Machine Learning Models for High-dimensional Biomedical Data

by

Sangdi Lin

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved April 2018 by the
Graduate Supervisory Committee:

George C. Runger, Chair
Adolfo R. Escobedo
Jean-Pierre A. Kocher
Rong Pan

ARIZONA STATE UNIVERSITY

August 2018

ABSTRACT

The recent technological advances enable the collection of various complex, heterogeneous and high-dimensional data in biomedical domains. The increasing availability of the high-dimensional biomedical data creates the needs of new machine learning models for effective data analysis and knowledge discovery. This dissertation introduces several unsupervised and supervised methods to help understand the data, discover the patterns and improve the decision making. All the proposed methods can generalize to other industrial fields.

The first topic of this dissertation focuses on the data clustering. Data clustering is often the first step for analyzing a dataset without the label information. Clustering high-dimensional data with mixed categorical and numeric attributes remains a challenging, yet important task. A clustering algorithm based on tree ensembles, CRAFTER, is proposed to tackle this task in a scalable manner.

The second part of this dissertation aims to develop data representation methods for genome sequencing data, a special type of high-dimensional data in the biomedical domain. The proposed data representation method, Bag-of-Segments, can summarize the key characteristics of the genome sequence into a small number of features with good interpretability.

The third part of this dissertation introduces an end-to-end deep neural network model, GCRNN, for time series classification with emphasis on both the accuracy and the interpretation. GCRNN contains a convolutional network component to extract high-level features, and a recurrent network component to enhance the modeling of the temporal characteristics. A feed-forward fully connected network with the sparse group lasso regularization is used to generate the final classification and provide good interpretability.

The last topic centers around the dimensionality reduction methods for time series data. A good dimensionality reduction method is important for the storage, decision making and pattern visualization for time series data. The CRNN autoencoder is proposed to not only achieve low reconstruction error, but also generate discriminative features. A variational version of this autoencoder has great potential for applications such as anomaly detection and process control.

I delicate this work to my mom who always believes in me and offered unconditional love and support over this journey.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. George Runger for opportunities to work with him and the tremendous support over my Ph.D. study. His passion to the scientific research and his collaborative work style inspires me, and I feel very lucky to be his student. I enjoyed every single discussion we have in the past three years, which will be missed the most after the graduation. I like to thank Dr. Jean-Pierre Kocher for the precious research opportunities that allows me to work on some very impactful health care projects with Mayo Clinic. I am very thankful for his patience in teaching me the fascinating biomedical insights about the projects and the generous funding support for my Ph.D. study. It has been my great pleasure to work with him. I like to thank our collaborator Dr. Chen Wang from Mayo Clinic for the detailed advice and support on our projects. I want to thank my committee members, Dr. Rong Pan and Dr. Adolfo Escobedo for the suggestions and advice given on this research and my Ph.D. study.

My fellow Ph.D. students and labmates, including Nathan Gaw, Ghazal Shams, Nooshin Shomal Zadeh, Maziar Kasaei Roodsari and many others, have been the greatest support and dearest friends for me over my Ph.D. study. My friends Jeffrey Djavadi, Leah Jorgenson, Ilana Hale and Sara Ranjbar from my volleyball group have been there for me through the ups and downs in this journey. For me, they are just like my family, and I like to express my deepest thanks to all of them.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

1.1 Overview

High-dimensional data is very common in the biomedical domain. For example, the profile of a patient can be considered as a high-dimensional data point with mixed attributes, as each patient can be described by their basic information, diagnosis history, medication history, test results and so on. Therefore, in a patient database, the profile of each patient is usually represented by a collection of both numeric and categorical features. Time series data is another special type of high-dimensional data frequently seen in biomedical domains. We collect various types of time series data for health monitoring and disease diagnosis. For example, electroencephalogram (EEG) has been used to monitor the heart condition, and electrocardiogram (ECG) has been to detect changes of the mental states [5] and the seizure onsets [6]. As the recent prevalence of the wearable devices, we are interested in integrating the time series data collected by the different sensors to make personalized health decisions. The attributes of time series data are in a specific order, and this property promotes some special approaches for effective analysis. As another example, with the recent advances of the genome sequencing technology, the research that investigates the relationship between the genome patterns and human diseases is receiving more and more attention. The genome sequencing data is another type of high-dimensional data, and can be considered as time series in a broader sense as their attributes are also in a meaningful order. In this dissertation, we discuss several supervised and unsupervised learning methods for analyzing and understanding high-dimensional

data. More specifically, we introduce a clustering method for high dimensionality and mixed attributes, a deep learning model for time series classification, a new data representation for genome sequencing data, and an autoencoder model for time series data. The proposed methods have important applications in not only the biomedical domains but also other industrial fields.

Clustering, as an unsupervised learning techniques, is usually the very first step for analyzing a dataset without label information. Clustering attempts to partition data such that similar data instances are in the same subset. After clustering data, useful patterns can be discovered by explaining and characterizing the partitions. As discussed earlier, many datasets in health care or biomedical domains are high-dimensional with mixed numeric and categorical attributes. However, clustering datasets with these characteristics in a scalable fashion still remains a challenging problem. First of all, the high dimensionality in many real-life datasets presents a specific challenge for clustering algorithms, especially those which are based on Euclidean distance. These spaces exhibit the "curse of dimensionality" where all objects are approximately equidistant [7] and the nearest neighbor problem can become ill-defined [8] [9]. Secondly, real data is usually noisy, containing some irrelevant or redundant information which hides the cluster patterns. The third challenge is that many algorithms that rely on the traditional distance measure are sensitive to different units of the attributes. Although data transformation [10] can be applied to alleviate the problem, this risks changing the distribution of data which might be an important clue for finding the clusters. Last but not least, it is not straightforward to measure the similarity of two instances when encountering mixed categorical and numeric attributes. In Chapter 2, we propose a tree-ensemble clustering algorithm, CRAFTER, to tackle these challenges. CRAFTER is able to handle mixed attributes simultaneously, and scales well with the dimensionality and the size of data

sets. We demonstrate the effectiveness of CRAFTER, through a series of experiments on both synthetic and real datasets including examples in the biomedical domain. In additional to the biomedical domain, CRAFTER can be useful to cluster data of different varieties in areas such as social network analysis, community detection or sensor integration. This work is published in [11].

Time series classification (TSC) has important applications across various domains. For example, in the biomedical domain, we are interested in the prediction of the seizure onsets using the EEG, the prediction of the heart attacks using the ECG, and the classification of the tumors based on the genome sequencing data. In manufacturing domain, the detection of a product defect usually relies on the classification of the profile time series signals collected by the sensors. TSC is a supervised learning problem that we train a classification model from a time series dataset where each time series is an instance associated with a class label. TSC is unique and challenging due to the high-dimensional and ordered attributes. The increase of publicly available temporal datasets (such as UCR time series archive [2]) promotes active research in TSC in the last decade. Researchers have tackled this problem through various techniques [12].

A good TSC model often relies on a discriminative data representation extracted from the massive number of attributes in the original sequence. In Chapter 3 and Chapter 4, we present two data representation methods for the time series data based on the case study of the tumor grade classification. In Chapter 3, driven by some biological insights, a data representation named as Bag-of-Segments (BoS) is proposed to extract predictive patterns from the whole genome sequencing dataset. Each instance in the dataset has over 25 thousand ordered attributes. When used for subsequent classification, our representation obtains over 98% average accuracy based on our leave-one-out cross-validation experiment. Our method specifically may be beneficial

in classifying ovarian serous carcinoma in patients with ambiguous histologic features. In such challenging cases, our prediction on low-grade versus high-grade carcinomas may provide valuable clue for patient clinical management. As an extension, in Chapter 4 we develop a location-aware supervised dictionary learning method (LSDF) to construct a stronger data representation for more complex classification tasks. The uterine corpus endometrial carcinoma (UCEC) data generated by the TCGA Research Network [4] is used as the case study for LSDF. The TCGA data is a microarray-based sequencing data, and it is nosier and more difficult to analyze compared to the low-coverage sequencing data we used in Chapter 3. Both methods can generalize to the classification of other time series which share similar characteristics.

Recently, deep learning techniques, particularly a Convolutional Neural Network (CNN) (originally developed by [13]) and a Recurrent Neural Network (RNN) [14, 15], have received renewed interest in areas such as computer vision and natural language processing. One important advantage of a deep learning model is that it often can process the raw data as the input without the exhausting feature engineering. However, the disadvantages include the relatively large dataset required for the model training and the difficult interpretation of the models. To advance the modeling for time series, in Chapter 5, we consider the applications of CNNs or RNNs towards TSC. Unlike the methods proposed in Chapter 3 and 4 where the feature extraction and classifier training are in two separate steps, the neural network models learn the features during the training. As an overview, the proposed model contains a convolutional network component to extract high-level features, and a recurrent network component to enhance the modeling of the temporal characteristics of time series data. In addition, a feed-forward fully connected network with the sparse group lasso regularization is used to generate the final classification. The proposed architecture not only achieves satisfying classification accuracy, but also obtained good

interpretability through the sparse group lasso regularization. All these networks are connected and jointly trained in an end-to-end framework, and it can be generally applied to time series classification tasks across different domains without the efforts of feature engineering. Our experiments in various time series datasets show that the proposed model outperforms the traditional convolutional neural network model for the classification accuracy, and also demonstrate how the sparse group lasso contributes to a better model interpretation. As a case study in the biomedical domain, we apply our model for the classification of a publicly available EEG seizure dataset. This work is published in [16].

As huge volume of the time series data is collected from the wearable devices and sensors at real time, we are facing challenges in the storage, analysis, and visualization of time series data. An effective dimension reduction method for time series data is a crucial step towards solving these challenges. Dimensionality reduction is an unsupervised learning problem. The goal of dimensionality reduction is to find a low-dimensional representation of the data which preserves the most important characteristics of the data and also minimizes the information loss. Dimensionality reduction, at the same time, identifies the underlying sources of variation patterns in the data, which can contribute to our understanding on the data generation mechanism and feature extraction. Because of the temporal characteristics of time series data (such as autocorrelation), they typically contain high redundancy and can be represented by a much smaller number of features than the original dimensionality. In addition, these temporal characteristics often contribute to the key features which can be used in clustering, classification and anomaly detection of time series data. In Chapter 6, we introduce a deep learning-based dimensionality reduction model, CRNN autoencoder, for time series data. Our discussion focuses on the ability of both reconstruction and generalization of the latent representation.

In general, the best parametrization for deep learning models are not very well understood, and the tuning of the parameter settings (such as the number of certain layers, filter size) are often based on trial and error. This dissertation selects some basic architectures of proposed deep learning models and shows good results. Potentially the results can be improved with fine-tuned parameter settings. However, the important properties we discovered regarding the proposed deep learning architectures are expected to generalize to other, related architectures.

1.2 Datasets and Applications

In this dissertation, we discuss machine learning methods (clustering, dimensionality reduction, classification) which can be applied to many biomedical problems and generalize to other industrial fields. To demonstrate the effectiveness of our methods, we tested our methods on both real and synthetic datasets. The real datasets include benchmark datasets from the UCI machine learning repository [1] and the UEA & UCR time series repository [2], as well as some more domain-specific datasets, such as the genome dataset from TCGA research network [4], the seizure EEG datasets from Bonn University [3] and the low-coverage whole genome sequencing data generated by our collaborators at Mayo Clinic [17]. In the following, we briefly introduce each dataset (or data source) and how they are used to support the claims in this dissertation. These datasets are good examples for the future applications of our methods.

1.2.1 UCI Machine Learning Repository [1]

The UCI machine learning repository is a collection of datasets that is frequently used by machine learning community for empirical comparisons of machine learning algorithms [1]. We compare Crafter to several other clustering methods on datasets

from this repository, which includes two biomedical datasets (Diagnostic Breast Cancer Dataset and Mice Protein Expression Dataset) and six datasets from various other domains. To evaluate the performance of different clustering algorithm, we use the known class labels of each dataset as the ground truth which are not used during the data clustering, and measure the concordance between the class labels and the found clusters.

1.2.2 UEA & UCR Time Series Classification Repository [2]

The repository provides a comprehensive collection of time series data for researchers to analyze and compare time series algorithms [2]. In Chapter 5, we compare different variants of GCRNN to the CNN and other time series classifiers over 14 datasets from this archive. In Chapter 6, to compare CRNN autoencoder to other dimensionality reduction methods, we focus on 4 datasets (including 3 ECG datasets and a synthetic control dataset) from the repository in our experiments.

1.2.3 Bonn EEG Seizure Dataset [3]

This is a publicly available seizure dataset provided by Bonn University [3]. The dataset consists of five subsets (set Z, O, N, F and S), and each has 100 single-channel EEG segment. Set S contains EEG segments during the seizure activity. Set Z and Set O are the EEG segments collected when the awake healthy volunteers with their eyes open and closed, respectively. Set N and set F contain the non-seizure segments from the seizure patients, recorded from two different brain regions. This dataset is used to form different classification tasks to compare GCRNN with other existing methods in literature in Chapter 5.

1.2.4 The Cancer Genome Atlas Data [4]

The Cancer Genome Atlas (TCGA) is a research collaboration that makes large number of genomic data of various types of cancers publicly available [4]. It helps the cancer researcher community to advance the diagnosis, treatment and prevention technologies of the cancers. In Chapter 4, the uterine corpus endometrial carcinoma (UCEC) data is used to form a binary classification problem (high grade vs. low grade) to illustrate that the supervised dictionary learning improves the prediction power of the representation in a complex classification problem.

1.2.5 Low Coverage Whole Genome Sequencing (LC-WGS) Data

Defined as somatic gain or loss of DNA regions, Copy Number Alterations (CNAs) are reflective of genomic instability, frequently affecting functionally important genes, such as tumor suppressors and oncogenes. The profiles of CNAs may provide a fingerprint specific to a tumor type or tumor class [18]. The availability of Next Generation Sequencing (NGS) technology platforms has enabled the study of CNAs at a genome wide scale and at an unprecedented level of resolution. Numerous methods are available to report CNAs from high-coverage whole genome sequencing and for low-coverage sequencing (LC-WGS). LC-WGS has recently gained interest since successfully translated into clinical applications. In Chapter 5, our study focuses on 34 LC-WGS data samples labeled into the low grade or the high grade. The samples were pre-processed with Wandy [17]. Wandy accumulates the sparse sequence reads into 10,000 base long bins and performs several noise reduction procedures to more accurately characterize changes in coverage characteristic of Copy Number Variations (CNVs). As a result, we have around 25 thousand points for each sample.

1.2.6 Other Domains

In additional to the biomedical domain, the machine learning methods developed in this dissertation can be applied to many other industrial fields. For instance, a scalable clustering algorithm like CRAFTER is important for areas such as social network analysis, community detection and recommendation system. GCRNN, as a time series classifier, can also be applied to the decision making about the financial time series data or control signals. The Bag-of-Segments representation can be used to represent other long time series data with small sample size. The time series autoencoder can be applied to process monitoring and defect detection in the manufacturing domain. As a toy example, we perform experiments to show that CRNN autoencoder can effectively identify abnormal from normal events based on a synthetic control dataset from the UCR repository. Discovering new domains or applications for the proposed methods will be an on-going research effort in the future.

1.3 Background

To make the dissertation more self-contained, in this section, background for tree-based ensembles, deep neural networks, and Bag-of-Features methods is provided.

1.3.1 Tree-based Ensembles

Ensemble methods are a type of supervised methods that combine the predictions from many weak base learners to present a stronger model. The condition for an ensemble model to outperform their individual members is the individual members are accurate (better than random guessing) and diverse [19]. Two models are diverse when their errors on new data points are uncorrelated. Therefore, we often make an ensemble of the base learners such as decision trees which has high variance.

To make the models less correlated, we can manipulate the training process of each base learner in several ways. The first way by sampling the training data. For example, *Bagging* is an ensemble method where each base learner is trained on a bootstrap sample of the original data. Adaboost [20] is another ensemble model which trains each base learner on a weighted sample of the data to focus on the misclassified instances. Another kind of manipulation is through modifying the objective function of the learning problem. For instance, a gradient boosting machine [21] with L^2 loss fits each based learner to model the current residuals iteratively.

Random Forest (RF) [22] is a bagging-style ensemble injected with more randomness, which is used in several methodologies in this dissertation. CRAFTER is developed based on both supervised and unsupervised RF models. A RF model is also used as the classification model in Chapters 3 and 4, and as an important component of the data representation method in Chapter 4. In the following we discuss the supervised RF and unsupervised RF models in more details.

A supervised RF is a ensemble of decision or regression trees [23]. A tree model splits the data based on the value of one feature at each node until a stopping criterion is met. Each tree in the ensemble is trained on a bootstrap sample of the dataset. Additional randomness is injected by randomly selected a subset of the features for evaluation at each node split during the training of each base learner. The final classification or prediction is made based on the majority of the votes or the average prediction made by its tree members.

Although a RF is a supervised learning model, it can be adapted for unsupervised problems. In this case it is used to measure the dissimilarity between each pair of instances in the dataset and provides input for a clustering algorithm. We need to first convert the unsupervised learning problem to a supervised setting. To this end, all the instances in the original dataset are labeled with class "0", and a synthetic

dataset of the same size is created and labeled with class "1". Each instance in the synthetic set is randomly drawn from the product distribution of the marginal attribute distributions from the original set. In other words, each attribute of the synthetic set follows the same marginal distribution as the corresponding one in the original set, but the dependencies among the attributes are broken through random permutations [24]. A RF classifier is trained to distinguish these two classes. The proximity between i-th and j-th instances of the original dataset, denoted as $PROX_{ij}$, is estimated by the proportion of trees in which both instances fall into the same leaf node. The RF dissimilarity between these two instances is defined as $\sqrt{1 - PROX_{ij}}$.

1.3.2 Deep Neural Networks

The development of Chapter 5 and Chapter 6 are based on the existing deep learning models. In the following we give a brief introduction on the neural network models including artificial neural networks (ANNs), convolutional neural networks (CNNs), the basic recurrent neural networks (RNNs), long-short term memory (LSTM) and gated recurrent unit (GRU). In the end of this section, an overview on autoencoder models is also provided.

ANN

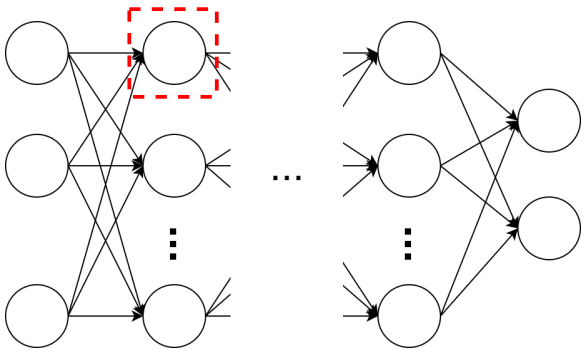


Figure 1.1: ANN Architecture

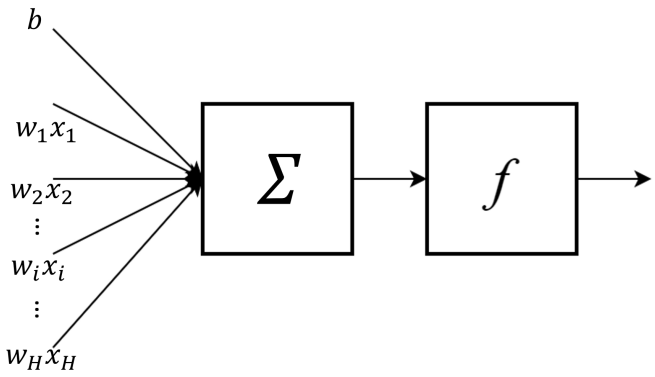


Figure 1.2: Perceptron Structure

An ANN, also known as Multilayer Perception (MLP) can be represented by the architecture in Fig. 1.1. The relationship between the input and the output of each node (e.g. the red box in Fig. 1.1) is described by Fig. 1.2. Suppose the input for a node is $x_1, x_2, ..., x_h$, the output of the node can be expressed as $o = f(\sum_{i=1}^{H} x_i w_i + b)$. Here f is also known as an activation function, for which we often use a non-linear function such as the logistic sigmoid function, the hyperbolic tangent function or the rectified linear unit function (ReLU) [25]. The activation in the hierarchical structure makes the neural network flexible to estimate the complex non-linear relationship. Because the nodes between the adjacent layers in ANN are fully connected, the number of parameters can be extremely large and the network may be difficult to train as the network grows deeper.

CNN

CNNs [13] use the local connectivity and the weight sharing to reduce the number of parameters in a deep neural network. Such structure is called a convolutional layer. Furthermore, CNNs use the max-pooling layer to reduce the number of connections/parameters. Fig. 1.3 provides a mini example for CNN, which is composed of two convolutional layers (layer 1 and layer 3) and one max-pooling layer (layer 2).

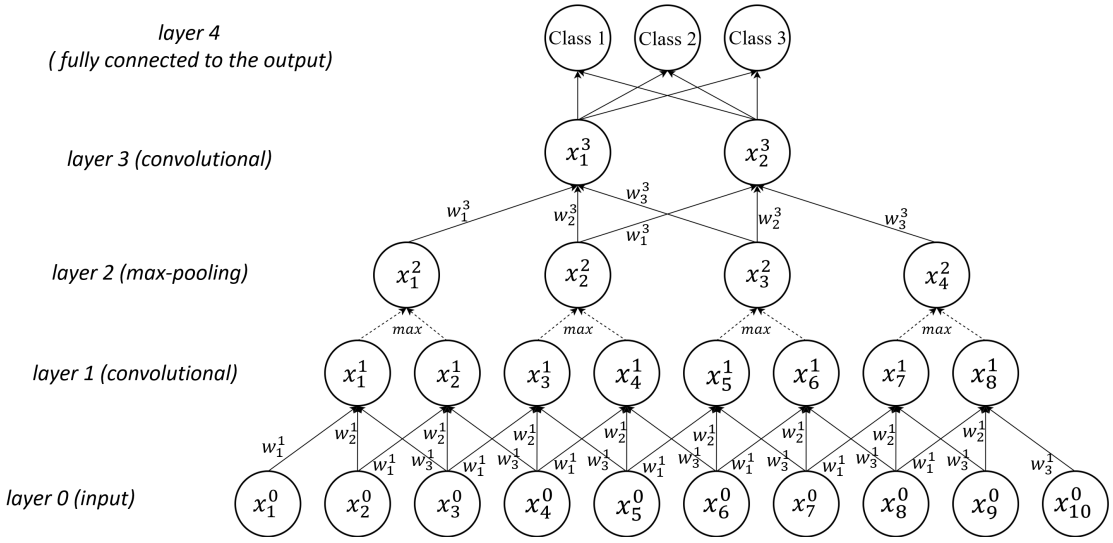


Figure 1.3: A CNN Example: The network is composed of 2 convolutional layers, 1 max-pooling layer and 1 fully connected layer. The network takes length-10 time series as the input.

Local connectivity means that each node in layer l is only connected to a subset of the nodes in layer $l-1$. With the weight sharing, each node at the same layer uses the same set of weights in the local connection. The relationship between the two adjacent layers becomes a convolution operation whose filter is the shared parameters $\boldsymbol{w^l} = [w_1^l, ..., w_D^l]$ learned from the data. For instance, node i in the layer l takes the subset of outputs from the previous layer, $x_{i+d-1}^{l-1}, d = 1, ..., D$, as its input. The convolution operation can be expressed as

$$y_i^l = \sum_{d=1}^{D} w_d^l x_{i+d-1}^{l-1}. \tag{1.1}$$

An activation function is then applied for the nonlinearity $x_i^l = f(y_i^l)$.

A max-pooling reduces the number of units by a factor of k by extracting the maximum ouput of each k non-overlapping nodes in the previous layer. A CNN network usually consists of alternating convolutional layer and max-pooling layer at the beginning, and a few fully connected layers to generate the final output.

RNN

ANN and CNN assume that the input variables are independent of each other, but in reality, many data are sequential. RNNs are able to model the nature of sequence [14, 15], therefore they are incorporated in developing the time series models in Chapter 5 and Chapter 6. A typical RNN looks like Fig. 1.4: on the left it is a compact representation of a recurrent layer, and it can be unrolled to the full network on the right that takes sequence data $\boldsymbol{x} = [x_1, x_2, ..., x_T]$ as the input. The "memory" of the network at time step t, denoted as s_t, is calculated based on the previous "memory" s_{t-1} and the input at step t: $s_t = f(Ux_t + Ws_{t-1})$. The function f is for the nonlinearity. The output at step t, o_t, is calculated as $o_t = Vs_t$. Note that U, V, W and V are shared across all nodes from 1 to T. Depending on the application, we can use the outputs at all time steps (e.g. the GCRNN in Chapter 5) or use only the output at the last time step T (e.g. the CRNN autoencoder model in Chapter 6).

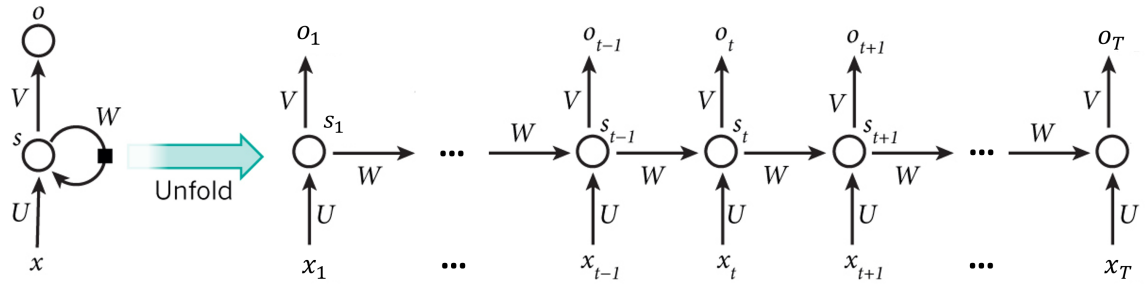


Figure 1.4: A Basic RNN Architecture for Input Time Series $\boldsymbol{x} = [x_1, x_2, ..., x_T]$. Modified from the figure in [26]

RNNs are hard to train as they suffer the vanishing and exploding gradient problem due to their deep architecture in the time axis [27]. As the result, the ability of RNNs to learn long-term dependency is degraded. Long short term memories (LSTMs) [28] and gated recurrent units (GRUs) [29] use a special structure called a gate to regulate the adding and removing information to the "memory" of the network. A LSTM unit uses 3 gates while a GRU uses 2 gates. For simplicity, we can

consider that using these two special unit structures is essentially replacing each node in Fig. 1.4 with a more complex module, but the input/output structure ("the big picture") remains very similar. The next subsection introduces LSTM and GRU with more details.

LSTM and GRU

The key idea of the LSTM is a memory cell that maintains its state over time, and gating units that regulate the incoming and outgoing information flow of the cell. The LSTM structure most commonly used in literature has incorporated the changes introduced by [30] and [31] into the original LSTM architecture in [28]. More specifically, the first major change is the forget gate introduced by [30]. It enables the LSTM to reset its state. The second major change is the "peephole connections" (connections from the internal cell state to the gates) introduced by [31]. These connections allow all the gates to inspect the cell states even when the output gate is closed. The resulted LSTM structure after these major changes can be presented in Fig. 1.5.

