KERNEL METHODS FOR TIME SERIES DATA

by

FENGZHEN TANG

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
June 2015

Abstract

Kernel methods are powerful learning techniques with excellent generalization capability. This thesis develops three advanced approaches within the generic SVM framework in the application domain of time series data.

The first contribution presents a new methodology for incorporating privileged information about the future evolution of time series, which is only available in the training phase. The task is prediction of the ordered categories of future time series movements. This is implemented by directly extending support vector ordinal regression with implicit constraints to leaning using privileged information paradigm.

The second contribution demonstrates a novel methodology of constructing efficient kernels for time series classification problems. These kernels are constructed by representing each time series through a linear readout model from a high dimensional state space model with a fixed deterministically constructed dynamic part. Learning is then performed in the linear readout model space.

Finally, in the same context, we introduce yet another novel time series kernel by co-learning the dynamic part and a global metric in the linear readout model space, encouraging time series from the same class to be represented by close model representations, while model representations of time series from different classes to be well-separated.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr. Peter Tiňo, who provided me with substantial support from the beginning of my PhD all the way to the end. His enthusiasm has inspired me to become enthusiastic and confident about myself in the research. This thesis would never have been completed without his continuous encouragement, careful supervision and constructive guidance. The second thanks are given to my co-supervisor Dr. Huanhuan Chen for kindly giving me many valuable comments and suggestions in numerous discussions on both my research and my future career.

I would also like to thank my Thesis Group Members, Dr. Ela Claridge (RSMG representative) and Dr. Ata Kaban, who have given time, thoughts and interesting research ideas about this thesis development. I am grateful to Dr. Pedro Antonio Gutiérrez for our research collaboration and for providing me with helps in the first part of my research. Many thanks also to Dr. Xin Yao for his valuable suggestions and financial support during my visit of the University of Science and Technology of China (USTC).

Thanks very much to our Machine Learning Reading Group members, Dr. Ata Kaban, Liyan Song, Yuan Shen, Luca Rossi, Momodou Sanyang and so on for the discussion and comments on my presentation and research. Many thanks to Dr. Frank-Michael Schleif, Michael Denzel and Lukas Adam for proof reading my thesis.

Special thanks go to the China Scholarship Council who granted me a generous financial support throughout my PhD study.

I am grateful to my officemates for many interesting conversations which inspired my work. Many thanks to Chuangjie Xu for bearing my complaints when I met problems and sharing useful information with me. Thanks very much to my house mates for helping me to correct mistakes I made in my English. Thanks Michael LoCicero for helping me to polish my English.

I am also grateful to my friends for their companionship and their patience. Thanks Liyan Song and Mengmeng Qi for comforting me whenever I met problems in both my life and PhD research. Many thanks to Gurpreet Grewal-Kang who encouraged me to participate social events, making my PhD study more colourful.

I would also like to show my deepest gratitude to my parents and my sister, who always tried to listen to me and encourage me even though far away. The completion of this thesis would have been impossible, without the support and encouragement from my beloved fiancée Daming Wang. I really appreciated his effort to understand the problems I met in my research and give me inspirations from time to time.

Contents

List of Figures

List of Tables

List of Abbreviations

ANN: Artificial Neural Network

AR: Autoregressive

CRJ: Cycle Reservoir with Regular Jumps

DACO: Difference between Auto-correlation Operators

DTW: Dynamic Time Warping

ESN: Echo State Network

GMMRV kernel: Gaussian mixture model based reservoir kernel

HMM: Hidden Markov Model

KKT: Karush-Kuhn-Tucker

KL: Kullback-Leibler

kNN k-Nearest Neighbour

LDS: Linear Dynamic System

LiMS: Learning in the Model Space

LSM: Liquid State Machine

LUPI: Learning Using Privileged Information

MAE: Mean Absolute Error

MDS: Multidimensional scaling

MMAE: Macroaveraged Mean Absolute Error

MMCL: Model-Metric Co-Learning

MZE: Mean Zero-one Error

RBF:	Radial Basis Function
RC:	Reservoir Computing
RLS:	Recursive Least Squares
RNN:	Recurrent Neural Network
RV:	Reservoir
SamplingRV kernel:	Sampling based reservoir kernel
SCR:	Simple Cycle Reservoir
SMO:	Sequential Minimal Optimization
STM:	State Transition Mapping
SVM+	Support Vector Machine plus (using) Privileged Information
SVM:	Support Vector Machine
SVORIM+:	Support Vector Ordinal Regression with Implicit Constraints Using Privileged Information
SVORIM:	Support Vector Ordinal Regression with Implicit Constraints
SVOR:	Support Vector Ordinal Regression
VAR:	Vector Autoregressive

List of Notations

A: the global metric in the linear readout space, 101

C: penalty hyper-parameter in SVMs, 30

E: the error function, 88

E_r: regularized error function, 92

J: number of classes in ordinal regression, 42

L: length of time series, 12

L_i: length of the i-th time series, 12

M: dimension of output, 17

Q: overall cost, 103

Q_p: the representability cost , 101

Q_s: separability cost, 102

R: RNN internal units connection matrix, 17

V: RNN input weight matrix , 17

W: RNN output weight matrix, 17

X: input space, 44

X^*: privileged input space, 42

Φ: kernel related mapping, 32

Φ^*: feature map induced by the kernel operating on X^*, 44

$\boldsymbol{r}$: state transition mapping parameter vector, 100

$\boldsymbol{w}$: the norm vector to the separating hyperplane in SVM, 27

$\boldsymbol{w}^*$: the normal vector to the correcting function in SVM+, 44

$\boldsymbol{w}_j^*$: the normal vector to the j-th correcting function in SVORIM+ , 49

$\boldsymbol{w}_m$: readout parameter of the m-th time series , 101

$\boldsymbol{x}_i^k$: i-th example of k-th category, 47

$\boldsymbol{x}_i^{*k}$: privileged input of example $\boldsymbol{x}_i^k$, 49

$\boldsymbol{x}_i$: i-th example, 12

η: the tradeoff between representability and separability , 103

γ: kernel scale parameter , 72

λ: ridge regression parameter , 72

$\mathbf{y}(t)$: the target at time stamp t, 89

$\mathcal{K}$: kernel function, 32

$\mathcal{K}^*$: kernel function in X^* , 45

$\mathcal{L}$: Lagrangian function, 28

$\mathcal{L}_D$: the dual function of the Lagrangian, 50

ρ: hyper-parameter used to control the capacity of the correcting functions, 45

σ: kernel width in Gaussian kernel in X, 32

σ^*: kernel width in Gaussian kernel in X^*, 53

$\xi^j(\boldsymbol{x}^*)$: the j-th correcting function in SVORIM+, 49

ξ^j_{ki}: the slack variable with respective to $\boldsymbol{x}^k_i$, 47

b: the bias term in SVM, 27

b^*: the bias term of the correcting function in SVM+, 44

b^*_j: the bias term of the j-th correcting function in SVORIM+, 49

b_j: j-th threshold in SVORIM and SVORIM+, 42

d: dimension of input examples., 27

$d_A(m, q)$: the squared distance between readout parameters $\boldsymbol{w}_m$ and $\boldsymbol{w}_q$ under the matric A, 101

n: number of examples, 27

n^k: number of examples in k-th category, 47

r_i: the shared absolute value of input connection weight of CRJ model, 20

r_c: the shared cycle connection weight of CRJ model, 20

r_j: the shared jump weight of CRJ model, 20

$s(t)$: value of time series at time stamp t, 12

y_i: the desired output of the i-th example, 12

CHAPTER 1

Introduction

This thesis presents research on kernel methods, a popular family of machine learning algorithms. Machine learning is a very active subdiscipline within artificial intelligence that involves study and development of systems which can learn from example data or past experience, rather than following explicit programming instructions [1, 64, 82]. The system is often defined as a model with parameters. Learning is the process of optimizing those parameters using the training data or past experience under a performance criterion (ideally the expected future performance). In a supervised learning setting, the training data presented to the system consist of example inputs and their desired outputs provided by a supervisor. The goal is to learn a mapping from the inputs to outputs. Depending on the nature of outputs, supervised learning can be classified into regression where the outputs are continuous and classification where the outputs only take a small number of discrete values. Recently, a new supervised learning setting that bridges regression and classification, referred as ordinal regression or ranking, has drawn people's attention [61, 83, 86, 88]. Ordinal regression problems are essentially multi-class classification problems, but a natural order among the multiple categories can be observed [16], for example, learning to grade the exchange rate of currency. In this setting, the training examples are labeled by a finite set of ranks, which exhibits a natural order among different categories. Consequently, ordinal regression bears resemblance to both regression and classification. However, in contrast to metric regression, the ranks are of finite and discrete type, and the exact amounts of difference among ranks are not defined. The existence of ordering information in the class labels makes ordinal regression also different from classification.

Kernel methods have become very popular in machine learning over last a couple of decades [6], since they are able to detect non-linear relations in the dataset without losing efficiency that has previously been reserved for linear algorithms. These methods first map the data into a high-dimensional feature space where linear relations can then be discovered [36]. The linear relations in the feature space correspond to non-linear relations in the input space, given the kernel is defined by a non-linear mapping. Kernels can evaluate the inner product between the images of the two inputs in a feature space without explicitly computing their coordinates. Therefore, kernel based non-linear algorithms can avoid extra computational effort to detect the non-linear relations in the data. Many linear algorithms such as Fisher discriminant and principal components analysis have their kernelized extensions [91]. Kernel-based support vector machines (SVMs) are powerful kernel based learning machines and have gained wide popularity over the last decade. They have shown to be effective for many problems on numerous applications such as digit recognition, face detection, speaker identification, time series prediction, and so on [13].

Recently, kernel methods have received considerable attention in the machine learning community dealing with structured data, such as image, graphs, texts, or voice signals. However, as an important ubiquitous data type in science and engineering, time series have received relatively less research in the kernel literature [26]. In this thesis, we present two different aspects of improving the generalization performance of kernel methods, especially SVMs, dealing with time series data. Time series prediction and classification are of particular interest in this work. Time series prediction is to estimate future values from the past observations within a single sequence while time series classification is to predict a class label for the entire sequence.

1.1 Motivation

In some learning problems, there exists some additional informative knowledge about the training examples which is unavailable in the test phase. Such information would be discarded in the traditional learning paradigm, since traditional learning algorithms require examples in test phase to be characterized in the same way as in the training stage. The incorporation of privileged knowledge into the learning process was first proposed by Vapnik [101, 102], motivated by human learning where a teacher will provide students not only with examples but also with additional information hidden in the explanations, comments, comparisons etc. This new advanced learning paradigm is called *learning using privileged information* (LUPI) [101]. It has been realized in the context of SVM framework for both regression and classification through modelling the slack variables using the privileged information [101,102]. However, LUPI has not been applied in ordinal regression setting yet. Incorporating privileged information in ordinal regression is very promising research question in the domain of time series prediction. In some time series prediction problems, instead of predicting the real future values, we are only interested in ordered categories of movements – e.g. extreme up, up, down and extreme down. In this context, an ordinal regression problem can be formed. Moreover, future events present in the training stage but unavailable during the test phase can form privileged information.

In a different context, classifying time series into predefined classes has received significant attention in the kernel literature [11, 17, 65, 108]. Classification of time series is an important problem arising in many application domains [39, 56, 89]. In this context, each example is a time series with possibly correlated values. Moreover, in many real applications, the time series are often of variable length and can be very long. Therefore, classification of time series is quite different from classifying traditional static data. Consequently, classic kernels such as the Gaussian kernel, which were designed for static data, might fail to provide good similarity measurements. Recently, a new trend has emerged in the machine learning community, using models that are fitted on parts of data as more stable and parsimonious data representations. Learning is then performed directly in the model space, instead of the original data space. For example, Brodersen et al. [11] used a generative model of brain imaging data to represent fMRI measurements of different subjects through subject-specific models. They subsequently employed SVM on the models' parameters to distinguish aphasic patients from healthy controls.

Several kernels of this kind, which use generative models to represent the time series, have been developed, e.g. Fisher kernel [48], autoregressive kernel [26], probability product kernel [52], or Kullback-Leibler (KL) divergence based kernels [17, 65]. However, these approaches depend on the particular parametric model class. For example, *Fisher* kernel maps individual time series into score functions of the single generative model that is assumed to be able to "explain" most of the data. The generative model employed in Fisher kernel is often a hidden Markov model (HMM) with a fixed number of states [47]. In some situations the assumption of the particular generative model underlying the data can be too strong. Yet another approach based on autoregressive kernel [26] uses a vector autoregressive (VAR) model of a given order to generate an infinite family of features from the time series. Each time series is represented by its likelihood profile of VAR across all possible parameter settings (under a matrix normal-inverse Wishart prior). The kernel function is then defined as the dot product of the corresponding likelihood profiles. Still several approaches have been developed such as probability product kernels [52], KL divergence based kernels [17, 65] which represent each sequence using a generative model, opposing to a single model for all sequences employed in the Fisher kernel. However, many of the existing time series kernels such as Fisher kernels [47] and auto-correlation operators (DACO) kernel [35] are computationally demanding, thus may fail in the application of very long time series data. Fisher kernels [47] requires the calculation of metric tensor (inverse of Fisher information matrix) in the tangent space of the generative model manifold. The "practical" Fisher kernel used in most of the time replaces the metric tensor with an identity matrix. This can result in a loss of valuable information in the data [99]. DACO kernel [35] proposed recently by Gaidon et al. for action recognition, compares the dynamic aspects of two time series by using the difference between their auto-correlations. The kernelized DACO inevitably needs to invert a matrix of size related to the time series length. Thus the kernel can be used for relatively short time series only. Therefore, computational efficient time series kernels that can deal with time series of variable (possibly long) length and simultaneously can capture the dynamic aspects of the time series are in demand.

1.2 Contributions

The key contributions of this thesis are listed as follows:

- **Development of a new method that incorporates privileged information in SVM for ordinal regression.**

  In particular, extending support vector ordinal regression with implicit constraints (SVORIM), which classifies data into ordered categories, to the case where privileged information is available. This approach is applied in predicting future movements of ordinal scale where future events act as privileged information. Experimentally, the utility of privileged information improves the generalization performance in ordinal regression.

- **Construction of a novel efficient time series kernel based on reservoir

models.

The proposed time series kernel is constructed by representing the time series through linear readout models of the echo state network (ESN) with a shared deterministically constructed reservoir and the difference of two time series will be defined by the model distance between their corresponding linear readout models. The proposed kernel can naturally handle time series of variable length. Additionally, our kernel construction for time series do not need the specification of a particular parametric model class for the time series because reservoir models are flexible enough to be used for a variety of data types. Moreover, compared with most time series kernels, our kernels are computationally very efficient, since only the linear readout on top of the reservoir needs to be trained and the model distances between linear readouts can be formulated analytically under some assumptions. Furthermore, with the recursive least squares algorithm to train the readout mapping of reservoir models, our kernels can be operated in an on-line fashion, with the ability to efficiently handle extremely long time series.

- **Learning the deterministically constructed echo state network.**

  A hybrid algorithm has been developed to learn the deterministically constructed echo state network with a simple architecture of cycle connection with regular jumps. Empirically, the proposed algorithm tremendously reduces the computational time without jeopardizing the generalization performance.

- **co-Learning the reservoir model and the metric in linear readout model space.**

  Following the previous contribution of kernel construction based on echo state network with a highly constrained dynamic part, we develop a hybrid model-metric co-learning approach. This approach learns the global metric in the linear readout model space, alongside with the shared dynamic part. The learning objective is that time series from the same class are represented by “close” readouts, while readouts of time series from different classes are well-separated and simultaneously, the models should well represent the time series. The existing model based time series kernel formulations either used a single model fitted on the full data/individual classes, or fit a full linear dynamic model on each sequence. The models fitted on the full data/classes can be more complex than those fitted on individual sequences. The price to be paid is the potential inadequacy of such model to capture individual variations among the data items. On the other hand, individual sequence models have to be relatively simple (e.g. with linear dynamics) to keep the model estimation stable. Our approach tries to span the two extremes in an attempt to keep the best of both worlds. In our approach, the final representation of a sequence is a linear readout mapping, but the full underlying model is a non-linear dynamic system. In this way our methodology reduces the computational demands without the loss of the computational ability of non-linear models. Moreover, it treats the model parameters adaptation and model distance designing jointly rather than adapting the model parameters first and defining the model distance afterwards in two independent steps. Furthermore, it has the similar motivation as other discriminative kernels [99], but differs in its goal of utilizing models that both represent the time series well and at the same time best separate the time series classes. Extensive numerical experiments have been conducted to show the superior generalization performance of the proposed approach.

1.3 Outline of the Thesis

Chapter 2 reviews the basic information and research related to this document.

This chapter begins by a short introduction of time series, followed by a detailed description of SVMs. At the meantime, optimization theory that is used in the SVM framework such as Lagrangian and Karush-Kuhn-Tucker (KKT) conditions are introduced. Then it goes on to a brief description of recurrent neural network – a predictive time series model that will be used in our kernel construction. This chapter ends by a concise depiction of several exiting kernels that are exclusively designed for time series data.

Chapter 3 introduces a novel method that incorporates privileged information in the SVM framework for ordinal regression.

This chapter initially reviews the literature related to the LUPI paradigm in the context of binary SVM. Then a main focus is given to the proposed algorithm that incorporates privileged information in the framework of SVORIM, which constructs multiple parallel separating hyperplanes defined through ordered thresholds. The privileged information is exploited during training by modelling the slacks through correcting functions for each of the hyperplanes separating the ordered classes. The primal problem is formulated and then is transformed into its dual. Evaluation metrics that are used to assess the performance of our approaches are described and a number of numerical experiments on benchmark ordinal datasets and real world time series datasets have been conducted with the aim of verifying the proposed algorithm.

Chapter 4 introduces a novel and efficient time series kernel based on reservoir model.

This chapter at first describes the deterministically constructed reservoir models. Then it presents the methodology of constructing the time series kernel using reservoir model. The deterministically constructed reservoir models are used as representation of the time series. Each time series is represented by a linear readout model from the Echo State Network with a shared dynamic reservoir. In our model representation, the reservoir is fixed as the same for all the time series. The reservoir will transform the time series into a higher dimensional dynamic feature space and the varying aspects of each time series will be captured through variation in the linear readout models trained in such dynamic feature spaces using the time series. The distance between two time series is then defined by the model distance between their corresponding linear readout models. We will show how the model distance can be calculated analytically or efficiently estimated: under the assumption of uniformly distributed states, a closed form calculation of distances can be obtained, while in the case of non-uniformly distributed states, mixture of Gaussians and sampling techniques are used. The kernel similarity is defined by plugging the model distance obtained above into the Gaussian function. Moreover, the Fisher kernel based on reservoir model is introduced. Intensive numerical experiments have been conducted on UCR time series classification datasets in order to verify the efficiency of the proposed kernels. Furthermore, a fast version of the proposed kernel is described. This fast version is implemented by training the reservoir readouts in an on-line fashion using recursive least squares method.

Chapter 5 introduces an adaptive model metric co-learning methodology of constructing kernel for time series data.

An initial investigation of learning the deterministically constructed echo state networks is presented. The basic algorithm where the linear readout weights are determined by linear regression, while the reservoir weights are found using a nonlinear optimization techniques is described. Readout regularization and an early stopping strategy is presented in order to obtain good generalization performance. Numerical experiments have been conducted to verify the efficiency and generalization performance of the proposed algorithm. A main focus is placed on co-learning the dynamic part of the reservoir model and the global metric in the linear readout space. The learning objective is to encourage time series from the same class to have close representations and those from different class to be well separated, as well as the model provides faithful representation for the time series. A number of numerical experiments have been performed on UCR time series classification datasets in order to assess the performance of the proposed kernel.

Finally, Chapter 6 summarizes the presented work and suggests a future research plan.

1.4 Publications From the Thesis

- Journal Publications:
  - Fengzhen Tang, Peter Tiňo, Pedro Antonio Gutiérrez and Huanhuan Chen: The Benefits of Modelling Slack Variables SVMs, Neural Computation, 2014, accepted.
- Conference Publications:
  - Fengzhen Tang, Peter Tiňo and Huanhuan Chen: Learning Deterministically Constructed Echo State Networks. The IEEE World Congress on Computational Intelligence (IEEE WCCI), 2014, pp. 77 - 83 .
  - Fengzhen Tang, Peter Tiňo, Pedro Antonio Gutiérrez and Huanhuan Chen: Support Vector Ordinal Regression using Privileged Information. In Proceedings of the 2014 European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN2014), pp. 253-258
  - Huanhuan Chen, Fengzhen Tang, Peter Tiňo and Xin Yao: Model-based Kernel for Efficient Time Series Analysis. 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'13), Accepted for oral presentation, 2013.

CHAPTER 2

Background and Related Work

2.1 Introduction

Kernel-based support vector machines (SVMs) are powerful learning machines [100]. They are effective for many problems on numerous applications such as digit recognition, face detection, speaker identification and so on [13]. In the application domain of time series, SVMs have been successfully utilized for both time series prediction [55, 66] and classification [11, 17, 65, 108]. A time series is a collection of observations made sequentially through time. Thus, there is temporal ordering within time series, making it different from other type of statistic data where the observations are usually expected to be independent. Consequently, it is non-trivial to apply kernel methods such as SVMs to time series data.

SVMs have been applied for time series *prediction* successfully on the basis of the sliding window method [55, 66]. Time series prediction, which is to predict future values from the past observations, is an important task of time series analysis. The sliding window method transforms the time series prediction into classical supervised learning problem. Given a time series $s(1), ..., s(t), ..., s(L)$, where $s(t)$ is value of time series at time stamp t, the method will split the time series into windows of fixed length and then constructs training examples as input-output pairs, e.g. $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{L-p}$, where $\boldsymbol{x}_i = (s(i), s(i+1), ..., s(i+p-1))^T$ and $y_i = s(i+p)$, with the window length p. The classical supervised learning algorithm such as Support Vector Machine, Relevant Vector Machine [73] and Gaussian process [9,10] can then be utilized directly on the constructed examples.

Another important task applied on time series data is *classification*. In this task, the problem is to predict a single label y that applies to an entire input sequence $(s(1), ..., s(L))$. For example, given a time series of ECG data (the time series of heart rates), the task is to identify the health condition of the person from whom the time series of ECG data is measured. In these kind of problems, each training example consists of a pair $(\boldsymbol{x}_i, y_i)$, where $\boldsymbol{x}_i$ is a sequence $\boldsymbol{x}_i = (s(1), ..., s(L_i))$ and each y_i is a class label (such as a person's health condition– healthy or ill).

This chapter first gives a brief introduction on time series. Basic concepts of time series and methods of modeling time series are demonstrated. Then it moves on to SVM which is the basic classifier in this thesis. SVMs for classification are described in details. After that kernels which are exclusively designed for time series data are reviewed. Finally, statistic tests for comparing performance of two methods are briefly introduced.

2.2 Time series

A time series is a collection of observations made sequentially through time [20]. When the observations are taken continuously through time, the time series is said to be continuous, while the time series is claimed to be discrete when the observations are taken at specific times, commonly equally spaced intervals. This thesis involves discrete time series, where the observations are taken at equal intervals, for instance, daily air temperatures, monthly readings of electricity meter, yearly sales figures and so on. These time series where at every single time one observation is made are called univariate time series. However, there exist time series where each time multiple observations are measured, for example many different economic activities such as the retail price index, the level of unemployment and the gross domestic product are recorded at regular time intervals [20]. This kind of time series is named as multivariate time series and will be stored in a matrix rather than a vector that is used to store the univariate time series.

One of the most important methods of describing time series data is the autocorrelation coefficient, which is the correlation of the time series with a temporally shifted version of itself [35]. Given a time series of L observations $s(1), s(2), ..., s(L)$, under the assumption that the time series is stationary, i.e. the mean and variance of the time series are not changing through time, the sample auto-correlation can be computed as follows:

$$\hat{\rho}_\tau = \frac{\sum_{t=1}^{L-\tau}(s(t)-\bar{s})(s(t+\tau)-\bar{s})}{\sum_{t=1}^{T}(s(t)-\bar{s})^2}, \tag{2.1}$$

where τ is the time lag and $\bar{s} = \frac{1}{L}\sum_{t=1}^{L} s(t)$ is the sample mean. The autocorrelation coefficient is an important tool for describing the properties of a time series statistically [20]. It contains the information of temporal dependencies in time series. However, the assumption of stationarity in the calculation of autocorrelation limits its application to stationary time series. Moreover, autocorrelation coefficient only provides a linear dependence within the time series.

The autocorrelation is a distinguishing feature in time series, suggesting that future values of the time series depend, usually in a stochastic manner, on the past observations. Thus it provides possibilities to predict the future from the past. This raises the problems of predication or forecasting, an important task of time series analysis. Deterministic methods, e.g. SVMs [55, 66], artificial neural network (ANN) [31], and recurrent neural network (RNN) [97], have been applied for time series prediction. However, statisticians usually treat a time series as a realization from a stochastic process. Probabilistic models, for example, autoregressive (AR) model [20], hidden Markov model (HMM) [69], and linear dynamic system (LDS) [29], are designed to capture the underlying dynamics from which the observed data are generated and therefore the forecasting will be made.

2.2.1 Autoregressive Models

Autoregressive (AR) models are attractive time series models for they are easy to specify and estimate [41]. However they are limited to situations where the assumption of linear dependence is reasonable. Specifically, they are under the assumption that the future values of time series depend linearly on the finite number of immediate past (possibly noisy) observations [20]. Denote the observations of the time series measured at time t as $s(t)$, the autoregressive model of order p, denoted by AR(p) which is parametrized by a horizon p and a coefficient vector $\alpha \in \mathbb{R}^p$, assumes that the time series is generated according to the following model:

$$s(t) = \sum_{i=1}^{p} \alpha_i s(t-i) + \varepsilon(t) \tag{2.2}$$

where $\varepsilon(t)$ is white-noise, which means that $E\varepsilon(t) = 0$, $var(\varepsilon(t)) = \sigma^2$ and $cov(\varepsilon_i, \varepsilon_j) = 0$ for all $i \neq j$. AR model is essential a multiple regression model but it is regresses on past values rather than on separate predictor variables. The parameter $\alpha_i, i = 1, ..., p$ can be estimated by maximum likelihood or Bayesian inference [6].

