Simulation 1	10	1080	{40,120,200}	{20,40,90}	{4,6,10,20}	{6,10,15}	2
Simulation 2	10	1080	{40,120,200}	{20,40,90}	{4,6,10,20}	{(2,3),(4,6),(6,9)}	2
Simulation 3	10	360	{40,120,200}	[80,120]	{5,7,11,16}	{6,10,20}	2
Estimating # of clusters	3	216	{30,100}	{50,100}	{4,7,20}	{7,18}	{3,6,10}
Austrian Wage	-	1	9402	[2,32]	6	-	-
Protein Data	-	1	2112	[80,127]	20	-	2