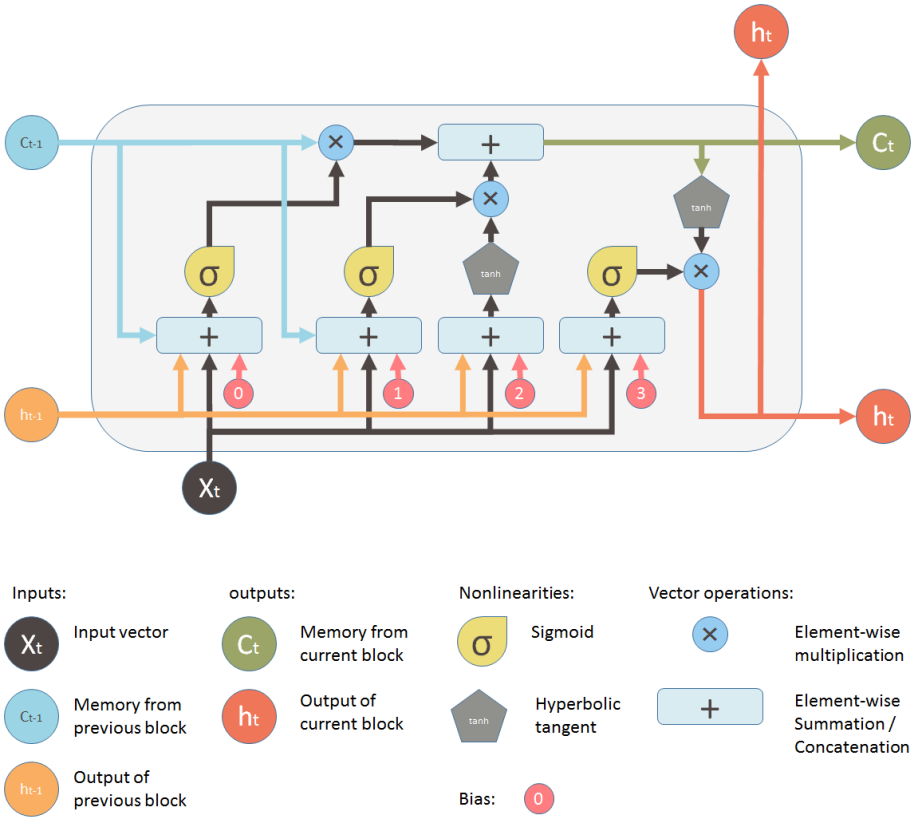


Figure 1.5: The Most Common LSTM Structure (figure from [32])

There are three gates in the most common LSTM structure, and each gate is a function of the cell state c_{t-1} or c_t, the current input x_t and the previous output y_{t-1}:

$$\text{input gate: } i_t = \sigma(W_i x_t + R_i h_{t-1} + p_i \odot c_{t-1} + b_i) \tag{1.2}$$

$$\text{forget gate: } f_t = \sigma(W_f x_t + R_f h_{t-1} + p_f \odot c_{t-1} + b_f) \tag{1.3}$$

$$\text{output gate: } o_t = \sigma(W_o x_t + R_o h_{t-1} + p_o \odot c_t + b_o) \tag{1.4}$$

The block input, cell state update and block output can be described by the following equations:

$$\text{block input: } z_t = \tanh(W_z x_t + R_z h_{t-1} + b_z) \tag{1.5}$$

$$\text{cell update: } c_t = i_t \odot z_t + f_t \odot c_{t-1} \tag{1.6}$$

$$\text{block output: } o_t \odot \tanh(c_t) \tag{1.7}$$

The $\odot$ operation stands for the element-wise multiplication.

Similar to LSTM, gated recurrent unit (GRU) proposed by [29] uses gating units to modulate the information flow into and out of the unit. GRU uses two gates, an update gate and a reset, both a function of the current input x_t and the previous output h_{t-1}:

$$\text{update gate: } z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{1.8}$$

$$\text{reset gate: } r_t = \sigma(W_r x_r + U_r h_{t-1} + b_r) \tag{1.9}$$

GRU first calculates a candidate output h_t^c. The final output is a linear interpolation between the previous output h_{t-1} and the new computed candidate h_t^c based on the update gate z_t. When computing the new candidate, the reset gate r_t determines how much previous information to combine. This update mechanism can be illustrated as follows.

$$\text{candidate: } h_t^c = \tanh(W_{hx} x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \tag{1.10}$$

$$\text{output: } h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t^c \tag{1.11}$$

Autoencoders

Autoencoder usually refers to a neural network model that performs an identity mapping. It forces the data to pass through a bottleneck layer with a much fewer number

of nodes than the original dimensionality of the data. The target of the network is required to be identical as the input so that the network is trained to minimize the reconstruction error. A traditional autoencoder is based on an ANN (or MLP). Fig. 1.6 depicts an ANN autoencoder with 3 hidden layers including a bottleneck layer with 2 nodes.

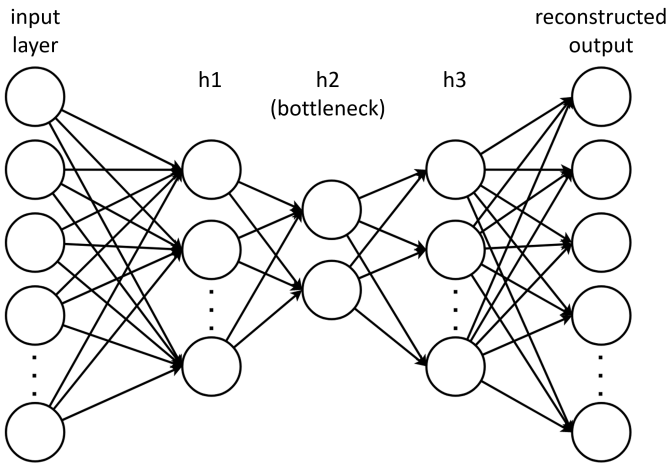


Figure 1.6: An ANN Autoencoder Example with 3 Hidden Layers (h1, h2 and h3). The Bottleneck Layer (h2 Layer) Has 2 Nodes.

Besides ANN, other neural network architectures have also been utilized in autoencoders for different applications. CNN is able to model the local relationship of the input by the convolutional operation and the hierarchy of the features by the pooling layers. CNN-based autoencoders have been used in image denoising [33] and image restoration [34]. Given the strength of RNNs in sequential modeling, the RNN-based autoencoder has been used in machine translation [35]. In Chapter 6, we propose a CRNN autoencoder which leverages the advantages of both CNNs and RNNs. The advantage of this new architecture in feature learning for time series is discussed.

1.3.3 Bag-of-Features Methods

Deep learning models trained on large datasets achieve the state-of-the-art performance across many domains. However, Bag-of-Features has proved its effectiveness in applications such as image and audio classification, particularly for more moderate sized datasets. Bag-of-Features is a group of feature representation methods. They usually consist of two steps: local feature extraction and dictionary construction. For images, the local features could be texture features such as HoG (short for Histograms of Gradients) [36] and SIFT (short for Scale-Invariant Feature Transform) [37]. For time series, they could be local statistics such as mean, slop, and standard deviation [38]. The dictionary can be constructed using either a supervised method [39] or an unsupervised method (e.g. clustering). In general, the dictionary is constructed by partitioning the extracted local features into different groups, and each group represents a "word" in the dictionary. Bag-of-Features representation is the frequency distribution over these groups, which can be used to train the classification models. Nowadays Bag-of-Features still plays an important role when the training data is limited or in a new domain. In addition, the domain knowledge can be easily incorporated into the framework based on the data or the application (as demonstrated in Chapter 3), and the resulted representation is often more interpretable compared to deep learning models.

Chapter 2

CRAFTER: A TREE-ENSEMBLE CLUSTERING ALGORITHM FOR MIXED ATTRIBUTES AND HIGH DIMENSIONALITY

2.1 Introduction

Clustering, as an unsupervised learning technique, attempts to partition data such that similar data instances are in the same subset. After clustering data, useful patterns can be discovered by explaining and characterizing the partition. Data clustering is usually the first step for analyzing an unlabeled dataset. There are still many remaining challenges in data clustering.

First of all, the high dimensionality in many real-life datasets presents a specific challenge for clustering algorithms, especially those which are based on Euclidean distance. These spaces exhibit the "curse of dimensionality" where all objects are approximately equidistant [7] and the nearest neighbor problem can become ill-defined [8] [9]. Secondly, data is usually noisy, containing some irrelevant or redundant information which hides the cluster patterns. The third challenge is that many algorithms that rely on the traditional distance measure are sensitive to different units of the attributes. Although data transformation [10] can be applied to alleviate the problem, this risks changing the distribution of data which might be an important clue for finding the clusters. Last but not least, clustering of categorical attributes remains an important task. Many applications deal with categorical or mixed data. For example, in a user profile database for an online application, each user is described by the categorical attributes such as region and hobby, as well as the numeric attributes such as daily online time.

Here we propose a clustering algorithm which addresses the challenges from a different start. Our algorithm begins with a sample clustering, and then it converts the unsupervised problem to a supervised learning setting under an iterative mechanism. More specifically, a Random Forest (RF) algorithm is iteratively applied to select representative instances to guide the overall clustering. We named our algorithm **CRAFTER**: **C**lustering algorithm using **RA**ndom **F**orest-based i**TER**ations. CRAFTER has many attractive properties. First of all, as a tree-ensemble method, it inherits the flexibility to handle mixed data and high dimensionality. It is also invariant to the units of numeric attributes. In addition, CRAFTER is less sensitive to irrelevant attributes, noise and outliers, and it scales well with the size of datasets.

Compared to Random Forest Clustering (RFC) [24], an existing tree-based approach and our major competitor, our advantage lies in both the scalability and the clustering quality. A limitation of RFC is that it only provides distances that then need to be clustered through a second algorithm. This second algorithm is restricted to those that can work only from the pairwise distance and is usually computationally expensive. Also, for large datasets, the size of the distance matrix and the computational demand increase dramatically. CRAFTER avoids both the distance matrix computations for the entire dataset and the second clustering algorithm with a scalable alternative. Our experiments show that CRAFTER performs equally good or better than RFC with a much lower computational complexity.

The remainder of this chapter is organized as follows. Section 2.2 reviews some related clustering methods in literature. Section 2.3 introduces the details of CRAFTER. In Section 5.4 and Section 2.5, an exhaustive set of experiments and comparisons on synthetic and real datasets is presented to demonstrate the effectiveness of CRAFTER. Section 2.6 discusses the limitations of our work and points out some future research. Finally, Section 5.7 provides conclusions.

2.2 Related Methods

Clustering algorithms can be divided into the four major categories: partitioning, hierarchical, density-based and grid-based [40]. In each category, efforts have been made to handle mixed attributes or high dimensionality.

K-means [41] is one of the most well-known partitioning algorithms that uses the coordinates of instances as its input. K-medoids [42] is more robust to outliers and noise as compared to K-means, because it chooses representative instances as the basis of clusters instead of means. K-medoid can work with an arbitrary-defined distance measure, which contributes to its role in RFC as discussed later. K-modes and K-prototypes [43] fall into the same category as extensions of K-means in categorical and mixed-attribute domains. K-modes algorithm uses a simple matching dissimilarity measure to handle categorical attributes. It replaces the means with the modes for each cluster, and the modes are updated using a frequency-based method. To cluster mixed numeric and categorical data, K-prototypes algorithm [43] uses a linear combination of two dissimilarity measures to integrate K-means and K-modes algorithms. More specifically, the component of numeric attributes is measured by Euclidean distance, while the categorical component is measured by simple matching. A suitable group of weights need to be specified to achieve a good balance between these two components.

Hierarchical clustering has two branches: agglomerative method and divisive method. Agglomerative methods are more commonly used. Clusters at a lower level merge into a larger cluster at a higher level based on a proximity measure. Hierarchical methods generate a dendrogram for the data, such that clusters in different levels can be extracted without the predetermined cluster number. In the general case, the computational complexity for agglomerative clustering is of $\mathcal{O}(N^3)$, which makes it

prohibitive for large datasets. For numeric attributes, Euclidean distance-based measure can be used to define the proximity. ROCK [44] extends agglomerative clustering to the categorical domain. It defines links between two instances based on the number of neighbors they share, and favors merging clusters/instances with more links. A pair of instances become neighbors if their similarity exceeds a certain threshold. However, ROCK doesn't scale well with the size of datasets due to its complexity of $\mathcal{O}(N^2 \, log \, N)$ [44]. To handle large datasets, a sampling approach needs to be adopted, which might leave out some important patterns. Moreover, the parameter for defining neighbors needs to be specified, which is not trivial.

In order to handle the high dimensionality, a group of subspace clustering approaches integrates feature selection into the clustering process [45]. For example, CLIQUE [46] is a grid-based algorithm for finding density-based clusters in subspaces with an APRIORI style approach. However, the quality of the clustering is greatly affected by the parameters such as the density threshold, and it doesn't scale well with the dimensionality of the output clusters [45]. From a different aspect, some research investigates good centrality measures for high dimensionality. For instance, Hubness [47], defined by the number of times a data point appears to one of the K nearest neighbors of other points, is shown to be a good local centrality measure in high-dimensional space. Based on hubness, several K-means-like algorithms in a deterministic or probabilistic manner were developed for high-dimensional data [48]. In this chapter, we utilize a tree-ensemble to tackle high dimensionality as the feature selection is embedded within the learning algorithm of the tree models.

Random Forest Clustering (RFC) [24], which inherits many desirable properties as a tree-ensemble method, is the most relevant method to our work. RFC is able to handle datasets with a large number of attributes of disparate types and different scales. First, an unsupervised Random Forest (RF) generates a unique dissimilarity

measure (RF distance), and then this distance is clustered through a method such as K-medoids clustering. RF distance is obtained by transforming the clustering problem (an unsupervised problem) to a supervised learning problem. To this end, all the instances in the original dataset are labeled with class "0", and a synthetic dataset of the same size is created and labeled with class "1". Each instance in the synthetic set is randomly drawn from the product distribution of the marginal attribute distributions from the original set. In other words, each attribute of the synthetic set follows the same marginal distribution as the corresponding one in the original set, but the dependencies among the attributes are broken through random permutations [24]. A RF classifier is trained to distinguish these two classes. The proximity between i-th and j-th instances of the original dataset, denoted as $PROX_{ij}$, is estimated by the proportion of trees in which both instances fall into the same leaf node. The RF distance between these two instances is defined as $\sqrt{1 - PROX_{ij}}$. RFC has proved its effectiveness in many applications, such as genomic sequence data clustering [49], tumor marker data clustering [50] and online user segmentation [51]. However, additional work is required to handle the distance matrix in large-instance applications, and, for the second step of RFC, a high computational cost is required for the K-medoids algorithm.

Partitioning Around Medoids (PAM) [42] is one of the best-known realizations of K-medoids algorithm. PAM begins with an arbitrary selection of K medoids from the dataset, and then it keeps swapping a non-medoid instance with a medoid which results in the greatest decrease in the cost function. This objective at each swap is to minimize

$$E = \sum_{i=1}^{K} \sum_{d \in C_i} distance(d, o_i) \tag{2.1}$$

where d represents a non-medoid instance in cluster C_i, and o_i is the medoid of this cluster.

The complexity for each iteration of PAM is of $\mathcal{O}(K(N-K)^2)$. To reduce the complexity, Clustering LARge Applications (CLARA) [42] utilizes a sampling-based scheme. CLARA draws multiple samples from the dataset, and applies PAM to each of them. The best K medoids giving the lowest cost function (Eq. (2.1)) evaluated at the whole dataset are returned. By reducing the complexity to $\mathcal{O}(KS^2+K(N-K))$ with sample size S for each iteration, CLARA makes K-medoids scheme applicable to large datasets, while the quality of the clusters can be compromised. Other than utilizing the sampling scheme, a K-medoids algorithm called CLATIN [52] utilizes the Triangular Irregular Network concept to reduce the computational time during the swap step of PAM. A K-means-like heuristic [53] was also proposed to update the medoids in a faster fashion. Similar to CLARA, these heuristics are not guaranteed to find the optimal K medoids.

2.3 CRAFTER Algorithm

CRAFTER begins with clustering a small sample of the dataset. In order to handle a wide variety of problems, a standard RFC is applied for clustering the small sample. Next, the clustering problem is converted to a supervised problem, with the selected sample as the training set and their initial cluster labels as the target attribute. A RF model is built to assign cluster labels to the rest of the dataset. The algorithm continues modifying the clusters in an iterative fashion, until either the clusters converge or a maximum iteration step is reached. More details about the initialization and main step of CRAFTER are presented in the following subsections.

2.3.1 Initialization

From a dataset D of size N, first a random sample of size n is drawn, and then RFC (unsupervised RF followed by PAM) is applied to this sample to generate K

initial clusters, where K is prespecified. When N is large, usually we have $n \ll N$, so that the computation for clustering the sample is negligible. With a small n, this process can be repeated M times to search for a good start at a low computational cost. Each sample and its clusters are evaluated by the PAM cost function (Eq. (2.1)), and the ones with the lowest cost function are chosen to initialize CRAFTER. Algorithm 1 summarizes the initialization step.

Other than RFC, there are several alternatives for generating clusters from the initial sample. For example, with RF distance, we can simply choose K instances far away from each other as the cluster medoids, and assign the other instances to their closest medoids. Besides, K-means can be used to cluster the samples for numeric datasets, and so can K-modes for categorical datasets.

Algorithm 1 RFC-based Initialization

procedure INITIALIZATION(D, K)
  for $m \leftarrow 1, M$ **do**
    Draw sample data $S_m \in D$
    $\{c_m, E_m\} \leftarrow$ RFC(S_m) ▷ E_m and c_m record the cost (Eq. (2.1)) and the clustering labels
    c_m and E_m are saved
  end for
  $T^{(0)}$ and $c^{(0)}$ are the sample and its labels corresponding to the minimum E_m
  return $T^{(0)}, c^{(0)}$
end procedure

2.3.2 Main Step

At this step, the clustering problem is transformed to a supervised learning problem. Consider the selected sample as the training set and the initial cluster labels as the target attribute. A RF classifier is trained on this sample, and used to assign a cluster label to the whole dataset (including the initial sample). The RF also provides

class probability estimates for each instance in the dataset, which is the proportion of the votes from the trees for assigning the cluster labels. For the training data, the votes are only based on the classification results from the trees where the instance is out-of-bag, and for the remainder of the dataset, the votes are based on the decisions made by all the trees. The class probability estimates provide the confidence of the cluster label assigned to each instance.

The *margin* is the difference between the maximum and the second greatest class probability estimate for each instance. An instance having a high margin is more likely to be labeled correctly by the RF. On the contrary, a low margin implies uncertainty regarding the cluster label of the instance. Therefore, CRAFTER takes advantage of those instances with high certainty in their cluster labels. That is, in each iteration, CRAFTER forms a new training set with the instances having high margin, trains a new RF model, and relabels the whole dataset including the training set. Although the initialization from a small sample may not be accurate, CRAFTER has the ability to revise the clustering iteration by iteration, by including the most representative instances into its training set and excluding the ones potentially being mislabeled.

More specifically, in the r-th iteration, we have a training set $T^{(r)}$ of size $n^{(r)}$ and the corresponding cluster labels $c^{(r)}$ as the target attribute. A RF classifier $RF^{(r)}$ is trained on $T^{(r)}$ and $c^{(r)}$. In the very first iteration, denoted by iteration 0, $T^{(0)}$ is the sample that provides the best initial clustering, and $c^{(0)}$ represents the initial clusters assigned by RFC. During the model training, the out-of-bag (OOB) class probability estimates for the training set, denoted by $P^{(r)} = \{p_{jk}^{(r)} | j = 1, ..., n^{(r)}, k = 1, ..., K\}$, are obtained, where $p_{jk}^{(r)}$ is the probability estimate that the j-th instance in the training set $T^{(r)}$ belongs to the k-th cluster. The cluster label of every instance in the remainder of the dataset $D - T^{(r)}$ is predicted by $RF^{(r)}$. The votes from the trees decide the class probability estimates for each instance in $D - T^{(r)}$ which we denote

by $Q^{(r)}$. Here $Q^{(r)}$ can also be considered OOB estimates, in the sense that the trees which make the estimates are trained on other data, $T^{(r)}$, rather than themselves, $D - T^{(r)}$. Combining these two concepts, the OOB class probability estimates for the whole dataset D are defined as

$$P_{OOB}^{(r)} = P^{(r)} \cup Q^{(r)} = \{p_{OOB,jk}^{(r)} | j = 1, \ldots, N, k = 1, \ldots, K\}. \tag{2.2}$$

Based on $P_{OOB}^{(r)}$, the cluster labels can be updated for the whole dataset. That is, each instance is assigned to the cluster label giving the highest probability estimate. Here $L^{(r)}$ represents the updated cluster labels for dataset D.

Also, $P_{OOB}^{(r)}$ plays an important role in selecting the new training set $T^{(r+1)}$ for the next iteration. Based on $P_{OOB}^{(r)}$, the set of probability margins, denoted by $P_{MG}^{(r)}$, can be computed as follows:

$$\begin{aligned} P_{MG}^{(r)} = \{p_{MG,j}^{(r)} = \max_k(p_{OOB,jk}^{(r)}) - \operatorname{secmax}_k(p_{OOB,jk}^{(r)}) | \\ j = 1, \ldots, N\}, \end{aligned} \tag{2.3}$$

where function secmax($\mathbf{x}$) retrieves the second greatest value from set $\mathbf{x}$. We set a threshold $\theta^{(r)}$ to select the representative instances into the new training set $T^{(r+1)}$. Only instances with probability margins which exceed $\theta^{(r)}$ are selected. That is

$$T^{(r+1)} = D[\text{indice}(P_{MG}^{(r)} > \theta^{(r)})], \tag{2.4}$$

where function indice(*cond.*) extracts the indices of the instances satisfying the given condition. The target attribute of the training set in the new iteration, $c^{(r+1)}$, is also extracted from the updated label set $L^{(r)}$ accordingly:

$$c^{(r+1)} = L[\text{indice}(P_{MG}^{(r)} > \theta^{(r)})]. \tag{2.5}$$

With this selection criterion, a new training set is formed by the instances whose labels CRAFTER is most certain about, while the potentially mislabeled ones with low margins are no longer a part of the new training set. With $T^{(r+1)}$ and $c^{(r+1)}$, CRAFTER is ready for the next iteration.

The iterations stop when clustering converges, in other words, when very few instances (less than 1%) change their cluster labels from the previous iteration. We also set the maximum iterations R in case the clustering doesn't converge within an acceptable time. Algorithm 2 summarizes the main process of CRAFTER.

Algorithm 2 Main Step Algorithm

procedure MAIN STEP($T^{(0)}, c^{(0)}, D$)
  for $r \leftarrow 0, R$ **do**
    $\{RF^{(r)}, P^{(r)}\} \leftarrow RandomForest(T^{(r)}, c^{(r)})$
    $Q^{(r)} \leftarrow ApplyModel(RF^{(r)}, D - T^{(r)})$
    $P^{(r)}_{OOB} = P^{(r)} \cup Q^{(r)}$
    $L^{(r)} \leftarrow UpdateClusters(P^{(r)}_{OOB})$
    if $r > 0$ & $CountDifference(L^{(r)}, L^{(r-1)}) < 1\% \times N$ **then**
      return $L^{(r)}$
    end if
    $P^{(r)}_{MG} \leftarrow CalculateMargins(P^{(r)}_{OOB})$ (Eq. (2.3))
    $T^{(r+1)} = D[\text{indice}(P^{(r)}_{MG} > \theta^{(r)})]$
    $c^{(r+1)} = L[\text{indice}(P^{(r)}_{MG} > \theta^{(r)})]$
  end for
  return $L^{(r)}$
end procedure

We can also understand CRAFTER from the clustering validation perspective. One common strategy to validate a clustering algorithm is splitting the data into halves, clustering each part, and evaluating how a classification model built on one half agrees with the clustering from the other half. A similar idea is adopted by

CRAFTER. A large class probability estimate implies approximate agreement among the tree models regarding the instance's cluster label. CRAFTER embraces the validation process into its algorithm, by selecting the most agreed upon instances as the training data.

2.3.3 Parameter Settings

In previous studies, many clustering results were reported with tuned parameters of different values for different datasets, such as [44] and [54]. However, tuning parameters to match the clustering with the real class labels violates the purpose of unsupervised learning. For CRAFTER, only K needs to be given by the user, and all the other parameters are specified as follows across all the datasets. We do not perform additional parameter tuning for each clustering scenario.

Initial sample size The sample size is kept small in the initialization step because of the $\mathcal{O}(n^2)$ computational complexity of RFC. Normally a sample size $n = 5\% \times N$ is adopted. In addition, we constrain the sample size to be no more than 500 and no less than $25 \times K$, so that the sample is not too large, and CRAFTER has enough data for each cluster at the start.

Threshold θ The threshold should decrease when there are more clusters because the class probability is distributed among more classes. We set $\theta^{(r)} = \frac{1}{K} + 0.35$ for all the iterations, which works well for all the clustering tasks in our experiments. Alternatively, we can set $\theta^{(r)}$ to a specific percentile of $P_{MG}^{(r)}$. By setting it to a higher percentile, less instances are selected into the training set in each iteration, which adds more disturbances to the clustering process and slows the convergence. For simplicity, we only used the first option in all our experiments.

Convergence criteria The clustering is considered converged when less than 1% of data changes their labels, and at most 20 iterations are allowed.

2.3.4 Computational Complexity

In the initialization step, applying RFC to the sample M times results in $\mathcal{O}(Mn^2)$ computational complexity because of the PAM clustering. The computational complexity of the main step is capped by the RF algorithm. In each iteration, it takes $\mathcal{O}(tFN \log N)$ time to train a RF model, where t is the number of trees and F is the number of features used at each split. Consequently, with maximum iterations R, the total computational complexity of CRAFTER is of $\mathcal{O}(Mn^2) + \mathcal{O}(RtFN \log N)$. In general, when N is large, we have $n, M, t, F, R \ll N$, so the computational complexity of CRAFTER can be concluded as $\mathcal{O}(N \log N)$. Later in the section of the scalability test, we show that the complexity of CRAFTER can be reduced to $\mathcal{O}(N)$ by adding constraints on the depth of the trees.

2.3.5 Miscellaneous Issues

CRAFTER can effectively handle outliers and noise. In each iteration, we select the training set based on the margin of class probability estimates so that outliers with relatively low margins usually won't be selected and, therefore, won't impact the clustering results. In addition, if clusters with fewer outliers are preferred, we can decide to remove those instances with low margins from the final clusters.

Tree-based methods select important features during their model building. Therefore, they are good at handling high dimensionality. CRAFTER shares the same advantage by having tree-ensembles in both the initial RFC step and the main step. According to Parsons et al.'s categorization scheme [45], we may consider CRAFTER a subspace clustering algorithm.