AR models are also under the condition of stationary time series. They can be expected to yield good short-term forecasts in many situations. The generalization of AR models to multivariate series is rather straightforward and called vector autoregressive (VAR) models.

2.2.2 Hidden Markov Model

Hidden Markov Model is a ubiquitous tool for modeling time series data. It is widely used in speech recognition, natural language processing and online handwriting recognition [6]. It can be regarded as a probabilistic state-space model with discrete hidden state variable, trying to represent the sequence of observations by probability distributions. It is mainly defined by two probability distributions: the observation distribution $p(\boldsymbol{s}(t)|\boldsymbol{z}(t))$ and the transition distribution $p(\boldsymbol{z}(t)|\boldsymbol{z}(t-1))$ where $\boldsymbol{s}(t)$ is the observation at time t and $\boldsymbol{z}(t)$ is the hidden state at time t. The probability distribution $p(\boldsymbol{s}(t)|\boldsymbol{z}(t))$ tells how the observation relates to the hidden states, i.e. the current observation $\boldsymbol{s}(t)$ is only depend on the current state $\boldsymbol{z}(t)$ but not on any other states and observations at other time instances, while $p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})$ indicates how the states evolve over time, i.e. is the value of current state $\boldsymbol{z}_t$ is dependent only on the previous state $\boldsymbol{z}_{t-1}$ and independent of all the states prior to $t-1$. This means that the hidden states satisfy the Markov property and the observations also satisfy a Markov property with respect to the states [38]. By assuming these Markov properties, the joint probability of a sequence of observations and states are given as follows:

$$p(S, Z) \quad = \quad p(\boldsymbol{z}_1)p(\boldsymbol{s}_1|\boldsymbol{z}_1)\prod_{t=2}^{T} p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})p(\boldsymbol{s}_t|\boldsymbol{z}_t) \tag{2.3}$$

where S denotes the observation sequence $\boldsymbol{s}_1, ..., \boldsymbol{s}_T$ and Z denotes the sequence of states $\boldsymbol{z}_1, ..., \boldsymbol{z}_T$.

Suppose that the discrete hidden state variable $\boldsymbol{z}_t$ can take on K values which can be denoted by the integers $\{1, ..., K\}$, then the HMM needs to specify a probability distribution over initial state $p(\boldsymbol{z}_1)$, the $K \times K$ state transition matrix that defines transition probability $p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})$ and emission matrix that defines observation probability $p(\boldsymbol{s}_t|\boldsymbol{z}_t)$. The problem of learning the parameters in an HMM can be formed as a problem of maximization of the following probability:

$$p(S) = \sum_{Z} p(S, Z) \tag{2.4}$$

by Expectation Maximization algorithm [7, 37, 94]. This is essentially to optimize the HMM such that it could best explain a given observation sequence. Usually, each probability in the state transition matrix and in the emission is assumed time independent, meaning that the matrices do not change in time as the system evolves. Therefore HMM is also restricted to stationary time series.

2.2.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) represent a class of artificial neural networks, where the connection topology of the network contains cycles [62]. This cyclic topology creates a time-dependent internal state for the network and enables the network to exhibit dynamic behaviour, consequently making RNNs highly promising for dealing with time series processing problems [62,97]. RNNs can be shown to universally appropriate dynamic systems, under fairly mild and general assumptions [62].

A simple form of fully recurrent network simply has the previous set of neuron (hidden unit) activations feeding back into the network, together with the inputs. In order to simplify the notation here, the bias term will not be explicitly written out, instead, the corresponding variables will be augmented by adding an element equal to one. The notation $\tilde{\mathbf{s}}(t) = [\mathbf{s}(t); 1]$ is adopted. The behaviour of such RNN can be described as a dynamic system mathematically as follows:

$$\mathbf{z}(t) = f_{\mathbf{z}}\left(R\,\mathbf{z}(t-1) + V\;\tilde{\mathbf{s}}(t)\right) \tag{2.5}$$

$$\boldsymbol{o}(t) = f_{\mathbf{o}}\left(W\tilde{\mathbf{z}}(t)\right) \tag{2.6}$$

where $\mathbf{z}(t) \in \mathbb{R}^N$ is a vector of neuron activations (state vector); $\mathbf{s}(t) \in \mathbb{R}^d$ is the input vector; $\boldsymbol{o}(t) \in \mathbb{R}^M$ is the actual output vector of the network ; $f_{\boldsymbol{z}}(\cdot)$ and $f_{\mathbf{o}}(\cdot)$ are the non-linear activation functions, usually $\tanh(\cdot)$ or sigmoid function, applied element-wise and $R \in \mathbb{R}^{N\times N}$, $V \in \mathbb{R}^{N\times(d+1)}$ and $W \in \mathbb{R}^{M\times(N+1)}$ are three weight matrices. The weight matrix R collects the connections between internal units, matrix V describes the connection between the input units and matrix W represents the connections between internal units and output units. The network is commonly started with the initial state $\boldsymbol{z}(0) = \mathbf{0}$.

RNNs can be trained by gradient-descent based algorithms. The gradients can be estimated using real-time recurrent learning [106] where the gradients are recursively computed at each time step. The gradients can also be computed using backpropagation through time [79]. However, the training process of fully RNN is very computationally expensive and often stuck in local minima [3].

Reservoir Computing (RC) For the purpose of accelerating the training process, reservoir computing (RC), which represents a class of new approaches of designing RNN, have been developed. The training techniques applied in RC methods make a conceptual and computational separation of the whole process into two parts [62, 77]: 1) representation of temporal structure in the input stream through a non-adaptable dynamic reservoir, 2) a linear recurrence-free readout that produces the desired output from the reservoir. In these methods, such as echo state networks (ESNs) [49] and liquid state machines (LSMs) [63], a recurrent neural network is randomly constructed and fixed through the training process. Such randomly generated and fixed RNN is called dynamic reservoir. The reservoir is excited by the input signal and maintains a non-linear transformation of the input history in its state. The desired output is described as a linear combination of the neuron activations (readout) from the input excited reservoir. Only the readout weights need to be obtained through training, usually by linear regression using the teacher signal as a target. Therefore RC methods are very computationally efficient. Reservoir models [62] have been extensively shown to be able to successfully process and model time series of a surprisingly wide variety of types (from deeper memory deterministic chaotic systems, to shorter memory stochastic sequences) [76, 96].

Echo State Networks (ESNs) ESN [50, 51] is one of the pioneering reservoir computing methods. It is of simple form but very effective. The network architecture for ESN is show in Figure 2.1.

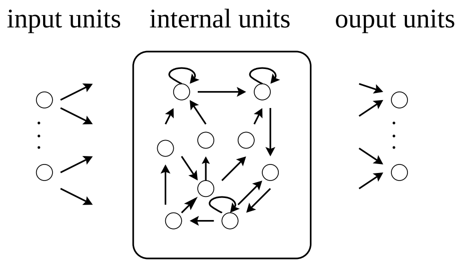


Figure 2.1: The network architecture of ESN. In this architecture, the input units to internal units are randomly connected and internal units are randomly connected to each other.

ESN is essentially a recurrent neural network with a randomly generated and fixed sparse recurrent part (the reservoir) and a very simple linear readout. The ESN reservoir model can be formulated as:

$$\boldsymbol{z}(t) = \tanh(R\,\boldsymbol{z}(t-1) + V\tilde{\boldsymbol{s}}(t)), \tag{2.7}$$

$$\boldsymbol{o}(t) = W\tilde{\boldsymbol{z}}(t), \tag{2.8}$$

In ESM model, the output neuron activation function is identity function, which suggests that the output $\boldsymbol{o}(t)$ is a linear function of states $\boldsymbol{z}$. ESN has a "non-trainable" recurrent part ("the reservoir") described by equation (2.7) and a simple linear readout shown as equation (2.8). Typically, the reservoir weights R and the input weights V to the reservoir are randomly generated so that the "echo state property" is satisfied. Loosely speaking, this means that the reservoir output would be independent of the initial conditions [49]. Training of ESN can be efficiently performed through linear regression [62], which is to find the least square solutions of a system of linear equations given as follows:

$$W\tilde{Z} = Y \tag{2.9}$$

through minimizing a quadratic error of true target Y and the model $W\tilde{Z}$. As before, $\tilde{Z}$ is the extended version of Z by adding a constant row equal to one in order to accommodate a bias term, where $Z \in \mathbb{R}^{N\times L}$ are all the network states $\boldsymbol{z}(t)$ produced by providing input $\boldsymbol{s}(t)$. $Y \in \mathbb{R}^{M\times T}$ are all the true targets $\boldsymbol{y}(t)$. Both matrices are collected during training period $t = 1, ..., L$, after the initial wash-out[1] [49]. A naive solution of Equation (2.9) is given as follows:

$$W = Y\tilde{Z}^T(\tilde{Z}\tilde{Z}^T)^{-1} \tag{2.10}$$

and a more elegant solution with regularization on parameter W, which is known as ridge regression, is given as follows:

$$W = Y\tilde{Z}^T(\tilde{Z}\tilde{Z}^T + \lambda I)^{-1} \tag{2.11}$$

where $I \in \mathbb{R}^{(N+1)\times(N+1)}$ is the identity matrix and λ is a regularization factor.

Ridge regression is a preferable method for learning W in ESN [62]. However it is a batch learning method. Some applications require even faster model adaptation, where recursive least squares (RLS), which is a fast online model adaptation method known for linear systems, can be adopted [51].

ESN has been successfully applied in many time-series prediction task, such as speech

[1] The data of $\boldsymbol{z}(t)$ from the beginning of the training run are dismissed, in order to remove the impact of the initial transients.

recognition, dynamic pattern prediction and language modelling [62, 77].

The downside of reservoir models is that their construction is largely driven by a series of randomized model building stages. Recently, Rodan et al. [76] proposed to use a simple deterministically constructed cycle reservoir with regular jumps (CRJ). This reservoir architecture has been shown to be comparable (or better) than the traditional ESN on a wide variety of time series modelling and prediction tasks [76]. In CRJ the reservoir nodes are connected in a uni-directional cycle with bi-directional shortcuts (jumps) (Figure 2.2). All cyclic connections have the same weight denoted by r_c, all jumps share the same weight denoted by r_j, and the input connections have the same absolute value denoted by r_i with an aperiodic sign pattern. This results in a sparse and deterministically constructed and simple coupling reservoir weight matrix R, i.e. R is a very sparse matrix with r_c and r_j spread over, e. g. in a network of 10 internal units with 2 as jump size, the matrix R is of the form as follows:

$$\begin{pmatrix}
0 & 0 & r_j & 0 & 0 & 0 & 0 & 0 & r_j & r_c \\
r_c & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
r_j & r_c & 0 & 0 & r_j & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & r_c & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & r_j & r_c & 0 & 0 & r_j & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & r_c & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & r_j & r_c & 0 & 0 & r_j & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & r_c & 0 & 0 & 0 \\
r_j & 0 & 0 & 0 & 0 & 0 & r_j & r_c & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & r_c & 0
\end{pmatrix}$$

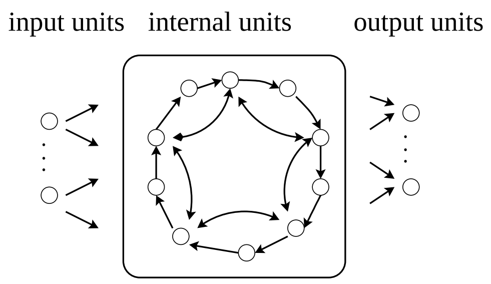


Figure 2.2: The network architecture of CRJ. In this architecture, the input units are connected to internal units by the same weight with aperiodic sign. The internal units are connected by directed cycle with regular jumps. All the cycle collection weights are the same and all the jump weights are the same.

2.3 Support Vector Machines

SVMs are very popular supervised learning machines [2, 13, 18, 66, 71]. The basic idea of SVM is to construct a separating hyperplane as the decision boundary that separates positive examples from negative ones with maximum margin. To deal with the case of overlapping classes, SVM formulations utilize non-negative slack variables to tolerate misclassification in training data. Non-linear class separation structure can be addressed through the so-called "kernel trick" – kernels map the input data into a higher dimensional feature space where a linear separation hyperplane can be applied. SVM can be extended for multi-class classification problems via ad hoc approaches such as one-versus-all or one-versus-one. It has been extended for ordinal regression by constructing multiple parallel separating hyperplanes through multiple ordered thresholds [23, 24, 44, 90]. SVM for ordinal regression is one of the main focuses in this thesis and more detailed description will be given in Chapter 3. This section first briefly introduces the convex optimization theory related to SVMs. Then it describes SVM through three classical cases – linear SVM trained on separable data, linear SVM trained on non-separable data, and non-linear SVM.

2.3.1 Convex Optimization Theory

The determination of model parameters in an SVM amounts to solve a constrained convex optimization problem. Thus, convex optimization theory that relates to SVMs is briefly described in this section.

Definition 2.1 (convexity [8]) *Function $f(\boldsymbol{x})$ is called convex if for any two points $\boldsymbol{x}$ and $\boldsymbol{y}$ in the domain of f, the Jensen inequality*

$$f(a\boldsymbol{x} + (1-a)\boldsymbol{y}) \leqslant af(\boldsymbol{x}) + (1-a)f(\boldsymbol{y}), 0 \leqslant a \leqslant 1, \tag{2.12}$$

holds true.

The secant line of the $f(\boldsymbol{x})$ passing the points $(\boldsymbol{x}, f(\boldsymbol{x}))$ and $(\boldsymbol{y}, f(\boldsymbol{y}))$ is $y' = \frac{f(\boldsymbol{y})-f(\boldsymbol{x})}{\boldsymbol{y}-\boldsymbol{x}}(\boldsymbol{x}'-\boldsymbol{y}) + f(\boldsymbol{y})$. The point $a\boldsymbol{x} + (1-a)\boldsymbol{y}$, where $0 \leqslant a \leqslant 1$, is located any where on the straight line between $\boldsymbol{x}$ and $\boldsymbol{y}$ (including the points $\boldsymbol{x}$ and $\boldsymbol{y}$ themselves) with the corresponding secant function value of $af(\boldsymbol{x}) + (1-a)f(\boldsymbol{y})$. Therefore, the definition of the convexity of a function can be interpreted in the way that the function value is no greater than its corresponding secant function value.

Consider a constrained convex optimization problem in the standard form as in [8]:

$$\min_{\boldsymbol{x}} \quad f_0(\boldsymbol{x}) \tag{2.13}$$

$$\text{s.t.} \quad f_i(\boldsymbol{x}) \leqslant 0, i = 1, ..., m, \tag{2.14}$$

$$h_k(\boldsymbol{x}) = \boldsymbol{a}_k^T\boldsymbol{x} - b_k = 0, k = 1, ..., q, \tag{2.15}$$

where $f_i(\boldsymbol{x}), i = 0, 1, ..., m$ are convex. In other words, a convex optimization problem requires that the objective function and the inequality constraint functions (2.14) are convex, while the equality constraint functions (2.15) are affine (linear). The feasible set $\mathcal{D}$ of the optimization problem (2.13)-(2.15) is given as follows:

$$\mathcal{D} = \{\boldsymbol{x} \mid f_i(\boldsymbol{x}) \leqslant 0, i = 1, \ldots, m; h_k(\boldsymbol{x}) = 0, k = 1, \ldots, q; \boldsymbol{x} \in \mathbb{R}^d\}. \tag{2.16}$$

We assume the feasible set $\mathcal{D}$ is nonempty and denote the optimal solution by p^* ($p^* = \inf\{f_0(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{D}\}$).

Instead of explicitly enforcing the constraints, Lagrange suggested to take the constraints into account by augmenting the objective function with a weighted sum of the constraints [6, 100], given as follows:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{v}) = f_0(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{x}) + \sum_{k=1}^{q} v_k h_k(\boldsymbol{x}), \tag{2.17}$$

where λ_i and v_k are the Lagrange multipliers associated to the i-th inequality constraint $f_k(\boldsymbol{x}) \leqslant 0$ and the k-th equality constraint $h_k(\boldsymbol{x}) = 0$, respectively. The vectors $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_m)$ and $\boldsymbol{v} = (v_1, ..., v_q)$ are called the dual variables or Lagrange multiplier vectors. The Lagrangian dual function is defined as follows:

Definition 2.2 (Dual Function [8]) *Suppose $g : \boldsymbol{R}^m \times \boldsymbol{R}^q \to R$, g is said to be the dual function of the Lagrangian $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{v})$ if it has the minimum value of the Lagrangian with respect to $\boldsymbol{x}$: for $\boldsymbol{\lambda} \in \boldsymbol{R}^m, \boldsymbol{v} \in \boldsymbol{R}^q$,*

$$g(\boldsymbol{\lambda}, \boldsymbol{v}) = \inf_{\boldsymbol{x} \in \mathbb{R}^d} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{v}) = \inf_{\boldsymbol{x} \in \mathbb{R}^d} \left(f_0(\boldsymbol{x}) + \sum_{k=1}^{m} \lambda_k f_k(\boldsymbol{x}) + \sum_{k=1}^{q} v_k h_k(\boldsymbol{x}) \right). \tag{2.18}$$

The dual function yields lower bound on the optimal value p^* of problem (2.13): For any $\boldsymbol{\lambda} \geqslant \boldsymbol{0}$ and any $\boldsymbol{v}$, we have $g(\boldsymbol{\lambda}, \boldsymbol{v}) \leqslant p^*$. This property is easy to verify. If $\boldsymbol{x} \in \mathcal{D}$ and $\boldsymbol{\lambda} \geqslant \boldsymbol{0}$, then we have $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{v}) \leqslant f_0(\boldsymbol{x})$. It is straightforward to see from the definition of p^* that:

$$g(\boldsymbol{\lambda}, \boldsymbol{v}) = \inf_{\boldsymbol{x}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{v}) \leqslant \inf_{\boldsymbol{x}\in\mathcal{D}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{v}) \leqslant \inf_{\boldsymbol{x}\in\mathcal{D}} f_0(\boldsymbol{x}) = p^*. \tag{2.19}$$

The dual function gives us a lower bound on the optimal p^* that depends on the parameters $\boldsymbol{\lambda}$ and $\boldsymbol{v}$, and the best lower bound is closest to the optimal p^*. This leads to the optimization problem:

$$\begin{aligned} \max \quad & g(\boldsymbol{\lambda}, \boldsymbol{v}) \\ \text{s.t.} \quad & \boldsymbol{\lambda} \geqslant \mathbf{0}. \end{aligned} \tag{2.20}$$

Denote the optimal value of the Lagrangian dual problem (2.20) as d^*, which by definition is the best lower bound on p^* that can be obtained from the Lagrangian dual function, we trivially always have

$$d^* \leqslant p^*. \tag{2.21}$$

This property is called weak duality, which means there may be a gap between the optimal value of the original (primal) problem and the optimal value of the dual function. This gap is called optimal duality gap. If the equality $d^* = p^*$ holds, i.e. the optimal duality gap is 0, then we can say strong duality holds true. Strong duality suggests that the dual optimal value obtained from (2.20) exactly equals to the primal optimal value obtained from (2.13). However, strong duality usually does but not always hold in a convex optimization problem. Further conditions beyond convexity need to be specified to guarantee strong duality. These conditions are called constraint qualification. One simple constraint qualification is the Slater's condition, which is given as follows:

Definition 2.3 (Slater's Condition [8]) *Suppose we have an optimization problem with inequality constraints $f_i(\boldsymbol{x}) \leqslant 0, i = 1, ..., m$; and equality constraints $h_k(\boldsymbol{x}) = 0, k =$*

$1, \ldots, q$, *we say the constraints meet Slater's condition if there exists an* $\boldsymbol{x}_0$ *such that*

$$f_i(\boldsymbol{x}_0) < 0, i = 1, ..., m; \quad h_k(\boldsymbol{x}_0) = 0, k = 1, \ldots, q, \tag{2.22}$$

holds. The Slater's condition can be refined when some of the inequality constraints are linear (affine), as follows:

Definition 2.4 (Refined Slater's Condition [8]) *Suppose we have an optimization problem with inequality constraints* $f_i(\boldsymbol{x}) \leqslant 0, i = 1, ..., m$; *and equality constraints* $h_k(\boldsymbol{x}) = 0, k = 1, \ldots, q$, *and the first* j *inequality constraint functions* $f_1, ..., f_j$ *are affine, we say the constraints meet Slater's condition if there exists an* $\boldsymbol{x}_0$ *such that*

$$f_i(\boldsymbol{x}_0) \leqslant 0, i = 1, ..., j; \quad f_i(\boldsymbol{x}_0) < 0, i = j+1, \ldots, m; \quad h_k(\boldsymbol{x}_0) = 0, k = 1, \ldots, q. \tag{2.23}$$

The refined Slater's condition suggests that affine inequalities do not need to hold with strict inequality to guarantee strong duality. Thus, the refined Slater's condition reduces to feasibility if all the constrains (both of equality and inequality types) are linear.

If a convex optimization problem with differentiable objective and constraint functions satisfying the Slater's condition, the optimality of the solution is characterized by Karush-Kuhn-Tucker (KKT) conditions [8] given as follows:

Definition 2.5 (Karush-Kuhn-Tucker conditions [8]) *Consider the convex optimization problem demonstrated by equation* (2.13). *Suppose that the objective function* $f_0(\boldsymbol{x})$ *and the constraint functions* $f_i(\boldsymbol{x})$ *and* $h_k(\boldsymbol{x})$ *are continuously differentiable at a point* $\boldsymbol{x}^*$ *and* $\boldsymbol{x}^*$ *is an optimal solution to problem* (2.13) *that satisfies Slater's condition, we say* $\boldsymbol{x}^*$ *meets KKT conditions if there exist Lagrange multipliers* $\boldsymbol{\lambda}^* = (\lambda_1^*, ..., \lambda_m^*)$ *and* $\boldsymbol{v}^* = (v_1^*, ..., v_q^*)$ *such that the corresponding Lagrangian function given by equation* (2.17) *satisfies the following conditions:*

(a) Stationality:

$$\frac{\partial \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{v}^*)}{\partial \boldsymbol{x}} = \frac{\partial f_0(\boldsymbol{x}^*)}{\partial \boldsymbol{x}} + \sum_{k=1}^{m} \lambda_k \frac{\partial f_k(\boldsymbol{x}^*)}{\partial \boldsymbol{x}} + \sum_{k=1}^{p} v_k \frac{\partial h_k(\boldsymbol{x}^*)}{\partial \boldsymbol{x}} = 0, \tag{2.24}$$

(b) Primal feasibility:

$$f_k(\boldsymbol{x}^*) \leqslant 0, k = 1, ..., m, \tag{2.25}$$

$$h_k(\boldsymbol{x}^*) = 0, k = 1, ..., p, \tag{2.26}$$

(c) Dual feasibility:

$$\lambda_k^* \geqslant 0, k = 1, ..., m, \tag{2.27}$$

(d) Complementary slackness:

$$\lambda_k^* f_k(\boldsymbol{x}^*) = 0, k = 1, ..., m. \tag{2.28}$$

KKT conditions are also sufficient for the points $\boldsymbol{x}^*$ and $(\boldsymbol{\lambda}^*, \boldsymbol{v}^*)$ to be primal and dual optimal, i.e. if $\boldsymbol{x}^*$, $\boldsymbol{\lambda}^*$ and $\boldsymbol{v}^*$ are any points that satisfy the KKT conditions (2.24)–(2.28), then $\boldsymbol{x}^*$ and $(\boldsymbol{\lambda}^*, \boldsymbol{v}^*)$ are primal and dual optimal, with strong duality (zero duality gap).

Theorem 2.1 (Karush-Kuhn-Tucker Theorem [8]) *Consider the convex optimization problem demonstrated by equation (2.13). Suppose that the objective function $f_0(\boldsymbol{x})$ and the constraint functions $f_i(\boldsymbol{x})$ and $h_k(\boldsymbol{x})$ are continuously differentiable at a point $\boldsymbol{x}^*$. The sufficient and necessary condition for $\boldsymbol{x}^*$ to be an optimal solution to problem (2.13) is that it meets KKT conditions.*

KKT conditions suggest a way to solve the convex optimization problem described in (2.13) which is to solve the equations defined in KKT conditions.

2.3.2 Support Vector Machine for Classification

In this section, we review SVM for binary classification problems [13]. Given a training set of n examples, represented by input-output pairs $(\boldsymbol{x}_i, y_i)$, $\boldsymbol{x}_i \in R^d$, $y_i \in \{+1, -1\}$, the aim of SVM for classification is to construct a decision boundary (separating hyperplane) that separates positive examples from the negative ones with maximum margin. SVM will be described iteratively by three cases: linear SVM trained on separable data, linear SVM trained on non-separable data and non-linear SVM.

We start with the simplest case where a linear decision boundary can be found with no error in the training set. The linear decision boundary also known as separating hyperplane can be defined as $\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$, where $\boldsymbol{w}$ is normal to the hyperplane and b is the bias term. The task is to find that $\boldsymbol{w}$ and b such that each positive example $\boldsymbol{x}_i$ (with class label $y_i = +1$) satisfy the inequality $\boldsymbol{w} \cdot \boldsymbol{x}_i + b > 0$, while each negative example $\boldsymbol{x}_i$ (with class label $y_i = -1$) satisfy the inequality $\boldsymbol{w} \cdot \boldsymbol{x}_i + b < 0$. However, in order to enable the classifier to have better generalization performance, during the learning process, every positive point $\boldsymbol{x}_i$ is required to meet the inequality $\boldsymbol{w} \cdot \boldsymbol{x}_i + b \geqslant +1$ (it can be any positive number but all of them are equivalent since rescaling will not change the hyperplane) and every negative point $\boldsymbol{x}_i$ is required to meet the inequality $\boldsymbol{w} \cdot \boldsymbol{x}_i + b \leqslant -1$. These two types of inequalities can be collectively formulated as one set of inequalities as $y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geqslant 1$. Only the points which exactly lie on the hyperplanes $\boldsymbol{w} \cdot \boldsymbol{x}_i + b = +1$ or hyperplane $\boldsymbol{w} \cdot \boldsymbol{x}_i + b = -1$ contribute to finding $\boldsymbol{w}$ and b therefore they are called support vectors. The perpendicular distance between the decision boundary and any of the support vectors is defined as margin and the margin is simply $1/||\boldsymbol{w}||$. The separating hyperplane can be constructed in many ways. Since an approximation as good as possible is desired, the separating hyperplane which maximizes the margin is chosen. Maximizing the margin $||\boldsymbol{w}||^-1$ is equivalent to minimizing $||\boldsymbol{w}||^2$. So, the problem of SVM can be formulated as an optimization problem with inequality constraints as follows:

$$\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2 \tag{2.29}$$

$$\text{s. t.} \quad 1 - y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \leqslant 0, \ \forall\ i, \tag{2.30}$$

where $||\boldsymbol{w}||$ is the Euclidean norm of $\boldsymbol{w}$. The objective function is convex quadratic and the constraint functions are linear. Thus, the optimization problem described by equations (2.29) and (2.30) is a convex optimization problem whose constraints meet the Slater's condition. Therefore this optimization problem can be solved using KKT conditions as demonstrated in Section 2.3.1.

First, the Lagrangian is constructed with Lagrange multipliers $\alpha_i \geqslant 0$,

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 - \sum_{i=1}^{n} \alpha_i [y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) - 1], \tag{2.31}$$

and then it is minimized with respect to $\boldsymbol{w}$, b to obtain the dual function. The minimization of $\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda})$ requires the gradient of $\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda})$ with respect to $\boldsymbol{w}$ and b vanish:

$$\frac{\partial \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda})}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i = 0, \tag{2.32}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda})}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0, \tag{2.33}$$

which gives the following conditions:

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i x_i, \quad \sum_{i=1}^{n} \alpha_i y_i = 0. \tag{2.34}$$

Since these are equality constraints, they can be substituted into the Lagrangian (2.31) to obtain the Lagrangian dual:

$$\begin{aligned} g(\boldsymbol{\alpha}) &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i\cdot\boldsymbol{x}_j - \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i\cdot\boldsymbol{x}_j + \sum_{i=1}^{n}\alpha_i - b\sum_{i=1}^{n} y_i\alpha_i \\ &= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i\cdot\boldsymbol{x}_j. \end{aligned}$$

Then, the Lagrangian dual is maximized as suggested in section 2.3.1. This leads to the following optimization problem:

$$\begin{aligned} &\max_{\boldsymbol{\alpha}}\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i\cdot\boldsymbol{x}_j \\ &\text{s. t. } \sum_{i=1}^{n}\alpha_i y_i = 0,\ \alpha_i \geqslant 0,\ \forall\ i. \end{aligned} \tag{2.35}$$

Since the primal problem (2.29) is a convex optimization problem satisfying constraint qualification mentioned in Section 2.3.1, the optimal solution of the dual problem (2.35) will be equivalent to the optimal solution of the primal problem (2.29). Consequently, the primal optimization will be solved by solving its corresponding dual problem. Once the optimal $\tilde{\boldsymbol{\alpha}}$ is obtained, $\tilde{\boldsymbol{w}} = \sum_{i=1}^{n}\tilde{\alpha}_i y_i \boldsymbol{x}_i$ and the KKT complementary slackness:

$$\tilde{\alpha}_i\left(1 - y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + \tilde{b})\right) = 0,\ \forall\ i, \tag{2.36}$$

can be used to determine the optimal bias $\tilde{b}$ by any pattern j for which $\tilde{\alpha}_j \neq 0$ through:

$$\tilde{b} = y_j - \sum_{i=1}^{n}\tilde{\alpha}_i y_i \boldsymbol{x}_i\cdot\boldsymbol{x}_j \tag{2.37}$$

Usually the average over all such points is taken. Finally, the decision function for a new input vector $\boldsymbol{x}_0$ is given by:

$$F(\boldsymbol{x}_0) = \operatorname{sgn}\left(\sum_{i=1}^{n} \tilde{\alpha}_i y_i \boldsymbol{x}_i \cdot \boldsymbol{x}_0 + \tilde{b}\right). \tag{2.38}$$

The method mentioned above is designed for linearly separable data. When it is applied to linearly non-separable data, there will be no feasible solution. However, the constrains can be relaxed when necessary by introducing non-negative slack variables $\xi_i, i = 1, ..., n$ and solve:

$$\begin{aligned} &\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}\xi_i \\ &\text{s. t.} \;\; 1 - \xi_i - y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) \leqslant 0,\; \xi_i \geqslant 0,\; \forall\, i, \end{aligned} \tag{2.39}$$

where $C \geq 0$ is a hyper-parameter chosen by user. In this case, besides maximizing the margin, minimizing the violation of the constraints is contained in the objective function. Still, this is a convex optimization problem satisfying the constraint qualification. Following the same procedure of the separable case, the Lagrangian is constructed with the presence of two kinds of multipliers $\alpha_i \geqslant 0, \beta_i \geqslant 0$ in this case:

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i[y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i) + b - 1 + \xi_i] - \sum_{i=1}^{n}\beta_i\xi_i. \tag{2.40}$$

Similarly, the Lagrangian $\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ is minimized with respect to $\boldsymbol{w}$, b and $\boldsymbol{\xi}$. In this case, besides the gradient of the Lagrangian with respect to $\boldsymbol{w}$ and b is required to vanish as stated in equations (2.32) and (2.33), the gradient of the Lagrangian with respect to $\boldsymbol{\xi}$ is required to vanish:

$$\frac{\partial\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial\xi_i} \;=\; C - \alpha_i - \beta_i = 0, i = 1, ..., n. \tag{2.41}$$

Then besides the conditions described by equation (2.34), another set of conditions is obtained as follows:

$$C - \alpha_i - \beta_i = 0, i = 1, ..., n. \tag{2.42}$$

Substitute the conditions (2.34) and (2.42) into the Lagrangian (2.40), the following dual optimization problem can be formulated:

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j \\ &\text{s. t.} \sum_{i=1}^{n} \alpha_i y_i = 0,\ 0 \leqslant \alpha_i \leqslant C,\ \forall\ i. \end{aligned} \tag{2.43}$$

Problem (2.43) is slightly different from problem (2.35), in the way that multipliers $\alpha_i, i = 1, ..., n$ are upper bounded by C. Consequently, the points with positive slacks will also contribute to the determination of decision boundary. The formulation that is used to compute the optimal bias remains the same, i.e. (2.37), but points with Lagrange multipliers $\tilde{\alpha}_i \in (0, C)$ are chosen to compute the bias. This follows from the KKT complementary slackness (2.44) and (2.45).

$$\tilde{\alpha}_i[y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i) + \tilde{b} - 1 + \tilde{\xi}_i] = 0,\ \forall\ i, \tag{2.44}$$

$$\tilde{\beta}_i \tilde{\xi}_i = 0,\ \forall\ i. \tag{2.45}$$

Any pattern j for which $\tilde{\alpha}_j \neq 0$ and $\tilde{\xi}_j = 0$ can be chosen to determine the optimal bias $\tilde{b}$ by equation (2.44). According to Eq. (2.45), $\tilde{\xi}_i = 0$ if $\tilde{\beta}_i \neq 0$. From (2.42), we can see if $\beta_i > 0$ (meaning that $\xi_i = 0$), then $\alpha_i < C$. Thus, we can take any training point for which $0 < \alpha_j < C$ to compute $\tilde{b}$, again the average over all such points is taken. The decision function for a new input vector $\boldsymbol{x}_0$ remains the same as (2.38).

The two learning methods mentioned above can be generalized to the case where the decision function is non-linear by using the "kernel trick". Notice that in the training problems (2.35) and (2.43), the inputs only appears in the form of dot products $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$. Suppose we map the data into higher dimensional space $\mathcal{H}$ where a linear separating hyperplane can be found, using a mapping Φ:

$$\Phi : \boldsymbol{R}^d \mapsto \mathcal{H}. \tag{2.46}$$

Then the learning algorithm in space $\mathcal{H}$ would only depend on the data through dot products, $\Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{x}_j)$. If there were a "kernel function" $\mathcal{K}$ such that $\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{x}_j)$, there is no need to know the explicit form of mapping Φ. We could simply use $\mathcal{K}$ in the learning algorithm. By introducing the kernels, $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$ everywhere in the method designed for the non-separable case is replaced by $\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The algorithm now will produce a linear support vector machine in the $\mathcal{H}$ space only roughly taking the same amount time as it would take to train on the un-mapped data. However, not every function can be used as a kernel function. The kernels have to posses some special properties, e.g. satisfying Mercer's condition [13] or the corresponding Gram matrix needs to be positive semi-definite. One of the most popular examples of a kernel function is the Gaussian radial basis function (RBF) $\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/2\sigma^2}$, where σ is a bandwidth parameter.

One of the most popular approaches to training support vector machines is sequential minimal optimization (SMO) [71]. LIBSVM is one of the most widely used library for support vector machines [19]. Its implementation is based on sequential minimal optimization. LIBSVM supports support vector classification in terms of both two-class and multi-class classification problem. Multi-class classification is implemented by "one-against-one" strategy, where given k classes, $k(k-1)/2$ binary classifiers are constructed. The classification is determined by a voting scheme, where a point is classified to a class with majority votes.

2.4 Kernels for Time Series Data

Another main task of time series analysis is classification, which maps the input sequences into predefined classes. The major difference between classifying time series data from static data is that the time series data has high correlation and is often high dimensional and noisy. Therefore classical machine learning algorithms cannot be directly applied to time series classification.

Kernel methods for dealing with time series data have received considerable attention in the machine learning community [80]. A crucial aspect of applying kernel methods on time series data is to find a good kernel similarity to distinguish between time series. A simple way is to treat the time series as static vectors, ignoring the time dependence, and simply employ a linear kernel or Gaussian radial basis kernel [13, 80]. This method can be simple and efficient provided the time series are short and of equal length. However, in many real-world applications, the time series of interest are of variable-length and can be quite long. It is therefore desirable to construct kernels capable of handling (possibly long) time series of variable length, e.g. dynamic time warping [4], where time series similarity is quantified through finding an alignment between variable-length time series.

A new trend has emerged in the machine learning community, using models that could capture the temporal information in the time series data as representations and kernels are subsequently defined on the fitted models, for example, autoregressive kernels [26], probability product kernels [52], and KL divergence based kernels [17, 65].

This section will give a brief review on several existing kernels exclusively designed for time series data.

Dynamic Time Wrapping Based Kernels Dynamic time warping (DTW) measures the “similarity” between two sequences of variable length by “warping” the time axis of one (or both) sequences to achieve a better alignment [4]. An alignment π of length p ($|\pi| = p$) between two sequences $\boldsymbol{x} = (x_i, ..., x_n)$ and $\boldsymbol{y} = (y_i, ..., y_m)$ is a pair of increasing p-tuples (π_1, π_2) such that $1 = \pi_1(1) <= ... <= \pi_1(p) = n$ and $1 = \pi_2(1) <= ... <= \pi_2(p) = m$ with some requirements, e.g. unitary increments and no simultaneous repetitions [28]. Denoting the set of all possible alignments between $\boldsymbol{x}$ and $\boldsymbol{y}$ as $\mathcal{A}(\boldsymbol{x}, \boldsymbol{y})$, the optimal alignment π^* in terms of mean-score given as follows:

$$\pi^* = \arg \max_{\pi \in \mathcal{A}(\boldsymbol{x}, \boldsymbol{y})} \frac{1}{|\pi|} S(\pi) \tag{2.47}$$

can efficiently computed by dynamic programming algorithms, where $S(\pi)$ is the DTW score and defined as $S(\pi) = \sum_{i=1}^{|\pi|} \phi(x_{\pi_1(i)}, y_{\pi_2(i)})$, where ϕ is an arbitrary conditionally positive-definite kernel such as Gaussian kernel. DTW has been successfully used in many applications [4, 27, 28]. However, it can sometimes generate unintuitive alignments by mapping a single point of one time series onto a large subsection of another time series, leading to inferior results [54]. Since DTW similarity measure is not essentially positive definite it cannot be directly used in a kernel machine. A time series kernel based on global alignment, motivated by DTW, has been proposed in [28], with an efficient version presented in [27]. The kernel takes advantage of the score values of all possible alignments

$$K(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\pi \in \mathcal{A}(\boldsymbol{x}, \boldsymbol{y})} \exp\{S(\pi)\} = \sum_{\pi \in \mathcal{A}(\boldsymbol{x}, \boldsymbol{y})} \exp\left\{ \sum_{i=1}^{|\pi|} \phi(x_{\pi_1(i)}, y_{\pi_2(i)}) \right\} \tag{2.48}$$

instead of a single optimal alignment as presented in [93]. This kernel is proved to be positive-definite under some mild conditions on ϕ [28].

Autoregressive Kernels Autoregressive (AR) kernels [26] are probabilistic kernels for time series data. In an AR kernel, the vector autoregressive model class is used to generate an infinite family of features from the time series. Given a time series $\boldsymbol{s}$ of dimension

d and of length L, the time series is supposed to be generated according to the following vector autoregressive (VAR) model of order p:

$$\boldsymbol{s}(t) = \sum_{i=1}^{p} A_i \boldsymbol{s}(t-i) + \boldsymbol{\varepsilon}(t), t = p+1, ..., L. \tag{2.49}$$

where $A_i \in \mathbb{R}^{d\times d}$ are the coefficient matrices, $\boldsymbol{\varepsilon}$ is a centred Gaussian noise with a covariance matrix $V \in \mathbb{R}^{d\times d}$. Then, the likelihood function $p_\theta(\boldsymbol{s})$ can be formulated as follows:

$$p_\theta(\boldsymbol{s}) = \frac{1}{2\pi|V|^{\frac{d(L-p)}{2}}} \prod_{t=p+1}^{L} \exp\left(-\frac{1}{2}\left(\boldsymbol{s}(t) - \sum_{i=1}^{p} A_i \boldsymbol{s}(t-i)\right)^T V^{-1}\left(\boldsymbol{s}(t) - \sum_{i=1}^{p} A_i \boldsymbol{s}(t-i)\right)\right).$$

For a given time series $\boldsymbol{s}$, the likelihood function $p_\theta(\boldsymbol{s})$ across all possible parameter setting (under a matrix normal-inverse Wishart prior $\omega(\boldsymbol{\theta})$) forms a representation of $\boldsymbol{s}$. Given two time series $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$, the kernel is defined as the dot product of the corresponding sequence representations:

$$\mathcal{K}_{AR}(\boldsymbol{s}_i, \boldsymbol{s}_j) = \int_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{s}_i) \cdot p_{\boldsymbol{\theta}}(\boldsymbol{s}_j) \omega(d\boldsymbol{\theta}).$$

Fisher Kernel The Fisher kernel [47] was proposed to combine the power of generative modelling with discriminant classifiers such as the Support Vector Machines. It has been successfully used in numerous applications, see annotated bibliography in [87]. Fisher kernel assumes that the generative model $p(\boldsymbol{s}|\boldsymbol{\theta})$ can explain all the data. The Fisher kernel maps each individual data point into a vector in the gradient log-likelihood space specified by this generative model. The feature vector (Fisher score) $U_{\boldsymbol{s}}$ is the gradient of the log-likelihood of the generative model (fit on the data set) for the time series $\boldsymbol{s}$:

$$U_{\boldsymbol{s}} = \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{s}|\boldsymbol{\theta}).$$

The Fisher kernel is then defined as follows:

$$\mathcal{K}(\boldsymbol{s}_i, \boldsymbol{s}_j) = U_{\boldsymbol{s}_i}^T \mathcal{I}^{-1} U_{\boldsymbol{s}_j},$$

where $\mathcal{I}$ is the Fisher information matrix and defined as follows:

$$\mathcal{I}_{i,j} = -\mathbb{E}\left[\frac{\partial \log p(\boldsymbol{s}|\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j}\right]. \tag{2.51}$$

The calculation of $\mathcal{I}$ can be computationally expensive. A routinely used practical "trick" is to use the identity matrix in place of $\mathcal{I}$ [91], which speeds up the computation at the cost of losing some important information [91].

Auto-correlation Operators kernel Recently, Gaidon et. al proposed *difference between auto-correlation operators (DACO) kernel* [35] in the context of (video) action recognition. DACO kernel compares the dynamic aspects of two time series $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$ by using the difference between their auto-correlations and plug the difference into Gaussian function to compute the final kernel value as follows:

$$\mathcal{K}_{ADCO}(\boldsymbol{s}_i, \boldsymbol{s}_j) = \exp\{-\frac{d_{DACO}(\boldsymbol{s}_i, \boldsymbol{s}_j)}{2\sigma^2}\}, \tag{2.52}$$

where $d_{DACO}(\boldsymbol{s}_i, \boldsymbol{s}_j) = \parallel \hat{\rho}_\tau^{(\boldsymbol{s}_j)} - \hat{\rho}_\tau^{(\boldsymbol{s}_i)} \parallel_{HS}$, the distance between their auto-correlations (see section 2.2) using Hilbert-Schmidt norm, which is defined by $\parallel A \parallel_{HS}^2 = \mathrm{Tr}(A^* A)$ where Tr is the trace function and A^T is the transpose of A. The time lag τ needs to be determined according to the nature of the datasets. In [35], τ was set to be 1 frame, as they argue that they dealt with short actions recondition which involves fast motions and potentially drastic change in a few frames. DACO is well suited to compare the dynamic aspects in time series and can also be easily kernelized. In the kernelized version the time series are mapped into the feature space on a element-by-element basis [35]. Such kernelization makes the kernel more expressive but at the cost of computational complexity ($L \times L$ matrix inversion, where L is the length of the time series).

Kernels mentioned above are specially designed for the comparison of time series data. These kernels, except DTW based ones, share one common characteristic, which is taking advantage of time time series models to make use of the time dependency in the time series data. Autoregressive kernels [26] and Fisher kernels [47] use the generative time series models, DACO kernels utilize the auto-correlations. Fisher kernel [47] use one generative model to represent all the data, while autoregressive kernel uses the generative models to extract infinite many features of the data point. Fisher kernel only looks at the change of the log-likelihood function under the generative model. If the log-likelihood functions of the two time series change in the similar way, the two time series are regarded as similar. It does not consider whether the generative model can characterize the example well or not. The Auto-regressive kernel looks at the difference of the generative models related to the two time series under all possible parameters. If the sum of the difference between the generative models corresponding to the two time series through all the possible parameter settings is small, the two time series are regarded as similar.

2.5 Statistic Tests

A classical procedure of comparing the performance of two methods is t-test [109], which is a parametric test under assumptions of normality and equal variances. Kolmogorov-Smirnv test [95], which is a nonparametric test for the equality of continuous one-dimensional probability distribution can be used to test the normality of the data. When the results of each method reject the normality hypothesis, t-test will be invalid. In this case, Wilcoxon signed-rank test [105], which is a nonparametric test can be used. Wilcoxon signed-rank test is equivalent to the dependent t-test, but without the assumption of normality in the data. The procedure of Wilcoxon signed-rank test is as follows:

- Compute the difference between each pair of scores.
- Rank the differences in order of magnitude neglecting signs and then assign a negative sign to which correspond to negative difference.
- Calculate the sum of the positive rank numbers W^+ and the sum of the negative rank numbers W^-, respectively, and choose $W = \min(W^+, |W^-|)$.
- Use the table of critical values for the Wilcoxon signed-rank test to determine whether the difference is significant. For instance, given the size of non-zero differences is 10, the critical value for a two-tailed test at the 0.05 significance level is 8. If the computed W is smaller or equal to 8, then the difference is significant, otherwise not.

2.6 Summary

This chapter provided a review on kernel methods (i.e. SVMs) applied to time series data in two different contexts: prediction and classification. In the first context, the task is to forecast the future values from the past observations. The application of SVMs to this task involves a sliding window method. This method rolls a finite fixed length window over a given time series (inputs), the future values of the time series serve as targets. In the second context, the aim is to predict a label that applies to the entire time series. The crucial aspect is to define a good kernel similarity between the time series, since in this case the time series (example input) may not be of equal length and possibly very long. Consequently, classic kernels such as the Gaussian kernel might not be qualified.

A particular effective way of constructing time series kernel is to fit time models using the time series and define kernel values on top of the fitted models. Recurrent neural networks which will be used in our kernel construction were introduced. Kernels which are exclusively designed to measure the similarity between time series were reviewed.

CHAPTER 3

Exploiting Privileged Information for Time Series Prediction

In Chapter 2.3, we reviewed SVM for binary classification problems. In some time series prediction problems, people are interested in multiple ordered categorical information where an ordinal regression problem is formed. Moreover, future events, which are present in the training stage but not in the test phase, can act as privileged information. Therefore, we propose a new methodology, called support vector ordinal regression with implicit constraints using privileged information (SVORIM+), for utilizing the privileged information to improve generalization performance in ordinal regression. In this chapter, LUPI paradigm which incorporates privileged information in training process is briefly described in section 3.2. Section 3.3 demonstrates support vector ordinal regression approaches, in particular SVORIM. Section 3.4 describes the proposed SVORIM+ method in details. Section 3.5 is devoted to experiments and the main findings are discussed and summarized in section 3.6.

3.1 Introduction

Machine learning algorithms for classification problems map the input $\boldsymbol{x}$, a set of attribute values into a categorical target value c, which is represented by the class label [25, 32, 34, 100]. In many practical applications of machine learning, an order may exist on the class labels (categories). For instance, when learning how to grade future movements of a financial indicator, instead of predicting the real values, we may only be interested in ordered categories of movements – e.g. extreme up, up, down and extreme down. A variety of algorithms (referred to as ordinal regression) have been developed that explicitly use the class order information, e.g. [22, 24, 34, 45, 59, 88, 90]. A simple method that enables standard classification algorithms to exploit the ordinal prediction was proposed in [34]. This method first transforms an J-class ordinal problem into $J-1$ binary classification problems and then combines the $J-1$ model predictions to estimate the probabilities of the J original ordinal classes for test instances. This method requires no modification of the standard binary classification algorithm. However, it cannot capture the overall structure among the ordinal classes, which may be suboptimal. A direct generalization of support vector machine approach for ordinal regression has been proposed by finding $J-1$ parallel class separation hyperplanes such that the input/feature space is partitioned into J ranked regions corresponding to the classes [90]. Unfortunately, there is no guarantee that the thresholds corresponding to these hyperplanes are properly ordered at the "optimal" solution. Thus, this approach has been further extended in support vector ordinal regression (SVOR) with explicit and implicit constraints [24] imposing the ordinal inequalities on thresholds[1] (defining the separation hyperplanes), $b_1 \leqslant b_2 \leqslant ... \leqslant b_{J-1}$.

Traditionally, these ordinal regression algorithms are to learn the decision function from sample given only by labelled points $(\boldsymbol{x}_i, y_i)$. However, additional privileged knowl-

[1]Threshold and bias are quite similar concepts but slightly different in the way of appearing in the model, i.e. in the model $ax + b = 0$, b is called bias while in the model $ax - b = 0$, b is named threshold.

edge (represented through 'privileged space' X^*) may contain substantial information that might be used when constructing the decision function. Recently, an advanced learning paradigm, called *learning using privileged information* (LUPI) has been suggested to incorporate the privileged knowledge into learning process [101]. This new learning framework is motivated by human learning where a teacher provides students with not only examples but also additional information hidden in the explanations, comments, comparisons, etc. For example, when a student learn a concept at school such as algebra, the teacher can provide additional explanations at any time. Hopefully, this will make the student learn faster than if the teacher would only pose questions and give their answers. However, when later in life, the student faces an algebra problem, or or she will bot be able to rely on the teacher's expertise anymore. In the LUPI framework, along with training inputs $\boldsymbol{x}$, we may have access to an additional information $\boldsymbol{x}^*$ about training examples, but this privileged information $\boldsymbol{x}^*$ will not be available for inputs $\boldsymbol{x}$ at the test stage. For instance, in the financial indicator prediction example above, during training we have access to both the past and future contexts of time instances within the raining set. An SVM based framework, SVM+, for incorporating privileged information during training has been suggested in [101, 102]. The privileged information is used in SVM+ to model the slack variables of training inputs through so-called correcting functions. The SVM+ framework was further extended for the case structured data [60] or multi-tasking learning [14, 15].

The existing support vector ordinal regression approaches are not able to directly utilize privileged information during training. Motivated by SVM+ [101], which incorporates privileged information by modelling the slack variables of training inputs through so-called correcting functions, we propose to exploit privileged information in support vector ordinal regression with implicit constraints (SVORIM) by constructing slack variable models for each parallel separation hyperplane, which will be referred to as SVORIM+.

3.2 Learning Using Privileged Information

Commonly, in human learning, a teacher will provide students with not only examples but also additional information hidden in the explanations, comments, comparisons etc. [101]. Taking this element in human learning into consideration, an advanced learning paradigm, called *learning using privileged information* (LUPI) [101], also known as learning using hidden information [102], has been introduced to deal with this kind of situations where additional (privileged) information $\boldsymbol{x}^* \in X^*$ about training examples $\boldsymbol{x} \in X$ is known during training but is unavailable in the test phase. Privileged information appears in several application domains [101, 102], for example in the time series prediction privileged information is the behaviour of the time series in the future; in cancer prediction using biopsy images, the privileged information is the pathologist's report etc.