To handle datasets with missing values, for categorical datasets, CRAFTER simply considers the missing values as a new categorical value, and for numeric datasets, the missing values are imputed by the attribute means.

CRAFTER also provides a compact method for organizing the clusters as it generates a supervised RF model $RF^{(R)}$ at the end of the clustering. If a small number of new data instances arrive, they can be assigned to the clusters based on the classification outputs of $RF^{(R)}$. If the amount of new data is large, we can continue the iterations with the new data included in the set $D - T^{(r)}$, starting from line 4 of Algorithm 2. In addition, the important features for clustering can be identified by the importance score generated by $RF^{(R)}$, which can help us gain deeper understanding of the dataset.

2.4 Experiments

2.4.1 Clustering Evaluation

When the class labels for the data are provided but not used to cluster the data, the cluster purity measure is one of the most commonly used criteria to evaluate the quality of clusters. Suppose the final number of clusters is K, we have the following definition for the clustering purity ([43], [54], [55] and [56]).

$$\text{purity} = \frac{\sum_{k=1}^{K} a_k}{N}. \tag{2.6}$$

where N is the total number of instances, and a_k is the number of instances of the majority class in the k-th cluster. Purity is the main evaluation criteria we considered. Additionally, we present the Rand Index [57] and Cluster Mapping Measure (CMM) [58] in our box plots (Fig. 2.2) to give other perspectives for the clustering evaluation. We use RF distance when calculating CMM.

2.4.2 Datasets

CRAFTER was tested on both synthetic datasets and real datasets to show its ability to handle a wide range of clustering tasks. A synthetic dataset, named as Noisy-set, was generated to test the performance of our algorithm in the presence of many irrelevant attributes. To demonstrate that CRAFTER can handle high-dimensional data effectively, we tested it on a group of synthetic datasets with Gaussian clusters in high-dimensional space (Dim-sets). A series of large-scale datasets generated by the data generator [59] was used to test the scalability of our algorithm. The details of the synthetic datasets are provided in the following.

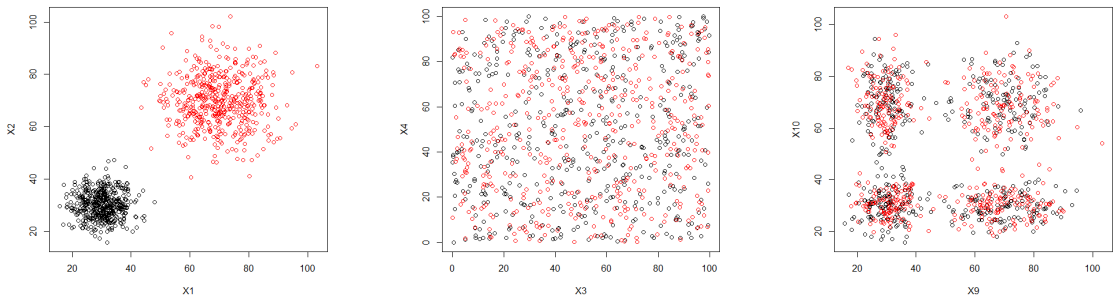
In addition, we covered three types of real datasets in our testing: numeric, categorical and mixed. All the real-life datasets tested in this chapter were obtained from the UCI Machine Learning Repository [1]. The characteristics of these datasets are summarized in Table 2.1. Note that the breast cancer dataset and the mice protein dataset belong to the biomedical domain.

Table 2.1: Characteristics of UCI Datasets for the Clustering Experiments (N: Numerical, C: Categorical)

Data type	dataset	Size	Attribute No. (N/C)	Missing value	Classes	Class distribution
Numeric	Breast cancer	569	30	None	Benign, Malignant	357 / 212
	Mice protein	1,080	77	Yes	CS, SC	525 / 555
	Pen digits	7,494	16	None	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	780/ 779/ 780/ 719/ 780/ 720/ 720/ 778/ 719/ 719
Categorical	Soybean(small)	47	35	None	D, C, R, P	17/ 10/ 10/ 10
	Voting	435	16	Yes	Republican, Democrat	168 / 267
	Mushroom	8,124	22	Yes	Edible, Poisonous	4,208 / 3,916
Mixed	Credit approval	690	15 (6/9)	Yes	Approved, Rejected	307 / 383
	KDD Cup 1999 sample	20,000	41 (34/7)	None	Attack, Normal	3872 / 16128

Synthetic Dataset with Irrelevant Attributes (Noisy-set)

We generated a dataset having 1000 instances and 10 numeric attributes. Among those 10 attributes, x_1 and x_2 primarily define the clusters. Here, (x_1, x_2) follows two Gaussian clusters centered on $(30, 30)$ and $(70, 70)$, respectively, as shown in Fig. 2.1a. The other 8 attributes are independent with the other attributes. We have x_3, x_4, x_5 drawn from a uniform distribution $U(0, 100)$ (Fig. 2.1b), and $x_6, x_7, x_8, x_9, x_{10}$ follow the same marginal distribution as that of x_1, but they are independent (Fig. 2.1c). We regard $x_3, ..., x_{10}$ as irrelevant attributes.



(a) $x_1 \, vs. \, x_2$ plot (b) $x_3 \, vs. \, x_4$ plot (c) $x_9 \, vs. \, x_{10}$ plot

Figure 2.1: Different Projections of Noisy-set - two true clusters are indicated in red and black

High-dimensional Datasets with Gaussian Clusters (Dim-sets)

Dim-sets are composed of 6 high-dimensional datasets: Dim32, Dim64, Dim128, Dim256, Dim512 and Dim1024. The numbers indicate the dimensionality. Each dataset contains 1024 data points allocated in 16 Gaussian clusters. They were originally introduced by [60]. We used Dim-sets to demonstrate the performance of CRAFTER in clustering high-dimensional data.

Large-scale Rule-based Data (Rule-sets)

Rule-sets are generated by the data generator available in [59]. The generator provides many options, in terms of the size of datasets, the number of attributes and the domain size (the number of possible categorical values for each attribute). Each class is specified by a conjunctive rule of the form $(Attribute_1 = v_1 \wedge Attribute_2 = v_2 \wedge \dots) \Rightarrow Class = C_1$. For our experiments, we generated datasets of different sizes (from 100 thousand to 5 million instances). Each dataset contains 20 attributes and 8 classes. The domain size of each attribute is 20.

2.4.3 Implementation Details and Comparison References

Both RFC and CRAFTER are able to handle datasets with different types of attributes. Therefore, we compare these two methods over every dataset. RFC is referred to as RF-PAM or RF-CLARA, depending on which K-medoids algorithm is used. When the size of the dataset is larger than 5,000, we only run RF-CLARA, because it becomes very computationally expensive to run RF-PAM for multiple replicates. In addition, we compare CRAFTER to K-means, K-modes, K-prototypes for numeric, categorical and mixed datasets, respectively. All of the testing was conducted on a laptop with Core i7-7500U Processor and 16GB RAM.

Our RF-PAM/CLARA implementation uses R package "randomForest"[61] to generate RF distances. We adopt the PAM implementation from the R package "cluster" [62], and we implement CLARA based on this PAM implementation. Our CLARA implementation employs 10 sampling processes with size of $\min(500, 10\% \times N)$. The K-means implementation we use is from R's basic "Stats" package, and we use R package "clustMixType" [63] for the experiments with K-modes and K-prototypes.

The implementation of CRAFTER is also based on the R package "randomForest" [61]. The RF-PAM implementation discussed above is used in the initialization step. In the main step, each RF model consists of 500 fully grown trees with the default setting.

In the testing, the initial cluster number K is always set to the number of classes in the dataset. Additional larger K values are considered for datasets whose initial purity is below 0.8. Such datasets include mice protein, pen digits, mushroom, credit and KDD 1999 sample datasets. For mice protein dataset, we add the experiments with $K = 4$ and 8, because this dataset may also be classified into 4 or 8 classes when additional labels are considered. For all the clustering methods, the mean and the standard deviation of the purity measure are reported based on 20 randomized experiments.

2.5 Results and Discussion

2.5.1 Testing Results for Synthetic Datasets

Table 2.2: Testing Comparisons on Synthetic Datasets (Noisy-sets and Dim-sets): each row compares the average clustering purity and its standard deviation (in parentheses) over 20 experiments using K-means, RF-PAM and CRAFTER

datasets	K-means with transformation	K-means without transformation	RF-PAM	CRAFTER
Noisy-set	0.951 (0.149)	0.761 (0.222)	0.806 (0.042)	0.969 (0.075)
Dim32	0.778 (0.066)	0.791 (0.065)	1.0 (0.0)	1.0 (0.0)
Dim64	0.759 (0.062)	0.791 (0.068)	1.0 (0.0)	1.0 (0.0)
Dim128	0.775 (0.062)	0.788 (0.065)	1.0 (0.0)	1.0 (0.0)
Dim256	0.750 (0.061)	0.722 (0.074)	1.0 (0.0)	1.0 (0.0)
Dim512	0.763 (0.090)	0.788 (0.074)	1.0 (0.0)	1.0 (0.0)
Dim1024	0.766 (0.053)	0.788 (0.080)	1.0 (0.0)	1.0 (0.0)

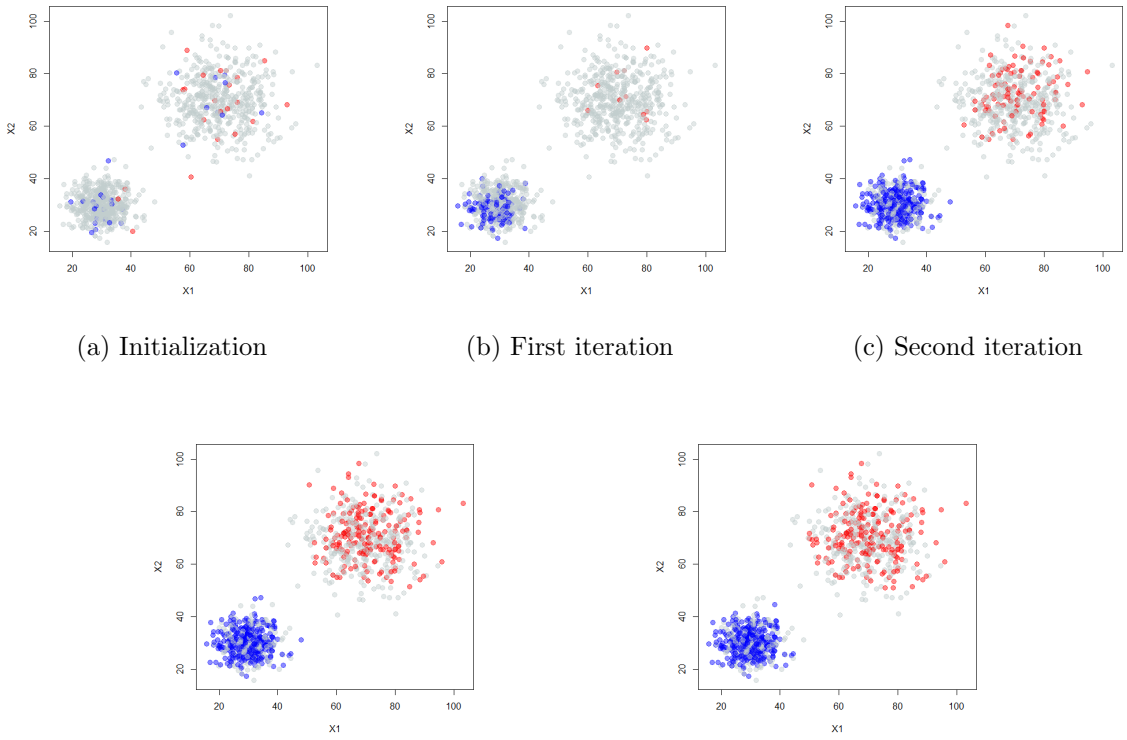
For Noisy-set and Dim-sets, the numeric attributes in each dataset are already in similar range, nevertheless, the results of K-means with and without transformation are both provided to present the best K-means outcome. After the transformation, all the attributes in the dataset have their mean equal to 0 and their standard deviation equal to 1.

By reference to Table 2.2, for all the Dim-sets, both CRAFTER method and RF-PAM fully discovered the 16 clusters with 0 error, which demonstrates the strong capability of the tree-ensemble method on handling high-dimensional cases. On the contrary, K-means only achieved 0.72-0.79 purity on the Dim-sets. Computational-wise, CRAFTER is more efficient than RF-PAM in order to achieve the same high-quality clusters.

For the Noisy-set, the clusters generated by CRAFTER and K-means (with transformation) are of similar quality. CRAFTER's clustering results are slightly better on average with less variation. This test demonstrates the robustness of CRAFTER in the presence of noise in the data.

2.5.2 Understand the Mechanism of CRAFTER

To get a better understanding about the mechanism of CRAFTER, Fig. 2.2 shows the "snapshots" of how CRAFTER clustered the Noisy-set in each iteration. As discussed, many methods can be used to generate the initial sample clustering which doesn't need to be perfect, because the main step revises the clustering through iterations until the clusters stabilize. In other words, CRAFTER is insensitive to the quality of the starting sample. To illustrate this point, we forced the algorithm to start with a weak initial clustering (generated by K-means) in this experiment. In Fig. 2.2a, the colored points represent the sample data $T^{(0)}$, and the "red" and "blue" colors indicate two initial sample clusters $c^{(0)}$. Unselected data remains in grey color.



(a) Initialization (b) First iteration (c) Second iteration

(d) Third iteration (e) Fourth iteration

Figure 2.2: Iteration-by-iteration Visualizations on the x_1 *vs.* x_2 Projection of Clustering Noisy-set Using CRAFTER: red and blue points are the training instances selected by CRAFTER in each iteration, and the color distinguishes the cluster labels of the training data. The grey points are not selected yet in that iteration, so they are assigned cluster labels by the maximum class probability estimate.

At this stage, data points of two colors were mixed inside the two true clusters, which was not an ideal initialization. As the algorithm proceeded, we observed that data points with incorrect labels at the beginning were excluded from the new training set $T^{(1)}$ based on the class probability margin criterion (Fig. 2.2b). At the same time, the data points with high margins started populating the two underlying clusters, which dominated the cluster structure in the next few iterations. Fig. 2.2c, Fig. 2.2d and Fig. 2.2e show that more and more data points received high margins and were included by the training set through the iterations. Eventually, the clustering converged in the fourth iteration.

2.5.3 Testing Results for Real-life Datasets

Table 2.3: Average and Standard Deviation (in Parentheses) of the Purity Measure Obtained by Different Clustering Algorithms on Different Datasets. The Last Row Summarizes the Counts of Win/Loss/Even Cases of CRAFTER Compared to Each Other Method Based on the t Tests with Corrected Level of the Significance

scenario	dataset	K	dataset type	K-means/modes/prototypes	RF-PAM	RF-CLARA	CRAFTER
1	Breast cancer	2	numeric	0.908 (0.003)	0.894 (0.016)	0.892 (0.020)	0.899 (0.023)
2	Mice protein	2	numeric	0.588 (0.000)	0.744 (0.039)	0.768 (0.071)	0.607 (0.101)
3	Mice protein	4	numeric	0.768 (0.040)	0.756 (0.039)	0.751 (0.038)	0.875 (0.118)
4	Mice protein	8	numeric	0.824 (0.030)	0.816 (0.025)	0.821 (0.028)	0.927 (0.072)
5	Pen digits	10	numeric	0.731 (0.038)	-	0.682 (0.022)	0.726 (0.026)
6	Pen digits	20	numeric	0.850 (0.017)	-	0.784 (0.017)	0.827 (0.037)
7	Voting	2	categorical	0.859 (0.007)	0.855 (0.010)	0.851 (0.012)	0.856 (0.023)
8	Soybean	4	categorical	0.799 (0.091)	0.949 (0.035)	0.951 (0.034)	1.000 (0.000)
9	Mushroom	2	categorical	0.715 (0.125)	-	0.771 (0.042)	0.774 (0.086)
10	Mushroom	3	categorical	0.787 (0.078)	-	0.835 (0.029)	0.876 (0.020)
11	Mushroom	4	categorical	0.845 (0.057)	-	0.867 (0.015)	0.876 (0.037)
12	Credit	2	mixed	0.705 (0.122)	0.673 (0.066)	0.695 (0.055)	0.739 (0.054)
13	Credit	3	mixed	0.731 (0.082)	0.702 (0.028)	0.692 (0.055)	0.726 (0.044)
14	Credit	4	mixed	0.765 (0.053)	0.713 (0.031)	0.710 (0.032)	0.679 (0.060)
15	KDD1999	2	mixed	-	-	0.806 (0.000)	0.806 (0.000)
16	KDD1999	3	mixed	-	-	0.976 (0.003)	0.980 (0.003)
17	KDD1999	4	mixed	-	-	0.975 (0.007)	0.979 (0.011)
# of win/loss/even of CRAFTER				4/2/8	4/1/4	8/1/8	

Table 2.3 summarizes the performance of different clustering algorithms on the 8 real-life datasets (17 scenarios). For each clustering method and each clustering scenario, we run the experiments for 20 times and the average and the standard deviation (in parentheses) of the purity measure are presented. In the table, the clustering results for K-means and its extensions, K-modes and K-prototypes, are presented in the same column for numeric, categorical and mixed datasets respectively. The K-prototypes in R package "clustMixType" failed to cluster the KDD1999 sample dataset, therefore their results are not provided here. We do not provide the results

of RF-PAM for pen digits, mushroom and KDD1999 datasets due to the high computational cost for larger datasets, especially for multiple replicates of experiments. For the numeric dataset, because the attributes in the breast cancer and mice protein data are of very different scales, these attributes were normalized to have 0 mean and 1 standard deviation before applying K-means clustering. All the attributes in the pen digits data had already shared the same range, so no transformation was performed. On the other hand, there was no need to perform any kind of data transformation before applying CRAFTER or RFCs due to their scale-invariant property.

For each scenario, we compare CRAFTER to each of the other methods using t tests. The level of significance is corrected to α/A_s based on the Bonferroni correction, where $\alpha = 0.05$, and A_s is the number of other clustering algorithms we compare to in scenario $s, s = 1, ..., 17$. The need of this correction in the multiple comparisons is highlighted in [64] and [65]. For example, in scenario 1, three pairs of t tests (CRAFTER vs. K-means/modes/prototypes, CRAFTER vs. RF-PAM, and CRAFTER vs. RF-CLARA) are performed, thus the null hypothesis is rejected only when p value $< 0.05/3 = 0.017$. Two methods are considered to perform evenly when the t test after the correction shows insignificance. A win or a loss is counted when the t test shows the significant difference. The last row of Table 2.3 summarizes the number of times that CRAFTER wins, loses or performs evenly compared to each of other clustering algorithms. From this summary, we observe that CRAFTER has more wins than losses compared to any other method included in the comparison. More specifically, CRAFTER has a slight advantage over K-means/modes/prototypes, and the advantage of CRAFTER lies more in handling various clustering tasks with only one approach. Compared to RF-CLARA, CRAFTER performs significantly better in 8 out of 18 scenarios with only 1 loss. Despite that the Bonferroni correction is quite conservative and results in many even cases, this comparison demonstrates that

CRAFTER's overall performance is stronger than that of RF-CLARA. The box-plots in Fig. 2.2 compares CRAFTER to RFCs in terms of not only purity, but also Rand Index and CMM. CMM is not calculated for KDD1999 dataset due to the high memory usage for this case. We find that the three measures give very similar evaluation for both the quality and the variation of the clustering results. We also observe that CRAFTER generates better clusters than RFCs for most testing scenarios, but sometimes results in a slightly higher variance (e.g. breast cancer, mice protein, pen digits and credit approval). It is worth mentioning that CRAFTER is a lot more scalable than RF-CLARA. For example, based on our implementation in R, it took CRAFTER around 30 seconds on average to cluster the 20,000 instances in the KDD1999 sample dataset into 4 clusters, but the same task took RF-CLARA around 8 minutes to complete. As observed in Fig. 2.2q, the clusters found by CRAFTER is slightly better with much less computational cost compared to RF-CLARA. This comparison shows that CRAFTER is a better scalable clustering solution than RF-CLARA which also leverages the advantages of a tree-based method.

2.5.4 Efficiency-effectiveness Trade-off

Constraining the depth of the tree (or the total number of leaf nodes) can reduce the computational complexity of CRAFTER to $\mathcal{O}(N)$ and makes it scalable for large dataset. The question is how this constraint would affect the quality of the clustering result. Two other parameters that affect the clustering speed are: 1) the number of iterations allowed and 2) the number of the trees in the RF model in each iteration. Here we investigate the impact of these three parameters based on breast cancer, credit approval and congressional voting datasets. The average clustering purity over 20 experimental replicates against the three parameters are presented in Fig. 2.3. The default setting is the unconstrained depth for each tree, 500 trees for each

RF model, and a maximum of 20 iterations. In our experiment, when changing one parameter, the other two parameters remained the default. From Fig. 2.3 we observe that the changes of these three parameters do not affect the clustering quality much. In fact, the most computational expensive setting does not necessarily produce the best clustering. The purity difference across different parameter settings are insignificant. Two conclusions can be drawn from this experiment: 1) the performance of CRAFTER is insensitive to the settings of these three parameters, 2) to cluster large datasets, we may consider the setting that has linear computational complexity.

2.5.5 Scalability Test

Rule-sets were used to test the scalability of CRAFTER. To reduce the computations for these large datasets, in each iteration of the main step, a RF composed of 100 trees with at most 30 leaf nodes was trained instead, and the maximum number of iterations allowed was also reduced to 10. Our experiments in Section 2.5.4 shows that the performance of CRAFTER is insensitive to the configuration change.

The analysis in Section 2.3.4 has concluded that the computational complexity for CRAFTER is $\mathcal{O}(N \log N)$, mostly determined by the computations of building the RF models in the main step. Here it is shown that the computational complexity can be further reduced to $\mathcal{O}(N)$ for large datasets. Fig. 2.4 depicts that the computing time increases with the sizes of the Rule-sets, and a nearly linear increasing trend is observed. The complexity of RF training becomes close to $\mathcal{O}(N)$ if constraints on the maximum number of leaf nodes or the maximum depth of trees are added. For tree-ensemble methods, this is a common strategy when dealing with large datasets. In the figure, the small deviations from the regression line are due to the different numbers of iterations needed for the clustering to converge on different datasets. In conclusion, in practice, CRAFTER can be used as an approximately $\mathcal{O}(N)$ algorithm.

2.6 Limitations and Future Work

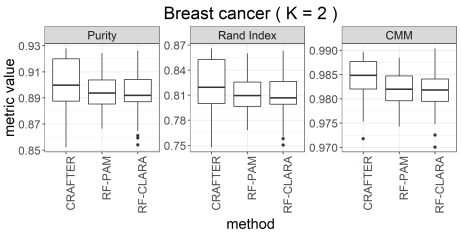
CRAFTER is not a deterministic algorithm, so in terms of the quality of the generated clusters, variation exists from run to run. However, as shown in the box plots in Fig. 2.2, the variance of the clustering results is acceptable as the clustering quality is better in general. Another limitation of CRAFTER is that the it is not designed to optimize a specific objective function but based on a heuristic, so only empirical evaluations and comparisons are allowed.

For the future work, we will aim to improve the performance of CRAFTER for special cases, and empirically compare it to more specialized algorithms, such as ROCK [44] for categorical datasets and Birch [66] for numeric datasets. Also, as the final clusters are organized by a black-box ensemble model, more work can be done to better interpret the found clusters.

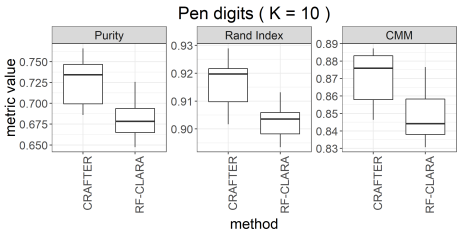
2.7 Conclusion

We have proposed a new tree-ensemble clustering algorithm which can effectively handle large datasets with mixed attributes and high dimensionality. Different from traditional clustering methods, CRAFTER tackles the unsupervised problems from a supervised-learning perspective. It adopts tree-ensembles in both the initialization and main steps, and inherits the desirable properties of tree-based methods throughout the entire algorithm. Compared to RFC, an existing tree-based clustering algorithm, CRAFTER first avoids the computations of the pair-wise distances for the entire dataset with a sampling approach, and then it replaces the computationally intensive K-medoids with a simple supervised RF-ensemble. All of these enhancements make CRAFTER a scalable and storage-efficient clustering algorithm. In terms of the quality of the clusters, our experiments demonstrate that CRAFTER is either com-

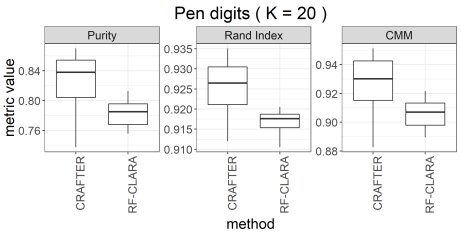
parable with or better than K-means, K-modes, K-prototypes and RFC in their own applicable domains. With all the desirable properties discussed above, CRAFTER has the potential to handle different clustering tasks in a wide variety of applications in the future.