An extension of SVM learning algorithm, known as SVM+, has been suggested as a candidate for LUPI in [101, 102]. In SVM+, the slack variables for inputs in X are determined by a correcting function operating in the privileged space X^*:

$$\xi(\boldsymbol{x}^*) = \boldsymbol{w}^* \cdot \Phi^*(\boldsymbol{x}^*) + b^*, \tag{3.1}$$

where Φ^* is the feature map induced by the kernel operating on X^*, $\boldsymbol{w}^*$ is the normal vector to the correcting function, and b^* is the bias term of the correcting function. Replacing the slacks in (2.39) by the slack variable model defined above, the primal problem becomes:

$$\min_{\boldsymbol{w},\boldsymbol{w}^*,b,b^*} \frac{1}{2}[\| \boldsymbol{w} \|^2 + \rho(\| \boldsymbol{w}^* \|^2)] + C \sum_{i=1}^{n} [\boldsymbol{w}^* \cdot \Phi^*(\boldsymbol{x}_i^*) + b^*] \tag{3.2}$$

s. t.

$$y_i[\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_i) + b] \geqslant 1 - [\boldsymbol{w}^* \cdot \Phi^*(\boldsymbol{x}_i^*) + b^*],\ \forall\, i, \tag{3.3}$$

$$\boldsymbol{w}^* \cdot \Phi^*(\boldsymbol{x}_i^*) + b^* \geqslant 0,\ \forall\, i, \tag{3.4}$$

where ρ is a hyper-parameter used to control the capacity for the correcting function in X^* space. The second term in (3.2) relates to the capacity of the correcting function and the third term in (3.2) relates to minimization of the slack values. The justification of the problem has been given in [101]. Generally speaking, if the real slack values were known, the learning problem become a separable case which has a faster rate of convergence than the non-separable one. However, a teacher does not know the real slack values. Instead he can provide privileged information and the admissible set of correcting functions that contains the correcting function which defines the true slacks. The privileged information helps to find that correcting function.

By introducing two dual variables $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, where α_i is the i-th element of vector $\boldsymbol{\alpha}$ that relates to the i-th constraint of the first type (3.3) while β_i is the i-th element of vector $\boldsymbol{\beta}$ that relates to the i-th constraint of the second type (3.4), the Lagrangian can be written as follows:

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{w}^*, b, b^*, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \tfrac{1}{2}[\| \boldsymbol{w} \|^2 + \rho(\| \boldsymbol{w}^* \|^2)] + C \textstyle\sum_{i=1}^{n} [\boldsymbol{w}^* \cdot \Phi^*(\boldsymbol{x}_i^*) + b^*] \tag{3.5}$$
$$- \textstyle\sum_{i=1}^{n} \alpha_i \{ y_i[\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_i) + b] - 1 + [\boldsymbol{w}^* \cdot \Phi^*(\boldsymbol{x}_i^*) + b^*] \} - \sum_{i=1}^{n} \beta_i [\boldsymbol{w}^* \cdot \Phi^*(\boldsymbol{x}_i^*) + b^*],$$

with the corresponding dual:

$$\begin{aligned} & \max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ & -\frac{1}{2\rho} \sum_{i,j=1}^{n} (\alpha_i + \beta_i - C)(\alpha_j + \beta_j - C)\mathcal{K}^*(\boldsymbol{x}_i^*, \boldsymbol{x}_j^*) \\ \text{s. t.} & \\ & \sum_{i=1}^{n} (\alpha_i + \beta_i - C) = 0,\ \sum_{i=1}^{n} y_i \alpha_i = 0,\ \alpha_i \geqslant 0,\ \beta_i \geqslant 0,\ \forall\ i. \end{aligned} \tag{3.6}$$

$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $\mathcal{K}^*(\boldsymbol{x}_i^*, \boldsymbol{x}_j^*)$ are kernels in X and X^* spaces, respectively. SVM+ have been successfully used on a variety of data sets with privileged information, e.g. [60, 74].

3.3 Support Vector Ordinal Regression

SVM has been altered for ordinal regression setting where a natural order exists in multiple classes known as support vector ordinal regression (SVOR) [23, 24, 44, 90]. While the key concept of the SVM classifier is the hyperplane separating the positive examples from the negative ones with maximum margin, SVOR classifier extends this idea by constructing multiple parallel hyperplanes separating the adjacent classes (in the class order).

Shashua and Levin [90] maximize the margin between the adjacent classes and find $J-1$ thresholds that divide the real line into J consecutive intervals corresponding to the J ordered categories. The margin can either be the same for all $J-1$ separating hyperplanes, or can be different for each adjacent class separation, in which case the sum of the margins is maximized.

However, in this approach it cannot be guaranteed that the optimized thresholds will preserve the category order, i.e. $b_1 \leqslant b_2 \leqslant \ldots \leqslant b_j \ldots \leqslant b_{J-1}$.

To address this problem, Chu and Keerthi [23, 24] proposed explicit and implicit constraints enforcing the inequalities on the thresholds. The explicit constraints directly enforce order on the adjacent thresholds, while the implicit constraints ensure the threshold order implicitly through the data by stipulating that the j-the hyperplane (corresponding to threshold b_j) separates all points from classes $\leq j$ from all points of classes $> j$. Ling and Lin [59] introduced a unified reduction framework from ordinal regression to binary classification. The explicit and implicit (SVORIM) approaches of [24] can be regarded as special instances of this framework.

Consider an ordered set of classes $\{1, 2, ..., J\}$, following [24], in SVORIM, there are two sets of slack variables $\boldsymbol{\xi}$ and $\boldsymbol{\nu}$, where $\boldsymbol{\xi}$ collects the slack variables of inputs from lower categories to the current separating hyperplane, referred as 'left' slacks, while $\boldsymbol{\nu}$ collects the slack variables of inputs from upper categories to the current separating hyperplane, referred as 'right' slacks. Then, the primal problem is formulated as follows:

$$
\begin{aligned}
&\min_{\boldsymbol{w},\boldsymbol{b},\boldsymbol{\xi},\boldsymbol{\nu}} \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 + C \sum_{j=1}^{J-1} \left(\sum_{k=1}^{j} \sum_{i=1}^{n^k} \xi_{ki}^j + \sum_{k=j+1}^{J} \sum_{i=1}^{n^k} \nu_{ki}^j \right), \\
&s.t. \\
&\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_i^k) - b_j \leqslant -1 + \xi_{ki}^j,\ \xi_{ki}^j \geqslant 0, \\
&\text{for } j = 1, ..., J-1,\ k = 1, ..., j \text{ and } i = 1, ..., n^k, \\
&\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_i^k) - b_j \geqslant +1 - \nu_{ki}^j,\ \nu_{ki}^j \geqslant 0, \\
&\text{for } j = 1, ..., J-1,\ k = j+1, ..., J,\ i = 1, ..., n^k,
\end{aligned}
\tag{3.7}
$$

where $\boldsymbol{w}$ is the normal vector of the separating hyperplanes, $\boldsymbol{x}_i^k$ is the i-th example of k-th category, ξ_{ki}^j and ν_{ki}^j are 'left' and 'right' slacks, respectively, for the i-th point in class k with respect to the separating hyperplane between classes j and $j+1$ and $\Phi(\boldsymbol{x})$ is the image of the input $\boldsymbol{x}$ under the feature map defined by the used kernel $\mathcal{K}(\cdot, \cdot)$, n^k is the number of examples in k-th category.

Note that since in *SVORIM* there is a slack variable for each *(data point, decision boundary)* pair, there is no need to have different notations for the 'left' and 'right' slacks, $\boldsymbol{\xi}$ and $\boldsymbol{\nu}$, respectively. The left-right slacks are necessary for the explicit constraint for-

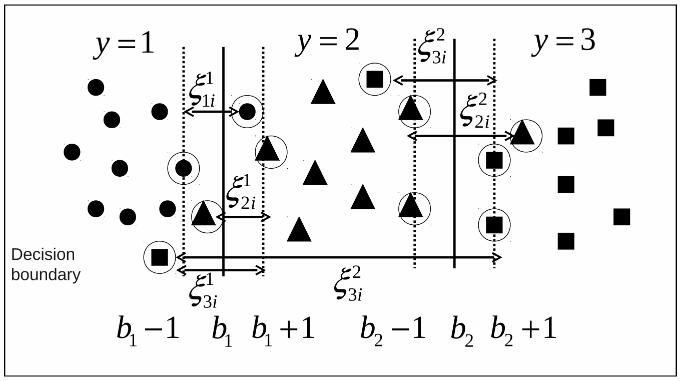


Figure 3.1: Illustration of SVORIM. All the examples are mapped to their function values $\boldsymbol{w} \cdot \Phi(\boldsymbol{x})$ along the horizontal axis [24].

mulation, but not for the implicit one. Similar to [24] but with only one set of slack variables, the definition of slack variable in *SVORIM* is given in Figure 3.1: for a threshold b_j, the function values $\boldsymbol{w} \cdot \Phi(\boldsymbol{x})$ of all samples from all the lower categories should be less than the lower margin $b_j - 1$ and the function values $\boldsymbol{w} \cdot \Phi(\boldsymbol{x})$ of all samples from all the upper categories should be greater than the upper margin $b_j + 1$. Slacks of each sample with respect to every threshold are allowed to relax the constraints. Therefore, in our approach, we need to model one set of slack variables $\boldsymbol{\xi}^j$ for each threshold b^j (j-the separating hyperplane).

3.4 SVORIM+ Approach

In this section, we apply the LUPI framework in the SVORIM approach. We chose the implicit SVORIM formulation instead of the explicit one (see section 3.3), since in the explicit SVOR each hyperplane $j = 1, 2, ..., J-1$, is constrained by the slacks of patterns $\boldsymbol{x}_i^j$ and $\boldsymbol{x}_i^{j+1}$ from adjacent classes only. On the other hand, in SVORIM patterns from all classes contribute to the slacks, thus constraining all separating hyperplanes. Since the key aspect of incorporating privileged information into SVOR is modelling of slacks via models operating on the privileged space, the SVORIM framework can provide more flexibility in using the privileged information through greater number of correcting functions.

Additionally, it was empirically found in [24] that the explicit constraint approach performed better in terms of the basic zero-one loss (classification error), while the SVORIM was preferable in terms of Mean Absolute Error (MAE), which is more important in the context of ordinal regression. Following notation in [101], we will refer to the SVORIM algorithm with privileged information as SVORIM+.

3.4.1 Primal Problem

Suppose, as in section 3.3, that we have observations classified into J ordered categories and there are n^k examples $\boldsymbol{x}_i^k \in X$, $i = 1, ..., n^k$, in the k-th category, $k = 1, 2, ..., J$. To simplify the presentation we assume that each example $\boldsymbol{x}_i^k$ has an associated privileged information $\boldsymbol{x}_i^{*k} \in X^*$. Extension to the case where only a subset of training examples has privileged information is straightforward. For points without privileged information the slack values would be obtained from the optimization programme as in SVORIM.

As there is no need to have a different notation for the "left" and "right" slacks, $\boldsymbol{\xi}$ and $\boldsymbol{\nu}$, respectively, we rewrite (3.7) as

$$\begin{aligned}
&\min_{\boldsymbol{w},\boldsymbol{b},\boldsymbol{\xi}} \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 + C \sum_{j=1}^{J-1} \sum_{k=1}^{J} \sum_{i=1}^{n^k} \xi_{ki}^j, \\
&s.t. \text{ for } j = 1, ..., J-1, \\
&\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_i^k) - b_j \leqslant -1 + \xi_{ki}^j,\ \xi_{ki}^j \geqslant 0, \quad k = 1, ..., j \text{ and } i = 1, ..., n^k, \\
&\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_i^k) - b_j \geqslant +1 - \xi_{ki}^j,\ \xi_{ki}^j \geqslant 0, \quad k = j+1, ..., J,\ i = 1, ..., n^k.
\end{aligned} \tag{3.8}$$

Therefore, in our approach, we only have one set of slack variables $\boldsymbol{\xi}^j$ for each threshold b_j (j-th separating hyperplane). Following [101], for $\boldsymbol{x}_i^k \in X$ with privileged information $\boldsymbol{x}_i^{*k} \in X^*$, the slack values are obtained through models ("correcting functions") of the form $\xi_{ki}^j = \boldsymbol{w}_j^* \Phi^*(\boldsymbol{x}_i^{*k}) + b_j^*$, operating on the privileged space X^*. In total $J-1$ correcting functions $\xi^j(\boldsymbol{x}^*) = \boldsymbol{w}_j^* \Phi^*(\boldsymbol{x}^*) + b_j^*$ are needed, $j = 1, 2, ..., J-1$, one for each decision boundary.

In analogy to [101], the primal problem of proposed SVORIM+ can be formulated as:

$$
\begin{aligned}
&\min_{\boldsymbol{w},b,\boldsymbol{w}^*,\boldsymbol{b}^*} \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 + \frac{\rho}{2} \sum_{j=1}^{J-1} (\parallel \boldsymbol{w}_j^* \parallel^2) + C \sum_{j=1}^{J-1} \sum_{k=1}^{J} \sum_{i=1}^{n^k} (\boldsymbol{w}_j^* \cdot \Phi^*(\boldsymbol{x}_i^{*k}) - b_j^*),\\
&s.t. \text{ for every } j = 1, ..., J-1,\\
&\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_i^k) - b_j \leqslant -1 + (\boldsymbol{w}_j^* \cdot \Phi^*(\boldsymbol{x}_i^{*k}) - b_j^*), \quad \text{for } k = 1, ..., j \text{ and } i = 1, ..., n^k,\\
&\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_i^k) - b_j \geqslant +1 - (\boldsymbol{w}_j^* \cdot \Phi^*(\boldsymbol{x}_i^{*k}) - b_j^*), \quad \text{for } k = j+1, ..., J \text{ and } i = 1, ..., n^k,\\
&\boldsymbol{w}_j^* \cdot \Phi^*(\boldsymbol{x}_i^{*k}) + b_j^* \geqslant 0 \quad \text{for } k = 1, ..., J \text{ and } i = 1, ..., n^k.
\end{aligned}
\tag{3.9}
$$

Note that unlike in SVORIM, in the proposed formulation of SVORIM+ the slack variables are reduced to one set per threshold and are replaced by correcting functions defined in the privileged information space. The term $\sum_{j=1}^{J-1}(\parallel \boldsymbol{w}_j^* \parallel^2)$ corresponds the capacity of the correcting functions and is controlled by the parameter $\rho \geq 0$, tuned via using cross-validation.

3.4.2 Dual Problem

Following the standard SVM practice, the primal problem will be transformed into its (more manageable) dual formulation [6, 8, 100]. Lagrangian for the primal problem can be formulated as (3.10):

$$
\begin{aligned}
\mathcal{L} \;=\;& \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{\rho}{2}\sum_{j=1}^{J-1}\|\boldsymbol{w}_j^*\|^2 + C\sum_{j=1}^{J-1}\sum_{k=1}^{J}\sum_{i=1}^{n^k}(\boldsymbol{w}_j^*\cdot\Phi^*(\boldsymbol{x}_i^{*k}) + b_j^*) \\
-\;& \sum_{j=1}^{J-1}\sum_{k=1}^{j}\sum_{i=1}^{n^k}\alpha_{ki}^j\left(-1 + \boldsymbol{w}_j^*\cdot\Phi^*(\boldsymbol{x}_i^{*k}) + b_j^* - \boldsymbol{w}\cdot\Phi(\boldsymbol{x}_i^k) + b_j\right) \qquad (3.10)\\
-\;& \sum_{j=1}^{J-1}\sum_{k=j+1}^{J}\sum_{i=1}^{n^k}\alpha_{ki}^j\left(-1 + \boldsymbol{w}_j^*\cdot\Phi^*(\boldsymbol{x}_i^{*k}) + b_j^* + \boldsymbol{w}\cdot\Phi(\boldsymbol{x}_i^k) - b_j\right) \\
-\;& \sum_{j=1}^{J-1}\sum_{k=1}^{J}\sum_{i=1}^{n^k}\beta_{ki}^j(\boldsymbol{w}_j^*\cdot\Phi^*(\boldsymbol{x}_i^{*k}) + b_j^*),
\end{aligned}
$$

where α_{ki}^j and β_{ki}^j are non-negative Larangian multipliers. The KKT conditions for the primal problem require the following conditions hold true:

$$
\begin{aligned}
\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}} \;=\;& \boldsymbol{w} + \sum_{j=1}^{J-1}\sum_{k=1}^{j}\sum_{i=1}^{n^k}\alpha_{ki}^j\Phi(\boldsymbol{x}_i^k) - \sum_{j=1}^{J-1}\sum_{k=j+1}^{J}\sum_{i=1}^{n^k}\alpha_{ki}^j\Phi(\boldsymbol{x}_i^k) = 0, \qquad (3.11)\\
\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}_j^*} \;=\;& \rho\boldsymbol{w}_j^* + C\sum_{k=1}^{J}\sum_{i=1}^{n^k}\Phi^*(\boldsymbol{x}_i^{*k}) - \sum_{k=1}^{j}\sum_{i=1}^{n^k}\alpha_{ki}^j\Phi^*(\boldsymbol{x}_i^{*k}) \qquad (3.12)\\
&- \sum_{k=j+1}^{J}\sum_{i=1}^{n^k}\alpha_{ki}^j\Phi^*(\boldsymbol{x}_i^{*k}) - \sum_{k=1}^{J}\sum_{i=1}^{n^k}\beta_{ki}^j\Phi^*(\boldsymbol{x}_i^{*k}) = 0,\\
\frac{\partial\mathcal{L}}{\partial b_j} \;=\;& -\sum_{k=1}^{j}\sum_{i=1}^{n^k}\alpha_{ki}^j + \sum_{k=j+1}^{J}\sum_{i=1}^{n^k}\alpha_{ki}^j = 0, \forall j, \qquad (3.13)\\
\frac{\partial\mathcal{L}}{\partial b_j^*} \;=\;& \sum_{k=1}^{J}\sum_{i=1}^{n^k}(\alpha_{ki}^j + \beta_{ki}^j - C) = 0, \forall j, \qquad (3.14)\\
\alpha_{ki}^j \;\geqslant\;& 0,\ \beta_{ki}^j \geqslant 0, \forall i, \forall j. \qquad (3.15)
\end{aligned}
$$

According to condition (3.11), we can have the solution of $\boldsymbol{w}$ as follows:

$$
\begin{aligned}
\boldsymbol{w} \;=\;& -\sum_{j=1}^{J-1}\sum_{k=1}^{j}\sum_{i=1}^{n^k}\alpha_{ki}^j\Phi(\boldsymbol{x}_i^k) + \sum_{j=1}^{J-1}\sum_{k=j+1}^{J}\sum_{i=1}^{n^k}\alpha_{ki}^j\Phi(\boldsymbol{x}_i^k)\\
=\;& \sum_{k,i}\left(\sum_{j=1}^{k-1}\alpha_{ki}^j - \sum_{j=k}^{J-1}\alpha_{ki}^j\right)\Phi(\boldsymbol{x}_i^k). \qquad (3.16)
\end{aligned}
$$

According to condition (3.12), we can have the solution of $\boldsymbol{w}_j^*$ as follows:

$$\begin{aligned}
\boldsymbol{w}_j^* &= \frac{1}{\rho}\Bigg(-C\sum_{k=1}^{J}\sum_{i=1}^{n^k}\Phi^*(\boldsymbol{x}_i^{*k}) + \sum_{k=1}^{j}\sum_{i=1}^{n^k}\alpha_{ki}^j\Phi^*(\boldsymbol{x}_i^{*k}) \\
&\quad + \sum_{k=j+1}^{J}\sum_{i=1}^{n^k}\alpha_{ki}^j\Phi^*(\boldsymbol{x}_i^{*k}) + \sum_{k=1}^{J}\sum_{i=1}^{n^k}\beta_{ki}^j\Phi^*(\boldsymbol{x}_i^{*k})\Bigg) \\
&= \frac{1}{\rho}\sum_{k=1}^{J}\sum_{i=1}^{n^k}(\alpha_{ki}^j + \beta_{ki}^j - C)\Phi^*(\boldsymbol{x}_i^{*k}). \qquad (3.17)
\end{aligned}$$

By substituting equations (3.16) and (3.17), together with conditions (3.13) and (3.14) into (3.10), we obtain the dual:

$$\begin{aligned}
\mathcal{L}_D &= -\frac{1}{2}\sum_{k,i}\sum_{k',i'}\left(\sum_{j=1}^{k-1}\alpha_{ki}^j - \sum_{j=k}^{J-1}\alpha_{ki}^j\right)\left(\sum_{j=1}^{k'-1}\alpha_{k'i'}^j - \sum_{j=k'}^{J-1}\alpha_{k'i'}^j\right)\mathcal{K}(\boldsymbol{x}_i^k, \boldsymbol{x}_{i'}^{k'}) \\
&\quad -\frac{\rho}{2}\sum_{j=1}^{J-1}\sum_{k,i}\sum_{k',i'}\left(\alpha_{ki}^j + \beta_{ki}^j - C\right)\left(\alpha_{k'i'}^j + \beta_{k'i'}^j - C\right)\mathcal{K}(\boldsymbol{x}_i^{*k}, \boldsymbol{x}_{i'}^{*k'}) \\
&\quad +C\sum_{j=1}^{J-1}\sum_{k=1}^{J}\sum_{i=1}^{n^k}b_j^* + \sum_{j=1}^{J-1}\sum_{k=1}^{j}\sum_{i=1}^{n^k}\alpha_{ki}^j - \sum_{j=1}^{J-1}\sum_{k=1}^{j}\sum_{i=1}^{n^k}\alpha_{ki}^j b_j^* - \sum_{j=1}^{J-1}\sum_{k=1}^{j}\sum_{i=1}^{n^k}\alpha_{ki}^j b_j \\
&\quad +\sum_{j=1}^{J-1}\sum_{k=j+1}^{J}\sum_{i=1}^{n^k}\alpha_{ki}^j - \sum_{j=1}^{J-1}\sum_{k=j+1}^{J}\sum_{i=1}^{n^k}\alpha_{ki}^j b_j^* + \sum_{j=1}^{J-1}\sum_{k=j+1}^{J}\sum_{i=1}^{n^k}\alpha_{ki}^j b_j - \sum_{j=1}^{J-1}\sum_{k=1}^{J}\sum_{i=1}^{n^k}\beta_{ki}^j b_j^*) \\
&= \sum_{k,i,j}\alpha_{ki}^j - \frac{1}{2}\sum_{k,i}\sum_{k',i'}(\sum_{j=1}^{k-1}\alpha_{ki}^j - \sum_{j=k}^{r-1}\alpha_{ki}^j)(\sum_{j=1}^{k'-1}\alpha_{k'i'}^j - \sum_{j=k'}^{r-1}\alpha_{k'i'}^j)\mathcal{K}(\boldsymbol{x}_i^k, \boldsymbol{x}_{i'}^{k'}) \\
&\quad -\frac{1}{2\rho}\sum_{j=1}^{r-1}\sum_{k,i}\sum_{k',i}(\alpha_{ki}^j + \beta_{ki}^j - C)(\alpha_{k'i'}^j + \beta_{k'i'}^j - C)\mathcal{K}(\boldsymbol{x}_i^{*k}, \boldsymbol{x}_{i'}^{*k'})
\end{aligned}$$

Thus, the learning problem of the SVORIM+ becomes the following optimization problem:

$$
\begin{aligned}
&\max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \sum_{k,i,j} \alpha_{ki}^{j} - \frac{1}{2}\sum_{k,i}\sum_{k',i'}(\sum_{j=1}^{k-1}\alpha_{ki}^{j} - \sum_{j=k}^{r-1}\alpha_{ki}^{j})(\sum_{j=1}^{k'-1}\alpha_{k'i'}^{j} - \sum_{j=k'}^{r-1}\alpha_{k'i'}^{j})\mathcal{K}(\boldsymbol{x}_i^k, \boldsymbol{x}_{i'}^{k'}) \\
- \quad &\frac{1}{2\rho}\sum_{j=1}^{r-1}\sum_{k,i}\sum_{k',i}(\alpha_{ki}^{j} + \beta_{ki}^{j} - C)(\alpha_{k'i'}^{j} + \beta_{k'i'}^{j} - C)\mathcal{K}(\boldsymbol{x}_i^{*k}, \boldsymbol{x}_{i'}^{*k'}) \qquad (3.18)\\
&s.t. \\
&\sum_{k=1}^{j}\sum_{i=1}^{n^k}\alpha_{ki}^{j} = \sum_{k=j+1}^{r}\sum_{i=1}^{n^k}\alpha_{ki}^{j}, \forall j \\
&\sum_{k=1}^{r}\sum_{i=1}^{n^k}(\alpha_{ki}^{j} + \beta_{ki}^{j} - C) = 0, \forall j \\
&\alpha_{ki}^{j} \geqslant 0, \beta_{ki}^{j} \geqslant 0, \forall i, \forall j
\end{aligned}
$$

where j runs over $1, ..., J-1$. This optimization problem can be solved by quadratic programming. Once the optimal solution of the above optimization problem is found, the value of discriminant function at (a new) input $\boldsymbol{x}$ is:

$$
F(\boldsymbol{x}) = \sum_{k,i}(\sum_{j=1}^{k-1}\alpha_{ki}^{j} - \sum_{j=k}^{r-1}\alpha_{ki}^{j})\mathcal{K}(\boldsymbol{x}_i^k, \boldsymbol{x}) \qquad (3.19)
$$

The correcting functions for each threshold have the form,

$$
\xi^{j}(\boldsymbol{x}^*) = f_j(\boldsymbol{x}^*) + b_j^*, \qquad (3.20)
$$

where $f_j(\boldsymbol{x}^*) = \frac{1}{\rho}\sum_{k=1}^{J}\sum_{i=1}^{n^k}(\alpha_{ki}^{j} + \beta_{ki}^{j} - C)\mathcal{K}^*(\boldsymbol{x}_i^{*k}, \boldsymbol{x}^*)$, and the bias b_j^* is computed by averaging over $-f_j(\boldsymbol{x}_i^{*k})$ for all the points which $\beta_{ki}^{j} > 0$, $j = 1, ..., J-1$. The threshold b_j can be computed by any $\alpha_{ki}^{j} > 0$, in the following way:

$$
b_j = \begin{cases} F(\boldsymbol{x}_i^k) + 1 - \xi^{j}(\boldsymbol{x}_i^{*k}) & k \leqslant j, \\ F(\boldsymbol{x}_i^k) - 1 + \xi^{j}(\boldsymbol{x}_i^{*k}) & k > j. \end{cases} \qquad (3.21)
$$

The threshold is taken the average for these points. Then, the predictive ordinal decision function is defined as:

$$\arg\min_i F(\boldsymbol{x}) < b_i. \tag{3.22}$$

3.5 Experimental Studies

We tested our methodology on 7 data sets of different nature and origin. The input vectors were normalized to zero mean and unit variance. RBF kernels were used in both X and X^* spaces with kernel widths σ and σ^*, respectively. In all experiments of this chapter,the parameter ranges were as follows: $\log_{10} C \in \{-2, -1, 0, 1, 2\}$, $\log_{10} \sigma \in \{-2, -1, 0, 1, 2\}$, $\log_{10} \sigma^* \in \{-2, -1, 0, 1, 2\}$ and $\log_{10} \rho \in \{-2, -1, 0, 1, 2\}$. Hyper-parameters were tuned via grid search based on 5-fold cross validation over the training set. The **cvx** matlab optimization routine[1] was used an optimization routine.

3.5.1 Evaluation Metrics

In the experiments we used three performance measures:

Mean Zero-one Error (MZE) (misclassification rate) fraction of incorrect predictions,

$$MZE = \frac{\sum_i^n I(y_i \neq \hat{y}_i)}{n}, \tag{3.23}$$

where n is the number of the test examples. $I(*)$ is the indicator function whose function value is 1, if the argument $*$ is true, 0 otherwise. y_i is the true label of the i-th test example and $\hat{y}_i$ is the corresponding predicted label, $y_i, \hat{y}_i \in \{1, 2, \ldots, r-1\}$.

Mean Absolute Error (MAE) average absolute deviation of the prediction from the true target,

$$MAE = \frac{\sum_i^n |y_i - \hat{y}_i|}{n} \tag{3.24}$$

[1]http://cvxr.com/cvx

Macroaveraged Mean Absolute Error (MMAE) specially designed for evaluating ordinal regression problems with imbalanced classes,

$$MMAE = \frac{1}{J}\sum_{j=1}^{J}\frac{\sum_{y_i=j}|y_i - \hat{y}_i|}{n^j} \tag{3.25}$$

where J is the number of classes and n^j is the number of examples of class j.

3.5.2 Benchmark datasets

We employed two benchmark ordinal datasets, *Pyrimidines* and *MachineCPU* used in [24]. Following [24], the continuous targets were discretized to 5 ordinal categories (equal-frequency binning). Each data set was randomly partitioned into training/test splits 10 times independently, yielding 10 re-sampled training/test sets of size 50/24 and 150/59 for *Pyrimidines* and *MachineCPU*, respectively. In order to demonstrate the advantage of the proposed method for incorporating the privileged information, an initial experiment is conducted which categorizes the input dimensions into 'original' and 'privileged' features in spaces X and X^*, respectively. Feature categorization is driven by a 'wrapper' approach. For each data set, we sort the input features in terms of their relevance for the ordinal classifier (in our case SVORIM). The first most relevant half of the features will form privileged information, the remaining half will constitute the original space X. Privileged features will only be incorporated in training of SVORIM+ and will be absent during the ordinal regression testing.

The average results over 10 randomized data sets splits (trials), along with standard deviations are shown in Table 3.1. Exploiting the privileged information slightly decreases the classification error MZE (by roughly 1%), decreases MAE (roughly by 7%) and decreases $MMAE$ (with about 13% of improvement for *Pyrimidines* and 7% for *MachineCPU*). To assess the significance of the performance differences, a statistical analysis was performed. The Kolmogorov-Smirnov (K-S) test, which is a nonparametric test for the equality of continuous one-dimensional probability distributions, was used to check if the hypothesis of normality could be assumed for the results. Normality hypothesis was rejected for all measures, so the non-parametric Wilcoxon signed-rank test [105] was applied to check whether the differences were statistically significant. The corresponding p-value is also included in Table 3.1, revealing that differences in MAE and MMAE were significant. We stress that both models are using the same set of features during the test phase, the privileged information being used only during training.

Table 3.1: Test results from 10 trials of the two algorithms on Pyrimidines and Machine CPU and p-values of the Wilcoxon signed rank test comparing SVORIM and SVORIM+

Dataset	Criteria	SVORIM	SVORIM+	$p-$value
Pyrimidines	MZE	0.5834±0.0651	0.5750±0.0756	0.5000
	MAE	0.7875±0.1249	0.7250±0.1306	0.0156$_\bullet$
	MMAE	0.9627±0.1609	0.8373±0.1421	0.0039$_\bullet$
MachineCPU	MZE	0.4390±0.0611	0.4356±0.0480	0.7656
	MAE	0.5220±0.0984	0.4848±0.0599	0.0547$_\circ$
	MMAE	0.5197±0.0991	0.4808±0.0601	0.0488$_\bullet$

$\circ$: Statistically significant differences with a level of significance of $\alpha = 0.15$.
$\bullet$: Statistically significant differences with a level of significance of $\alpha = 0.05$.

3.5.3 Time series datasets

In this section we employ time series data sets (see Table 3.2), as during the training information about the immediate future (known in the training phase) can be used as privileged information. The time series were quantized into a series of categories with natural order.

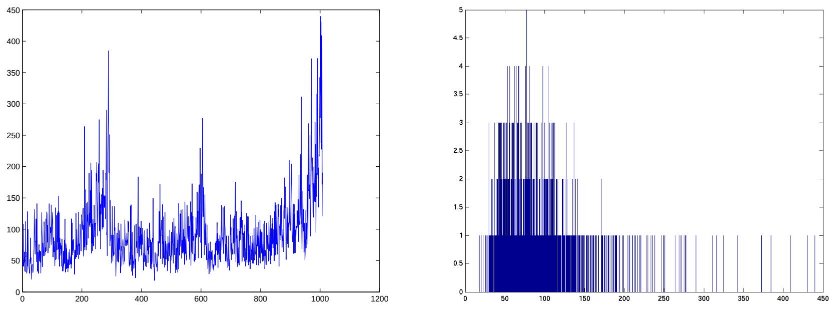
FTSE100 is a series of the spreads $\{s(t)\}$ (shown as Figure 3.2a) obtained from the daily highest $(H(t))$ and lowest $(L(t))$ prices, $s(t) = H(t) - L(t)$ in four-year historical daily prices between 1 October 2008 and 31 September 2012 on the FTSE 100 downloaded from Yahoo Finance. The distribution of *FTSE100* series is shown in Figure 3.2b, similar

Table 3.2: Description of the real datasets. d_1 is the number of previous values which function as basic information, and d_2 is the number of future values which function as privileged information.

Dataset	training/test	d_1	d_2	# trails
FTSE100	1 year/1 month	5	5	36
Fish	9 folds/1 fold	5	5	10
wine	9 folds/1 fold	5	5	10
SOI	300/200	5	5	7
Laser	500×10/4874	10	10	1
Birth	9 folds/1 fold	5	5	10

to [98], we quantized the real-valued series $\{s(t)\}$ into a symbolic stream $\{y(t)\}$ through

$$y(t) = \begin{cases} 1 \text{ (extremely small change)} & \text{if } s(t) < \theta_1 \\ 2 \text{ (small change)} & \text{if } \theta_1 \leqslant s(t) < \theta_2 \\ 3 \text{ (large change)} & \text{if } \theta_2 \leqslant s(t) < \theta_3 \\ 4 \text{ (extremely large change)} & \text{if } \theta_3 \leqslant s(t) \end{cases}$$



(a) The series of daily spreads values of FTSE100

(b) The Distribution of FTSE100 daily spread value series

Figure 3.2: The time series of FTSE100 daily spread values.

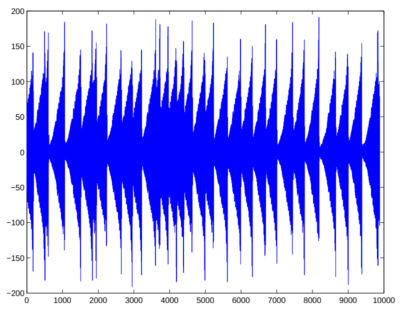
Since the extreme cases in reality are much rarer than mild changes, the cut values $\theta_1, \theta_2, \theta_3$ were chosen so that classes 1, 2, 3 and 4 contain 10%, 40%, 40% and 10% of sequence elements $s(t)$.

The *Fish* data contains 453 monthly values of estimated fish recruitment in the period 1950-1987; *Wine* data set contains Australian red wine sales in the period of 1980-1991.

SOI is monthly values of the Southern Oscillation Index (SOI), which indicates the sea surface temperature of the Pacific, in the period between January 1999 and October 2012. The *Laser* data, shown as Figure 3.3a with distribution shown as 3.3b, represents a cross-cut through periodic to chaotic intensity pulses of a real laser in a chaotic regime. The s*Birth* data set contains births per 10,000 of 23 year old women in U.S. in the period of 1917-1975. For each of the four time series, differenced time series $\{r(t)\}$ was quantized into a symbolic stream $\{y(t)\}$ through

$$y(t) = \begin{cases} 1 \text{ (extremely down)} & \text{if } r(t) < \theta_1 < 0 \\ 2 \text{ (normal down)} & \text{if } \theta_1 \leqslant r(t) < 0 \\ 3 \text{ (normal up)} & \text{if } 0 \leqslant r(t) < \theta_2 \\ 4 \text{ (extremely up)} & \text{if } \theta_2 \leqslant r(t) \end{cases}$$

The cut values θ_1 and θ_2 were chosen in the same manner as in the *FTSE100* experiment.


(a) Plot of Laser series

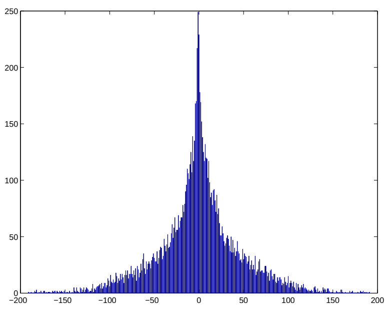

(b) The distribution of laser series

Figure 3.3: The time series of Laser.

For *FTSE100* and *SOI*, the training and test set sizes are described in Table 3.2. We used the rolling window methodology. The models are trained on the first continuous segment of training set size (e.g. 1 year for *FTSE100*) and then tested on the data window of test set size(e.g. 1 month for *FTSE100*) directly following the training set. The training window is then moved forward by the test set size. After training, the models are tested on the new test set directly following the training set, etc.

For smaller data sets, *Fish*, *Wine* and *Birth*, we used 10-fold cross-validation. For rather long Laser dataset, we trained an ensemble model consisting of 10 models independently trained on 10 non-overlapping folds of the training set (500 points in each fold). Ensemble approach was used for the reason that this dataset is too large, while the implementation of the proposed approach by quadratic programming can not process large datasets. Each of the 10 models was validated on the remaining folds. After training, the ensemble model predicted class labels on the hold-out test set of size 4874 using majority voting (among the 10 ensemble members, each having equal voting weight). In the case of a tie (e.g. 5 votes for class 1, 5 votes for class 2), we added validation errors of ensemble members behind each vote and predicted the class voted by ensemble members with smaller validation error.

The results shown given in Table 4.3, including the results of the Wilcoxon test (the KS-test rejected the null hypothesis of normality). Exploiting the privileged information decreases the classification error approximately by 3%, MAE approximately by 5% and $MMAE$ by about 2%. MAE of SVORIM and SVORIM+ on the *Fish* recruitment data are almost the same. This may be because the privileged information is not informative enough, so that the SVORIM+ solution reduces to the SVORIM one [102]. The improvement on *laser* data is significant, both the MZE and MAE decreased by up to about 17% and $MMAE$ decreased by up to about 46%, the differences being found statistically significant.

3.6 Discussion and Conclusion

How to utilize all available information during training to improve generalization performance of a learning algorithm is one of the main research questions in machine learning. This chapter presents a new methodology called SVORIM+, for utilizing privileged information of the training examples, unavailable in the test regime, to improve generalization

Table 3.3: Test Results of the two algorithms on real datasets

Datasets	Criteria	SVORIM	SVORIM+	$p-$value
FSTE 100	MZE	0.5884± 0.1470	0.5640±0.1311	0.0263$_\bullet$
	MAE	0.6799±0.2672	0.6425±0.2291	0.1493$_\circ$
	MMAE	0.7528±0.2409	0.7294±0.2610	0.0745$_\circ$
Wine	MZE	0.6428±0.1348	0.6040±0.1155	0.1250$_\circ$
	MAE	0.7804±0.1336	0.7194±0.1557	0.0156$_\bullet$
	MMAE	0.9871±0.1061	0.9258±0.1368	0.0625$_\circ$
Fish	MZE	0.5672±0.0273	0.5683±0.0474	0.6250
	MAE	0.6316±0.0356	0.6260±0.0672	0.6250
	MMAE	0.8318 ±0.0297	0.8294±0.0755	0.8125
MonthlySOI	MZE	0.5724±0.0497	0.5502±0.0344	0.2187
	MAE	0.6354±0.0649	0.6017±0.0470	0.2187
	MMAE	0.8549±0.0919	0.8380±0.0548	0.2187
Laser	MZE	0.0554	0.0460	0.0098$_\bullet$
	MAE	0.0558	0.0460	0.0040$_\bullet$
	MMAE	0.0852	0.0454	0.0019$_\bullet$
Birth	MZE	0.6333± 0.1076	0.6017± 0.1531	0.5000
	MAE	0.7778± 0.1834	0.6972± 0.1708	0.1406$_\circ$
	MMAE	1.0083± 0.2362	0.8696± 0.2001	0.0156$_\bullet$

$\circ$: Statistically significant differences with a level of significance of $\alpha = 0.15$.
$\bullet$: Statistically significant differences with a level of significance of $\alpha = 0.05$.

performance in ordinal regression.

Support vector ordinal regression algorithms have been developed to exploit the order structure in J class labels by constructing $J-1$ parallel discriminant hyperplanes (in the feature space), separating the J ranks/classes. The proposed approach incorporates the privileged information into support vector ordinal regression by constructing correcting functions for each hyperplane. This approach extends to ordinal regression the original SVM+ methodology. We formulate the proposed algorithm as a constrained convex optimization problem, which guarantees the global maximization in the optimization process. The experimental results on several benchmark and time series datasets confirmed that the generalization performance of SVORIM+ can indeed be superior to that of SVORIM. This is so even though the test inputs for both approaches are exactly the same - the only difference is that during the training SVORIM+ is able to model the slack variable values through correcting functions operating in the privileged space. The improvement of the generalization performance is due to utilization of useful additional information in the form of future evolution of time series (privileged information). As argued in [101] effectively using such information on time series turns the extrapolation problem into interpolation one which is easier and more stable to learn. However, compared with SVORIM, SVORIM+ requires longer training time because SVORIM only needs to solve an optimization problem of size $(J-1)n$, while the problem size of SVORIM+ is $2(J-1)n$, which is double of that SVORIM requires. Besides, there are more hyper-parameters to tune in SVORIM+. Making SVORIM+ training more efficient is a matter for our future work.

Our work opens the door to application domains where ordinal regression with privileged information can be readily employed. For example, in automated trading where only ordered categorical information is often needed [84,97,98], the privileged information during training can be utilized (as suggested in [101]) in the form of the known future development of the time series.

Learning using privileged information can be viewed as every special case of learning with missing data where the same kind of information is consistently missing in test phase, in contrast to, e.g. missing data at random [46].

CHAPTER 4

Learning in the Model Space (LiMS) Kernel For Time Series Classification

In chapter 3, we simply applied Gaussian kernel in our proposed method, since the example inputs obtained from the time series are of equal length and very short. However, in the context of time series classification, the example time series inputs may be of variable length and possibly very long. Therefore, kernels specially designed for time series are required. In this chapter, we introduce novel efficient model based kernels for time series data rooted in the reservoir computation framework. The kernels are implemented by fitting reservoir models sharing the same fixed deterministically constructed state transition part to individual time series. Learning is then perform in the reservoir model space. Reservoir computing models constructed in a simple deterministically manner is briefly described in section 2.2.3. Section 4.2 proposes time series kernels based on reservoir models. The experimental results and analysis are reported in Section 4.3. Section 4.4 studies on-line reservoir kernel construction for extremely long time series (more than one million). Finally, Section 4.5 discusses and concludes the results and main findings.

4.1 Introduction

Kernel methods have received considerable attention in the machine learning community dealing with structured data, such as image, graphs, texts or voice signals. However, as an important ubiquitous data type in science and engineering, time series have received relatively less research in the kernel literature [26].

There has been active research on quantification of the "similarity" or the "distance" between time series. However, these measures are not always applicable for kernel approaches as many of such similarity measures are not positive definite, which is a necessary basis for the reproducing kernel Hilbert space to be valid.

A simple way to distinguish between two time series of the same length is to treat the time series as vectors and simply employ a linear kernel or Radial basis kernel. This method can be simple and efficient provided the time series are short and of equal length. However, in many real-world applications, the time series of interest are of variable-length and can be quite long. It is therefore desirable to construct kernels capable of handling possibly long time series of variable length. For example, in dynamic time warping [4] time series similarity is quantified through finding an alignment between variable-length multivariate time series.

Another possibility is to use a generative probabilistic model of the time series data and then define the time series kernel through model parameters corresponding to different sequences, e.g. probability product kernel [52], KL divergence based kernels [17, 65] and Autoregressive kernel [26]. These approaches depend on the particular parametric model class. For example, *Fisher* kernel [47] maps individual time series into score functions of the single generative model that is assumed to be able to 'explain' most of the data. Often an HMM with a fixed number of states is employed. In some situations the assumption of the particular generative model underlying the data can be too strong. In addition, *Fisher* kernels [47] are computationally expensive because of the calculation of metric tensor (inverse of *Fisher* information matrix) in the tangent space of the generative model manifold. The 'practical' *Fisher* kernel used in most of the time replaces the metric tensor with an identity matrix. This can result in a loss of valuable information in the data [99].

The requirement of using a single generative model in kernel calculations is relaxed e.g. in the Autoregressive kernel [26]. Sequences are judged to be similar/dissimilar according to the corresponding likelihood profile of a vector autoregressive (VAR) model under a variety of parameter settings (controlled by the prior). In this case it is less crucial that the VAR model is a faithful model of the data since the base VAR model class is used as a 'feature extractor'.

Due to the requirements of many time series applications, the kernel evaluation should happen in real-time. Therefore, computational complexity of kernel construction and evaluation can play a critical role in applying kernel methods to time series data. However, many of the existing time series kernels are computationally demanding. For example DACO kernel [35] proposed recently by Gaidon et al. for action recognition, compares the dynamic aspects of two time series by using the difference between their auto-correlations. The kernelized DACO inevitably needs to invert a matrix of size related to the time series length. Thus the kernel can be used for relatively short time series only.

To address the problems mentioned above, we propose novel general time series kernels that can naturally and efficiently handle long time series data of variable length. The core idea is to transform the time series into a higher dimensional "dynamical feature space" via reservoir computation models [62] and then represent varying aspects of the signal through variation in the linear readout models trained in such dynamical feature spaces. In this way each time series will be represented by the corresponding readout model of the same *fixed* reservoir. Hence, unlike in the *Fisher* kernel, there will be a different dynamic model for each time series, but all such models will share the same dynamical reservoir.

The sequence-specific dynamic models will differ only in the corresponding linear readout models from the reservoir. The intuition is that while the general fixed dynamic reservoir provides a unique and rich pool of dynamic features for the whole data set, the individual readout models bring enough flexibility to represent specifics of different time series, thus providing a platform for wide applicability across time series of different characteristics and origins.

One can, of course, argue that our approach is yet another variation on model-based kernel construction for time series based on a particular class of dynamic (reservoir) models. However, unlike parametric time series models of a particular from, reservoir models have been extensively shown to be "generic" in the sense that they are able to represent a wide variety of dynamical features of the input signals, so that given a task at hand only the linear readout on top of the reservoir needs to be retrained [62]. As stated above, in our formulation, the underlying dynamic reservoir will be the *same* for all time series - the differences in the signal characteristics in different time series will be captured solely by the linear readout models and will be quantified in the function space of such models.

There are several advantages of such reservoir based time series kernels:

1. The proposed kernels can naturally handle time series of different length;

2. General reservoir model is flexible enough so that it can be used for a variety of data types without the need to specify a particular parametric model class for the time series;

3. Since only the linear readout on top of the reservoir needs to be trained, compared with most time series kernels, our kernels are computationally very efficient;

4. With recursive least squares algorithm to train readout mapping of reservoir models, our kernels can be operating in an on-line fashion, with the ability to efficiently

handle extremely long time series;

5. Under some assumptions, the model distances between linear readouts can be formulated analytically.

4.2 Reservoir Model Based Kernels

This section will introduce new time series kernels based on a general 'temporal filter' implemented by echo state network (ESN) with the simple deterministically constructed reservoir architecture described in section 2.2.3. As shown as figure 4.1, the main idea is that, provided the reservoir is able to represent a rich set of features of the input time series, the model-based representation of a particular time series $\boldsymbol{s}$ will be given by the linear readout mapping $f(\boldsymbol{z})$ operating on reservoir activations $\boldsymbol{z}$. The linear readout mapping is defined as $f(\boldsymbol{z}(t)) = W\boldsymbol{z}(t) + \boldsymbol{a}$ [1], where the activation $\boldsymbol{z}(t)$ is determined by the inputs and the previous states as $\boldsymbol{z}(t) = \tanh(R\ \boldsymbol{z}(t-1) + V\tilde{\boldsymbol{s}}(t))$. As reservoir architecture is fixed (R and V are fixed), the norm of the linear readout mapping W and the bias $\boldsymbol{a}$ can be obtained by linear regression through minimizing the mean square error (MSE) between the model predictions $f(\boldsymbol{z}(t))$ and the true targets $\boldsymbol{s}(t+1)$. The linear readout mapping of $\boldsymbol{s}_i$ is denoted by $f_i(\boldsymbol{z})$. The distance between a pair of time series $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$ is then be calculated using a model distance between the corresponding readouts $f_i(\boldsymbol{z})$ and $f_j(\boldsymbol{z})$. Finally, the kernel can be constructed using the distance through exponentiation.

4.2.1 Distance in the Reservoir Model Space

Uniform State Distribution Using Euclidean metric on the readout parameters to calculate the distance between two readout mappings is not satisfying since one should

[1]Here, we explicitly write out the bias term, for the purpose of investigating its influence on model distance between two linear readouts.

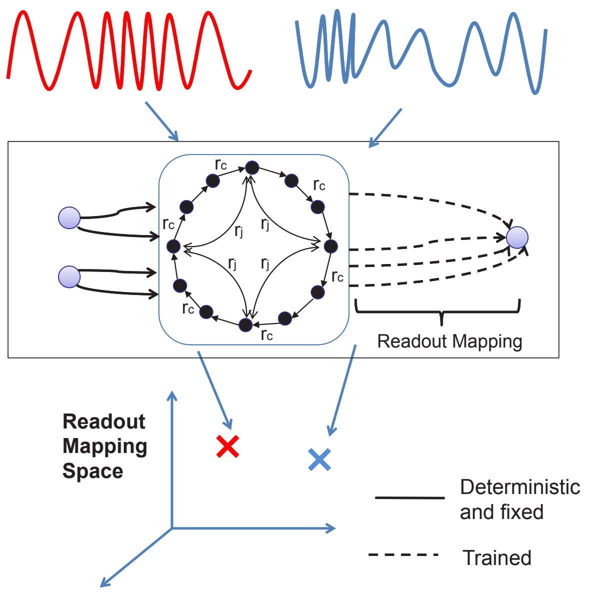


Figure 4.1: Illustration of the time series kernel in the deterministic reservoir model space. The first stage is to train the readout mapping of reservoir models using time series, i.e. generate individual points in the model space to represent time series. The second stage is to construct the kernel by investigating the model distance.

be interested in the model distance in the function space of the readout models, rather than the distance between the model parameterizations.

We will use the L_2 distance in the model space for its intuitive nature [21], although our framework is general and can be applied to any appropriate function distance between the readouts. Consider two mappings $f_1(\boldsymbol{z})$ and $f_2(\boldsymbol{z})$, $f_1,\ f_2 : \mathbb{R}^N \rightarrow \mathbb{R}^M$, where N is the number of reservoir units, M is the output dimensionality. Their L_2 distance is defined as:

$$L_2(f_1, f_2) = \left(\int_{\mathcal{Z}} \|f_1(\boldsymbol{z}) - f_2(\boldsymbol{z})\|^2 \, dP(\boldsymbol{z}) \right)^{1/2} \tag{4.1}$$

where $P(\boldsymbol{z})$ is a measurement on the input (reservoir) domain $\mathcal{Z}$. Since *tanh* transfer function is used to calculate the activation of the neurons, $\mathcal{Z} = [-1, +1]^N$.

First, the measurement $P(\boldsymbol{z})$ is assumed to be uniform. Later this assumption will be relaxed by considering non-uniform $P(\boldsymbol{z})$ to reflect the fact that the state space activations $\boldsymbol{z}$ in the reservoir can follow a more complex distribution.

The readout model takes the form of an affine mapping:

$$f(\boldsymbol{z}) = W\boldsymbol{z} + \boldsymbol{a}, \tag{4.2}$$

where $\boldsymbol{z} = [z_1, \cdots, z_N]^T$ is the state vector, W is the parameter matrix $(M \times N)$ and $\boldsymbol{a} = [a_1, \cdots, a_M]^T$ is the bias vector.