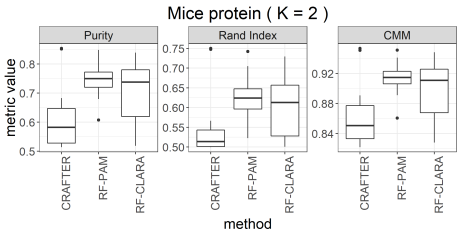
(a) breast cancer



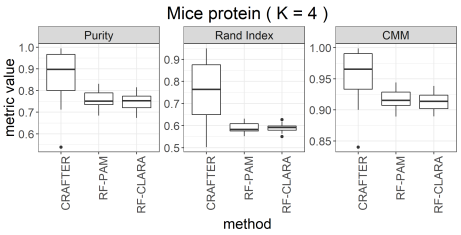
(b) pen digits (10 clusters)



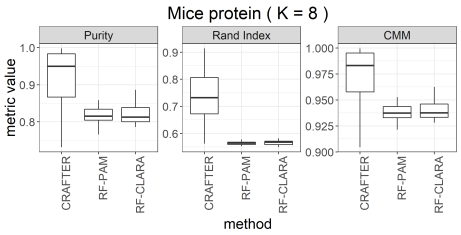
(c) pen digits (20 clusters)



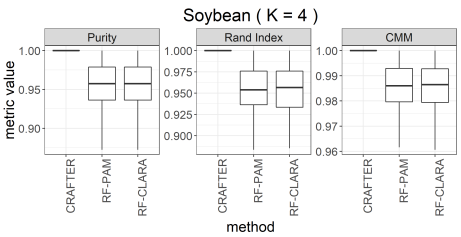
(d) mice protein (2 clusters)



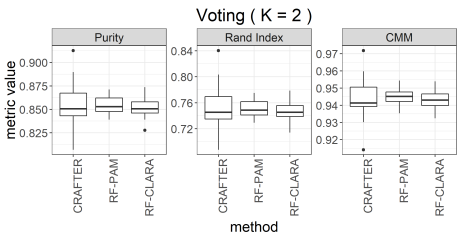
(e) mice protein (4 clusters)



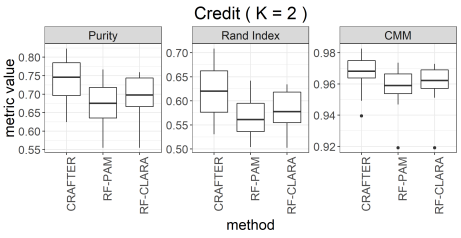
(f) mice protein (8 clusters)



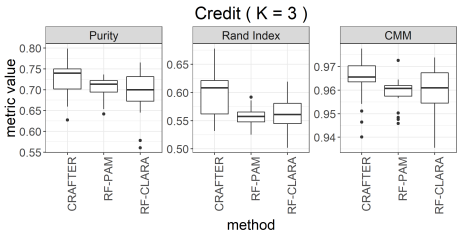
(g) soybean (small)



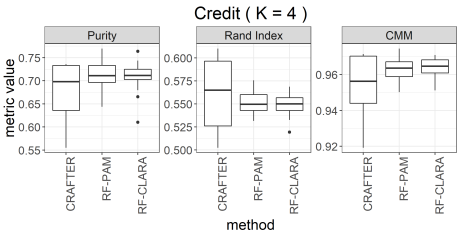
(h) congressional voting



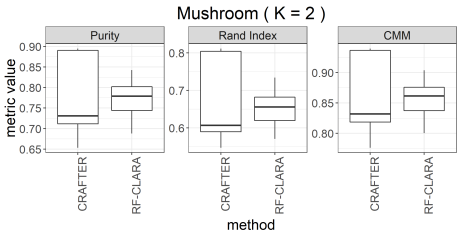
(i) credit approval (2 clusters)



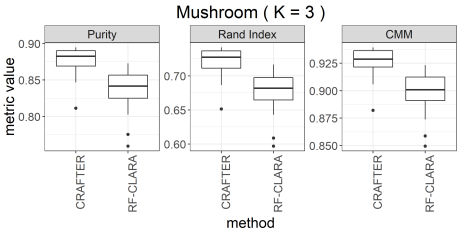
(j) credit approval (3 clusters)



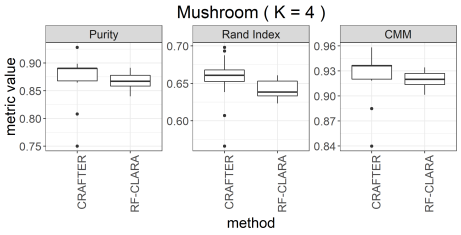
(k) credit approval (4 clusters)



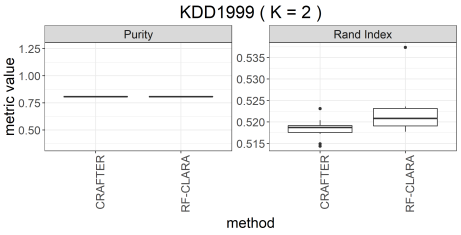
(l) mushroom (2 clusters)



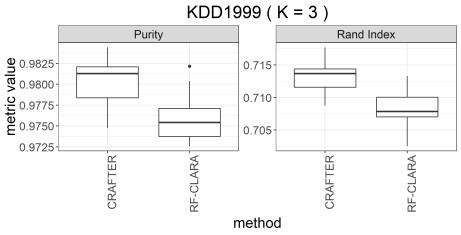
(m) mushroom (3 clusters)



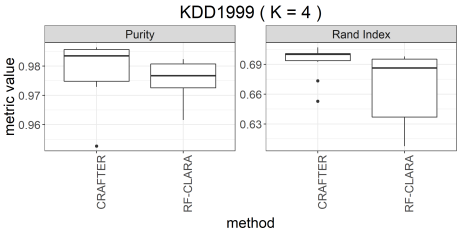
(n) mushroom (4 clusters)



(o) KDD 1999 (2 clusters)

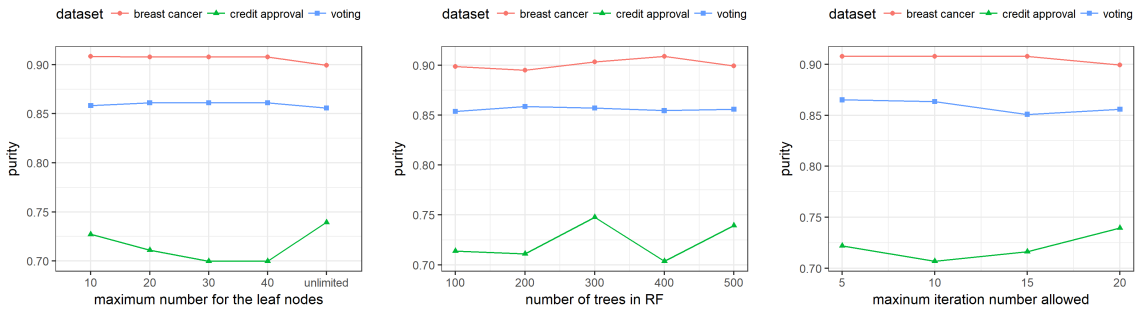


(p) KDD 1999 (3 clusters)



(q) KDD 1999 (4 clusters)

Figure 2.2: Box-plot Comparison Between CRAFTER and RFCs Based on Purity, Rand Index and CMM



(a) purity vs. max leaf nodes (b) purity vs. ntrees (c) purity vs. max iterations

Figure 2.3: Relationship Between the Performance of CRAFTER (Estimated by Average Purity) and 3 Parameters: (a) the total number of the leaf nodes for each tree in the RF model, (b) the number of trees in the RF model in each iteration, (c) the maximum number of iterations allowed

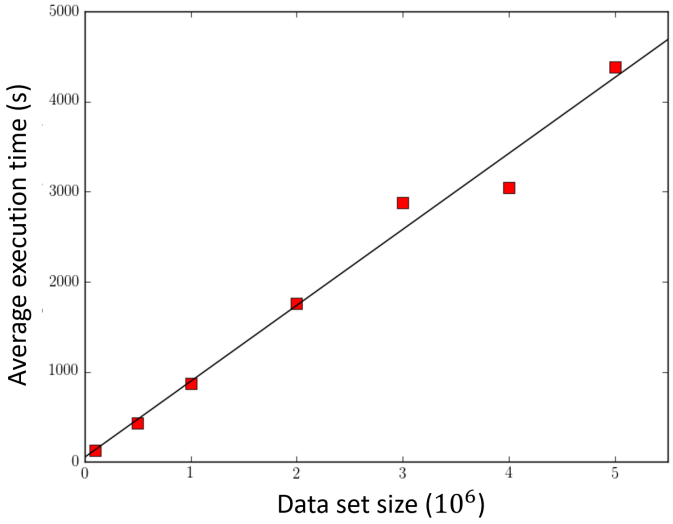


Figure 2.4: Execution Time of CRAFTER vs. Sizes of Datasets: estimated based on the average execution time over 10 replicates (in seconds) on different sizes of synthetic Rule-sets. The regression line indicates a approximately linear relationship.

Chapter 3

BAG-OF-SEGMENTS REPRESENTATION FOR GENOME SEQUENCING DATA

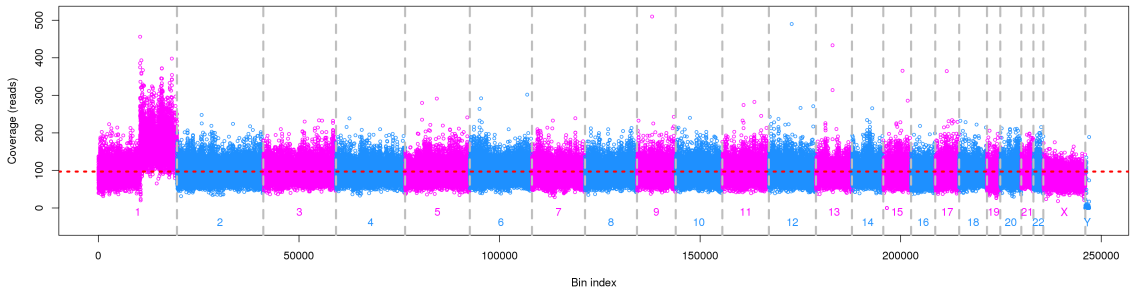
3.1 Introduction

Genome sequencing data can be considered as a special type of time series data as their attributes are in biological order. The human genome contains approximately 3 billion of base pairs, and that implies that each instance may have as many as 3 billion features. However, the number of instances (e.g. patients, tumor samples, etc.) for a medical study is usually smaller compared to other studies due to the higher cost. The nature of high dimensionality and small data presents a specific challenge for the classification problems involving genome sequencing data. In this chapter, we introduce a compact data representation for the genome sequencing data. The method can generalize to other time series classification problems which share similar properties (very long time series and small dataset). We named this new representation Bag-of-Segments (BoS). With this new representation, standard machine learning algorithms can be applied to obtain a classification model.

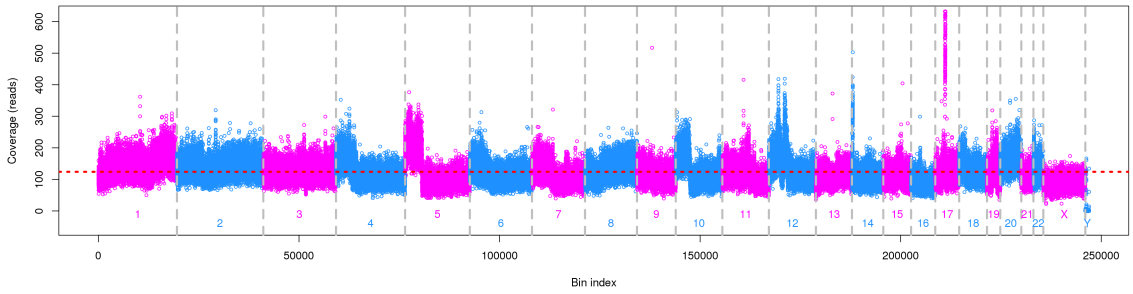
BoS data representation is similar to the family of Bag-of-Features methods. Bag-of-Features methods have been extensively used in the classification of images [67], time series [38] and text documents [68]. Here we extend the methodology to the classification problem of genome sequencing data or very long time series.

In this chapter, we study the grade classification problem of the ovarian serous carcinoma in order to demonstrate the effectiveness of BoS representation. The input data is the sequencing profile of the Copy Number Alterations (CNAs) (as shown in

Fig. 3.1). Although the sequencing data is obtained through low-coverage sequencing (1000 base pairs represented by one attribute), it contains over 250 thousand ordered attributes per instance. On the contrary, we only have 34 instances, significantly smaller than the number of attributes. Our experiment shows that the BoS representation can effectively extract CNA patterns predictive of tumor grades and lead to high classification accuracy. Furthermore, the analysis of the BoS features contributes to the differentiation highlighting two different underlying biological processes. One involves large scale deletions or amplifications suggesting abnormal mitotic events while the other involves local amplification and deletions commonly associated to DNA repair malfunctions.



(a) A low grade example



(b) A high grade example

Figure 3.1: Low-grade and High-grade Examples for the CNA Profile Data: colors are used only to distinguish the consecutive chromosomes.

3.2 Backround

3.2.1 Copy Number Alterations (CNAs)

Defined as somatic gain or loss of DNA regions, Copy Number Alterations (CNAs) are reflective of genome instability, frequently affecting functionally important genes, such as tumor suppressors and oncogenes. CNAs are also associated to the early onset of tumor. CNAs include both deletions and amplifications of large or small genome regions. Large scale CNA events involving whole chromosome or chromosome arms alterations also referred as aneuploidy. Small deletion events may target local regions of the genome harboring tumor suppressor genes locations, while amplifications preferentially target oncogenes locations [69]. As the consequence of tumor progression and evolutions, CNAs are not randomly distributed across the genome. The profiles of CNAs may provide a fingerprint specific to a tumor type or tumors class [18].

3.2.2 Ovarian Serous Carcinoma

Ovarian serous carcinoma is a group of heterogeneous diseases that are further classified into low grade and high grade serous carcinoma based on their histologic features. These two groups are thought to be mutually exclusive based on their molecular characteristics. The majority (96%) of high grade serous carcinomas have TP53 mutations and show high level of chromosomal copy number changes through the entire genome, whereas low grade serous carcinomas dont have TP53 mutations, show KRAS and BRAF mutations and in most cases are near diploid [70].

3.2.3 Low-coverage Whole Genome Sequencing

The availability of Next Generation Sequencing (NGS) technology platforms has enabled the study of CNAs at a genome wide scale and at an unprecedented level

of resolution. Not only the precision of the CNAs detection is enhanced but also the number of copy changes can be more accurately defined. Numerous methods are available to report CNAs from high-coverage whole genome sequencing and for low-coverage sequencing (LC-WGS). LC-WGS has recently gained interest since successfully translated into clinical applications. The Non-Invasive Prenatal Test (NIPT) is one example where cell free DNA of pregnant woman is sequenced a low coverage (< 1x) to report the presence of fetal DNA aneuploidy. The expertise acquired in our group in the processing of LC-WGS as led us to explore how CNAs reporting from LC-WGS combine with machine learning techniques can be used to stratify tumor biospecimens of different grades.

3.2.4 Bag-of-Features and Bag-of-Segments

Bag-of-Features approaches usually consist of two steps: local feature extractions and dictionary construction. For images, the local features could be texture features such as HoG (short for Histograms of Gradients) [36] and SIFT (short for Scale-Invariant Feature Transform) [37]. For time series, they could be local statistics such as mean and standard deviation [38]. The dictionary can be constructed using either a supervise method [39] or an unsupervised method (e.g. clustering). In general, the dictionary is constructed by partitioning the local features into different groups, and each group represents a "word" in the dictionary. The Bag-of-Features representation is the frequency distribution over these groups. Although currently surpassed by other methods such as deep learning, Bag-of-features remains a strong method when dealing with small sample sizes like in the case for our study.

The proposed BoS is a variant of the Bag-of-Features methods for the CNA profile data. In this case, the local features are the width and the height of the segments of the sequencing after a segmentation step. The development of this method incorporates

the domain knowledge regarding the CNAs profiles and aims to reflect the scale differentiation of the deletion or amplification associated with tumor grades.

3.3 Data and Pre-processing

In this study, we have collected and processed 34 sequencing coverage profiles from patients with ovarian serous carcinoma. Among these patients, 14 cases were diagnosed with high-grade and the remaining 20 with low-grade ovarian serous carcinoma based on the histologic review of the surgical material from tumor debulking surgery. In each case, area of tumor was macrodissected from the Formalin-Fixed-Paraffin-Embedded (FFPE) tissue blocks and DNA was extracted using Qiagen extraction kit. Sequencing reads were produced by Hiseq 4000, with multiplexing 8 samples per lane. Samples were preprocessed by Wandy [17]. Wandy accumulates the sparse sequence reads into 10000 base long bins and performs several noise reduction procedures to more accurately characterize changes in coverage characteristic of CNVs. As a result, each point in the input sequencing data is the coverage of WGS in 10kb genome window.

3.4 Method

The proposed method for the tumor grade classification based on LC-WGS data consists of three major steps: 1) the segmentation of CNA profiles and extraction of the local features (the segment width and the segment height) 2) BoS representation. 3) the training of the classification model using the BoS representation as the input.

3.4.1 Segmentation

We apply a top-down regression tree (CART algorithm [23]) to segment and identify the step-wise changes in the sequencing data. More specifically, the regression

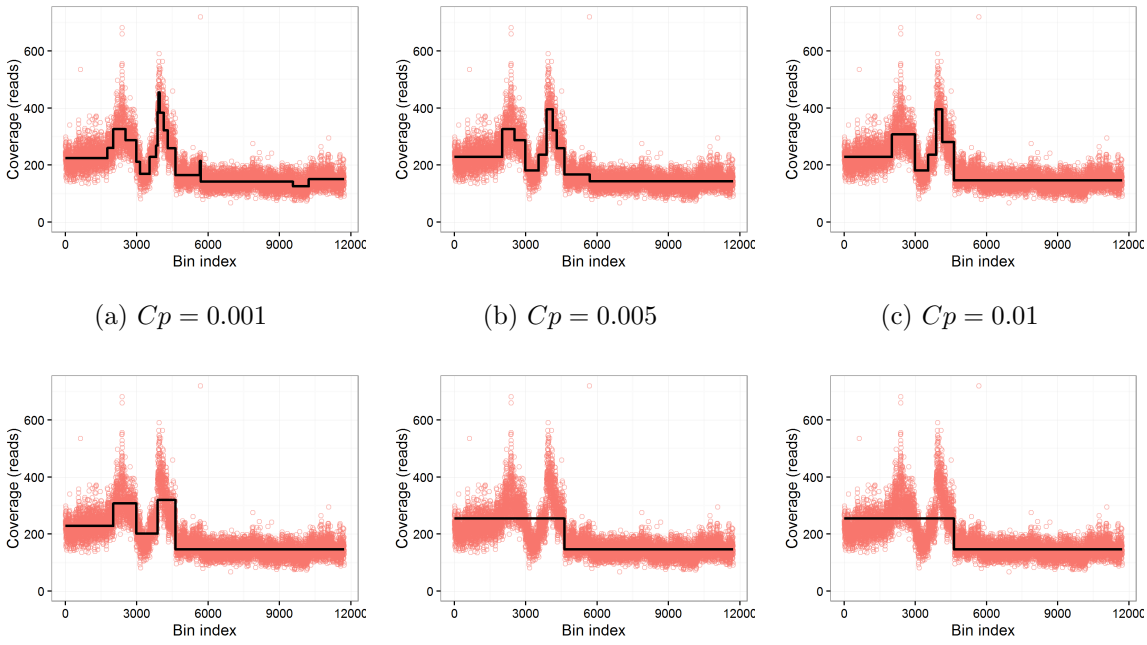
tree model is fitted to the sequence coverage of each chromosome resulting into 23 regression tree models per sample. During this process bin index is considered as the predictor variable, and the coverage value at each bin is considered as the target variable. Each leaf node of a regression tree represents a CNA segment. To obtain a step-wise signal of a proper level of complexity, the CART algorithm is tuned by modifying the cost-complexity parameter (Cp). When Cp is too large, the segmentation may be too coarse to detect the high-grade signal. On the contrary, if Cp is too small, the segmentation may overfit the noise in the data. Fig. 3.2 uses the CNA profile data of the chromosome 12 from one tumor sample as the example to illustrate the effect of Cp.

Each segment is then characterized by a width (in proportion to the chromosome length) and a height (in log2 ratio to the median coverage). Fig. 3.3 gives a segmentation example for the entire CNA profile (23 chromosomes) and the corresponding 2D distribution of the height and the width of its segments.

3.4.2 Bag-of-Segments

The CNA segments from all the samples are aggregated to produce a single 2D distribution of heights and widths, as illustrated in Fig. 3.4. The CNA segment dictionary is defined based on the 2D distribution as follows.

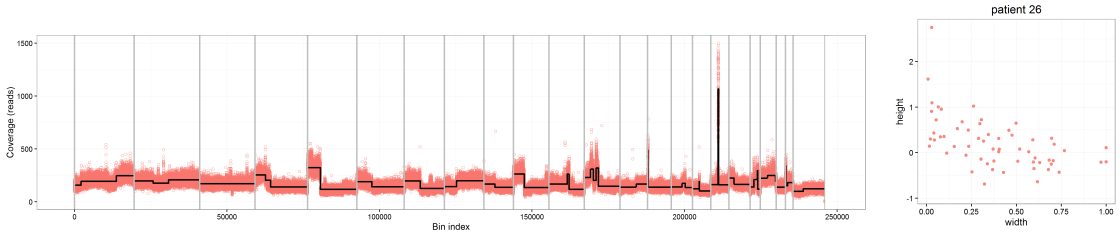
In analogy to the dictionary for a text document, the dictionary for the sequencing segments is a collections of CNA segment classes. Each CNA segment class represents the segments with certain characteristics, similar to a "word" in the dictionary. We define the CNA segment classes in an unsupervised manner. More specifically, let h_α and $h_{1-\alpha}$ denote the α and $1-\alpha$ percentiles of the segment heights, and w_α and $w_{1-\alpha}$ denote the α and $1-\alpha$ percentiles of the segment widths. The black lines in the right subfigure of Fig. 3.4 indicate these quantiles. Using the quantiles of both



(a) $Cp = 0.001$ (b) $Cp = 0.005$ (c) $Cp = 0.01$

(d) $Cp = 0.05$ (e) $Cp = 0.1$ (f) $Cp = 0.2$

Figure 3.2: Results of Regression-tree Segmentation with Different Cp Values Based on a Chromosome-12 Sample



(a) (b)

Figure 3.3: Segmentation Result of a Whole Genome Sequencing Example and its 2D Representation: (a) each red dot is coverage of WGS in 10 kb genome window. Black lines indicate the segmentation results fitted by CART. (b) the 2D distribution of the segment width and height based on the segmentation results in (a).

the width and the height, 9 CNA segment classes (NA, MA, WA, NN, MN, WN, ND, MD, WD) are defined as based on the rules described in Table 3.1. For each sample, its segment frequency distribution over the segment classes generates the fix-length BoS representation. We use quantiles instead of a clustering algorithm to partition the space of segment height and segment width for better interpretability.

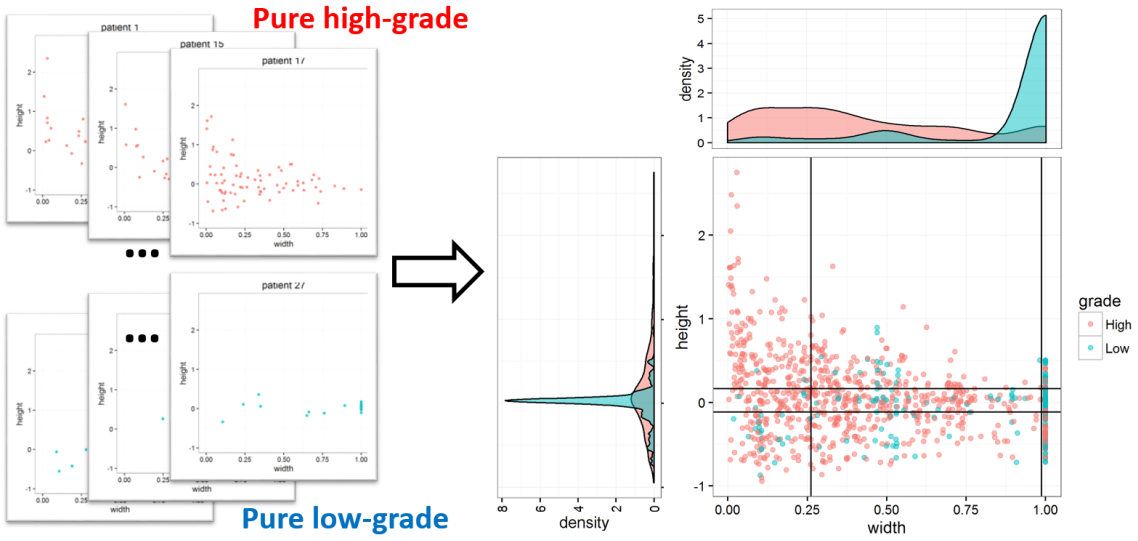


Figure 3.4: Workflow of Bag-of-Segments: the 2D distributions in the height-width space of the segments from different samples (on the left) are aggregated into a single distribution (on the right). The dictionary is defined by the 9 cells divided by the black lines which were drawn based on the percentiles of the width and the height. The marginal density estimates for the width and height are shown on the top and the left of the aggregated distribution plot respectively. The points for the segments and their marginal density estimates are denoted in blue for the low grade samples, and those are denoted in red for the high grade samples.

Table 3.1: Categorization for the Segments

	$w < w_\alpha$	$w_\alpha \leq w < w_{1-\alpha}$	$w \geq w_{1-\alpha}$	row sum
$h \geq h_{1-\alpha}$	Narrow Amplified (NA)	Medium Amplified (MA)	Wide Amplified (WA)	Amplified
$h_\alpha \leq h < h_{1-\alpha}$	Narrow Normal (NN)	Medium Normal (MN)	Wide Normal (WN)	Normal
$h < h_\alpha$	Narrow Deleted (ND)	Medium Deleted (MD)	Wide Deleted (WD)	Deleted
column sum	Narrow	Medium	Wide	

3.4.3 Classification Model

A Random Forest (RF) model [22] was trained on the BoS representation. In addition to the classification, the RF model measures the variable importance using the average Gini information gain enabling the identification of important features. RF also provides continuous score between 0 and 1 for the grade classification, which is meaningful in the real experimental environment, as the tumor samples could be diluted by normal cells, and that may reduce the confidence of the classification.

Table 3.2: Examples for the Final Bag-of-Segments Representation: the upper table provides examples for the representation based on the raw counts over 9 segment classes; the lower table presents the final BoS representation based on the frequency distribution

	MA	MD	MN	NA	ND	NN	WA	WD	WN	Grade
1	9	16	9	9	1	2	2	1	1	High
2	1	1	2	1	2	0	1	1	18	Low
3	0	4	2	0	2	1	4	3	12	Low
4	0	0	1	0	1	0	3	0	19	Low
...	...	...	...	...	...	...	...	...	...	...

	MA	MD	MN	NA	ND	NN	WA	WD	WN	Grade
1	0.1800	0.3200	0.1800	0.1800	0.0200	0.0400	0.0400	0.0200	0.0200	High
2	0.0370	0.0370	0.0741	0.0370	0.0741	0.0000	0.0370	0.0370	0.6667	Low
3	0.0000	0.1429	0.0714	0.0000	0.0714	0.0357	0.1429	0.1071	0.4286	Low
4	0.0000	0.0000	0.0417	0.0000	0.0417	0.0000	0.1250	0.0000	0.7917	Low
...	...	...	...	...	...	...	...	...	...	...

3.5 Experimental Results

3.5.1 Model Evaluation and Sensitivity Analysis

Because of the small sample size, we estimate the generalization of our method through leave-one-out cross-validation (LOOCV). The Cp and α are two critical parameters affecting our results: Cp controls the complexity of the regression tree in the segmentation step, and α is the quantile percentage for defining CNA segment classes. We perform sensitivity analysis with different Cp values (0.001, 0.005, 0.01, 0.05, 0.1, 0.2) and different α values (0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45). For each combination, LOOCV using RF model was repeated 10 times to obtain the average accuracy and the results are presented in Fig. 3.5.