Consider two readouts from the *same* reservoir

$$\begin{aligned} f_1(\boldsymbol{z}) &= W_1\boldsymbol{z} + \boldsymbol{a}_1, \\ f_2(\boldsymbol{z}) &= W_2\boldsymbol{z} + \boldsymbol{a}_2. \end{aligned}$$

Then,

$$L_2(f_1, f_2) = \left(\int_{\mathcal{Z}} \|W\boldsymbol{z}\|^2 + 2\boldsymbol{a}^T W\boldsymbol{z} + \|\boldsymbol{a}\|^2 \, d\boldsymbol{z} \right)^{1/2}$$

where $W = W_1 - W_2$, and $\boldsymbol{a} = \boldsymbol{a}_1 - \boldsymbol{a}_2$.

Note that since $\mathcal{Z} = [-1, 1]^N$, for any fixed $\boldsymbol{a}$ and W

$$\int_{\mathcal{Z}} \boldsymbol{a}^T W\boldsymbol{z} \, d\boldsymbol{z} = 0.$$

Therefore, it can be shown that (see Appendix A)

$$L_2(f_1, f_2) = \left(\frac{2^N}{3} \sum_{j=1}^{N} \sum_{i=1}^{M} w_{i,j}^2 + 2^N \|\boldsymbol{a}\|^2 \right)^{1/2} \tag{4.3}$$

where $w_{i,j}$ is the (i, j)-th element of W.

In order to compare the difference between model distance in the function space and Euclidean distance in the function's parameter space, scaling of the squared model distance ($L_2^2(f_1, f_2)$) by 2^{-N} is applied:

$$\frac{1}{3}\sum_{j=1}^{N}\sum_{i=1}^{M} w_{i,j}^2 + \|\boldsymbol{a}\|^2 ,$$

We can see it differs from the squared Euclidean distance on the readout parameters

$$\sum_{j=1}^{N}\sum_{i=1}^{M} w_{i,j}^2 + \|\boldsymbol{a}\|^2 ,$$

by the factor 1/3 applied to the differences in the linear part W of the affine readouts. Hence, more importance is given to the 'offset' than 'orientation' of the readout mapping.

Non-uniform State Distribution In the above, we assumed that the probability distribution of reservoir states $\boldsymbol{z}$ is uniform in $\mathcal{Z}$. As mentioned before, it is likely that the state probability distribution $P(\boldsymbol{z})$ will be non-uniform. We denote the corresponding probability density function as $\mu(\boldsymbol{z})$ and we will introduce two approaches for allowing general $\mu(\boldsymbol{z})$ - modelling of the probability density function μ by a mixture of Gaussians and numerical approximation of the integral (4.1) by sampling using bootstrapped input series.

For non-uniformly distributed states $\boldsymbol{z}$, a K-component Gaussian mixture model can be employed to approximate the corresponding density distribution $p(\boldsymbol{z})$:

$$\begin{aligned} p(\boldsymbol{z}) &= \sum_{k=1}^{K} \pi_k \, \mathcal{N}(\boldsymbol{z}|\boldsymbol{\eta}_k, \Sigma_k), \\ \mathcal{N}(\boldsymbol{z}|\boldsymbol{\eta}_k\Sigma_k) &= \frac{\exp\left(-\frac{1}{2}(\boldsymbol{z}-\boldsymbol{\eta}_k)^T\Sigma_k^{-1}(\boldsymbol{z}-\boldsymbol{\eta}_k)\right)}{(2\pi)^{N/2}\left|\Sigma_k\right|^{1/2}}, \end{aligned}$$

where π_k are mixture coefficients with $\sum_{k=1}^{K}\pi_k = 1$, $\boldsymbol{\eta}_k$ is mean of the k-th component and Σ_k is the covariance matrix of the k-th component.

Then, the distance $L_2(f_1, f_2)$ can be obtained as follows:

$$
\begin{aligned}
L_2(f_1, f_2) &= \left(\int_{\mathcal{Z}} ||W\boldsymbol{z} + \boldsymbol{a}||^2 dP(\boldsymbol{z})\right)^{1/2} && (4.4)\\
&= \left(\int_{\mathcal{Z}} \left(\boldsymbol{z}^T W^T W \boldsymbol{z} + 2\boldsymbol{a}^T W \boldsymbol{z} + \boldsymbol{a}^T \boldsymbol{a}\right) p(\boldsymbol{z}) d\boldsymbol{z}\right)^{1/2} && (4.5)\\
&= \left(\sum_{k=1}^{K} \pi_k \left(\mathrm{Tr}(W^T W \Sigma_k) + \boldsymbol{\eta}_k^T W^T W \boldsymbol{\eta}_k + 2\boldsymbol{a}^T W \boldsymbol{\eta}_k + \boldsymbol{a}^T \boldsymbol{a}\right)\right)^{1/2}. && (4.6)
\end{aligned}
$$

More detailed derivation is given in Appendix A.

We employed the mixture model construction proposed by Figueiredo et. al [33] that automatically selects the appropriate number of mixture components in a top-down manner.

Alternatively, the integral can be numerically approximated by using reservoir activations collected while processing the input time series. Assume that for a given time series $\boldsymbol{s}$, after the initial wash-out (initial activations upto certain steps need to be dismissed, in order to wait the effects of the initial state to die out) [49], m state activations are collected $\boldsymbol{z}(1), \cdots, \boldsymbol{z}(m)$. Then,

$$
L_2^2(f_1, f_2) \approx \frac{1}{m} \sum_{t=1}^{m} \|f_1(\boldsymbol{z}(t)) - f_2(\boldsymbol{z}(t))\|^2 . \qquad (4.7)
$$

However, in some applications the length of the time series is not sufficient to yield a good approximation. We therefore adopted the circular block-resampling bootstrap for time series [58] to construct sufficiently long input series. This bootstrap procedure first wraps the time series around in a circle, and then extracts L blocks from the wrapped time series, where L is the length of the original time series, and finally, samples with replacement from the obtained set of blocks. The resulting time series is concatenated from the resampled blocks. The block length in bootstrapping was automatically determined by following [72], based on spectral estimation via the flat-top lag-windows.

4.2.2 Time Series Kernels via Reservoir Models

In the above, the function distance between readout mappings of reservoir models is formulated. Therefore, the three kernels can be defined as follows:

$$\mathcal{K}(\boldsymbol{s}_i, \boldsymbol{s}_j) = \exp\left\{-\gamma \cdot L_2^2(f_i, f_j)\right\},$$

where $L_2^2(f_i, f_j)$ can be Equations (4.3), (4.6) and (4.7) as reservoir (RV) kernel, Gaussian mixture model based reservoir ($GMMRV$) kernel ,

and sampling based reservoir ($SamplingRV$) kernel. γ is the kernel scale parameter. The main algorithm is summarized below:

Algorithm 1 Model based kernel algorithm (*RV, GMMRV, SamplingRV*)

```
1: Input: Set of sequences $\boldsymbol{s}_1, \cdots, \boldsymbol{s}_n$; parameters (number of reservoir units $N$; CRJ
   weights $(r_c, r_j, r_i)$; ridge regression parameter $\lambda$; kernel scale parameter $\gamma$;
2: Output: Kernel (Gram) matrix $\mathcal{K}$.
3: for each time series $\boldsymbol{s}_i$, $i = 1, \cdots, n$ do
4:     Drive the reservoir state evolution with the input sequence $\boldsymbol{s}_i$.
5:     Fit the linear readout $f_i$ using ridge regression for the next item prediction task on
       $\boldsymbol{s}_i$.
6: end for
7: Calculate the pairwise model distance matrix $L_2(f_i, f_j)$ $i, j = 1, \cdots, n$, via eqs. (4.3)
   ($RV$), (4.6) ($GMMRV$), or (4.7) ($SamplingRV$) - Section 4.2.1.
8: Calculate the kernel matrix as $\mathcal{K}(\boldsymbol{s}_i, \boldsymbol{s}_j) = exp\{-\gamma \cdot L_2^2(f_i, f_j)\}$.
```

4.2.3 Fisher Kernel Based on Reservoir Model

Besides the model distance based kernels introduced above, we also considered the *Fisher* kernel obtained with the reservoir model ($FisherRV$).

Endowing the readout with a noise model yields a generative time series model of the form:

$$\begin{aligned} \boldsymbol{z}(t) &= g(R\,\boldsymbol{z}(t-1) + V\boldsymbol{s}(t)), \\ \boldsymbol{s}(t+1) &= W\boldsymbol{z}(t) + \boldsymbol{a} + \boldsymbol{\varepsilon}(t), \end{aligned}$$

Assume the i.i.d. noise model $\boldsymbol{\varepsilon}(t)$ follows a Gaussian distribution,

$$\boldsymbol{\varepsilon}(t) = \mathcal{N}(0, \sigma^2 I).$$

Then,

$$\begin{aligned} & P((\boldsymbol{s}(t+1) \mid \boldsymbol{s}(1..t)) = P((\boldsymbol{s}(t+1) \mid \boldsymbol{z}(t)) \\ = \quad & (2\pi\sigma^2)^{-M/2} \exp\left\{ -\frac{\|\boldsymbol{s}(t+1) - W\boldsymbol{z}(t) - \boldsymbol{a}\|^2}{2\sigma^2} \right\}, \end{aligned}$$

where $\boldsymbol{s}(1..t)$ denotes the time series $\boldsymbol{s}(1), \boldsymbol{s}(2), \cdots, \boldsymbol{s}(t)$.

Slightly abusing mathematical notation, the model likelihood $p(\boldsymbol{s}(1..L))$ given the time series $\boldsymbol{s}$ of length L can be written as follows:

$$\begin{aligned} & p(\boldsymbol{s}(1..L)) = \prod_{t=1}^{L} P(\boldsymbol{s}(t) \mid \boldsymbol{s}(1..t-1)) \\ = \quad & \prod_{t=1}^{L} (2\pi\sigma^2)^{-M/2} \exp\left\{ -\frac{\|\boldsymbol{s}(t) - W\boldsymbol{z}(t-1) - \boldsymbol{a}\|^2}{2\sigma^2} \right\}. \end{aligned}$$

Therefore, the partial derivative of log likelihood $\log p(\boldsymbol{s}(1..L))$ can be obtained as

$$\begin{aligned} U &= \frac{\partial \log p(\boldsymbol{s}(1..L))}{\partial W} \\ &= \sum_{t=1}^{L} \frac{(\boldsymbol{s}(t) - \boldsymbol{a})\ \boldsymbol{z}(t-1)^T - W\boldsymbol{z}(t-1)\boldsymbol{z}(t-1)^T}{\sigma^2}. \end{aligned}$$

Note that the partial derivative U is an $(M \times N)$ matrix. As presented in [91], the practical Fisher kernel is defined as the dot product of two corresponding Fisher score vectors. In our case, the partial derivative U is an $(M \times N)$ matrix. Therefore, element-wise product is involved. The 'practical' *Fisher* kernel for two time series $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$ with scores U_i and U_j, respectively, can be formulated as

$$\mathcal{K}(\boldsymbol{s}_i, \boldsymbol{s}_j) = \sum_{k=1}^{M} \sum_{l=1}^{N} (U_i \circ U_j)_{k,l},$$

where $\circ$ is Hadamard (element-wise) product. This definition is equivalent to vectorize the matrices U_i and U_j first, and then define the Fisher kernel as dot product between the obtained vectors as in [91]. In practice, the noise variance σ^2 can be estimated from the original time series and the output of the fitted readout model. If the output of the fitted readout model is denoted as $\boldsymbol{o}(t)$, the noise variance σ^2 can be estimated as follows:

$$\sigma^2 \approx \frac{1}{L-1} \sum_{t=1}^{L-1} ||\boldsymbol{s}(t+1) - \boldsymbol{o}(t)||^2 \tag{4.8}$$

4.3 Experimental Studies

This section presents experimental results of the proposed kernels, *RV*, *GMMRV*, *SamplingRV*, *FisherRV*, and other existing time series kernels, including autoregressive (*AR*) kernel, *Fisher* kernel with hidden Markov models (*Fisher*), and dynamic time warping based kernel (*DTW*).

All hyperparameters, such as the kernel width γ and order p in the *AR* kernel, number of hidden states in the HMM based *Fisher* kernel etc. have been set by 5-fold cross-validation on the training set. The search ranges for parameters of each algorithm are detailed in Table 4.1. In the reservoir based kernels, we used a fixed topology reservoir (cycle with jumps) [76] for all data sets: $N = 100$, 15 jumps. The cycle weight r_c, jump

Table 4.1: Parameters for all kernels. γ is the parameter in RBF function, ξ in *AR* kernel is the weight of the negative definite kernel [26], p is the order of the vector autoregressive model, *state* is the number of states for HMM in *Fisher* kernel, λ is the ridge regression parameter.

Kernel	Parameters	Parameter range
DTW	γ	$\gamma \in \{10^{-6}, 10^{-5}, \cdots, 10^{1}\}$,
AR	γ, ξ, p	$\gamma \in \{10^{-6}, 10^{-5}, \cdots, 10^{1}\}$, $\xi \in \{0.1, 0.2, \cdots, 0.9\}$, $p \in \{1, 2, \cdots, 10\}$
Fisher	*state*	*state* $\in \{1, 2, \cdots, 10\}$
RV, *FisherRV*, *GMMRV*, *SamplingRV*	γ, λ	$\gamma \in \{10^{-6}, 10^{-5}, \cdots, 10^{1}\}$, $\lambda \in \{10^{-5}, 10^{-4}, \cdots, 10^{1}\}$

weight r_j, input weight r_i and readout were obtained on the training set. The readout mapping was trained via ridge regression (hyperparameter λ tuned via cross-validation). To evaluate the readout model distance in the *SamplingRV* kernel, except for long time series in the PEMS data set (Section 4.4), the bootstrapped time series were 5 times longer than the original ones[1].

The implementation of *AR* kernel was obtained from Marco Cuturi's website[2]. *Fisher* kernel was obtained Maaten's website[3].

We employ a well-known, widely accepted and used implementation of SVM – LIBSVM [19]. In LIBSVM, we use cross validation to tune the regularization parameter C. The candidates of C were $C \in \{10^{-3}, 10^{-2}, ..., 10^{3}\}$. After model selection using cross-validation on the training set, the selected model class representatives were retrained on the whole training set and were evaluated on the test set. Multi class classification is performed via the one-against-one strategy (default in LIBSVM). Kernel matrices of different kernels were precomputed and presented to SVM using the option of precomputed kernel in LIBSVM.

[1] $m = 5\ |\boldsymbol{s}|$, where $|\boldsymbol{s}|$ indicates the length of the time series.
[2] `http://www.iip.ist.i.kyoto-u.ac.jp/member/cuturi/AR.html`(12/11/2014)
[3] `http://homepage.tudelft.nl/19j49/Software.html`(12/11/2014)

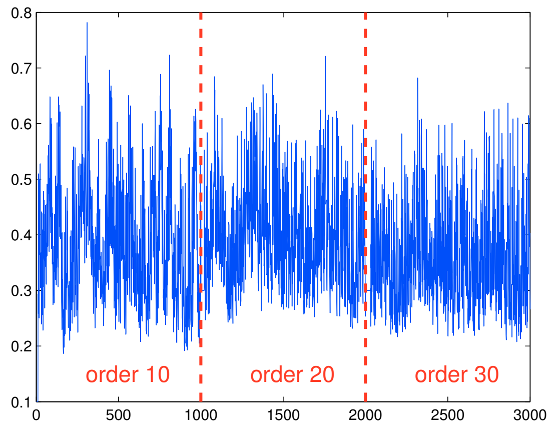


Figure 4.2: Illustration of three NARMA sequences with different orders (10, 20 and 30).

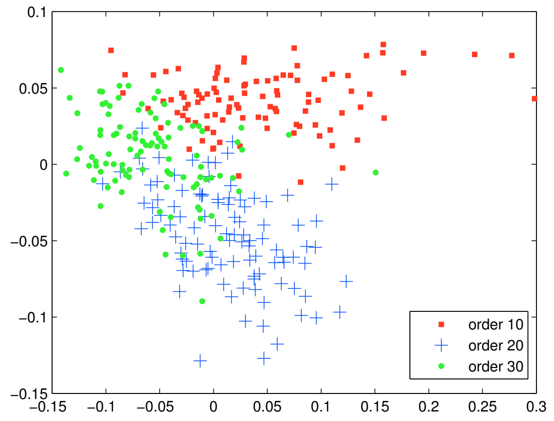


Figure 4.3: Illustration of MDS on the model distance among reservoir weights. It shows the separability of the three time series in the model space.

4.3.1 Synthetic Data

We employed three NARMA time series models of orders 10, 20 and 30, given by:

$$s(t+1) = 0.3s(t) + 0.05s(t)\sum_{i=0}^{9} s(t-i) + 1.5u(t-9)u(t) + 0.1,$$

$$\begin{aligned} s(t+1) \quad &= \quad \tanh(0.3s(t) + 0.05s(t)\sum_{i=0}^{19} s(t-i) + \\ &\quad\quad +1.5u(t-19)u(t) + 0.01) + 0.2, \end{aligned}$$

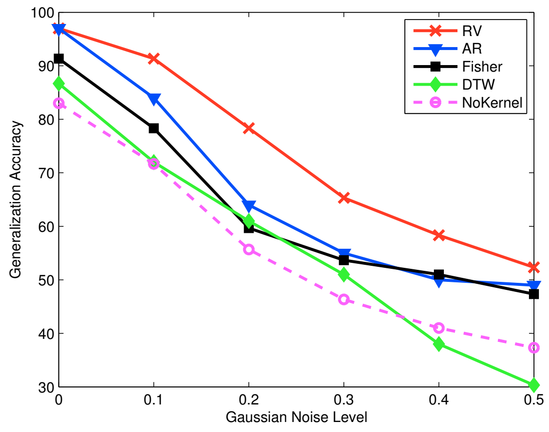


Figure 4.4: Illustration of the performance of compared kernels with different noise levels.

$$s(t+1) = 0.2s(t) + 0.004s(t)\sum_{i=0}^{29} s(t-i) + 1.5u(t-29)u(t) + 0.201,$$

where $s(t)$ is the output at time t, $u(t)$ is the input at time t. The inputs $u(t)$ form an i.i.d stream generated uniformly in the interval $[0, 0.5)$. We use the same input stream for generating the three long NARMA time series (60,000 items), one for each order. The three sequences are illustrated in Figure 4.2. The time series are challenging due to non-linearity and long memory.

For each order, the series of 60,000 numbers is partitioned into 200 non-overlapping time series of length 300. The first 100 time series for each order are used as training set, and the other 100 time series form the test set.

As apparent from Figure 4.2, distinguishing the three NARMA models using the original time series may be challenging. However, when viewing the time series through the model space of fitted reservoir models, the three time series classes become separated, as illustrated in Figure 4.3 showing 2-dimensional multi-dimensional scaling[1] representation of the pair-wise readout model distances. Multidimensional scaling is a dimension re-

[1]Multidimensional scaling (MDS) aims to preserve the pairwise distance between points, which is suitable to preserve the *model distance* for visualization.

duction technique which projects points in high-dimensional space into lower dimension, preserving pair-wise distance between points.

In order to study robustness of the kernels we corrupt the time series with additive Gaussian noise (zero mean, standard derivation varies in [0.1,0.5]). Figure 4.4 shows the test set classification accuracy against the noise level. As a baseline we also include results by SVM operating on the time series directly (300-dimensional inputs) - *NoKernel*. The *RV* reservoir based kernel outperforms the baseline and the other time series kernels.

4.3.2 Benchmark Data

Table 4.2: Description of the data sets

Dataset	Length	Classes	Train	Test
Symbols	398	6	25	995
OSULeaf	427	6	200	242
Oliveoil	570	4	30	30
Lighting2	637	2	60	61
Beef	470	6	30	30
Car	576	4	60	60
Fish	463	8	175	175
Coffee	286	2	28	28
Adiac	176	37	390	391

Table 4.3: Comparison of *DTW*, *AR*, *Fisher* (with hidden Markov models), *RV*, *FisherRV*, *GMMRV*, and *SamplingRV* kernels on nine benchmark datasets by accuracy. The best performance for each data set has been boldfaced.

Dataset	*DTW*	*AR*	*Fisher*	*RV*	*FisherRV*	*GMMRV*	*SamplingRV*
Symbols	94.77	91.15	94.42	**98.08**	95.96	97.31	95.77
OSULeaf	**74.79**	56.61	54.96	69.83	64.59	56.55	63.33
Oliveoil	83.33	73.33	56.67	86.67	83.33	84.00	**90.00**
Lighting2	64.10	77.05	64.10	77.05	75.41	78.69	**80.33**
Beef	66.67	78.69	58.00	80.00	68.00	79.67	**86.67**
Car	58.85	60.00	65.00	76.67	72.33	78.33	**86.67**
Fish	69.86	60.61	57.14	79.00	74.29	78.00	**85.71**
Coffee	85.71	**100.00**	81.43	**100.00**	92.86	96.43	**100.00**
Adiac	65.47	64.45	68.03	72.63	71.61	74.94	**76.73**

Table 4.4: CPU Time (in seconds) of *DTW*, *AR*, *Fisher* (with hidden Markov models), *RV*, *FisherRV*, *GMMRV*, and *SamplingRV* kernels on nine benchmark datasets.

Dataset	*DTW*	*AR*	*Fisher*	*RV*	*FisherRV*	*GMMRV*	*SamplingRV*
Symbols	1,318	2,868	2,331	202	236	374	808
OSULeaf	6,030	1,375	3,264	98	111	186	447
Oliveoil	295	113	832	11	19	27	43
Lighting2	918	151	1,143	33	46	61	95
Beef	107	54	87	10	17	23	40
Car	679	442	902	27	42	50	84
Fish	3,353	495	1,998	81	96	159	286
Coffee	21	25	145	3	3	7	19
Adiac	550	8131	1,122	201	213	394	699

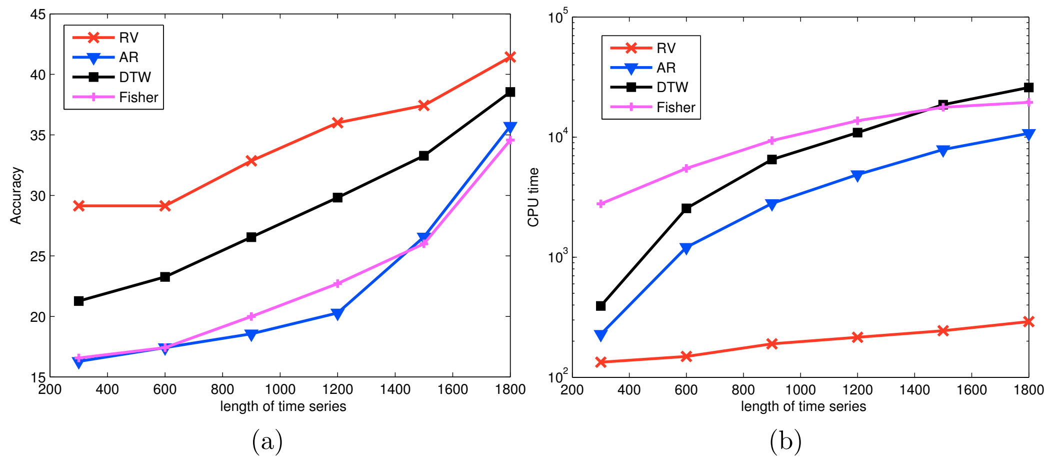


Figure 4.5: Comparison of generalization accuracy (a) and CPU time (b) in seconds of *RV*, *AR*, *DTW* and *Fisher* kernels on InlineSkate data set.

We used 9 datasets from UCR Time Series Repository [53]. Each data set has already been split into training and test sets (see Table 4.2).

Table 4.3 reports performance of the time series kernels on the benchmark data in terms of test set classification accuracy. *SamplingRV* kernel outperforms the other kernels on 7 datasets; *RV* is superior on 2 datasets and *DTW* outperforms the other kernels on 1 data set. In terms of computation time[1], the reservoir kernels are clearly the most

[1]The computational environment is Windows XP with Intel Core 2 Duo 1.66G CPU and 4G RAM.

efficient. Table 4.4 shows the average CPU time taken to evaluate the kernels in seconds[1]. *SamplingRV* kernel is obviously the most expensive among the reservoir kernels. Still, it is faster than its state-of-art competitors.

To further compare the computational effectiveness of the kernels, a relatively large data set, *InlineSkate* from UCR time series repository, has been employed. The data set contains 650 time series (100 training, 550 test) of length 1882, belonging to 7 classes. The influence of time series length on the classification performance and computational complexity was studied by considering from each training time series only the first ℓ elements, with ℓ growing from 300 to 1800 in increments of 300. The resulting accuracy and CPU times are shown in Figure 4.5. Relatively to the other kernels, the reservoir *RV* kernel has the lowest computational cost, while achieving competitive performance.

4.3.3 Multivariate Time Series

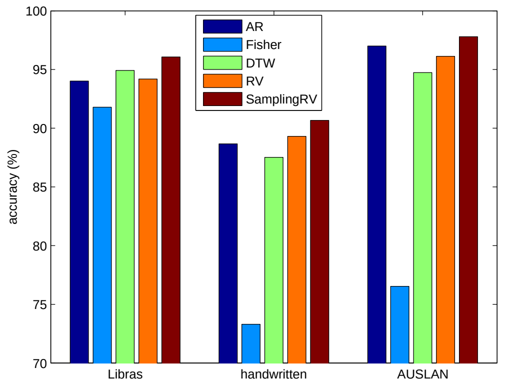
Table 4.5: Summary of multivariate (variable length) time series classification problems.

Dataset	dim	length	classes	train	test
Libras	2	45	15	360	585
handwritten	3	60-182	20	600	2258
AUSLAN	22	45-136	95	600	1865

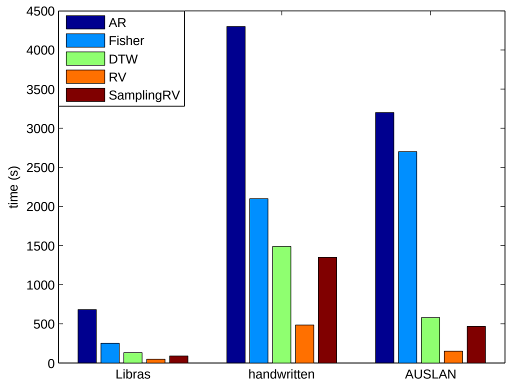
Datasets used so far involved univariate time series. In this section, we perform classification on three multivariate time series - Brazilian sign language (*Libras*), *handwritten* characters and Australian language of signs (*AUSLAN*). Unlike the other data sets, the *handwritten* characters and *AUSLAN* data sets contain time series of variable length. Following [26] (previous *AR* kernel study) we split the data sets into training and test sets as detailed in Table 4.5.

The results are shown in Figure 4.6. *SamplingRV* is superior on all three data sets. *RV* kernel is outperformed by *DTW* and *AR* kernels on *Libras* and *AUSLAN* data sets,

[1]We do not record the cross validation time for SVM.



(a)



(b)

Figure 4.6: Comparison of generalization accuracy (a) and CPU time (b) in seconds of *AR*, *Fisher*, *DTW*, *RV* and *SamplingRV* kernels on 3 multivariate time series.

respectively. In terms of CPU time, *RV* kernel usually uses the least and *AR* consumes the most computation time.

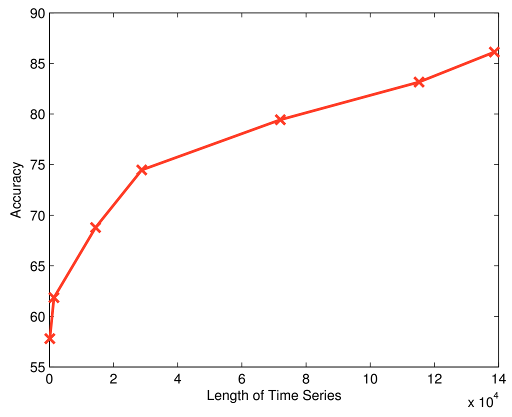


Figure 4.7: Generalization accuracy of on-line *RV* kernel on PEMS time series.

4.4 On-line Reservoir Kernel

Reservoir readouts can be trained in an on-line fashion using recursive least squares (RLS). In RLS, the readout weights W are recursively updated at every time step t (Detailed derivation is given in Appendix B):

$$\begin{aligned}
\boldsymbol{k}(t) &= \frac{\Lambda(t-1)\ \tilde{\boldsymbol{z}}(t)}{\tilde{\boldsymbol{z}}^T(t)\ \Lambda(t-1)\ \tilde{\boldsymbol{z}}(t)+\varsigma} \\
\Lambda(t) &= \varsigma^{-1}\Big(\Lambda(t-1)-\boldsymbol{k}(t)\ \tilde{\boldsymbol{z}}^T(t)\ \Lambda(t-1)\Big) \\
W(t) &= W(t-1)+[\boldsymbol{s}(t+1)-\boldsymbol{o}(t)]\ \boldsymbol{k}^T(t),
\end{aligned}$$

where $\boldsymbol{k}(t)$ stands for the innovation vector; $\boldsymbol{s}(t+1)$ and $\boldsymbol{o}(t)$ correspond to the desired (next-item prediction) and calculated (readout) output; $\Lambda(t)$ is the error covariance matrix. 'Forgetting parameter' $0 < \varsigma < 1$ is usually set to a value close to 1.0. In this work ς is set by cross validation.

This enables us to construct and refine reservoir kernels on-line, as more and more data become available. This can be particularly convenient in situations where individual items to be classified (time series) are not fixed, but appear in an on-line manner.

After observing sufficiently long initial segments of the time series it is possible to train the classifier and perform initial classification. As more and more data arrives, the reservoir kernels can be updated recursively, without the need to re-construct the kernels from scratch.

We illustrate this approach on a set of long series *PEMS-SF* (UCI machine learning repository) with 440 time series of length 138,672. The data reports the occupancy rate of different car lanes of San Francisco freeways within 15 months. The generalization performance of on-line *RV* kernel is reported in Figure 4.7. As expected, the generalization improves monotonically with increasing amount of data. On full data *RV* kernel achieves 86.13% accuracy. This compares favourably with the best reported performance levels (82% $\sim$ 83%) [26] among a variety of time series kernels, such as *AR*, global alignment kernel [28], splines smoothing kernel [57] and Bag of vectors kernel [43].

4.5 Discussion and Conclusion

In this chapter efficient kernels have been proposed to tackle the challenges in time series classification through kernel machines. Instead of constructing the kernel directly in the original data space, this paper introduces a 'kernel in the deterministically constructed reservoir model space' that represents each time series as a reservoir model with the common dynamic part.

We demonstrated the application of the distance definition in the (function) model space of linear readout models. The model distance is different from the Euclidean distance of the readout parameters, indicating that more importance is given to the 'offset' than 'orientation' of the readout mapping. We also estimated the model distance by using either sampling methods or a Gaussian mixture model when the reservoir state distribution is non-uniform.

The proposed kernels were compared with other competitors on synthetic and benchmark data sets. The results confirm the effectiveness of reservoir based kernels. The on-line reservoir kernels proposed in Section 4.4 can process extremely long time series efficiently.

In general, the simple RV kernel that build upon the closed form of distance in model space is the most efficient[1]. However, it is obtained under the (rather unrealistic) assumption of uniform state distribution and the tolerable increase in computational cost by the *SamplingRV* kernel is well offset by the increase in the classification accuracy. The *GMMRV* kernel can also be analytically obtained via approximating the state distribution by a Gaussian mixture. Of course, the quality of this kernel depends on how well the state distribution is captured by the Gaussian mixture model used.

It is interesting that the *Fisher* kernel based on the reservoir model achieves better

[1]It is worth noting that there also exist fast implementations of non-kernelized variations of DACO and global alignment kernels.

performance than the *Fisher* kernel based on the HMM model with continuous (Gaussian distributed) emissions. The principal difference between the reservoir model and HMM is that in the reservoir model the state space is infinite (uncountable) with deterministic input-driven dynamics. In HMM the state space is finite and latent, with probabilistic state transitions.

In conclusion, reservoir model based time series kernels can achieve competitive performance in terms of both generalization accuracy and computational time, without the need for explicit specification of the parameterized model class for the time series data. This is potentially of great benefit in cases of very large data sets of long time series where the underlying parametric model is unknown. Reservoir kernels stand and fall on the ability of the particular dynamic reservoir to generate a rich pool of dynamical features sufficiently representing the variety of time series occurring in a given task. If the echo state property - a cornerstone of reservoir modelling - is not an appropriate modelling assumption, the reservoir kernels cannot be expected to perform well. However, as has been demonstrated numerous times, for most real-world data the fading memory assumption (encapsulated in the echo state property) is appropriate.

The kernels designed in this chapter can be viewed as an special instance of model-based kernels. These kind of time series kernels are constructed upon time series models. Time series models act as representations of data and kernel similarity is then build using the model representations. Other time series model such as hidden Markov model (HMM) can replace the reservoir model in the kernel construction and the distance between two model representation needs to change accordingly.

CHAPTER 5

Adaptive LiMS Kernel For Time Series Classification

The previous chapter presented the model based kernels for time series classification based on the echo state network with a simple deterministically constructed reservoir structure. In this chapter, we present an adaptive model based time series kernel, termed *model-metric co-learning* (MMCL). This methodology is also developed within the framework of *learning in the model space* - each data item (sequence) is represented by a predictive model from a carefully designed model class. The difference is that this method co-learns the dynamic reservoir and a global metric in the linear readout model space. Before we introduce this MMCL methodology in section 5.3, we demonstrate an initial investigation on learning the dynamic reservoir in the deterministically constructed echo state network in section 5.2. Finally, Section 5.4 discusses and concludes the main findings.

5.1 Introduction

So far approaches to model based sequence classification either used a single model fitted on the full data (i.e. Fisher kernel [47])/individual classes [89], or fit a full linear dynamic model on each sequence [81]. The models fitted on the full data/classes can be more complex than those fitted on individual sequences. The price to be paid is the potential inadequacy of such a model to capture individual variations among the data items. On the other hand, individual sequence models have to be relatively simple (e.g. with linear dynamics) to keep the model estimation stable. We propose a new hybrid approach spanning the two extremes in an attempt to keep the best of both worlds. Following the core idea presented in the previous chapter, we represent individual sequences through linear readout models from an adaptive high-dimensional non-linear state space model with a highly constrained dynamic part. The dynamic part is the same for all data items and acts as a temporal filter providing a rich pool of dynamic features that can be selectively extracted by individual (static) linear readout mappings representing the sequences. Alongside learning the dynamic part, we also learn the global metric (distance function) in the readout model space. The overall goal is to adapt the shared dynamical system and metric tensor on the readout maps (standing for the sequences) so that sequences from the same class are represented by 'close' readouts, while readouts of sequences from different classes are well-separated. We term our methodology *model-metric co-learning* (MMCL).

The MMCL framework for sequence classification builds on ideas of model based representation for sequences [17] and discriminative learning [11, 99] but differs mainly in three aspects: First, the final representation of the sequences is linear readout mapping, but the full underlying model is a non-linear dynamic system. In this way our methodology reduces the computational demands without the loss of the computational ability of non-linear models. Second, it treats the model parameters adaptation and model distance designing jointly, rather than adapting the model parameters first and defining the model distance afterwards in two independent steps. Third, it has the similar motivation as other discriminative kernels [99], but differs in its goal of utilizing models that both represent the time series well and at the same time best separate the time series classes.

5.2 Learning the Deterministically Constructed Echo State Networks

Echo state network [50,51], a very simple form of reservoir computing method, has already been introduced in section 2.2.3. The randomization of generating the reservoir in ESN cause it to be poorly understood, as a result leaving the room for further investigation on what exactly a reservoir structure leads to good performance for a given problem [42, 68, 78]. A simple cycle reservoir (SCR) introduced in [77] shows comparable performances to the traditional randomized ESN. One extension of the cycle reservoir by adding regular bidirectional jumps (shortcuts) (CRJ) introduced in [76] has shown superior performance to those of the traditional randomized reservoir models in non-linear system identification, real time series prediction and speech recognition [76]. Moreover, the characterizations of the selected reservoir model is well understood.

However, in the original CRJ model introduced in [76], the parameters that govern the design of reservoir are tuned by costly and potentially unstable cross-validation technique using an exhaustive grid search method. Furthermore, the selected parameters from grid search may not be the optimal, because of the discretization of the continuous parameters.

We propose a hybrid optimization strategy to learn the parameters in the CRJ network. The linear output weights of the network are determined by ridge regression, while the reservoir weights are found using a nonlinear optimization technique. Regularization on the output weights and early stopping strategy have been incorporated in this proposed training strategy. The experimental results show that the new learning method tremendously improves the computational efficiency and can achieve comparable, sometimes even better, performance to the original cross-validation CRJ fitting.

5.2.1 Basic Algorithm

The mathematical formulation of the deterministically constructed CRJ model is given as follows:

$$\begin{aligned} \mathbf{z}(t) &= \tanh(R\,\mathbf{z}(t-1) + V\ \mathbf{s}(t)), & (5.1)\\ \mathbf{o}(t) &= W\mathbf{z}(t) & (5.2)\end{aligned}$$

where $\boldsymbol{z}(t) \in \mathbb{R}^N$, $\boldsymbol{s}(t) \in \mathbb{R}^d$ and $\boldsymbol{o}(t) \in \mathbb{R}^M$ are the state vector, input vector and output at time t, respectively; R is a $(N \times N)$ dynamic coupling matrix; $V \in \mathbb{R}^{N\times d}$ and $W \in \mathbb{R}^{M\times N}$ are the input and output weight matrices [1], respectively. The function tanh is applied element-wise on the resulting matrix. Here, we allow that the dimensionality of the input vector can be different from the dimensionality of output vector, making the model more flexible.

As mentioned in section 2.2.3, in the CRJ architecture, all cyclic connections have the same weight denoted by r_c, all jumps share the same weight denoted by r_j, and the input connections have the same absolute value denoted by r_i with an aperiodic sign pattern. Thus, the dynamic coupling matrix R is determined by two free parameters r_c and r_j, where r_i determines the input weight matrix V. Consequently, only three free parameters, r_c,r_j, and r_i, need to be learned during training.

The problem of training the network can be formulated as a problem of the minimization of an error function E. For convenience, r_c, r_j and r_i are grouped together into a single vector $\mathbf{r}$. We choose the sum-of-squares error function when training the network. Assuming the network is running from time step 1 up to time step L, the sum-of-squares

[1]We can add the state vector $\boldsymbol{z}(t)$ with a constant element (e.g. 1) and W with a column to account for the bias term. To simplify the presentation, we omit the bias term.

error is given as follows:

$$E = \sum_{t=1}^{L} ||\mathbf{o}(t) - \mathbf{y}(t)||^2 \tag{5.3}$$

where $\mathbf{y}(t) \in \mathbb{R}^M$ is the target at time step t and $||\cdot||$ denotes the Euclidean norm. Since the value of the sum-of-squares error function depends on the number of patterns, we consider a normalized error function for assessing the performance of the trained network. As suggested in [5], we choose the normalized mean square error function as follows:

$$\tilde{E} = \frac{\langle ||\mathbf{o}(t) - \mathbf{y}(t)||^2 \rangle}{\langle ||\mathbf{y}(t) - \langle \mathbf{y}(t) \rangle ||^2 \rangle} \tag{5.4}$$

where $\langle \cdot \rangle$ denotes the empirical mean.

The network we considered here is a recurrent network with linear output units. Since the dependence of the network mapping on the final-layer weight is linear, the partial optimization of sum-of-squares error function with respect to these weights can be performed by linear methods. The computational effort involved in linear methods is often very much less than that required for general non-linear optimization. Therefore, we adopt a hybrid procedure for optimizing the weights in the network, where linear method is used for obtaining the final layer weights, and non-linear method is used for acquiring all the other parameters [103].

The error function E is a quadratic function of W. For any given value $\boldsymbol{r}$, where $\mathbf{r} = \{r_c, r_j, r_i\}$, we can perform a one-step exact minimization with respect to the W using linear regression, in which $\mathbf{r}$ is held fixed. The gradient of E with respect to W is as follows (details are given by C.1):

$$\frac{\partial E}{\partial W} = 2 \sum_{t=1}^{L} (\mathbf{o}(t) - \mathbf{y}(t)) \mathbf{z}^T(t) \tag{5.5}$$

If we collect the network states $\mathbf{z}(t)$, for a given $\mathbf{r}$, into a matrix Z and the target values $\mathbf{y}(t)$ in a vector (matrix if the output is multi-dimensional) Y, by making the gradient of

E with respect to W equal to zero, the output weights can be computed as follows:

$$W = YZ^T(ZZ^T)^{-1} \tag{5.6}$$

where $Y = [\mathbf{y}(1), ..., \mathbf{y}(L)]$ and $Z = [\boldsymbol{z}(1), ..., \boldsymbol{z}(L)]$ respectively. Since the output weights W are regarded as a function of $\mathbf{r}$ and can be chosen using Equation (5.6), we can regard E as a nonlinear function of $\mathbf{r}$ only and a nonlinear function optimization method, e.g. conjugate gradient descent [5], is employed to find these weights by minimizing E with respect to $\mathbf{r}$. The gradient of E with respect to $\mathbf{r}$ is computed using real-time recurrent learning [106].

Since the total error is the sum of the errors at the each time steps, we can compute the gradient by summing up the gradient at each time step via (details see Appendix C.1)

$$\frac{\partial E}{\partial \theta} = 2\sum_{t=1}^{L}(\mathbf{o}(t) - \mathbf{y}(t))^T W \frac{\partial \mathbf{z}(t)}{\partial \theta} \tag{5.7}$$

The gradient of $\mathbf{z}(t)$ with respect to θ, where $\theta = r_c, r_j$ or r_i, can be computed iteratively as:

$$\frac{\partial \mathbf{z}(t)}{\partial r_c} = \operatorname{sech}^2(R\mathbf{z}(t-1) + Vs(t)).* \left(R\frac{\partial \mathbf{z}(t-1)}{\partial r_c} + \frac{\partial R}{\partial r_c}\mathbf{z}(t-1)\right), \tag{5.8}$$

$$\frac{\partial \mathbf{z}(t)}{\partial r_j} = \operatorname{sech}^2(R\mathbf{z}(t-1) + Vs(t)).* \left(R\frac{\partial \mathbf{z}(t-1)}{\partial r_j} + \frac{\partial R}{\partial r_j}\mathbf{z}(t-1)\right), \tag{5.9}$$

$$\frac{\partial \mathbf{z}(t)}{\partial r_i} = \operatorname{sech}^2(R\mathbf{z}(t-1) + Vs(t)).* \left(\frac{\partial V}{\partial r_i}s(t) + R\frac{\partial \mathbf{z}(t-1)}{\partial r_i}\right), \tag{5.10}$$

where sech is Hyperbolic secant function and $.*$ stands for element-wise matrix multiplication (MATLAB notation). As usual in real time recurrent learning, the initial conditions for the update equations can be set to:

$$\frac{\partial \mathbf{z}(0)}{\partial r_c} = 0, \ \frac{\partial \mathbf{z}(0)}{\partial r_j} = 0, \ \frac{\partial \mathbf{z}(0)}{\partial r_i} = 0. \tag{5.11}$$

The reservoir weights $\mathbf{r}$ are obtained using a non-linear optimization algorithm while the output weights W are regarded as a function of $\mathbf{r}$ and are chosen using Equation (5.6). Every time the value of $\mathbf{r}$ is changed, the weights W need to be recomputed. Thus, the optimization strategy in this section proceeds the training process on two timescales: Longer timescale, where the weights $\mathbf{r}$ are adjusted to minimize the error function, and a short timescale, where the output weights are changed to minimize the error as a function of the weights $\mathbf{r}$ alone.

The hybrid optimization strategy of combining linear methods and non-linear methods together is chosen here mainly for two reasons [5]: First, the dimensionality of the effective search space for the non-linear algorithm is reduced to only 3 parameters. Thus, it is possible that the time taken for the nonlinear optimization scheme to find a minimum of the error will be reduced. Second, the network obtained in this way is always in a state where the error is at a global minimum in the space of output weights. This may help the network to reach a minimum more rapidly and to reach a shallow local minimum less often. The approach can be characterized as a group-coordinate-wise descent on the error function, where the parameters are divided into two groups - output readout weights and the reservoir/input weights.

5.2.2 Readout regularization

If the training set contains noise, the network with an access of many free coefficients tends to generate mappings which have a lot of curvature and structure, as a result of over-fitting to the noise on the training data. In order to avoid over-fitting, we introduce a quadratic regularization term in the error function to encourage smoother network mapping, as follows:

$$E_r = \sum_{t=1}^{L} ||\mathbf{o}(t) - \mathbf{y}(t)||^2 + \lambda ||W||^2 \tag{5.12}$$

where $||\cdot||$ is the Frobenius norm. L_2 regularization is used here in order to be consistent with ridge regression mentioned in Section 2.2.3. The redundant weights of the network will get smaller as the training proceeds. Ideally, the structure of the network will be simplified while the accuracy remains. Thus, the generalization ability can be improved [67]. We only penalize the output weights W, since the input weights V and cycle connection weights R have already been pruned when the highly constrained reservoir is designed. Hence, addition of regularization term will not change the optimization of the reservoir weights $\mathbf{r}$. The output weights W will be computed as follows :

$$W = YZ^T(ZZ^T + \lambda I)^{-1} \tag{5.13}$$

which is known as ridge regression [92]. The regularization parameter λ can be tuned via cross-validation. The regularization will not change the partial derivative of E with respect to θ where $\theta = r_c$, r_j or r_i. More details are presented in Appendix C.2.

5.2.3 Early Stopping

Another way to improve the generalization ability is the procedure of early stopping [5,6]. The non-linear optimization process of learning the parameter $\mathbf{r}$ corresponds to an iterative reduction of the error function with respect to the training dataset. For many of the optimization algorithms, e.g. conjugate gradient descent, the value of the error function is a nonincreasing function of the iteration index. However, the error measured with respect to a dataset independent of the training, i.e. a validation set, often decreases first and then increases as the network starts to over-fit. Therefore, in order to have a good generalization ability, training needs to be stopped at the point of the smallest error with respect to the validation dataset, rather than the minimum point with respect to the training set.

We leave out a validation set. The training process is terminated when the error measured on the validation set starts to increase, in order to optimize the generalization performance of the network.

The overall structure of the training process is briefly demonstrated in Algorithm 2 .

Algorithm 2 The algorithm of the training process

```
1: Initialize $r_c, r_j$ and $r_i$.
2: repeat
3:     Generate CRJ model using $r_c, r_j$ and $r_i$ to obtain the state matrix $X$, and then
       compute the linear readout weight $W$ using Equation (5.13).
4:     Assess the performance of the network on the validation set $D_{validation}$ via the
       normalized mean square error $\tilde{E}(D_{validation})$.
5:     Update the reservoir parameters $r_c, r_j$ and $r_i$ using non-linear optimization algo-
       rithm, such as conjugate gradient descent.
6: until $\tilde{E}(D_{validation})$ starts to increase.
```

5.2.4 Experimental Studies

In this section, we evaluate our proposed algorithm on a variety of time series prediction tasks. The topology of the network was fixed, with 50 internal units and with the jump size of 15. For comparison, the original CRJ had the same network topology, but the cycle connection weight r_c, jump weight r_j and input weight r_i were chosen via cross-validation through exhaustive grid search and the linear readout weights were fitted using ridge regression. The range of reservoir weights are: r_c, r_j and $r_i \in$ {0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1}. Not every combination of the parameters leads to a network that satisfies the echo state property, but the optimum or selected parameters do. Standard randomized ESN with the same number of internal units was considered. The linear readout weights of ESN was also learned using ridge regression. The range of the ridge regression parameter is: $\lambda \in \{0.0001, 0.001, 0.01, 0.1, 1, 10, 100\}$. For ESN, we reported the average performance for 10 trials. Since, CRJ and TCRJ are deterministically methods, we only report one result[1].

Synthetic data We first tested our algorithm with a nonlinear system identification task, i. e. a 10th-order NARMA system [51], given by:

$$s(t+1) = 0.3s(t) + 0.05s(t)\sum_{i=0}^{9} s(t-i) + 1.5u(t-9)u(t) + 0.1,$$

where $s(t)$ is the output at time t, $u(t)$ is the input at time t. The inputs $u(t)$ form an i.i.d stream generated uniformly in the interval $[0, 0.5]$. The current output depends on both the input and the previous output. The time series is challenging due to non-linearity and long memory. To make the task harder, we added zero mean and 0.01 variance Gaussian noise to the output stream. The networks were trained to predict output $s(t)$ based on $u(t)$.The NARMA sequence has a length of 4000 items where first 2000 were used for training, the following 1000 for validation and the remaining 1000 for testing.

Then the nonlinear Mackey Glass chaotic time series model was used to generate a time series on which we evaluated the proposed algorithm. The series is a solution of the following equation:

$$\frac{dx(t)}{dt} = -ax(t) + \frac{bx(t-\tau)}{1 + x^{10}(t-\tau)} \tag{5.15}$$

where a, b and τ are the parameters of the equation. We used the Mackey-Glass series with parameters $a = 0.1$, $b = 0.2$, $\tau = 30$ and the initial condition $x(\tau) = 1$. A length of 4000 items of this time series was generated and zero mean and 0.05 variance Gaussian noise is added to it. As we did for NARMA sequence, the first 2000 items were used for training, the following 1000 for validation and the remaining 1000 for testing.

The experimental results on these two artificial time series are presented in Tables 5.1

[1] we use coarse grid search to initialize the reservoir parameters in TCRJ. The time for initialization is included when presenting the experiments

and 5.2. Table 5.1 shows that our proposed algorithm of learning the reservoir parameters in CRJ slightly outperforms the method of obtaining the reservoir parameters in CRJ through exhaustive grid search in terms of generalization error. As shown in Table 5.2, our proposed algorithm is much better than that of the original CRJ in terms of the computational time, decreasing by 70% to 80%.

Table 5.1: Normalized mean square error of the artificial time series

Dataset	ESN	CRJ	Learned CRJ
NARMA	0.2539 ± 0.0308	0.1019	**0.0981**
MG	0.0900 ± 0.0004	0.0900	**0.0884**

Table 5.2: Computational time (s) of the artificial time series

Dataset	ESN	CRJ	Learned CRJ
NARMA	279	4594	1632
MG	280	4648	962

Real-world Time Series Datasets Three real time series downloaded from the website [1] have been used to evaluate the proposed algorithm.

Darwin-SLP represents the monthly values of the Darwin sea level pressures from 1882 to 1998. This series is of length 1400 and a key indicator of climatological patterns. The first 1000 patterns were used for training, the following 200 patterns were used for validation and the remaining 200 patterns were used for test.

Oxygen Isotope Level (*OIL*) series contains measurements of relative abundance of oxygen isotope to oxygen from the deep ocean cores from various geographical locations over a period of about 2.5 million years, whose geological time variations relate to patterns of variation in global ice volume and ocean temperature [104]. This series contains 866 patterns. Given the limited size of the dataset, 5-fold cross validation were performed to tune the parameters, instead of "train-validation-test" approach. The first 600 patterns

[1] `http://isds.duke.edu/~mw/ts˙data˙sets.html` (12/11/2014)

were used for training and the rest 266 patterns were used for test.