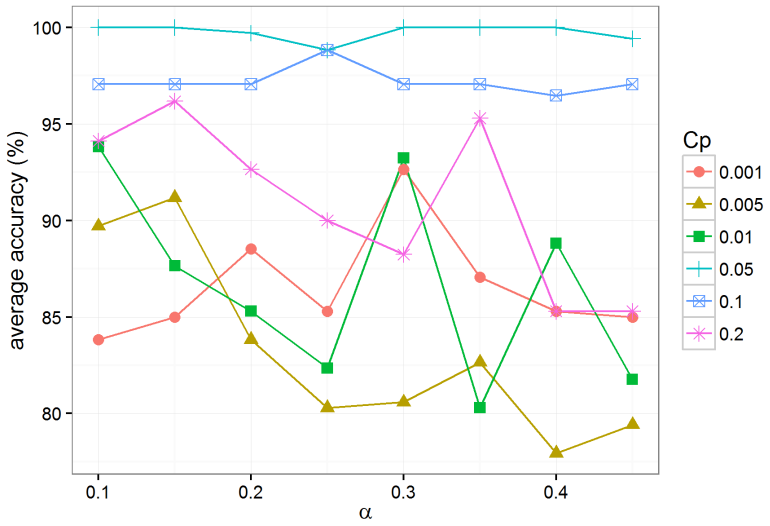


Figure 3.5: Sensitivity Analysis with Various α and Cp: for each combination the average leave-one-out cross-validation accuracy over 10 experimental replicates is reported

Our method shows high accuracy on the ovarian serous carcinoma classification study. Over 98% LOOCV accuracy was obtained in various parameter combinations. It is shown that our method performs the best when $Cp = 0.05$. It achieved 100% accuracy with multiple different α values. We also observe that the accuracy of our method is more sensitive to the choice of Cp than the choice of α. When Cp is properly selected ($= 0.1$ or 0.05), our method works well with a wide range of α values. From this experiment, we see that many combinations of these two parameters can work well in the context of CNA sequencing data classification. It shouldn't be difficult to find a good parameter choice for another problem. For this study of ovarian serous carcinoma, $Cp = 0.05$ is recommended.

3.5.2 Evaluation on the Reduced Feature Sets

In addition to building a classifier, we may be interested in a reduced number of features as clinical metrics. Three sets of features derived from the original 9-feature BoS representations are considered here for the evaluation. The first feature

set consists of the most important features based on the RF importance score as depicted in Fig.3.6c. According to RF importance, feature Narrow Amplified (NA) and feature Wide Normal (WN) are the most important features. The second feature set, amplified, deleted, normal, is another BoS representation where the segment classes are only defined by the height quantiles. Similarly, the third feature set is narrow, medium, wide which is defined by the width quantiles. We can also consider the second and the third feature sets as a linear combination of the 9-feature BoS representation as explained in Table 3.1. In our experiments, $Cp = 0.05$ and $\alpha = 0.25$ are used, and LOOCV with RF model was repeated 10 times to obtain the average accuracy for each feature set.

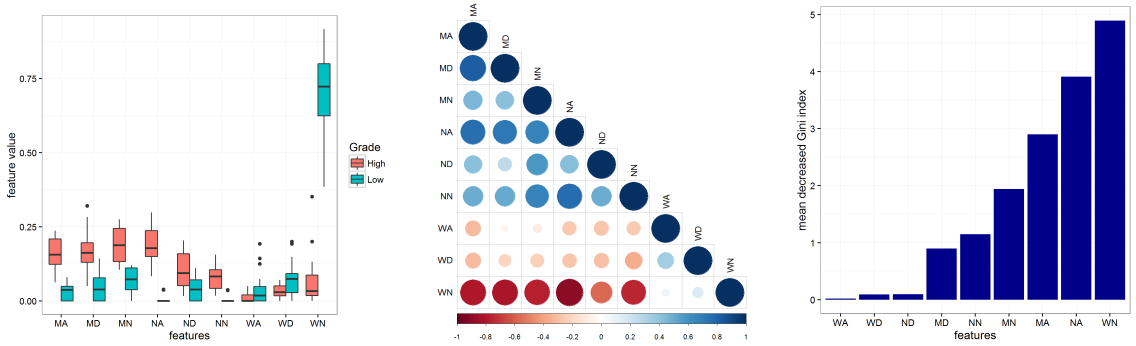
Table 3.3: Classification Accuracy Comparison with Different Feature Sets

features sets	mean LOOCV accuracy (%)
all	98.82353
Top 2 RF features {Narrow Amplified, Wide Normal}	99.11765
height features (deleted, normal, amplified)	91.47059
width features (narrow, medium, wide)	100

The average LOOCV accuracies of the three feature sets are presented in Table 3.3. The top 2 RF features features can already achieve 99.12% average LOOCV accuracy. Also, we observe that the width-defined features are more important than the height-defined features. The width-defined features themselves are already able to achieve 100% accuracy on average, while the average accuracy of the height-defined features is only 91.47%. Therefore, {Narrow Amplified, Wide Normal} or the width features can be potentially used as clinical metrics.

The good performance of the width-defined features also implies that our method can still be useful when the tumor samples are diluted by the normal cells. In those scenarios, the signal contained in the segment heights gets weakened, while the signal in the segment width is still preserved.

3.5.3 *Biological Interpretation of Bag-of-Segments*



(a) high-grade vs. low grade (b) correlation plot (c) RF feature importance plot

Figure 3.6: The Comparisons on the Bag-of-Segment Features

To understand the biology behind the BoS features, we provide the feature correlation plot in Fig.3.6b and the feature comparison box-plot between the low grade- and the high grade-samples in Fig.3.6a. From Fig.3.6b we observe that narrow-width and medium-width features are positively correlated with each other, and negatively correlated with the wide-width features. This divides the features into two groups, representing two underlying biological patterns. The narrow- and medium- width features are associated with local amplification and deletion, while the wide-width features describe the process of large scale deletion or amplifications. Furthermore, from the boxplot in Fig.3.6a, we discover that there are more local amplification and deletion in high-grade samples, compared to the low-grade samples where more largescale deletion and amplification appear.

3.6 Conclusion

In this chapter, we develop a method to classify the ovarian serous carcinoma into high grade and low grade with high accuracy. BoS is proposed to represent the information of the CNAs sequencing coverage data. The new representation

summarizes each sequencing coverage data into 9 features predictive of the tumor grades. The relationship between the extracted features and the underlying biological processes are discussed. Furthermore, we identify some smaller sets of the features which can potentially be used as clinical metrics in practice. This method specifically may be beneficial in classifying ovarian serous carcinoma in patients with ambiguous histologic features. In such challenging cases, prediction of a low versus high grade disease by this method may provide valuable clue for patient clinical management. In the next chapter, we discuss the limitations of BoS, and extend BoS to a more powerful representation method with less assumption on the data.

Chapter 4

LSDF: LOCATION-AWARE SUPERVISED DICTIONARY LEARNING BY FOREST FOR SEQUENCING DATA

4.1 Motivation

In the previous chapter, we developed a simple data representation method, BoS, for the grade classification of the ovarian serous carcinoma. This representation leads to high classification accuracy. BoS assumes that the location of a specific type of segment is unimportant for the classification. For example, the two time series segmentations shown in Fig. 4.1 would have identical BoS representation. This assumption may be too strong for the applications where the locations of the local events are actually informative for the classification decision.

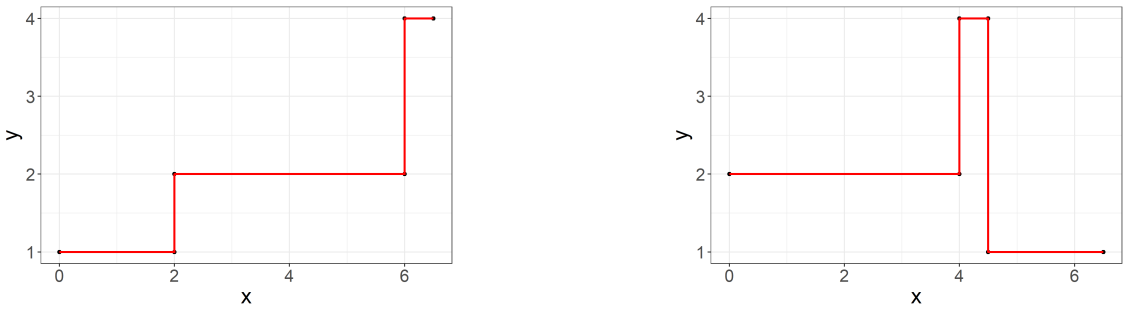
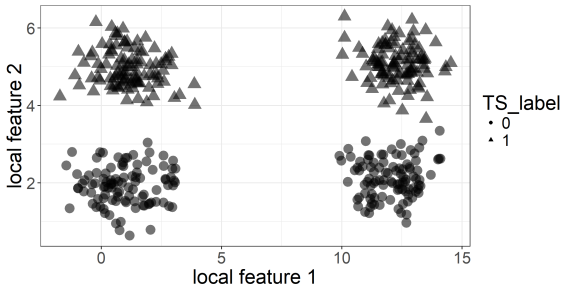


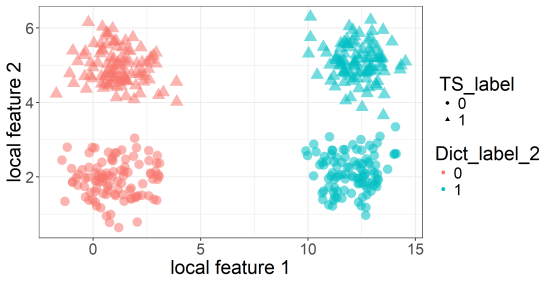
Figure 4.1: Two Time Series Segmentation Results that Have Identical BoS Representation

Also, in BoS, the dictionary (the collection of several types of segments) is defined in an unsupervised manner, which doesn't ensure the prediction power of the representation. For example, suppose 2 local features extracted from the segments obtained from two time series of different labels are distributed as that in Fig. 4.2a. If we generate a dictionary of size 2 using a Euclidean-distance-base clustering al-

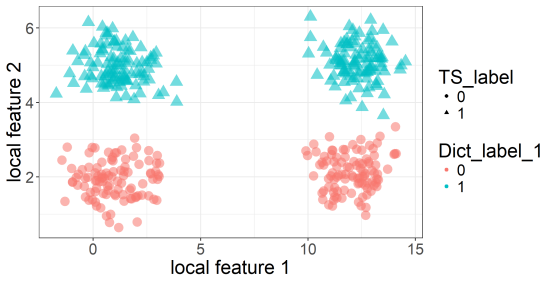
gorithm, we can obtain the clusters shown in Fig. 4.2b, and two time series have identical representation (0.5, 0.5). If we defines the dictionary differently as shown in Fig. 4.2c, the representation for two time series will be (0, 1) and (1, 0). This is a better representation as it is discriminative of the class labels. The second representation can be obtained by a supervised method which takes the labels of the original time series into account.



(a) Local features



(b) Unsupervised dictionary learning



(c) Supervised dictionary learning

Figure 4.2: An Example to Demonstrate the Advantage of the Supervised Dictionary Learning: the different shapes indicate different time series classes, and the colors indicate the groups used to define dictionary. Here a supervised method (c) will result in a more discriminative representation compared to an unsupervised method (b).

With these motivations, in this short chapter we present a Location-aware Supervised Dictionary learning algorithm using Random Forest (LSDF) as the extension to the previous chapter. We apply our method to the grade classification problem using the UCEC dataset obtained from the TCGA network [4] to demonstrate its effectiveness.

4.2 Method

Similar to the work flow in the previous chapter, the classification method using LSDF representation consists of three steps: 1) segmentation of the time series, 2) constructing the LSDF representation and 3) training a classification model.

The first and the third steps are identical to the ones for BoS as discussed in Section 3.4.1 and Section 3.4.3. As a reminder, we used a regression-tree based approach for the segmentation and a random forest model as the final classification model. Here our discussion focuses on the LSDF representation.

To learn the dictionary supervisely, the first step is to construct a labeled segment-level dataset. Each segment is described by its height, width as well as the location of the segment center if the location is considered important for the classification. For the training set, each segment is labeled by the class label of the time series from which it is segmented. The resulted segment-level dataset has the form of the data table shown in Table 4.1.

Table 4.1: Example for a Segment-level Dataset

segment ID	width	height	center location	label
1	100	50	50	A
2	40	10	120	A
...	...	...	...	...
26	10	100	5	B
...	...	...	...	...

A RF model consisted of T trees, with each tree containing D leaf nodes, is trained on this segment-level dataset. For a time series instance, each tree in the RF generates a length-D vector representation using the leaf node distribution of its segments when passed through the tree model. The final representation from

the RF model is generated by concatenating all the tree representations. Therefore, the final LSDF representation for each time series instance has a length of $T \times D$. Table 4.2 gives a few examples for the final representation. Each feature in the final representation represents a certain type of segment (at certain region of the time series, within certain ranges for the segment height and the segment width).

Table 4.2: Examples for the LSDF Representation

times series ID	tree 1				tree 2			...	tree T			label
	leaf 1	leaf 2	...	leaf D	leaf 1	...	leaf D	...	leaf 1	...	leaf D	
1	20%	10%	...	50%	5%	...	20%	...	10%	...	30%	A
2	10%	15%	...	5%	50%	...	10%	...	40%	...	5%	B
3	0%	10%	...	50%	10%	...	20%	...	8%	...	25%	A

4.3 TCGA UCEC Data

For our experiments, we use uterine corpus endometrial carcinoma (UCEC) data generated by the TCGA Research Network [4] to form a classification problem. Compared to the CNA profile data used in Chapter 3, this data is subjected to more noise. However, it has more genome sequencing samples than the previous study and therefore allows more complex modeling techniques. We consider a binary classification problem, where the original labels "G1" and "G2" are relabeled as the low-grade class, "G3" and higher grades are relabeled as the high-grade class. As the result, we have 191 low-grade instances and 301 high-grade instances.

The sequencing data provided by TCGA has already been segmented, so we skip the segmentation step. When considering the location information, we need to generate the dictionary and representation for each chromosome separately and then concatenate them, resulting in a total of $23 \times T \times D$ features. If the location is not considered, the number of features in the final representation can be reduced to $T \times D$.

4.4 Experimental Results

We perform 5-fold cross-validation (CV) to estimate the performance of each method. A default RF model with 500 trees is used as the classification in all the experiments. The following 3 data representation methods are compared: 1) unsupervised dictionary learning with K-means clustering; 2) supervised dictionary learning without the location information; 3) location-aware supervised dictionary learning method (LSDF).

For the unsupervised dictionary learning with K-means algorithm, different K values are tested. Because the location information is not considered, the clustering is only based on the segment height and segment width after the standardization. The 5-fold cross-validation accuracy for different K values is presented in Table 4.3. The highest accuracy is obtained when $K = 500$. The best CV accuracy for the unsupervised dictionary learning method is 0.715.

Table 4.3: CV Accuracy for K-means Dictionary Learning with Different K

K	10	20	50	100	500	1000	2000	3000	5000
CV accuracy	0.685	0.663	0.667	0.687	**0.715**	0.699	0.671	0.691	0.665

For the supervised dictionary learning, different combinations of T and D values are experimented. The number of the trees T considered for LSDF (method 3) is smaller than method 2 (the version without considering location information) because the total number of features is scaled up by factor of 23 due to the number of the chromosomes.

As we can observe from the results, the supervised method improve the accuracy over the unsupervised method, and the accuracy is further improved by taking the location information into account in this particular case. The best CV accuracy is obtained using the LSDF method with $T = 10$ and $D = 30$. The best CV accuracy

Table 4.4: CV Accuracy for Supervised Dictionary Learning without Location Information with Different T and D Values

		D					
		10	20	30	50	100	200
T	10	0.689	0.709	0.713	0.715	0.726	**0.734**
	50	0.691	0.722	0.709	0.715	0.722	0.711
	100	0.695	0.715	0.711	0.717	0.72	0.709

Table 4.5: CV Accuracy for Supervised Dictionary Learning with Location Using Different T and D Values

		D					
		10	20	30	50	100	200
T	5	0.738	0.717	0.74	0.736	0.728	0.734
	10	0.732	0.752	**0.758**	0.742	0.726	0.711
	50	0.742	0.732	0.746	0.754	0.742	0.709

Table 4.6: Confusion Matrix for the LSDF Representation with $T = 10$ and $D = 30$

	predicted	
actual	low grade	high grade
low grade	124	67
high grade	52	249

obtained by LSDF is 0.758, higher than that obtained by the unsupervised method (0.715). The corresponding confusion matrix is presented in Table 4.6. From the confusion matrix, we observe that the high grade samples are easier to predict than the low grade samples.

4.5 Summary

This chapter is an extension of the BoS representation method proposed in the previous chapter. It aims to extract more complex and more discriminative representation from time series data using a supervised method, which may require a larger dataset in practice. The case study on the TCGA UCEC dataset demonstrates that supervised dictionary learning and incorporating the location information can improve the predictive power of the representation and result in better performance for the machine learning models.

Chapter 5

GCRNN: GROUP-CONSTRAINED CONVOLUTIONAL RECURRENT NEURAL NETWORK FOR TIME SERIES CLASSIFICATION

5.1 Introduction

Time series classification (TSC) is a classification task where the predictor attributes are ordered. Compared to traditional classification problems, TSC is more challenging due to the high-dimensional, ordered attributes. Typically the attributes are auto-correlated, which implies redundancy in the data. Also, in contrast to other classification tasks, a good TSC solution might rely on features related to the order of the attributes.

TSC is important in many domains such as health care, manufacturing, finance, etc. For example, for the seizure onset detection in health care, the primary tool is based on the analysis of the EEG signal. In manufacturing systems, most monitoring systems are based on the time series data collected from sensors.

The increase of publicly available temporal datasets promotes active research in TSC in the last decade. Many researchers have tackled this problem through various techniques. For example, there are distance-based classifiers such as Weighted Dynamic Time Warping [71], Mover-Split-Merge [72], dictionary-based classifiers such as Bag of Patterns [73], shapelet-based classifiers such as Shapelet Transform [74], and interval-based classifiers such as Time Series Forest [75] and Learned Pattern Similarity [76]. A good review for the recent developments is provided by [12].

Also, previous research did not focus on the interpretation of the time series patterns, which, however, is important for some applications. For instance, in a task

where the genomic sequences are classified to different cancer types, we not only want to obtain high accuracy, but also to understand where the discriminative patterns appear in the sequence. One notable method along this effort is Sparse Multivariate Tree (SMT) [77], where the time series classifier also provides location information for the mean, slope and deviation patterns. The importance of the interpretability of the time series classifier also motivates the development of this work.

Recently, deep learning techniques, particularly a Convolutional Neural Network (CNN) (originally developed by [13]) and a Recurrent Neural Network (RNN) proposed in the 80s [14] [15], have received interest in areas such as computer vision and natural language processing. Refinements for RNNs use long short term memories (LSTMs) [28] and gated recurrent units (GRUs) [29].

Consider the applications of CNNs or RNNs to time series data. CNNs are attractive for their feature learning ability. However, they do not fully explore the temporal characteristics of time series data. RNN can effectively model the sequencing data, but the dependencies in the raw time series may be too long for RNN to capture. Combining two networks allows us to leverage the advantages of both CNNs and RNNs. That is what this research aims to accomplish.

Here we present a new architecture, a general end-to-end deep neural network model for general TSC tasks. We named our neural network model Group-constrained Convolutional Recurrent Network (GCRNN). GCRNN takes the raw time-series as the input without any feature engineering. Both the features and the classification model are learned jointly through the training of the entire neural network. GCRNN not only leads to good classification accuracy, it also highlights the most important and discriminative time-series regions for the TSC task. That is a novel contribution of this work, because neural networks are usually considered "black-box" models, and are often difficult to interpret.

GCRNN breaks down to three key components: a CNN module, a RNN module and a fully connected module with a group lasso penalty. More specifically, the CNN module learns the task-specific features through the training, and the RNN module enhances the modeling of the temporal characteristics of time series. To connect a CNN with a RNN, the outputs of the CNN module need to be redistributed in time. This is handled with a new architecture. Finally, the outputs of RNN layers are fully connected to the final layer of different classes. A sparse group lasso (SGL) penalty is imposed on this final layer to discover the most discriminative regions as well as to reduce the model complexity. Therefore, there are two key contributions in this work. First, we propose a new architecture, an end-to-end system for general time series classification. Second, our model is designed to be interpretable to highlight the discriminative patterns.

The remainder of this chapter is organized as follows. Section 5.2 reviews the related work and background knowledge. Section 5.3 introduces the GCRNN model. Section 5.4 presents the experimental results and Section 5.5 discusses how our model can be applied to real-world problems. Finally, Section 5.7 provides conclusions.

5.2 Related Work

5.2.1 Time Series Classification

At a higher level, most existing work for TSC follows two types: distance-based and feature-based. The key part of distance-based methods is to measure the similarity between two time series, and apply kernel-based classifiers such as k-Nearest Neighbors (kNN) [78] or Support Vector Machine (SVM) [79]. A well-known approach to measure time-series similarity is through Dynamic Time Warping (DTW) [80], based on an optimal matching. For feature-based approaches, features are ex-

tracted from different window sizes at different regions of the time series. The features could be based on simple statistics such as mean and variance [75] [38], or more complicated ones like the spectrogram. These methods separate the feature extraction part from the classification part, and the performance relies heavily on the quality of these handcrafted features. Different tasks may require different sets of features to obtain good classification performance. A principled and systematic way to design features according to a specific dataset is not obvious. Ideally, the features should be learned according to the task, which motivates the development of our model. GCRNN learns the task-specific features through training, which frees the modeling from feature engineering.

5.2.2 Deep Neural Networks

CNN has been extensively studied in image-related tasks, such as object detection [81] and face verification [82]. Previous study has shown that CNN has strong capability to learn meaningful features [83]. However, its applications to TSC is more recent. In [84], a multi-channel CNN was proposed, where a time series is put into multiple CNNs and then the extracted features are concatenated. To train a complex CNN model like this, large datasets are needed, which, unfortunately, is not the case in many domains. RNN has led to impressive results in various tasks with sequence data, such as machine translation [35], time series prediction [85] and time series anomaly detection [86]. The invention of LSTM [28] and GRU [29] units makes RNN applicable to model long-term dependency and overcomes the vanishing and exploding gradient problem [27] of training deep networks. Applying RNN to time series can benefit from the extraction of high-level features from the raw time series, which further reduces the burden of modeling long-term dependency. In the proposed GCRNN model, the features are learned jointly through the CNN module.

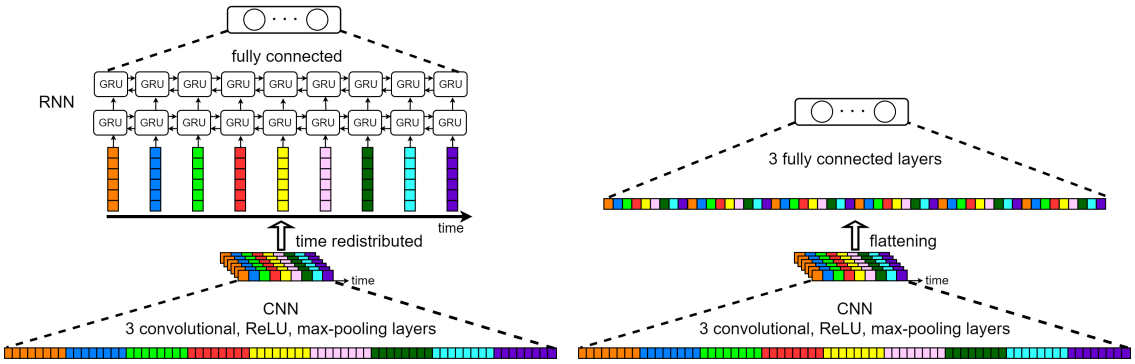
There are some existing researches that connect CNN and RNN in an end-to-end system. For image domain, such system was used in image captioning [87] and visual question answering [88]. For speech recognition, [89] utilized both a CNN and a RNN, and used the spectrogram of the speech signal as the network input. Instead of forcing the model to use the full spectrum, the input of the proposed GCRNN model is the raw time series with no feature engineering. With the unique network architecture, only the relevant frequencies are utilized through the learned filters in the convolutional layers.

In spite of the strong performance of neural network models, they are discouraged in some applications due to lack of interpretability. Along the efforts to understand how neural networks work, an attention mechanism was leveraged in [87] to uncover attended image regions of the neural network when generating captions for images. An attention mechanism was also introduced in [90] to model which part of sequence to attend to in a machine translation task. For a similar goal, in this work, we propose to use the sparse group lasso to understand the network's attention in TSC tasks.

5.3 GCRNN Architecture

5.3.1 Overview

A time series of length T is denoted as $\boldsymbol{x} = \{x_1, x_2, \ldots, x_T\}$, where x_t is the value(s) at time stamp t, and x_t is a scalar for a single-channel time series or a vector for time series with multiple channels. We assume that multiple time series are of the same length (otherwise they are down-sampled or interpolated to equal length). A time series dataset is denoted as $D = \{\{\boldsymbol{x}_n, y_n\}_{n=1}^{N}\}$ which contains N time series, each associated with label y_n. For a C-class classification problem, $y_n \in \{1, ..., C\}$.



(a) GCRNN network (b) Benchmark CNN network

Figure 5.1: Network Architectures of the GCRNN and the Benchmark CNN. The bottom layer is the input time series with 1 channel, but both networks can work with multi-channel time series. The CNN component includes 3 convolutional, ReLU and max-pooling layers which are not shown here. With 3 max-pooling layers with non-overlapping size-2 windows, each input region of 8 time steps maps to one unit before the RNN layers

A robust classifier for time series data should be invariant to small distortion and translation in the data. It should model the temporal characteristics of the time series, but be aware of the redundancy contained in the data. Moreover, interpretability is also a desirable property for a TS classifier. In some applications, we are interested in not only the good classification accuracy, but also what discriminative patterns contribute to the classification tasks.

GCRNN achieves these goals through three stacked modules: a CNN module, a RNN module and a fully connected module with sparse group lasso penalty. The CNN module generates the task-specific features. Thanks to the max-pooling layers, the model is invariant to small distortion and translation in the data [91]. The RNN module contributes to the modeling of the temporal characteristics. The SGL imposed on the fully connected layer uncovers the important regions of the time series, which makes the model more interpretable while less complex. In greater details, the following subsections introduce each of the three modules.

5.3.2 CNN Module

The CNN part is very similar to the architecture in LeNet [13][92], but in one dimension. It consists of K convolutional, rectified linear units (ReLU) [25] and max-pooling layers. The inputs of the CNN module are the raw time series. Suppose that each input time series has m_0 channels, and let m_k and s_k denote the number of filters and the length of the filters at the k-th convolutional layer, for $k = 1, ..., K$. Therefore, the trainable parameters of the CNN module are composed of the weight matrices of size $m_k \times m_{k-1} \times s_k$ and threshold vectors of size m_k, for $k = 1, ..., K$.

The ReLU activation function, $f(x) = \max(0, x)$, is applied to the outputs of each convolutional layer for the nonlinearity, followed by a max-pooling over non-overlapping intervals of length s_{MP}. A max-pooling operation pools the maximum over each interval as its output. Thus, the length of the output is reduced by a factor of s_{MP}. For instance, as depicted in Fig. 5.1a, after three max-pooling layers with $s_{MP} = 2$, the feature vectors at the outputs of the CNN module are approximately of length $T' \approx T/8$.

5.3.3 Connecting CNN with RNN (Redistributed in Time)

The feature vectors at the output of the CNN module are denoted as $\mathbf{F}_j, j = 1, \ldots, m_K$. Each feature vector has length T', so we have $\mathbf{F}_j = (F_{j,1}, F_{j,2}, ..., F_{j,T'})$. Note that each feature vector is a higher level summary of the original time series with a shorter length along the time axis. For example, in Fig. 5.1a, the first element of each feature vector (in orange) approximately summarizes the information from the starting of the time series (also in orange). Therefore, we can redistribute the elements of the feature vectors along the time axis. Specifically, we reconstruct T' feature vectors, and each corresponds to a time stamp from 1 to T'. They are $\mathbf{F}(t) =$

$(F_{1,t}, ..., F_{m_K,t}), t = 1, ..., T'$. The new feature vectors are in the meaningful order of time, which can be further modeled by the RNN module.

5.3.4 RNN Module

The RNN module consists of two layers of bidirectional RNNs, and each layer can be unrolled to T' GRU units along the time axis. Because of the bidirectional property, each unit accumulates the information from both sides of the time series, with an emphasis on its own corresponding region. So each unit captures both local and global information, which makes it reasonable to apply a group lasso penalty (discussed later). The extensive comparison in [93] showed no significant difference between the LSTM unit and the GRU unit. The time series data sets studied here are not large (with less than 1000 instances). Therefore, we choose the GRU because it has fewer parameters to train.

5.3.5 Fully Connected Module with Sparse Group Lasso

The output vectors from all the GRU units in the second RNN layer are fully connected to the C units in the final layer, each representing a class in the TSC task. Also, Dropout [94] is applied to this fully connected layer to mitigate overfitting.

Our final loss function to be minimized consists of two parts: a cross-entropy term representing the classification error and a penalty term for the regularization

$$\text{loss} = \text{cross entropy} + \text{penalty}. \tag{5.1}$$

Suppose that passing $\{\boldsymbol{x}_n, y_n\}$ through the network gives the outputs $o_n(1), o_n(2), \ldots, o_n(C)$, and the batch stochastic gradient decent is used, then the cross-entropy term can be written as

$$\text{cross entropy} = -\frac{1}{|\text{Batch}|} \sum_{n \in Batch} \log o_n(y_n). \tag{5.2}$$

We propose to use a sparse group lasso (SGL) penalty [95] to regularize the weights of the final, fully connected layer. We consider the weights associated with each GRU unit as a group. A SGL penalty blends the lasso (L1 norm) with the group lasso (L2 norm) penalties in order to achieve a sparse solution at both the group and the individual levels. The group lasso (L2) shrinks all the weights from the same GRU if its corresponding region in the time series is not discriminative. A relatively high L2 norm from a group implies the high importance of a specific region. Within a group, lasso (L1) also selects the important individual features. A parameter α determines the weights between these two components, and a higher α increases the penalty on groups.

The regularization term is

$$\text{penalty} = \lambda\alpha \sum_{t=1}^{T'} \sqrt{p_t}||\mathbf{W}_t||_2 + \lambda(1-\alpha)||\mathbf{W}||_1, \tag{5.3}$$

where $\mathbf{W}_t$ is the weight matrix between the output vector of the t-th GRU unit and the C class units. Specifically, $\mathbf{W}_t = \{\mathbf{w}_{t1}, \mathbf{w}_{t2}, ..., \mathbf{w}_{tC}\}$ for $t = 1, ..., T'$. Each $\mathbf{w}_{tk}$ is a vector whose length depends on the output dimension of the GRU unit, and p_t is the total number of parameters in $\mathbf{W}_t$. Here $\mathbf{W}$ represents the union of all the parameters at this fully connected layer. That is $\mathbf{W} = \{\mathbf{W}_1, ..., \mathbf{W}_{T'}\}$. The threshold terms are neglected here for the simplicity.

Because every unit encodes the information from the entire time series, but with a different concentration/attention, a simpler model using only a few groups may have captured all the important information for the classification. This is the objective of the SGL regularization. Furthermore, the SGL provides feedback on the location of the discriminative patterns in the original time series that conveys some useful insights about the data, e.g., how the classes differ from each other.

5.3.6 *Benchmark Networks*

This section lists the benchmark networks to be compared in our experiments. GCRNN is first compared to a conventional CNN whose architecture is presented in Fig. 5.1b. The CNN has identical structure for the CNN module compared to GCRNN, but the two RNN layers in GCRNN are replaced with two fully connected layers. This CNN architecture corresponds to the 1D CNN described by [96]. The purpose of this benchmark is to demonstrate the effectiveness of RNN layers in the temporal modeling, as well as the advantage of our joint CNN-RNN architecture.

In addition, several variations of GCRNN are considered in our experiments. When $\lambda = 0$, GCRNN doesn't have any penalization. This simplified model is refered to as a Convolutional Recurrent Neural Network (CRNN) hereafter. When $\lambda > 0$, different values for α also result in different models. One special case is when $\alpha = 0$, the SGL penalty is simplified to a lasso penalty without any group-level regularization. Both variations are compared in our experiments.

5.4 Experiments

For the convolutional module in both the CNN and the GCRNN models, we have $K = 3$ convolutional, ReLU, max-pooling layers. For the size of the filters, we set $s_1 = 21, s_2 = 11, s_3 = 5$, and for the number of the filters, we set $m_1 = 5, m_2 = 10, m_3 = 20$. The window size for the max-pooling, s_{MP}, is set to 2. For GCRNN, the dimension of the hidden memory size of the GRU unit is set to 20. Dropout [94] is applied on the last fully connected layer in the GCRNN model and the last two fully connected layers in the CNN model to alleviate the overfitting problem. The Adagrad method [97] is used in the batch stochastic gradient descent, and the batch size is set to 50 for all the data sets.

In this preliminary study, the same network structure for GCRNN is tested for all the data sets, , resulting in competitive accuracies. Additional tuning in the network settings for each data set could potentially achieve better accuracy. However, because the focus here is to illustrate the advantages of combining RNN with CNN, we did not experiment with many other parameter settings.

5.4.1 Time Series Datasets

We experimented on 14 time series datasets from the UCR time series repository archive [2]. Their characteristics are summarized in Table 5.1. Each data set has been divided to a training set and a testing set by the provider. Only the training set is involved in the training process of all the networks, and the testing set is tested after each training epoch to provide the results presented in this chapter.

All the time series datasets have only one channel, so that $m_0 = 1$ in our models. But it is worth noting that our model is capable to handle multivariate time series in an intuitive manner.

5.4.2 Experimental Results

We compare the following 6 models on the UCR datasets: GCRNN with $\alpha = 0.0$, GCRNN with $\alpha = 0.25$, GCRNN with $\alpha = 0.50$, GCRNN with $\alpha = 0.75$, CRNN, and CNN. We use $\lambda = 0.001$ for all the GCRNN models. Each model is trained and tested 10 times with the default training-testing sets split. Fig. 5.4 presents the median testing accuracies over the first 200 training epochs for each model on each dataset. From Fig. 5.4, it can be observed that GCRNN/CRNN models outperform CNN with a significant advantage in the testing accuracies in 10 out of 14 datasets. In the other 4 datasets (Ham, Haptics, Yoga, HandOutlines), GCRNN/CRNN and CNN perform almost equally at the end of the training. This comparison provides

Table 5.1: Characteristics of the UCR Time Series Datasets

Datasets	Number of classes	Size of training set	Size of testing set	Time series length
Computers	2	250	250	720
FISH	7	175	175	463
FordB	2	810	3636	500
Ham	2	109	105	431
HandOutlines	2	370	1000	2709
Haptics	5	155	308	1092
LargeKitchenAppliances	3	375	375	720
OSU Leaf	6	200	242	427
RegrigerationDevices	3	375	375	720
ScreenType	3	375	375	720
uWaveGestureLibrary-X	8	896	3582	315
uWaveGestureLibrary-Y	8	896	3582	315
uWaveGestureLibrary-Z	8	896	3582	315
Yoga	2	300	3000	426

strong evidence for the effectiveness of the proposed GCRNN architecture for TSC. Fig. 5.4 also reveals that different variations of GCRNN perform quite similarly. One variation might performs slightly better than the others for a specific dataset.

To facilitate the comparison across different datasets, Fig. 5.2 shows box plots of different models in terms of the normalized accuracies. We define the normalized accuracy as the following. For each dataset, the median accuracy in the last 50 training epochs for each model is first calculated, then normalized by the highest median accuracy among the 6 models. Therefore, the best model achieves 1 for the normalized accuracy, while the others receive a value less than 1. From Fig. 5.2, similar conclusions to the previous ones can be drawn. That is, GCRNN/CRNN models significantly outperform the CNN model. Comparing the GCRNN model with $\alpha = 0.75$ to the CRNN model, we observe that the SGL regularization slightly improves the classification accuracy.

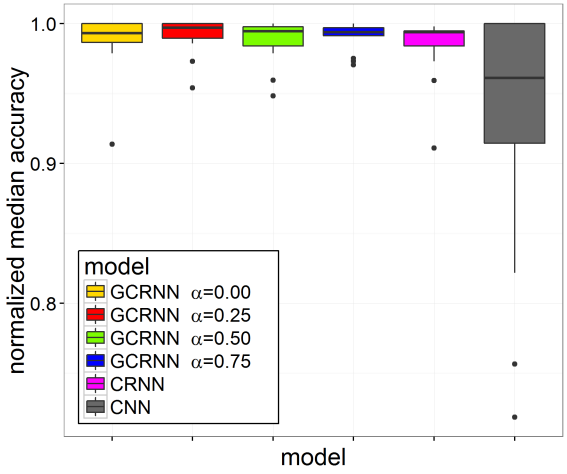


Figure 5.2: Normalized Median Testing Accuracy for 6 Neural Network Models on 14 TS Datasets

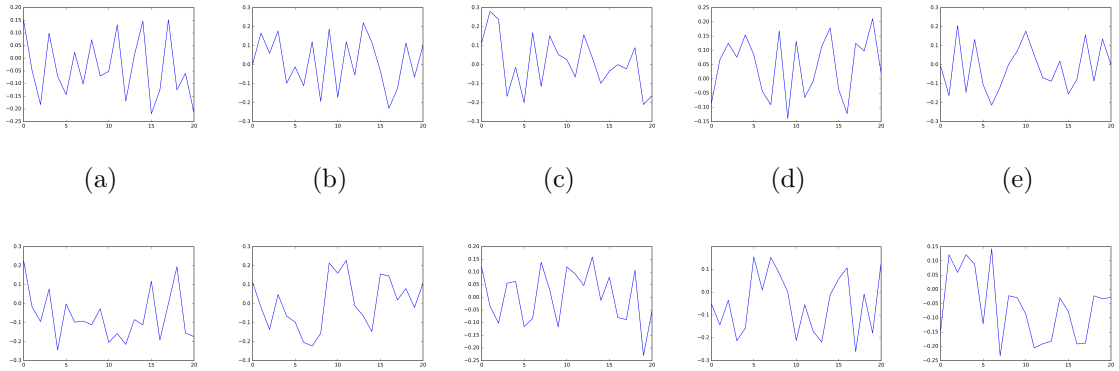
Although the goal of this work is not to improve the prediction results in specific datasets, for reference purpose Table 5.2 compares the median testing accuracy of the last 50 epochs of the GCRNN and CNN models to the accuracies reported by other popular TSC methods. Here two versions of Nearest Neighbour Classifier with DTW distance are considered: NNDTWBest [98] searches the best warping windows using the training data, while NNDTWNoWin doesn't use warping window. We should note that DTW is considered a strong solution for TS problems in many domains [99]. In addition, we compare them to TSBF (a TS classifier with Bag-of-Features algorithm) [38] for the datasets also reported in their paper. TSBF is considered one of the state-of-the-art approaches for TSC. We observe that GCRNN models outperform the two NNDTW methods by a significant advantage for 12 out of 14 datasets (except for the Computer and Yoga datasets). Although GCRNN methods only win TSBF in 1 out of 7 datasets, the difference between their accuracies is quite small across all the datasets. Considering that the performance of GCRNNs can be further improved by applying early stopping [100] or tuning the network architectures, GCRNN has the potential to be a very strong time-series classifier.

Table 5.2: Performance Comparison Among GCRNN/CRNN, CNN, NNDTWBest, NNDTWNoWin and TSBF: the median testing accuracy of the last 50 epochs is report for GCRNN/CRNN and CNN, and the testing accuracy of NNDTWBest, NNDTWNoWin and TSBF were reported in [2] and [38]

Datesets	GCRNN $\alpha = 0.00$	GCRNN $\alpha = 0.25$	GCRNN $\alpha = 0.50$	GCRNN $\alpha = 0.75$	CRNN	CNN	NNDTW -Best	NNDTW -NoWin	TSBF
Computers	0.680	0.692	0.688	0.688	0.680	0.592	0.620	0.7	*
FISH	0.931	0.926	0.926	0.926	0.926	0.886	0.846	0.823	0.853
ForB	0.906	0.905	0.906	0.904	0.904	0.883	0.586	0.594	*
Ham	0.695	0.686	0.676	0.686	0.686	0.705	0.600	0.467	*
HandOutlines	0.851	0.850	0.847	0.848	0.847	0.851	0.803	0.798	*
Haptics	0.448	0.451	0.448	0.455	0.455	0.458	0.412	0.377	0.515
LargeKitchenAppliances	0.841	0.845	0.853	0.832	0.819	0.613	0.795	0.795	*
OSULeaf	0.764	0.764	0.760	0.760	0.760	0.579	0.612	0.591	0.671
RefrigerationDevices	0.493	0.499	0.491	0.484	0.491	0.456	0.440	0.464	*
ScreenType	0.424	0.443	0.440	0.464	0.423	0.381	0.411	0.397	*
uWaveGestureLibrary_X	0.809	0.813	0.812	0.811	0.810	0.802	0.773	0.727	0.836
uWaveGestureLibrary_Y	0.735	0.738	0.740	0.734	0.736	0.712	0.699	0.634	0.751
uWaveGestureLibrary_Z	0.745	0.750	0.743	0.745	0.744	0.717	0.678	0.658	0.783
Yoga	0.842	0.833	0.839	0.840	0.839	0.842	0.845	0.836	0.851

5.4.3 Task-specific Features

The CNN module learns task-specific features through the training. To illustrate this point, the learned filters at the first layer of the CNN module are presented in Fig. 5.3 for two datasets. These filters were obtained from the GCRNN model with $\alpha = 0.75$. Fig. 5.3 (a)-(e) depict the filters of the first convolutional layer for the FordB dataset, and Fig. 5.3 (f)-(j) present the filters for the LargeKitchenAppliances dataset. We observe that different patterns are learned for different classification tasks. For the same dataset, different filters are learned to capture different characteristics. For instance, Fig. 5.3 (g) represents a relatively low-frequency pattern learned by the GCRNN model, while a relatively high-frequency pattern is shown in Fig. 5.3 (h).



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure 5.3: Learned Filters for the First Convolutional Layer for FordB Dataset and LargeKitchenAppliances Dataset: (a)-(e) are the filters for the FordB dataset, and (f)-(g) are the filters for the LargeKitchenAppliances dataset

5.4.4 Model Interpretation

With the SGL, the improvement in the classification accuracy is not large. However, the gain in terms of the model interpretation is much more valuable. Consider the examples in Fig. 5.5. Each sub-figure presents the time series of a dataset, with different colors indicating different classes. The depth of the background color tells the attention/importance of the regions uncovered by the SGL for the classification model. The deeper the color is, the more important that region is for the TSC task. Here a larger λ value ($= 0.01$) was used to emphasize the effects of SGL. The attention/importance is measured by the L2 norm of the weights associated with each GRU unit, specifically $||\mathbf{W}_t||_2, t = 1, ..., T'$, which are rendered at the corresponding time series regions in the background.

The importance information helps us understand the time series data. For example, in Fig. 5.5a, it shows that the shifting patterns between two classes are the most discriminative. From Fig. 5.5b, our model identifies two regions which are relatively more important for the classification task. The first region is at the starting 50 time

stamps of the time series, and the second one is roughly from time stamp 600 to the end. As another example, from Fig. 5.5c, we see that the network utilizes the information throughout the entire time series, with a stronger focus in the end.

5.5 Applications to Real-world Problems

So far we have demonstrated the ability of GCRNN on the benchmark datasets where all the time series from the training and testing sets have an equal length. To apply GCRNN to a real-world scenario, we need to define a proper length for the time series beforehand, and use it to collect a training dataset from the history. The length can usually be defined based on the domain knowledge. For example, in the problem of seizure detection using EEG, an EEG abnormality is usually required to persist and evolve for a minimum of 6-10 s before considering the abnormality of a seizure [6]. If minimizing the detection latency is also desirable, 6 s is a reasonable choice for the length. A training dataset consists of randomly sampled 6-s multivariate EEGs from both normal and abnormal period in the historical data. For online testing, as the EEG stream arrives, the GCRNN takes the most recent 6-s EEG as the input and provides the classification output.

Without the domain knowledge, the length of the time series can be chosen by cross validation using the historical data. More specifically, datasets with the same class distribution but different time-series lengths are collected from the historical data. The length which generates the best cross-validation accuracy (or other metrics) is selected.

5.6 Case Study in Biomedical Domain: EEG Seizure Classification

As a case study, we apply our method to a publicly available seizure dataset provided by Bonn University [3]. The dataset consists of five subsets (set Z, O,

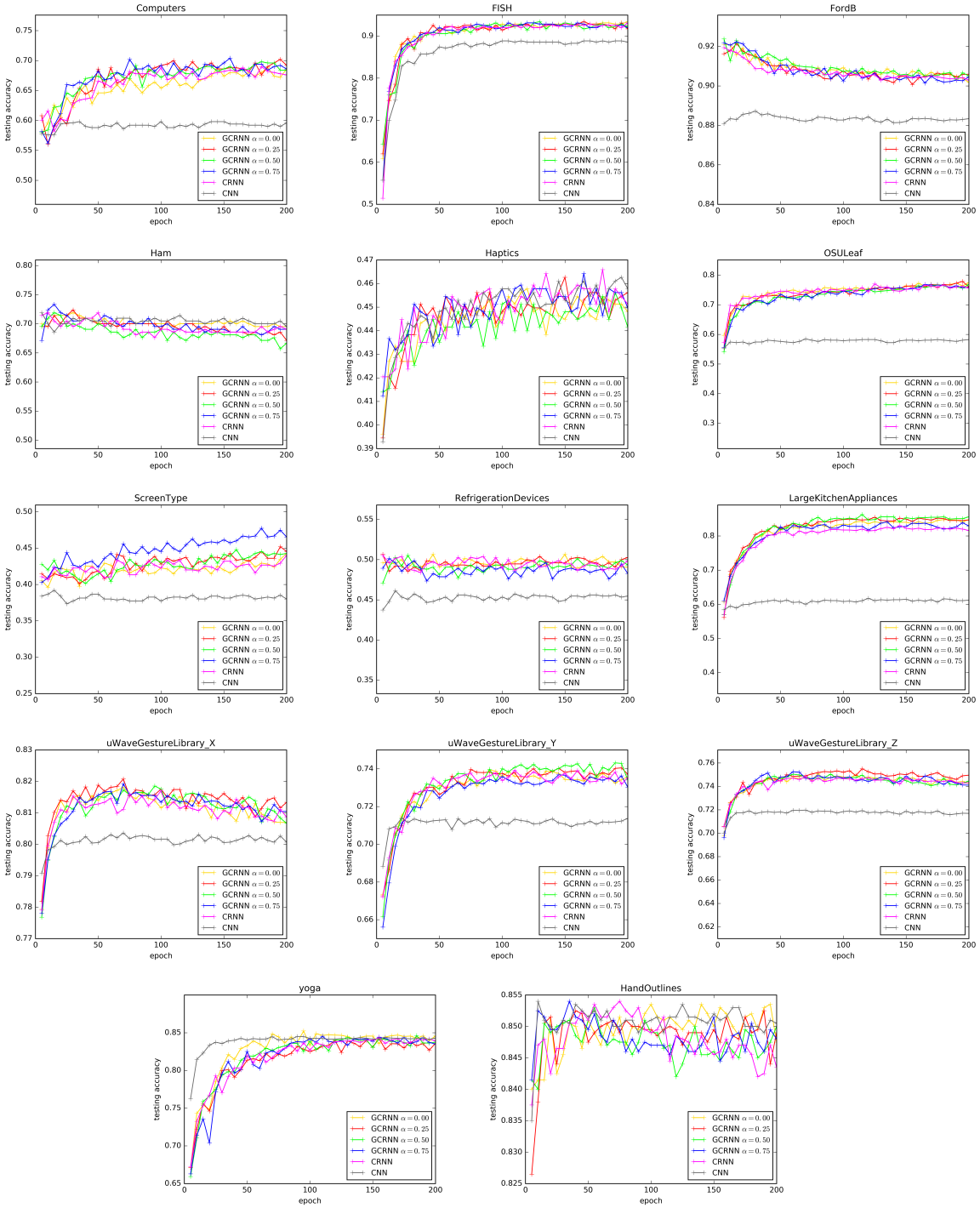
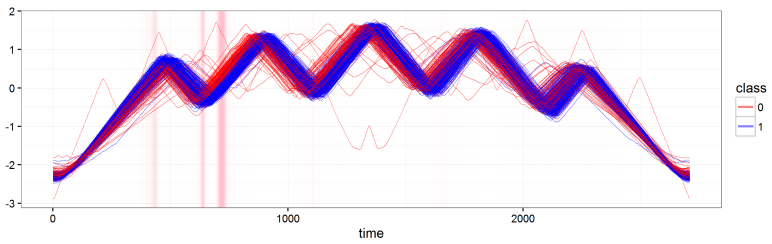
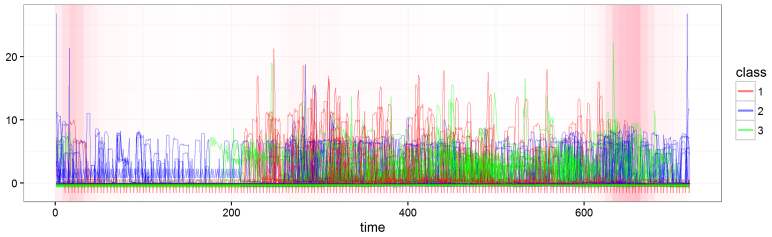


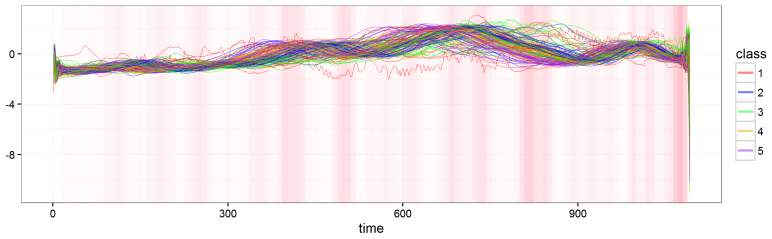
Figure 5.4: The Testing Accuracies of GCRNN/CRNN and CNN Models over 200 Training Epochs on 14 TS Datasets



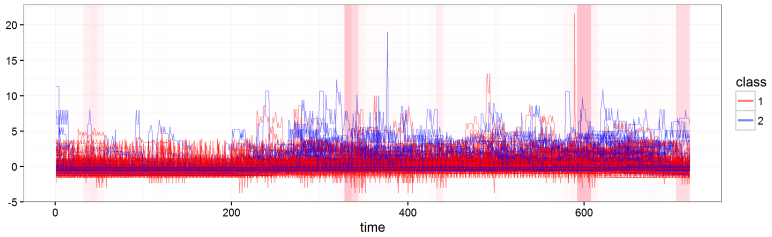
(a) HandOutlines



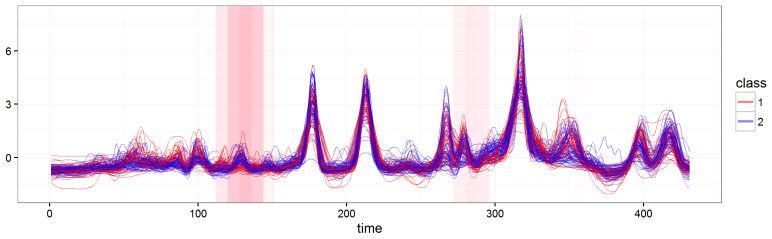
(b) LargeKitchenAppliances



(c) Haptics



(d) Computers



(e) Ham

Figure 5.5: Region Importance for 5 Time Series Datasets Identified by GCRNN($\lambda = 0.01$)

N, F and S), and each has 100 single-channel EEG segment of equal length. Set S contains EEG segments during the seizure activity. Set Z and Set O are the EEG segments collected when the awake healthy volunteers with their eyes open and closed, respectively. Set N and set F contain the non-seizure segments from the seizure patients, recorded from two different brain regions. Three classification tasks (Z vs. S, ZNF vs. S, ZONF vs. S) are formed to compare GCRNN to other methods in the literature.

For each classification task, we randomly split the data into two halves for training and testing for each experiment. The average testing accuracy over 20 experiments is reported for GCRNN. We use $\lambda = 0$ as the location of the patterns are less meaningful for the EEG segments. We compared GCRNN to two methods based on artificial neural network (ANN) that take different sets of hand-crafted features as input [101] [102]. The results for these two ANN-based methods are obtained from [102].

Table 5.3: A Comparison of the Classification Accuracy (%) Obtained by GCRNN and Two Other ANN-based Method for Three Seizure Classification Tasks

classification tasks	time frequency features with ANN [101]	line length features with ANN [102]	GCRNN
Z vs. S	100	99.6	99.95
(Z, N, F) vs. S	-	97.75	98.6
(Z, O, N, F) vs. S	97.73	97.77	99.06

From the comparison in Table 5.3, we show that GCRNN achieves high accuracy across all three classification problems. It outperforms the others for tasks (Z, N, F) vs. S and (Z, O, N, F) vs. S and performs almost equally well as the time-frequency-ANN method for task Z vs. S. This case study provides a successful application of GCRNN towards biomedical time series data.

5.7 Conclusion and Future Work

Here, GCRNN, an end-to-end neural network model for TSC, is proposed. This method leverages both CNN and RNN for the feature learning and the temporal modeling. In addition, the SGL is deployed to reduce the model complexity, but also to understand the attention region of the model. This combination generates a unique, general architecture for time series classification. Our experiments compare different variations of GCRNN with the traditional CNN model to demonstrate that the proposed network architecture performs strongly for time series modeling.

Our model can be generally applied to a large variety of TSC tasks across different domains. For the future work, it is worthwhile to investigate more applications for the proposed GCRNN model, especially for those problems where understanding the discriminative patterns is also of interest. For example, it could be used to classify heartbeat signals (ECG) into the normal and abnormal classes. Along with the classification, the sparse group lasso component of our model could uncover where abnormal patterns appear in the signals. That could help us gain deeper understanding of heart diseases. As another example, GCRNN can be applied to genome sequencing data for the classification of different cancer types. In this particular task, it is important to understand where informative mutations occur and how different cancers are developed.

Chapter 6

CRNN AUTOENCODER FOR TIME SERIES

6.1 Introduction

Dimensionality reduction is useful for the analysis, visualization and understanding of high-dimensional data. The goal of dimensionality reduction is to find a low-dimensional representation of the data which preserves the most important characteristics of the data and minimizes the information loss.

Principal Component Analysis (PCA) is a linear method frequently used to extract low-dimesnioal features from high-diemnsioal data. However, PCA may fail to capture underlying nonlinear patterns and result in large reconstruction error for the nonlinear cases. An ANN autoencoder [103], is a non-linear generalization of PCA (NLPCA). The model is a multiple-layer perceptron which performs an identity mapping. It forces the data to pass through a bottleneck layer with a fewer number of nodes than the original dimensionality of the data. The target of the network is required to be identical as the input so that the network is trained to minimize the reconstruction error. A typical ANN architecture is presented in Fig. 1.6.

Dimensionality reduction is also used in discovering the variation patterns. The reduced number of features is considered to represent the sources of variations, which can enhance our understanding on the mechanism of the data generation. To further improve the interpretability of the patterns, previous works [104] [105] use different regularization strategies to make the features more distinct to each other.

As huge volume of the time series data are collected from the wearable devices and sensors at real time, a good dimensionality reduction method for time series data can

significantly improve the analysis, storage and pattern visualization for time series data. Time series is a special type of high-dimensional data with ordered attributes. Because of the temporal characteristics of time series data (such as autocorrelation), they typically contain high redundancy and can therefore be represented by a much smaller number of features than the original dimensionality (time series length T $\times$ the number of channels D). However, neither the PCA nor the ANN autoencoder utilizes the temporal characteristics of times series during the dimensionality reduction. They ignore the sequential property of the input variables and treat them as independent feature. A dimensionality reduction model that better utilizes the temporal characteristics of time series is expected to be more effective and efficient in summarizing the important time series features at a certain compression level. As pointed out in Chapter 5, convolutional neural networks (CNNs) are effective in extracting high-level features, and recurrent neural networks (RNNs) use their internal memory to model the sequential nature of the time series inputs. In this chapter, we introduce a convolutional recurrent neural networks (CRNN) autoencoder which leverages the advantages of both the CNN and the RNN architectures.

The proposed CRNN model can be considered as a generalization of the existing CNN autoencoder and the RNN autoencoder. A CNN autoencoder is a special variant of a CRNN autoencoder with no RNN components. CNN autoencoder models have been used in image denoising [33] and image restoration [34]. The RNN autoencoder (often known as the sequence-to-sequence model) is also a special variant of the CRNN autoencoder without the CNN components, which has been shown useful in the machine translation task [35].

In this chapter, we discuss different variants of the CRNN autoencoder. The advantage and disadvantage of each variant are discussed via empirical comparisons on 3 ECG datasets and a process control dataset. In addition, the application of

CRNN autoencoder towards the anomaly detection is illustrated based on a simulated study using the same process control dataset.

6.2 CRNN Autoencoder

6.2.1 Problem Statement

Suppose the input data has length T and D channels. The i-th instance in the time series dataset is denoted as $\boldsymbol{x}_i = [x_{i,1}, x_{i,2}, ..., x_{i,T}]$, where $x_{i,t}$ is a scalar for single-channel time series or a vector of length D for multi-channel time series. Dimensionality reduction is to identify a P-dimensional representation $\boldsymbol{v_i} = [v_{i,1}, v_{i,2}, ..., v_{i,P}]$, as well as a reconstruction function $\boldsymbol{f(v_i)} = [f_1(\boldsymbol{v_i}), f_2(\boldsymbol{v_i}), ..., f_T(\boldsymbol{v_i})] : \mathcal{R}^P \rightarrow \mathcal{R}^{TD}$ to map $\boldsymbol{v_i}$ to the original time series space.

From the view of variation pattern discovery [104], $\boldsymbol{v_i} = [v_{i,1}, v_{i,2}, ..., v_{i,P}]$ represents P underlying variation patterns, and the observed instance $\boldsymbol{x_i}$ as the recovered data corrupted by a noise component $\boldsymbol{w_i} = [w_{i,1}, ..., w_{i,T}]$. That is $\boldsymbol{x_i} = \boldsymbol{f(v_i)} + \boldsymbol{w_i}$.

6.2.2 General Architecture

Figure 6.1 describes the architecture of the proposed CRNN autoencoder. For the encoding part, the input time series is first processed by the convolutional and max-pooling layer(s). Their output is usually a multi-channel time series but with a shorter length due to the pooling layer(s). Next, the output of the convolutional-pooling layer(s) is processed by recurrent layer(s) in sequences. The final node of the recurrent layer(s) contains the encoded information from the entire time series, and is densely connected to the bottleneck layer consisted of P nodes.

The output of the bottleneck layer is considered as the compressed representation of the original time series $\boldsymbol{v_i}$. In the decoder, $\boldsymbol{v_i}$ is densely connected the decoding

recurrent layer(s) at each time step, and a few upsampling-convolutional layers are applied to reconstruct a time series of the same dimensionality as the input. The output of the network is denoted as $\boldsymbol{x'_i} = [x'_{i,1}, ..., x'_{i,T}]$. The decoding network learns a reconstruction function $\boldsymbol{f(v_i)} = [f_1(\boldsymbol{v_i}), f_2(\boldsymbol{v_i}), ..., f_T(\boldsymbol{v_i})] : \mathcal{R}^P \rightarrow \mathcal{R}^{TD}$, so we have $\boldsymbol{x'_i} = \boldsymbol{f(v_i)}$.

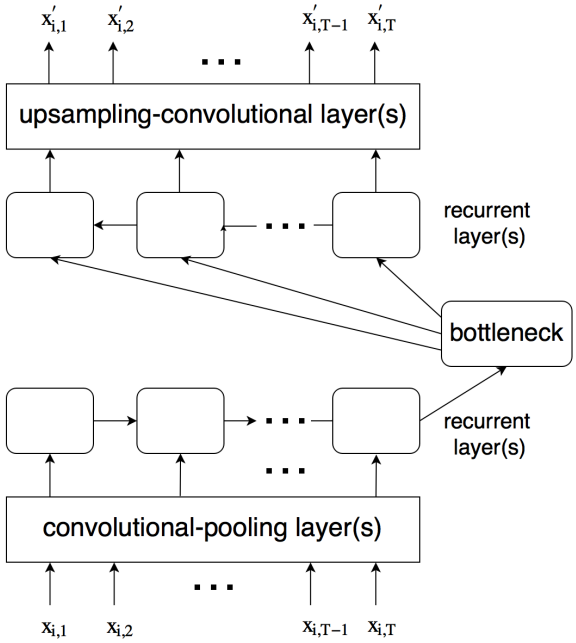


Figure 6.1: CRNN Autoencoder Architecture

6.2.3 Variants 1 and 2: CNN Autoencoder and RNN Autoencoder

A CNN autoencoder is a variant of the CRNN autoencoder without any recurrent layer. In the encoder network of the CNN autoencoder, the output of the last convolutional-pooling layer is fully connected to the bottleneck nodes. Likewise, a RNN autoencoder is another variant of the CRNN autoencoder that only contains recurrent layers.

6.2.4 *Variant 3: Variational CRNN Autoencoder*

The theoretical details of variational autoencoder can be found in [106]. In practice, to build a variational autoencoder, we consider that the bottleneck representation $\boldsymbol{v}_i$ is sampled from a normal distribution parametrized by vectors of mean and standard deviation, $\boldsymbol{\mu}_i$ and $\boldsymbol{\sigma}_i$. That is $\boldsymbol{v}_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i \mathbf{I})S$. The parameters $\boldsymbol{\mu}_i$ and $\boldsymbol{\sigma}_i$ are vectors of P elements generated by the encoder network. The variational autoencoder adds a regularization term to the loss function, which is the Kullback-Leibler (KL) divergence between the bottleneck distribution $\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i)$ and the standard normal distribution $\mathcal{N}(0, \mathbf{I})$. The final KL regularization term is equivalent ot the sum of the KL divergence at each latent dimension as expressed in Eq. 6.1 [106].

$$\text{KL Loss}_i = 0.5(\sum_{p=1}^{P} \sigma_{i,p}^2 + \mu_{i,p}^2 - \log \sigma_{i,p}^2 - 1) \tag{6.1}$$

Many existing statistical approaches for anomaly detection is to model the data points using a parametric distribution, and the points are determined to be anomaly when their likelihood is low. However, it becomes increasingly inaccurate to estimate the distribution of the data in high dimensionality [107]. Thanks to the KL regularization, variational CRNN autoencoders can map complex time series instances to a bottleneck representation in low-dimensional space which can be easily modeled by a Gaussian distribution. Furthermore, many control chart methods reply on normality assumptions to effectively detect anomalies. Variational CRNN autoencoders enable the future applications of the control chart methods on the bottleneck representation in order to monitor complex time series data.

6.3 Experiments

In the firs set of experiments, we compare CRNN autoencoder, CNN autoencoder, RNN autoencoder, ANN autoencoder and PCA based on two performance metrics, the reconstruction error and cross-validation error. Section 6.3.1 defines these two metrics.

In the second set of experiment, we simulate the anomaly detection scenario. More specifically, the autoencoder model is trained on the normal samples in the dataset and tested on both the normal and abnormal cases. The performance of the autoencoder is evaluated based on the cross-validation error to reflect the degree of the separation between the normal and the abnormal cases. We include variational CRNN autoencoder for this set of experiments. In the following, Section 6.3.2 introduces the parameter and training setting for each autoencoder. Section 6.3.3 provides a brief introduction for the datasets used in our experiments.

6.3.1 Evaluation Metrics

Reconstruction error

A good dimensionality reduction method should minimize the information loss during the data compression. We should be able to reconstruct the original data as close as possible from the data representation in the reduced dimensionality. Therefore, we calculate mean squared error (MSE) between the reconstruction and the original time series:

$$MSE = \frac{1}{NTD} \sum_{i=1}^{N} ||\boldsymbol{x_i} - \boldsymbol{f}(\boldsymbol{v_i})||^2, \tag{6.2}$$

where N is the number of times series instances in the dataset, T is the length of the time series and D is the number of time series channels.

However, an autoencoder giving a low reconstruction error does not necessarily generate a good low-dimensional representation for the machine learning tasks. Over-fitting may occur, as the deep autoencoder models in our experiments are very complex with thousands of parameters. The model could overfit the data, and the representation may not generalize well to an unseen instance. Therefore, in the next subsection we propose another metric based on the cross-validation performance to measure the generalibility of the autoencoder models.

Cross-validation classification error (CV error)

A good dimensionality reduction method should also capture the discriminative information for the further analysis and decision making. It should be able to generalize to an unseen case. For a labeled time series dataset, we can evaluate a dimensionality reduction algorithm by evaluating the classification performance of a simple classification model trained on the reduced features. In our experiment, we use 1-nearest-neighbor (1-NN) classifier and present the classification error from a 5-fold cross-validation.

6.3.2 Network Parameter and Training Settings

The CRNN autoencoder we experimented has 2 convolutional-pooling layers and 2 LSTM layers in the encoding network. For the decoding network, it consists of 2 upsampling-convolutional layers and 2 LSTM layers. Each convolutional layer has 10 feature maps. The filter sizes for the 4 convolutional layers are set to 5, 3, 3 and 5. The memory size of the LSTM unit is set to 20. The max-pooling/upsampling layer each reduces/increases the dimensionality by a factor of 2.

The CNN autoencoder is a variant of the CRNN autoencoder discussed above without the LSTM layers. The output of the encoder convolutional-pooling layers

is fully connected to the bottleneck nodes. The RNN autoencoder is another variant of this exact CRNN autoencoder without the convolutional, max-pooling and upsampling layers.

The ANN autoencoder we compare to contains 2 hidden layers in both the encoder network and the decoder network, with 20 and 10 hidden nodes respectively.

We train each autoencoder for 500 epochs. We use rmsprop [108] as the stochastic gradient method. As the stochastic nature of training a neural network models, we repeat each experiment 10 times and present the box-plots of the metrics or the median cases.

6.3.3 Datasets

The datasets used in our experiments are obtained from the UEA & UCR Time Series Repository [2]. They include 3 ECG datasets and a synthetic control chart dataset. Table 6.1 provides a brief summary of these datasets. Note that the synthetic control dataset contains control charts of six classes: 1. normal 2. cyclic 3. increasing trend 4. decreasing trend 5. upward shift 6. downward shift, so we also use it to simulate a testing scenario for the anomaly detection.

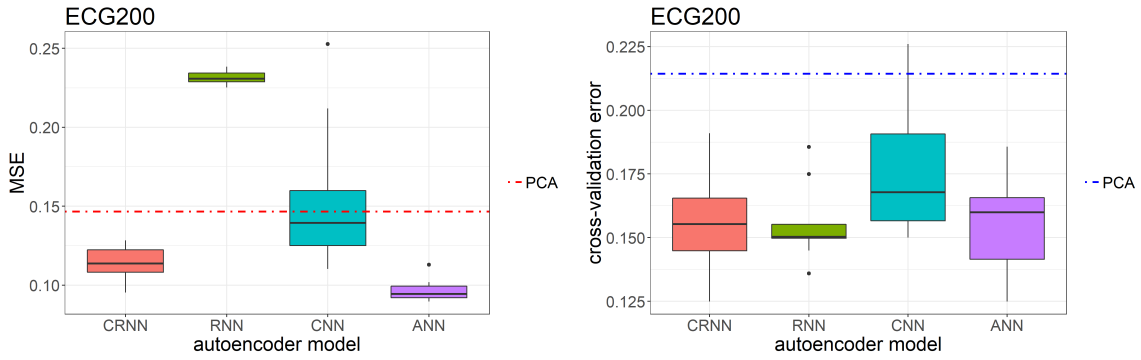
Table 6.1: Summary for the Time Series Datasets Used to Compare Different Autoencoder Models

dataset name	TS length	No. of instances	No. of classes
ECG200	96	200	2
ECGFiveDays	136	884	2
TwoLeadECG	82	1162	2
Synthetic Control	60	600	6

6.4 Experimental Results and Discussion

6.4.1 Performance Comparison on Both Metrics

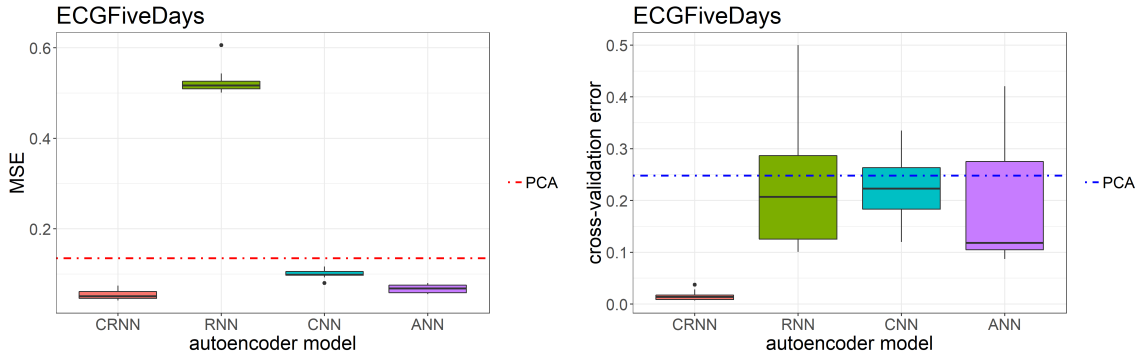
Figures 6.2, 6.3, 6.4 and 6.5 summarize the performance of CRNN, RNN, CNN and ANN autoencoders on both metrics for each dataset. At the same time, the results for PCA is provided as the benchmark. In this set of experiments, we set the number of bottleneck to 2 for all the autoencoder models. In the following, our discussion focuses on the comparison for CRNN autoencoder vs. ANN autoencoder, CRNN autoencoder vs. RNN autoencoder, and CRNN autoencoder vs. CNN autoencoder, respectively.



(a) Recontruction MSE (b) CV Error

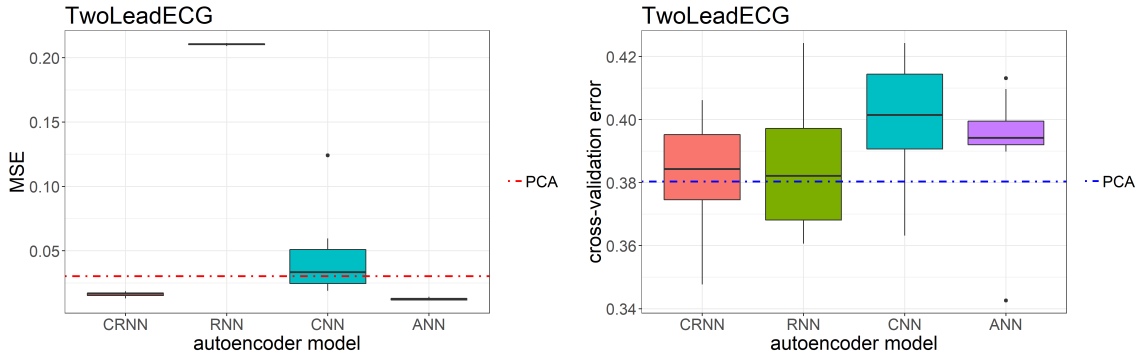
Figure 6.2: Performance Comparison for Different Dimensionality Reduction Models Based on ECG200 Dataset

Comparing CRNN autoencoder with ANN autoencoder, CRNN and ANN autoencoders both result in the lowest reconstruction MSE across 4 datasets. CRNN autoencoder gives slightly lower reconstruction error than ANN for ECGFiveDays and synthetic control datasets, and slightly higher reconstruction error for ECG200 and TwoLeadECG datasets. So two autoencoder models are comparable in terms of the reconstruction error. CRNN autoencoder is significantly better than ANN autoencoder in terms of the cross-validation classification performance. It presents a



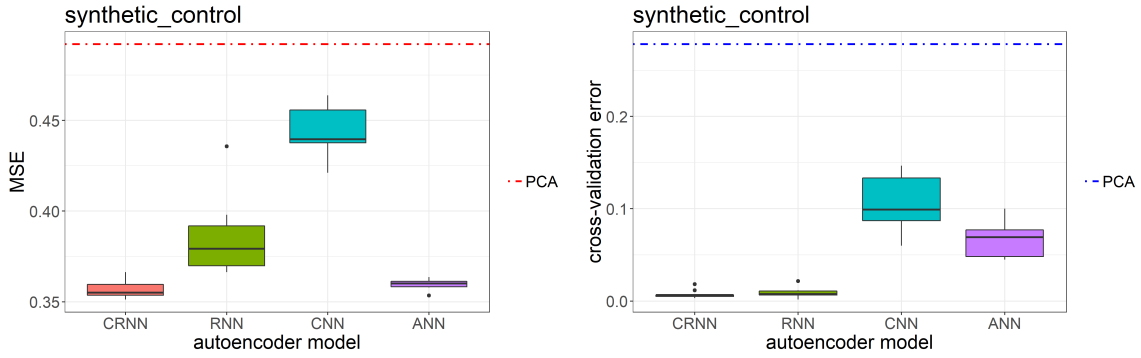
(a) Recontruction MSE (b) CV Error

Figure 6.3: Performance Comparison for Different Dimensionality Reduction Models Based on ECGFiveDays Dataset



(a) Recontruction MSE (b) CV Error

Figure 6.4: Performance Comparison for Different Dimensionality Reduction Models Based on TwoLeadECG Dataset



(a) Reconstruction MSE (b) CV Error

Figure 6.5: Performance Comparison for Different Dimensionality Reduction Models Based on Synthetic Control Dataset

much lower classification error on 3 datasets and a comparable result on the other dataset (ECG200). Our comparison shows that CRNN autoencoder has stronger ability in extracting discriminative features compared to ANN autoencoder, given a similar reconstruction error.

Comparing CRNN autoencoder with RNN autoencoder, they both present strong ability in feature extraction. They both have achieved the low CV error across all the datasets using the low-dimensional representation. A more in-depth comparison on the CV error shows that CRNN autoencoder is slightly better on the synthetic control dataset and significant better on the ECGFiveDays dataset. Our observation demonstrates that incorporating RNN into the autoencoder architecture enhances the extraction of the temporal characteristics of time series, which is often discriminative and informative for a classification task. However, RNN autoencoders result in a higher reconstruction error compared to CRNN autoencoders, as shown in our experiments on the 3 ECG datasets.

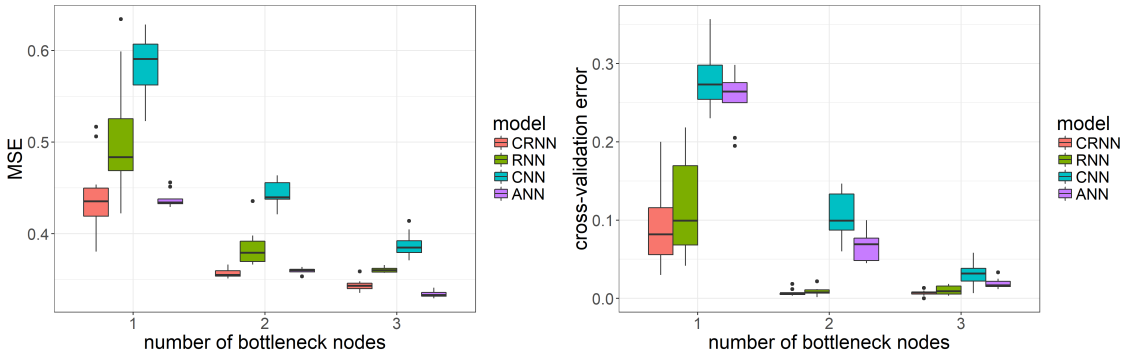
The box-plots show that the CRNN autoencoder consistently outperforms the CNN autoencoder in both metrics on all the 4 datasets. This comparison demonstrates the strong advantage of the CRNN autoencoder over the CNN autoencoder.

To conclude, CRNN is a strong model to reduce the dimensionality for time series data. The features obtained from the low-dimensional representation have strong ability in both the signal reconstruction and the generalization.

6.4.2 Comparison with Different Number of Bottleneck Nodes

Here we take the synthetic control dataset as the example and illustrate the effect of changing number of bottleneck nodes. For each autoencoder, we experiment with different numbers of bottleneck nodes (1, 2 and 3) and compare their performance in both the reconstruction MSE and CV error. As the number of bottleneck nodes

increases, the reconstruction decreases for all the autoencoders, which is expected as the bottleneck layer is allowed to store more information from the original time series. For the classification performance, the same decreasing trend is observed for ANN and CNN autoencoders, however a slight bounce is observed for CRNN and RNN autoencoders at 3 bottleneck nodes, along with a slight increase in the variance. That can be an indication for overfitting. This experiment also demonstrates that low reconstruction error doesn't always guarantee well-generalized features.



(a) Reconstruction MSE vs. bottleneck dimension (b) CV error vs. bottleneck dimension

Figure 6.6: Autoencoder Performance Metrics vs. Number of Bottleneck Nodes: reconstruction MSE and classification accuracy of different autoencoder models with 1, 2, 3 bottleneck nodes for synthetic control dataset.

6.4.3 MSE vs. CV Error

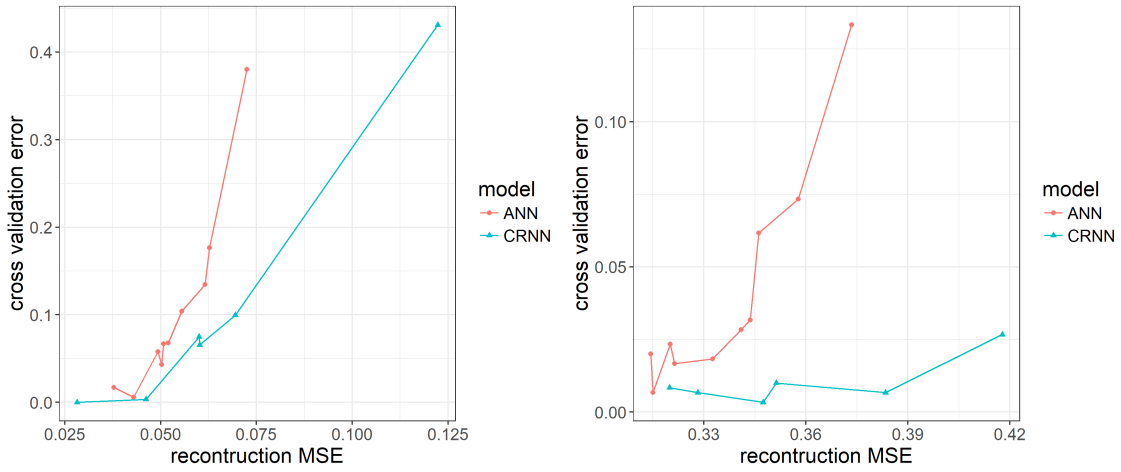
In this section we elaborate our discussion on the relationship among the reconstruction error, the CV error and the model complexity. Our discussion focuses on the CRNN and ANN autoencoders which share a similar level of the reconstruction error. Previous comparisons have made clear that the reconstruction error shouldn't be the sole criteria for evaluating a dimension reduction method, as the overfitting could occur. Therefore, the generalization of the reduced representation should also be taken into the evaluation.

Furthermore, the complexity of the model shouldn't be simply based on the number of parameters, because the constraints imposed on different neural network architecture vary a lot. A CRNN autoencoder model can be significantly less complex than an ANN model with the same number of parameters due to the constraints such as weight sharing in the convolutional and recurrent layers. Instead of the number of parameters, we measure the model complexity by the reconstruction error for different autoencoder models. The lower the reconstruction error, the more complex the autoencoder model.

With that being said, the advantage of the CRNN autoencoder over the ANN autoencoder indeed lies in the better features (measured by the CV error) given the same model complexity (measured by the reconstruction error). To demonstrate the point, we train ANN autoencoders with different number of hidden nodes and CRNN autoencoders with different recurrent memory sizes, and record the CV error and the reconstruction MSE for each parameter setting. The relationship between these two metrics are depicted in Fig. 6.7 based on the ECGFiveDays and Synthetic Control datasets. It is difficult to control the exact reconstruction error for each model, so the spaces of the reconstruction MSE covered by two models are not perfectly aligned. From the shared MSE regions, with similar reconstruction MSE (or similar model complexity), the features generated by CRNN have stronger predictive power.

6.4.4 Simulation Experiments for Anomaly Detection

Time series autoencoders can be used for anomaly detection tasks. For example, the monitoring signals for process control applications are often time series. Autoencoders can summarize the complex time series data with a much smaller number of features, so that the anomaly can be detected more easily.



(a) ECGFiveDays (b) Synthetic Control

Figure 6.7: Relationship Between the CV Error and the Reconstruction MSE for CRNN and ANN Autoencoders based on ECGFiveDays and Synthetic Control Datasets

Here we use the synthetic control dataset to simulate an anomaly detection experiment. In a real-life scenario for process control, a process often starts with normal state, and we want to detect the abnormal events when they occur. To mimic the real scenario, in our experiment, we train each autoencoder using only the normal samples. With this autoencoder, we can encode the samples from both the normal class and other abnormal classes. We evaluate the performance of the autoencoder by measuring how well each abnormal class is separated from the normal class. More specifically, the CV error is calculated between the normal class and each abnormal class. The lower the CV error, the more effective the autoencoder is in the anomaly detection task. Therefore, five classification tasks will be considered: normal vs. cyclic, normal vs. increasing trend, normal vs. decreasing trend, normal vs. upward shift, and normal vs. downward shift. Fig. 6.8 presents the average classification accuracy over 5 tasks. Fig. 6.9 shows the detailed results for each classification task.

In additional to the 4 autoencoders we considered in the previous experiments, we also consider the variational CRNN autoencoder, denoted as V-CRNN, in this

experiment. The variational autoencoder regularizes the bottleneck representation to be close to a standard normal distribution, and that enables the applications of control chart methods which often rely on the normality assumption. All the autoencoders are set to have 2 bottleneck nodes, and the distributions of the their bottleneck representation are visualized in Fig. 6.10.

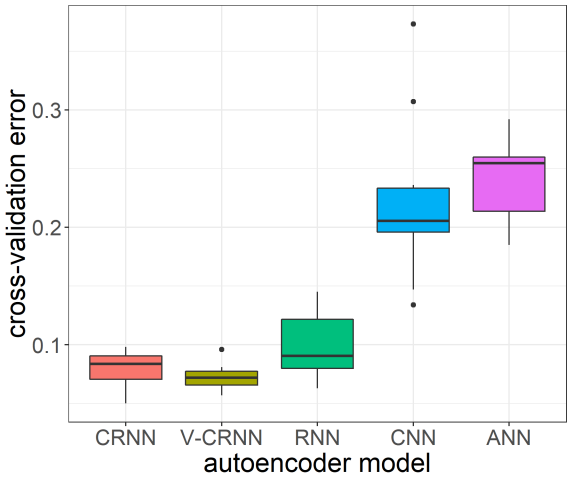


Figure 6.8: Overall CV Error for Different Autoencoders in the Anomaly Detection Experiments

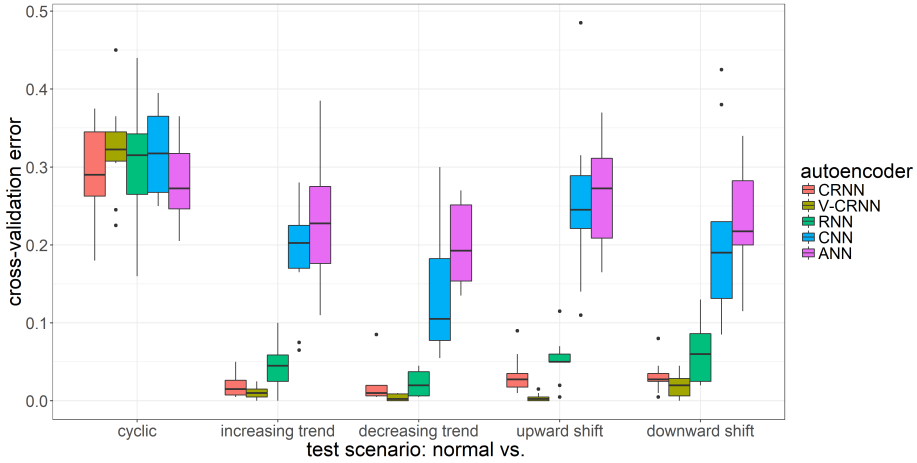
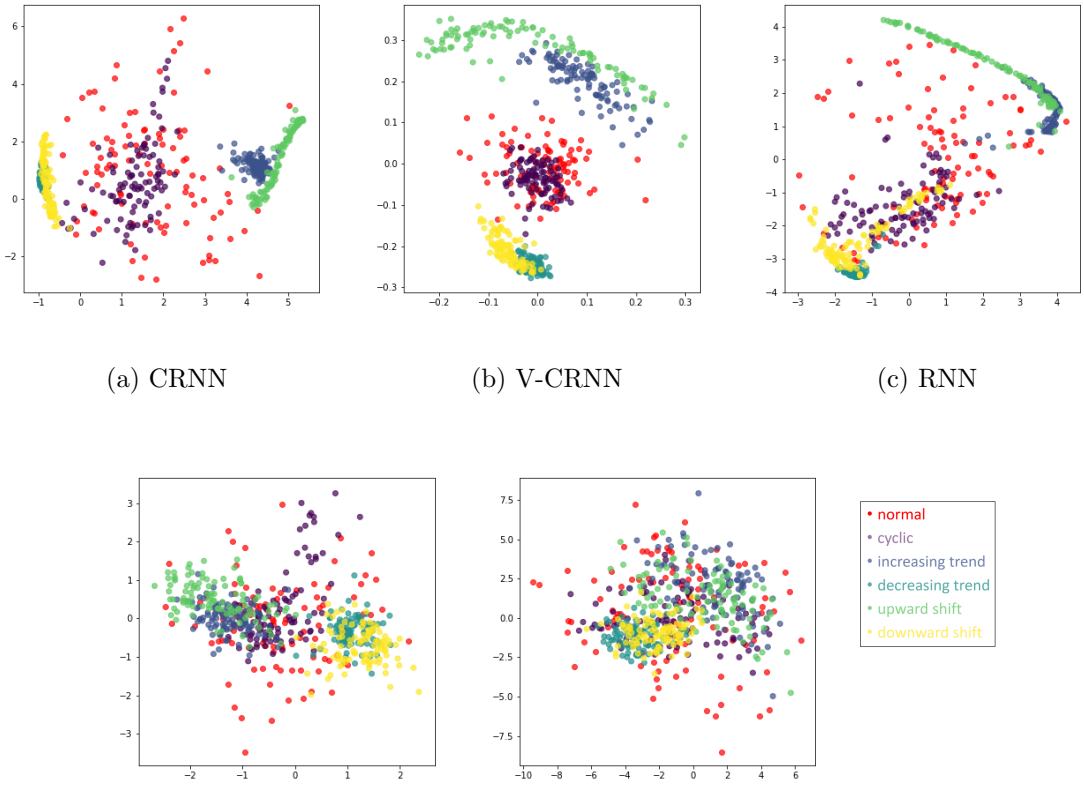


Figure 6.9: CV Error for Different Autoencoders in Each Anomaly Detection Task

From the experimental results, we observe that two CRNN autoencoders obtain the best separation between the normal and abnormal samples in the low-dimensional



(a) CRNN (b) V-CRNN (c) RNN

(d) CNN (e) ANN

Figure 6.10: Bottleneck Representations Generated by Different Autoencoders in the Process Control Experiment: each subfigure corresponds to the case with the median overall CV error. All the autoencoders are trained on normal samples (in red) and used to encode other abnormal samples

representation. This observation demonstrates that the CRNN autoencoder has the strongest ability in anomaly detection among all the autoencoder models included in this experiment. Furthermore, we observe that V-CRNN autoencoder outperforms the regular CRNN autoencoder, which indicates that it is beneficial to include the normality regularization on the bottleneck representation. This observation lays the foundation for the future incorporation of control chart methods. RNN also presents good performance on this task, but not as strong as the CRNN autoencoders. CNN and ANN autoencoders fail to separate the abnormal samples from the normal samples

for all the tasks and result in high CV error. From this experiment, we can conclude that CRNN autoencoder is effective dimensionality reduction method for anomaly detection in complex time series profile data.

6.5 Conclusion

In this chapter, we introduce a novel autoencoder architecture, CRNN autoencoder, for time series data. CRNN autoencoder can be considered a generalization of the existing CNN or RNN autoencoder models, however, this unique combination makes CRNN autoencoder a superior autoencoder model than these two variants. Through a series of experiments, we demonstrate that CRNN is a strong time series autoencoder model which gives the lowest reconstruction error and CV error at the same time. We highlight the advantages of CRNN over its variants (CNN and RNN autoencoders) and the ANN autoencoder in different aspects. More specifically, CRNN autoencoders result in lower reconstruction error than RNN autoencoders, lower CV error than ANN autoencoders, and superior performance in both metrics compared to CNN autoencoders.

Variational CRNN autoencoder is shown to have good potential in the field of process monitoring and anomaly detection. In the future, it can be combined with control chart methods as it imposes the normality of the bottleneck representation.

Based our experiments, we have pointed out that the reconstruction error should not be the only criterion of evaluating an autoencoder. Instead, we can consider the reconstruction error a measure for the model complexity. An over-complex autoencoder (very low reconstruction MSE) may potentially overfit the data, and the features induced by such autoencoder may not generalize well to the unseen cases. In the future, it will be interesting to investigate more metrics for a more comprehensive evaluation for the autoencoder models.

Chapter 7

FUTURE WORK

Although multiple research publications have been derived from the previous chapters, more work can be done to make our methods more comprehensive and applicable to real-world challenges. The following sections discuss how our work can be extended in different aspects in the future.

7.1 Data Clustering

In Chapter 2, we introduce CRAFTER, a scalable clustering algorithm to tackle high-dimensionality and mixed attributes. Similar to many other clustering algorithms, CRAFTER requires a prespecified number of clusters to detect, often denoted as K, as its input. The choice of K is not trivial depending on many characteristics of the data as well as the desired resolution of the clustering. CRAFTER works in an iterative way in updating the clusters. Potentially, the selection of K can also be updated iteratively based on some measure derived from the RF distance. That would make CRAFTER a more powerful clustering algorithm that requires no input from its users.

Compared to the development for new clustering algorithms, relatively less attention has been put on how to interpret the found clusters. However, that is an important step towards a comprehensive understanding for a dataset. Also it is a challenging problem when a dataset contains lots of mixed attributes. After data clustering, it is natural to ask the following questions: what are the most important features that cause the data clusters? What are the common characteristics for instances in each cluster? How can we assign the cluster label for a future instance?

The final supervised RF model obtained by CRAFTER can model and manage the found clusters quite easily, however, the interpretability is limited as the RF is an ensemble model. The variable importance generated by the RF model may be useful for the research in this direction.

Utilizing the clustering structure has shown beneficial for active learning. An active learning paradigm aims to train a good model with reduced labeling cost by selective queries. Stochastic Query-By-Forest (SQBF) [109] uses a tree ensemble to generate the uncertainty score to guide the sampling. Later Shams *et al.* [110] shows that incorporating the cluster information into SQBF promotes the search in diverse areas of the input space. CRAFTER is an ensemble method that evaluates the representativeness of the instances by the probability margin, which may be combined with the uncertainty score used in [109] and [110] to perform a cluster-guided search.

7.2 Deep Learning for Time Series Classification

In Chapter 5, we have shown that the overall performance of GCRNN is better than CNN based on our experiments on 14 TSC tasks. However, the improvement on some datasets are more significant than the others. That naturally leads us to a question: what characteristics of the the time series data causes one deep learning model to outperform the other? The answer to this questions can provide important guidance to the selection of deep learning tools for time series data.

To simplify the comparison between GCRNN and the other models, we use the same setting in the experiments with different datasets. In practice, how to systematically select the hyper-parameters is a key issue for neural network practitioners. In addition, the resolution of the identified important regions can affect the number of pooling layers. In the future, it is worth to investigate the relationships between the hyper-parameters and the model accuracy or the interpretability.

7.3 Deep Autoencoder for Time Series

The combination of the variational autoencoder and control chart methods can lead to a new anomaly detection method for monitoring complex time series profile. Many control chart methods reply on the normality assumption for the input data, and variational autoencoder is able to provide a near normal representation in a low-dimensional space. This new anomaly detection technique has many potential applications, such ash the detection for abnormal heart beats in ECG monitoring, the product defect detection based on control signals, and the seizure onset prediction based on EEG stream.

Dimensionality reduction is associated with variational pattern mining. Ideally each feature in the reduced feature space corresponds to a specific variation pattern. The interpretation of the feature can help us understand the underlying data generation mechanism. How well the solution can be interpreted with respect to the variation exhibited in the original feature space is important. However, the direct application of autoencoder models often results in confound features. That means that each feature is a combination of multiple sources of variations, thus the interpretation becomes difficult. The distinct feature/variation pattern discovery can be imposed in the model training via some regularization methods [104] [105]. Similar techniques can be applied to CRNN autoencoders for discovering distinct features for time series.

7.4 Discovering Future Applications

There are many potential applications for the developed algorithms which we haven't experimented on. For example, CRAFTER can be used to cluster heterogeneous patient data, which can help doctors to utilize the knowledge from similar past

cases in giving a diagnosis. CRAFTER can also be used to generate the matching groups for the matched case-control analysis which is widely used in clinical studies. Given the availability of a large genome sequencing dataset for different cancers, we can apply CRNN autoencoder to discover the underlying relationship between different cancer types in the embedded feature space. GCRNN can be used to classify different tumor types or tumor grades and identify the abnormal genome regions. To conclude, discovering new applications in not only the biomedical domain but also other domains will be an ongoing effort in the future.

REFERENCES

[1] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[2] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, "The ucr time series classification archive," July 2015, www.cs.ucr.edu/~eamonn/time_series_data/.

[3] R. G. Andrzejak, K. Lehnertz, F. Mormann, C. Rieke, P. David, and C. E. Elger, "Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state," *Physical Review E*, vol. 64, no. 6, p. 061907, 2001.

[4] Tcga research network. [Online]. Available: http://cancergenome.nih.gov/

[5] G. Lujan Moreno, "Analytical methods for high dimensional physiological sensors," January 2017. [Online]. Available: http://search.proquest.com/docview/1897020275/

[6] A. H. Shoeb and J. V. Guttag, "Application of machine learning to epileptic seizure detection," in *Proceedings of the 27th International Conference on Machine Learning*, 2010, pp. 975–982.

[7] E. Keogh and A. Mueen, "Curse of dimensionality," in *Encyclopedia of Machine Learning, C. Sammut and G. I. Webb, Eds.* Springer, 2017, pp. 314–315.

[8] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional spaces," in *ICDT*, vol. 1. Springer, 2001, pp. 420–434.

[9] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is nearest neighbor meaningful?" in *Proceedings of the 7th International Conference on Database Theory*, 1999, pp. 217–235.

[10] Z. S. Abdallah, L. Du, and G. I. Webb, "Data preparation," in *Encyclopedia of Machine Learning and Data Mining*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2017, pp. 318–327.

[11] S. Lin, B. Azarnoush, and G. Runger, "Crafter: a tree-ensemble clustering algorithm for static datasets with mixed attributes and high dimensionality," *IEEE Transactions on Knowledge and Data Engineering*, 02 2018.

[12] A. Bagnall, A. Bostrom, J. Large, and J. Lines, "The great time series classification bake off: An experimental evaluation of recently proposed algorithms," *arXiv preprint arXiv:1602.01711*, 2016.

[13] B. B. Le Cun, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems*. Citeseer, 1990.

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[15] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[16] S. Lin and G. C. Runger, "Gcrnn: Group-constrained convolutional recurrent neural network," *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

[17] Wandy: A program for cnv/aneuploidy detection from wgs sequencing data. [Online]. Available: http://bioinformaticstools.mayo.edu/research/wandy/

[18] R. Beroukhim, C. H. Mermel, D. Porter, G. Wei, S. Raychaudhuri, J. Donovan, J. Barretina, J. S. Boehm, J. Dobson, M. Urashima *et al.*, "The landscape of somatic copy-number alteration across human cancers," *Nature*, vol. 463, no. 7283, pp. 899–905, 2010.

[19] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE transactions on pattern analysis and machine intelligence*, vol. 12, no. 10, pp. 993–1001, 1990.

[20] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[21] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of Statistics*, pp. 1189–1232, 2001.

[22] L. Breiman, "Random forests," *Machines learning*, vol. 45, no. 1, pp. 5–32, 2001.

[23] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. US: CRC press, 1984.

[24] L. Breiman, "Manual–setting up, using and understanding random forests v4. 0," 2003. [Online]. Available: https://www.stat.berkeley.edu/~breiman/Using_random_forests_v4.0.pdf

[25] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning*, 2010, pp. 807–814.

[26] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[27] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of The 30th International Conference on Machine Learning*, 2013, pp. 1310–1318.

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[29] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[30] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[31] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002.

[32] S. Yan. (2016) Understanding lstm and its diagrams. [Online]. Available: https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714

[33] L. Gondara, "Medical image denoising using convolutional denoising autoencoders," in *Data Mining Workshops (ICDMW), 2016 IEEE 16th International Conference on*. IEEE, 2016, pp. 241–246.

[34] X. Mao, C. Shen, and Y.-B. Yang, "Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections," in *Advances in neural information processing systems*, 2016, pp. 2802–2810.

[35] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[36] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[37] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[38] M. G. Baydogan, G. Runger, and E. Tuv, "A bag-of-features framework to classify time series," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 11, pp. 2796–2802, 2013.

[39] F. Moosmann, B. Triggs, F. Jurie *et al.*, "Fast discriminative visual codebooks using randomized clustering forests," in *NIPS*, vol. 2, 2006, p. 4.

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

[41] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Applied Statistics*, pp. 100–108, 1979.

[42] L. Kaufman and P. J. Rousseeuw, "Partitioning around medoids (program pam)," *Finding Groups in Data: an Introduction to Cluster Analysis*, pp. 68–125, 1990.

[43] Z. Huang, "Extensions to the k-means algorithm for clustering large data sets with categorical values," *Data Mining and Knowledge Discovery*, vol. 2, no. 3, pp. 283–304, 1998.

[44] S. Guha, R. Rastogi, and K. Shim, "Rock: A robust clustering algorithm for categorical attributes," in *Proceedings of the 15th International Conference on Data Engineering*. IEEE, 1999, pp. 512–521.

[45] L. Parsons, E. Haque, and H. Liu, "Subspace clustering for high dimensional data: a review," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 90–105, 2004.

[46] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic subspace clustering of high dimensional data," in *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*. ACM, 1998, pp. 94–105.

[47] M. Radovanović, A. Nanopoulos, and M. Ivanović, "Hubs in space: Popular nearest neighbors in high-dimensional data," *Journal of Machine Learning Research*, vol. 11, no. Sep, pp. 2487–2531, 2010.

[48] N. Tomasev, M. Radovanovic, D. Mladenic, and M. Ivanovic, "The role of hubness in clustering high-dimensional data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 3, pp. 739–751, 2014.

[49] E. Allen, S. Horvath, F. Tong, P. Kraft, E. Spiteri, A. D. Riggs, and Y. Marahrens, "High concentrations of long interspersed nuclear element sequence distinguish monoallelically expressed genes," *Proceedings of the National Academy of Sciences*, vol. 100, no. 17, pp. 9940–9945, 2003.

[50] T. Shi and S. Horvath, "Unsupervised learning with random forest predictors," *Journal of Computational and Graphical Statistics*, vol. 15, no. 1, 2006.

[51] B. Azarnoush, J. M. Bekki, G. C. Runger, B. L. Bernstein, and R. K. Atkinson, "Toward a framework for learner segmentation," *Journal of Educational Data Mining*, vol. 5, no. 2, pp. 102–126, 2013.

[52] Q. Zhang and I. Couloigner, "A new and efficient k-medoid algorithm for spatial clustering," *Computational Science and Its Applications–ICCSA 2005*, pp. 207–224, 2005.

[53] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert systems with applications*, vol. 36, no. 2, pp. 3336–3341, 2009.

[54] J. Ji, W. Pang, C. Zhou, X. Han, and Z. Wang, "A fuzzy k-prototype clustering algorithm for mixed numeric and categorical data," *Knowledge-Based Systems*, vol. 30, pp. 129–135, 2012.

[55] Z. He, X. Xu, and S. Deng, "Attribute value weighting in k-modes clustering," *Expert Systems with Applications*, vol. 38, pp. 15 365–15 369, 2011.

[56] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining, (First Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[57] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical association*, vol. 66, no. 336, pp. 846–850, 1971.

[58] H. Kremer, P. Kranen, T. Jansen, T. Seidl, A. Bifet, G. Holmes, and B. Pfahringer, "An effective evaluation measure for clustering on evolving data streams," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 868–876.

[59] G. Melli, "The datgen dataset generator. version 3.1," 1999. [Online]. Available: http://www.datasetgenerator.com

[60] P. Franti, O. Virmajoki, and V. Hautamaki, "Fast agglomerative clustering using a k-nearest neighbor graph," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 11, pp. 1875–1881, 2006.

[61] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R News*, vol. 2, no. 3, pp. 18–22, 2002. [Online]. Available: http://CRAN.R-project.org/doc/Rnews/

[62] M. Maechler, P. Rousseeuw, A. Struyf, M. Hubert, and K. Hornik, *cluster: Cluster Analysis Basics and Extensions*, 2017, r package version 2.0.6 — For new features, see the 'Changelog' file (in the package source).

[63] G. Szepannek, "R package 'clustmixtype'," 2017. [Online]. Available: https://CRAN.R-project.org/package=clustMixType

[64] N. Japkowicz and M. Shah, *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, 2011.

[65] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.

[66] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: A new data clustering algorithm and its applications," *Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 141–182, 1997.

[67] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *Workshop on statistical learning in computer vision, ECCV*, vol. 1, no. 1-22. Prague, 2004, pp. 1–2.

[68] Y. H. Li and A. K. Jain, "Classification of text documents," *The Computer Journal*, vol. 41, no. 8, pp. 537–546, 1998.

[69] T. I. Zack, S. E. Schumacher, S. L. Carter, A. D. Cherniack, G. Saksena, B. Tabak, M. S. Lawrence, C.-Z. Zhang, J. Wala, C. H. Mermel *et al.*, "Pan-cancer patterns of somatic copy number alteration," *Nature genetics*, vol. 45, no. 10, pp. 1134–1140, 2013.

[70] I.-M. Shih and R. J. Kurman, "Ovarian tumorigenesis: a proposed model based on morphological and molecular genetic analysis," *The American journal of pathology*, vol. 164, no. 5, pp. 1511–1518, 2004.

[71] Y.-S. Jeong, M. K. Jeong, and O. A. Omitaomu, "Weighted dynamic time warping for time series classification," *Pattern Recognition*, vol. 44, no. 9, pp. 2231–2240, 2011.

[72] A. Stefan, V. Athitsos, and G. Das, "The move-split-merge metric for time series," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 6, pp. 1425–1438, 2013.

[73] J. Lin, R. Khade, and Y. Li, "Rotation-invariant similarity in time series using bag-of-patterns representation," *Journal of Intelligent Information Systems*, vol. 39, no. 2, pp. 287–315, 2012.

[74] J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall, "Classification of time series by shapelet transformation," *Data Mining and Knowledge Discovery*, vol. 28, no. 4, pp. 851–881, 2014.

[75] H. Deng, G. Runger, E. Tuv, and M. Vladimir, "A time series forest for classification and feature extraction," *Information Sciences*, vol. 239, pp. 142–153, 2013.

[76] M. G. Baydogan and G. Runger, "Time series representation and similarity based on local autopatterns," *Data Mining and Knowledge Discovery*, vol. 30, no. 2, pp. 476–509, 2016.

[77] H. Deng, M. G. Baydogan, and G. Runger, "Smt: Sparse multivariate tree," *Statistical Analysis and Data Mining*, vol. 7, no. 1, pp. 53–69, 2014.

[78] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[79] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

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

[81] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[82] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 815–823.

[83] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: An astounding baseline for recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014, pp. 806–813.

[84] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Time series classification using multi-channels deep convolutional neural networks," in *International Conference on Web-Age Information Management*. Springer, 2014, pp. 298–310.

[85] C. L. Giles, S. Lawrence, and A. C. Tsoi, "Noisy time series prediction using recurrent neural networks and grammatical inference," *Machine Learning*, vol. 44, no. 1-2, pp. 161–183, 2001.

[86] M. S. alDosari, "Unsupervised anomaly detection in sequences using long short term memory recurrent neural networks," Master's thesis, George Mason University, Fairfax, VA, USA, 2016.

[87] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *Proceedings of The 32nd International Conference on Machine Learning*, 2015, pp. 2048–2057.

[88] H. Xu and K. Saenko, "Ask, attend and answer: Exploring question-guided spatial attention for visual question answering," *arXiv preprint arXiv:1511.05234*, 2015.

[89] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," *arXiv preprint arXiv:1512.02595*, 2015.

[90] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[91] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International Conference on Artificial Neural Networks*. Springer, 2010, pp. 92–101.

[92] Y. LeCun and Y. Bengio, "The handbook of brain theory and neural networks," M. A. Arbib, Ed. Cambridge, MA, USA: MIT Press, 1998, ch. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258. [Online]. Available: http://dl.acm.org/citation.cfm?id=303568.303704

[93] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proceedings of The 32nd International Conference on Machine Learning*, 2015, pp. 2342–2350.

[94] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[95] J. Friedman, T. Hastie, and R. Tibshirani, "A note on the group lasso and a sparse group lasso," *arXiv preprint arXiv:1001.0736*, 2010.

[96] J. Y. Lee and F. Dernoncourt, "Sequential short-text classification with recurrent and convolutional neural networks," in *Proceedings of NAACL-HLT*, 2016, pp. 515–520.

[97] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[98] C. A. Ratanamahatana and E. Keogh, "Making time-series classification more accurate using learned constraints," in *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM, 2004, pp. 11–22.

[99] ——, "Three myths about dynamic time warping data mining," in *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, 2005, pp. 506–510.

[100] R. Caruana, S. Lawrence, and C. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Advances in neural information processing systems*, 2001, pp. 402–408.

[101] A. T. Tzallas, M. G. Tsipouras, and D. I. Fotiadis, "Automatic seizure detection based on time-frequency analysis and artificial neural networks," *Computational Intelligence and Neuroscience*, vol. 2007, 2007.

[102] L. Guo, D. Rivero, J. Dorado, J. R. Rabunal, and A. Pazos, "Automatic epileptic seizure detection in eegs based on line length feature and artificial neural networks," *Journal of neuroscience methods*, vol. 191, no. 1, pp. 101–109, 2010.

[103] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[104] P. Howard, D. W. Apley, and G. Runger, "Distinct variation pattern discovery using alternating nonlinear principal component analysis," *IEEE transactions on neural networks and learning systems*, 2016.

[105] P. Howard, D. Apley, and G. Runger, "Regularized learning for distinct feature discovery using autoassociative neural networks," Arizona State University, School of Computing, Informatics, Decisions System Engineering, Tech. Rep., 2017.

[106] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[107] C. C. Aggarwal and P. S. Yu, "Outlier detection for high dimensional data," in *ACM Sigmod Record*, vol. 30, no. 2. ACM, 2001, pp. 37–46.

[108] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," 2012.

[109] A. Borisov, E. Tuv, and G. Runger, "Active batch learning with stochastic query-by-forest (SQBF)," in *JMLR Workshop on Active Learning and Experimental Design*, 2011, pp. 59–69.

[110] G. Shams, N. Shomal Zadeh, E. Del Castillo, and G. Runger, "Multi-class clustering-guided active learning," Arizona State University, School of Computing, Informatics, Decisions System Engineering, Tech. Rep., 2017.