SOI is a series of monthly values of the Southern Oscillation Index during 1950-1995. This series consists of 540 observations on the SOI computed as the difference of the departure from the long-term monthly mean sea level pressures at Tahiti in the South Pacific and Darwin in Northern Australia. As we did for *OIL*, 5-fold cross validation were used to tune the parameters. The first 400 observations were used for training and the remaining 140 observations were used for test.

The experimental results on these three real time series are presented in Tables 5.3 and 5.4. As shown in Table 5.3, the proposed methodology appears to have comparable generalization ability to the original CRJ model where the reservoir parameters was obtaining through an exhaustive grid search method. The computational time of the proposed algorithm is much less than that of the original CRJ model.

Table 5.3: Normalized mean square error of the *Darwin-SLP*, *OIL* and *SOI* time series

Dataset	ESN	CRJ	TCRJ
Darwin-SLP	0.2467(0.0174)	0.1780	0.1843
OIL	0.2214(0.0008)	0.2094	0.2076
SOI	0.5613(0.0039)	0.5374	0.5374

Table 5.4: Computational time (s) of the *Darwin-SLP*, *OIL* and *SOI* time series

Dataset	ESN	CRJ	learned CRJ
Darwin-SLP	230	3539	726
OIL	167	2470	781
SOI	122	1622	381

Three datasets from UCR Time Series Repository [53] were used for the evaluation of our proposed algorithm. These datasets are briefly described in Table 5.5. They are mainly used for time series classification. Here we use the sequences to demonstrate predictive power of our models, so each sequence is divided into training, validation and test part as indicated in Table 5.5. The results presented in Tables 5.6 and 5.7 are the average results over the test parts of the sequences considered. The experimental results on the UCR time series presented in Tables 5.6 and 5.7 shows similar trends as we find before.

Table 5.5: Description of the UCR time series. L is the length of each sequences in the datasets and n is number of sequences in the datasets.

Datasets	L	n	Train/Validation/Test
CinC-ECG-torso	1639	40	1000/300/339
InlineSkate	1882	100	1000/400/482
MALLAT	1024	55	600/200/224

Table 5.6: Normalized mean square error of the UCR time series

Dataset	ESN	CRJ	learned CRJ
CinC-ECG-torso	0.0060(0.0154)	0.0054(0.0149)	0.0054(0.0150)
InlineSkate	0.0310(0.0695)	0.0104(0.0313)	**0.0056**(0.0134)
MALLAT	0.0175(0.0138)	0.0152(0.0166)	**0.0117**(0.0147)

Table 5.7: Computational time (s) of the UCR time series

Dataset	ESN	CRJ	learned CRJ
CinC-ECG-torso	155(1.51)	2488(28.14)	398(117.78)
InlineSkate	150(5.10)	2237(11.37)	372(81.56)
MALLAT	165(1.92)	2379(142.64)	353(23.18)

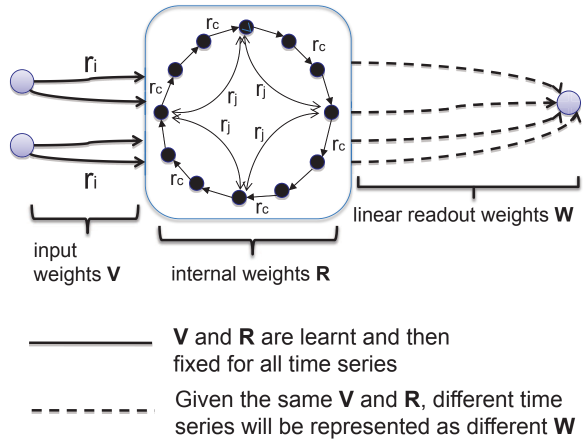
In this section, two synthetic time series, three real time series and three UCR time series datasets were used to test our algorithm compared with the original exhaustive search CRJ model and standard randomized ESN model. All the experimental results show the same trend that learning the parameters in CRJ model has comparable performance (sometimes even better) to choosing the parameters via exhaustive search in terms of generalization ability, and also learning the parameters is much more computational efficient.

The proposed algorithm does not jeopardize the generalization performance on all the datasets used in this section with the exception of *Darwin-SLP* time series. It may be because the error surface of this time series contains many shallow local minimums and the learning algorithm was stuck in one of the very bad local minimum, while the exhaustive search method reach the relative global minimum (global minimum on the discretized parameter space). One possible solution of this problem in the proposed approach is to add a small random value to the obtained R each step, hoping to jump out of the local minimal. However, the generalization performance of our proposed algorithm on *Darwin-SLP* time series is still much better than that of standard randomized ESN model.

The proposed algorithm tremendously reduces the computational time in the experiments, compared with the original exhaustive search CRJ model. The proposed hybrid optimization strategy keeps the computational advantage of ESN in terms of efficient computation of linear readout weights by the separation of the optimization of linear readout weights from the weights govern the structure of the reservoir. The readout weights are computed in linear techniques and as a result, the error of the network is always at a global minimum in the space of output weights, helping the network to reach a minimum fast. Since the output weights are solved using linear algorithm, the parameters left for non-linear optimization are reduced. The reduction of the search space for the non-linear optimization methods will lead to less number of iteration to terminate the training. Besides, the early stopping strategy potentially helps to reduce the computational time and improve the generalization performance.

5.3 Metric-Model Co-learning

In this section, the MMCL approach for time series classification is presented. The same as previous chapter, a high-dimensional parameterized non-linear state space model rooted in ESN with a simple deterministically constructed dynamic coupling structure [76] is used as the core model class for sequence representation. Still, each sequence is represented



r_i
r_i
r_c
r_j
input weights **V**
internal weights **R**
linear readout weights **W**
V and **R** are learnt and then fixed for all time series
Given the same **V** and **R**, different time series will be represented as different **W**

Figure 5.1: Illustration of the parameterized state space models used in this work. The parameterized state space model has a fixed topology, i.e. a uni-directional cycle with bi-directional jumps. There are only three parameters in this model: the cyclic connection weights $r_c > 0$; the jumps weight $r_j > 0$; and the input weight $r_i > 0$. The input weight matrix V only contains r_i, and the state transition matrix R contains r_c and r_j.

by the corresponding linear readout mapping acting on top of the common shared non-linear dynamical system. However, in this section, the difference between two sequences is measured through the distance of their linear readout weights weighted by a metric tensor. The shared dynamical system, as well as the metric tensor in the readout space are learned simultaneously.

The N-dimensional dynamical model we employ has the following form:

$$\boldsymbol{z}(t) = \tanh(R\,\boldsymbol{z}(t-1) + V\,\boldsymbol{s}(t)), \tag{5.16}$$

$$\boldsymbol{o}(t) = W\boldsymbol{z}(t) \tag{5.17}$$

where $\boldsymbol{z}(t) \in \mathbb{R}^N$, $\boldsymbol{s}(t) \in \mathbb{R}^M$ and $\boldsymbol{o}(t) \in \mathbb{R}^M$ are the state vector, input vector and output at time t, respectively; R is a $(N \times N)$ dynamic coupling matrix; $V \in \mathbb{R}^{N\times M}$and $W \in \mathbb{R}^{M\times N}$ are the input and output weight matrices[1], respectively. We will refer to (5.16) as the state transition mapping (STM).

[1]As usual, we add the state vector $\boldsymbol{z}(t)$ with a constant element (e.g. 1) and W with a column to account for the bias term. However, in order to simplify the presentation, we omit the bias term.

Provided the non-autonomous dynamics (5.16) is able to represent a rich set of features of the given time series, a particular time series $\boldsymbol{s}$ can be represented by the linear readout mapping parameterized by W (eq. (5.17)) operating on reservoir activations $\boldsymbol{z}$, specifically fitted to $\boldsymbol{s}$ on the next-item prediction task $\boldsymbol{o}(t) \sim \boldsymbol{s}(t+1)$ by minimizing the mean squared error (MSE) between the model predictions $\boldsymbol{o}(t)$ and targets $\boldsymbol{s}(t+1)$. The non-linear state space model used in this paper is illustrated in Figure 5.1.

In the original CRJ, the weights r_c, r_j, r_i were determined by cross-validation using grid search, which is computationally extensive in practice. In this contribution, given a sequence classification task, we aim to learn these parameters (common for the whole data set) so that **(1)** a faithful modeling of individual sequences is achieved (by allowing individualized readout mappings from the common state transition mapping), **(2)** the sequence classes in the space of readout models are well separated. **(1)** reflects the model representability of the proposed approach, i.e. the readout mappings should represent each sequence well, while **(2)** demonstrates the class separability of the approach, i.e the distance function in the readout model space should separate each class well. We will learn the weights (r_c, r_j, r_i) using real time recurrent learning as Section 5.2 and metric tensor[1] on an appropriate cost functional reflecting **(1)** and **(2)** above.

5.3.1 Cost Functional

We vectorize[2] parameters W of the readout mapping given by equation (5.17) into the parameter vector $\boldsymbol{w}$. Assume we are given a set of n labelled sequences $\{\boldsymbol{s}_m, y_m\}_{m=1}^{n}$. Using the fixed common state transition mapping (eq. (5.16)), each sequence $\boldsymbol{s}_m$ will be represented by the linear readout mapping parameterized by $\boldsymbol{w}_m$. The readout is trained on the next item prediction on $\boldsymbol{s}_m$ via ridge regression. The readout weights $\boldsymbol{w}_m$ are

[1]This can be viewed as metric learning [107] in the readout model space.

[2]For multivariate time series with dimensionality M, the linear readout weight matrix W is a $M \times N$ matrix. In this case, we vectorize the matrix W into a column vector $\boldsymbol{w} = [w_{11}, ..., w_{M1}, ..., w_{1N}, ..., w_{MN}]^T$.

thus determined by the model parameters $\boldsymbol{r} = (r_i, r_c, r_j)$, as well as the sequence $\boldsymbol{s}_m$. To simplify the notation, in the following this dependence[1] will not be made explicitly.

Representability:

A good non-linear state space model parameterized by $\boldsymbol{r} = (r_i, r_c, r_j)$ should minimize the representability cost with respect to all (i.e. n) training sequences:

$$Q_p(\boldsymbol{r}) = \sum_{m=1}^{n} \left(\sum_{t=1}^{L_m - 1} ||\boldsymbol{o}_m(t) - \boldsymbol{s}_m(t+1)||^2 + \lambda ||W_m||^2 \right), \tag{5.18}$$

where L_m is the length of sequence $\boldsymbol{s}_m$, and $\boldsymbol{o}_m$ is the output mapping (eq. (5.17)) determined by $\boldsymbol{w}_m$ (vectorized W_m). This equation aims to minimize the difference between actual output $\boldsymbol{o}(t)$ and desired output $\boldsymbol{s}(t+1)$

Separability:

We aim to learn a global metric A such that sequences from the same class are close to each other in the readout model space and sequences in different classes are far away in the readout model space. Given a training sequence $\boldsymbol{s}_m$, following [40], we introduce a conditional distribution over training sequence indexes $q \neq m$:

$$p_A(q|m) = \frac{e^{-d_A(m,q)}}{\sum_{k \neq m} e^{-d_A(m,k)}}, m \neq q,$$

where

$$d_A(m, q) = (\boldsymbol{w}_m - \boldsymbol{w}_q)^T A (\boldsymbol{w}_m - \boldsymbol{w}_q) \tag{5.19}$$

is the squared distance between readout parameter $\boldsymbol{w}_m$ and $\boldsymbol{w}_q$ under the (global) positive semi-definite metric A. If all readouts in the same class were mapped to a single point and

[1]The readout weights $\boldsymbol{w}_m$ is a function of the model parameters $\boldsymbol{r}$ and the sequence $\boldsymbol{s}_m$ as the variables.

infinitely far from points in different classes, we would have the ideal distribution [40]:

$$p_0(q|m) \propto \begin{cases} 1 & \text{if } y_m = y_q \\ 0 & \text{if } y_m \neq y_q \end{cases} .$$

We optimize the non-linear state space model (eq. (5.16)) and the metric A such that $p_A(q|m)$ is as close as possible to $p_0(q|m)$ with respect to KL divergence:

$$\min_A \sum_{m=1}^{n} D_{KL}\left[p_0(q|m) \| p_A(q|m)\right],$$
$$s.t. A \text{ is positive semi-definite.}$$

where $D_{KL}(p||q) = \sum_i p(i) \log \frac{p(i)}{q(i)}$.

$$\begin{aligned}\sum_{m=1}^{n} D_{KL}\left[p_0(q|m) \| p_A(q|m)\right] &= \sum_{m=1}^{n}\sum_{q=1}^{n} p_0(q|m) \log \frac{p_0(q|m)}{p_A(q|m)} \\ = \sum_{m=1}^{n}\sum_{q=1}^{n} p_0(q|m) \log p_0(q|m) &- \sum_{m=1}^{n}\sum_{q=1}^{n} p_0(q|m) \log p_A(q|m) \qquad (5.20)\end{aligned}$$

The first term in (5.20) is constant. Since $p_0(q|m) = 0$ when $y_q \neq y_m$ and $\sum_{q=1}^{n} p_0(q|m) = 1$ (the property of probability), the second term in (5.20) can be rewritten as follows:

$$\begin{aligned}&-\sum_{m=1}^{n}\sum_{q=1}^{n} p_0(q|m) \log p_A(q|m) = -\sum_{m=1}^{n}\sum_{q=1}^{n} p_0(q|m) \log \frac{e^{-d_A(m,q)}}{\sum_{k\neq m} e^{-d_A(m,k)}} \\ &= -\sum_{m=1}^{n}\sum_{q=1}^{n} p_0(q|m) \left(-d_A(m,q) - \log \sum_{k\neq m} e^{-d_A(m,k)}\right) \\ &= \sum_{m=1}^{n}\sum_{q=1}^{n} p_0(q|m) d_A(m,q) + \sum_{m=1}^{n}\sum_{q=1}^{n} p_0(q|m) \log \sum_{k\neq m} e^{-d_A(m,k)} \\ &= \sum_{\{(m,q):y_m=y_q\}} d_A(m,q) + \sum_{m=1}^{n} \log \sum_{k\neq m} e^{-d_A(m,k)} \sum_{q=1}^{n} p_0(q|m) \\ &= \sum_{\{(m,q):y_m=y_q\}} d_A(m,q) + \sum_{m=1}^{n} \log \sum_{k\neq m} e^{-d_A(m,k)}\end{aligned}$$

This results in a separability cost[1] $Q_s(\boldsymbol{r}, A)$ equal to

$$\sum_{\{(m,q):y_q=y_m\}} d_A(m,q) + \sum_{m=1}^{n} \log \sum_{k \neq m} e^{-d_A(m,k)}. \tag{5.21}$$

Overall cost functional:

Using the representation (eq. (5.18)) and separation costs (eq. (5.21)), respectively, we finally construct the cost functional to be minimized by the state space model with parameters $\boldsymbol{r}$ and metric A:

$$Q(\boldsymbol{r}, A) = Q_s(\boldsymbol{r}, A) + \eta Q_p(\boldsymbol{r}) \tag{5.22}$$

where parameter $\eta > 0$ controls the tradeoff between the *representability* Q_p and the *separability* Q_s. In practice we first minimize $Q(\boldsymbol{r}, A)$ (via gradient descent) and then project the resulting A onto the space of positive semi-definite matrices using eignvalue decomposition [40].

Gradients of Q with respect to STM parameters $\boldsymbol{r}$ and metric tensor A read (θ stands for r_i, r_c or r_j):

$$\frac{\partial Q}{\partial \theta} = \frac{\partial Q_s}{\partial \theta} + \eta \frac{\partial Q_p}{\partial \theta} \tag{5.23}$$

[1]assuming equally probable classes and ignoring constant terms

where

$$\begin{aligned}
\frac{\partial Q_s}{\partial \theta} &= \sum_{\{(m,q):y_q=y_m\}} (\boldsymbol{w}_m - \boldsymbol{w}_q)^T (A + A^T) \left(\frac{\partial \boldsymbol{w}_m}{\partial \theta} - \frac{\partial \boldsymbol{w}_q}{\partial \theta} \right) \qquad (5.24) \\
&\quad - \sum_{m=1}^{n} \frac{\sum_{q \neq m} e^{-(\boldsymbol{w}_m - \boldsymbol{w}_q)^T A (\boldsymbol{w}_m - \boldsymbol{w}_q)} (\boldsymbol{w}_m - \boldsymbol{w}_q)^T (A + A^T) \left(\frac{\partial \boldsymbol{w}_m}{\partial \theta} - \frac{\partial \boldsymbol{w}_q}{\partial \theta} \right)}{\sum_{k \neq m} e^{-(\boldsymbol{w}_m - \boldsymbol{w}_k)^T A (\boldsymbol{w}_m - \boldsymbol{w}_k)}} \\
&= \sum_{m,q}^{n} (p_0(q|m) - p_A(q|m)) (\boldsymbol{w}_m - \boldsymbol{w}_q)^T \left(A + A^T \right) \left(\frac{\partial \boldsymbol{w}_m}{\partial \theta} - \frac{\partial \boldsymbol{w}_q}{\partial \theta} \right) \\
& \qquad (5.25)
\end{aligned}$$

and according to previous section (more details are presented in Appendix C.2):

$$\frac{\partial Q_p}{\partial \theta} = \sum_{m=1}^{n} \sum_{t=1}^{L_m - 1} 2 (\boldsymbol{o}_m(t) - \boldsymbol{s}_m(t+1))^T W_m \frac{\partial \boldsymbol{z}(t)}{\partial \theta} \qquad (5.26)$$

$$\begin{aligned}
\frac{\partial Q}{\partial A} &= \frac{\partial Q_s}{\partial A} \qquad (5.27) \\
&= \sum_{\{(m,q):y_q=y_m\}} (\boldsymbol{w}_m - \boldsymbol{w}_q) (\boldsymbol{w}_m - \boldsymbol{w}_q)^T \qquad (5.28) \\
&\quad + \sum_{m=1}^{n} \frac{\sum_{q \neq m} e^{-(\boldsymbol{w}_m - \boldsymbol{w}_q)^T A (\boldsymbol{w}_m - \boldsymbol{w}_q)} (\boldsymbol{w}_m - \boldsymbol{w}_q) (\boldsymbol{w}_m - \boldsymbol{w}_q)^T}{\sum_{k \neq m} e^{-(\boldsymbol{w}_m - \boldsymbol{w}_k)^T A (\boldsymbol{w}_m - \boldsymbol{w}_k)}} \\
&= \sum_{m,q}^{n} (p_0(q|m) - p_A(q|m)) (\boldsymbol{w}_q - \boldsymbol{w}_m) (\boldsymbol{w}_q - \boldsymbol{w}_m)^T . \qquad (5.29)
\end{aligned}$$

According to the previous section, the readout parameters W_m representing sequence $\boldsymbol{s}_m$ are obtained as follows:

$$W_m = Y_m \; Z_m^T (Z_m \; Z_m^T + \lambda I)^{-1}.$$

where $Z_m = [\boldsymbol{z}_m(1), \cdots, \boldsymbol{z}_m(L_m - 1)]$ is the $(N \times (L_m - 1))$ STM state matrix storing state activations obtained while processing $\boldsymbol{s}_m$ as columns, $Y_m = [\boldsymbol{s}_m(2), \cdots, \boldsymbol{s}_m(L_m)]$ is the target matrix for the next-item prediction task and $\eta > 0$ is a regularization parameter. To ease the presentation, we omit the sequence index m in the following derivations. We have

$$\begin{aligned}
\frac{\partial W}{\partial \theta} &= Y\frac{\partial Z^T}{\partial \theta}(ZZ^T+\lambda I)^{-1} + YZ^T\frac{\partial (ZZ^T+\lambda I)^{-1}}{\partial \theta} \\
&= Y\left(\frac{\partial Z}{\partial \theta}\right)^T(ZZ^T+\lambda I)^{-1} - YZ^T(ZZ^T+\lambda I)^{-1}\frac{\partial (ZZ^T+\lambda I)}{\partial \theta}(ZZ^T+\lambda I)^{-1} \\
&= Y\left(\frac{\partial Z}{\partial \theta}\right)^T(ZZ^T+\lambda I)^{-1} \\
&\quad -YZ^T(ZZ^T+\lambda I)^{-1}\left(\frac{\partial Z}{\partial \theta}Z^T + Z\left(\frac{\partial Z}{\partial \theta}\right)^T\right)(ZZ^T+\lambda I)^{-1}
\end{aligned}$$

where

$$\frac{\partial Z}{\partial \theta} = \left[\frac{\partial \boldsymbol{z}(1)}{\partial \theta}, \frac{\partial \boldsymbol{z}(2)}{\partial \theta}, ..., \frac{\partial \boldsymbol{z}(L)}{\partial \theta}\right] \tag{5.30}$$

and $\frac{\partial \boldsymbol{z}(t)}{\partial \theta}$ is given by (5.7).

$$\frac{\partial \boldsymbol{w}}{\partial \theta} = \mathrm{vec}\left(\frac{\partial W}{\partial \theta}\right) \tag{5.31}$$

The model-metric co-learning can be achieved by alternating between performing state space model learning (eq. (5.23)) and metric learning in the readout model space (eq. (5.29)) to obtain a tradeoff between *representability* and *separability*[1]. This strategy is adopted for the reason that the two group of parameters representing different concepts. It is not optimal but a reasonable appropriation to the optimal.

[1]The learning process is implemented using *minimize* function provided by the website: http://www.di.ens.fr/~mschmidt/Software/minFunc.html(12/11/2014)

5.3.2 Sequence Classification

Having determined the appropriate STM and global metric tensor on the readout parameters, we can use any convenient distance-based classifier, e.g. kNearest Neighbour (kNN). The (squared) distance between two sequences $\boldsymbol{s}_m$ and $\boldsymbol{s}_q$ is simply $d_A(m,q)$ in eq. (5.19) , where, as explained above, the feature vector $\boldsymbol{w}_m$ of a sequence $\boldsymbol{s}_m$ is obtained by ridge regression from the dynamical system generated using the learned parameters r_i, r_c and r_j with the task of predicting the next item of the sequence. Of course, one can also adopt a kernel classification framework (e.g. SVM) using a sequence kernel

$$\mathcal{K}(\boldsymbol{s}_m, \boldsymbol{s}_q) = \exp\{-\gamma\ d_A(m,q)\}\,, \tag{5.32}$$

where $\gamma > 0$ is a scale parameter. In our experiments we employ both kNN operating in the readout space and SVM with the kernel defined in (5.32).

5.3.3 Experimental Studies

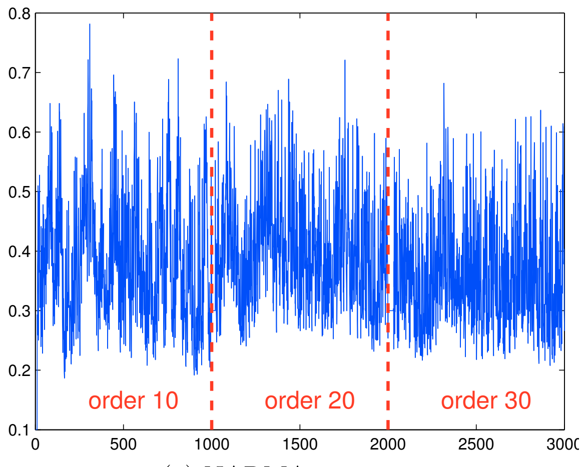
Table 5.8: Parameter search ranges for methods employed in the experiments. p is the order of the vector autoregressive model, ξ is an AR kernel parameter [26], $\#states$ is the number of states for HMM in *Fisher* kernel, λ is the ridge regression regularization parameter, η is the trade-off parameter. r_i, r_c and r_j are parameters of the deterministic state space mode used in *RV* kernel.

	Parameters	Parameter range
AR	ξ, p	$\xi \in \{0.1, 0.2, \cdots, 0.9\}$, $p \in \{1, 2, \cdots, 10\}$
Fisher	$\#states$	$\#states \in \{1, 2, \cdots, 10\}$
RV	r_i,r_c,r_j, λ	r_i,r_c,$r_j \in \{0.01, 0.05, \cdots, 1\}$ $\lambda \in \{10^{-5}, 10^{-4}, \cdots, 10^{1}\}$
MMCL	λ, η	$\lambda \in \{10^{-5}, 10^{-4}, \cdots, 10^{1}\}$ $\eta \in \{0, 10^{-1}, \cdots, 10\}$

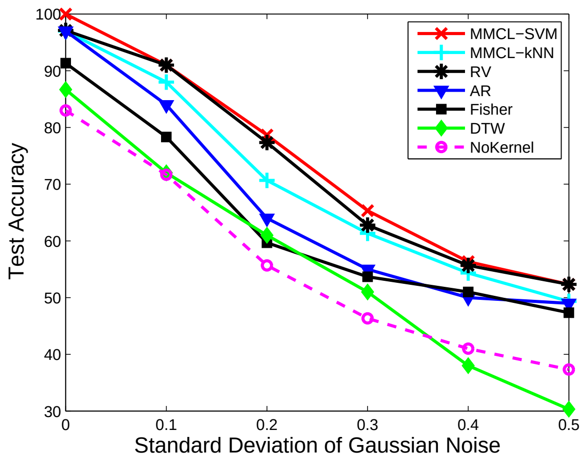
We compare our MMCL framework with several state-of-the-art methods for sequence classification such as *dynamic time warping (DTW)* based kernel [4], *autoregressive kernel (AR)* [26], the *Fisher kernel (Fisher)* [47] and the *reservoir kernel (RV)*. The RV kernel can be viewed essentially as the MMCL kernel, expect for the dynamic parameters r_i, r_c, r_j are simply set by costly cross-validation without any regard for class separability and metric tensor learning. The *AR*, *Fisher*, *RV* and *DTW* based kernels were used in a SVM classifier.

In MMCL, the dynamic state space topology was fixed to $N = 50$ and 10 jumps (making the jump length 5). The jump length could be set by cross-validation as in [76], but to demonstrate the power of the framework, we simply fixed the topology for all experiments without pre-testing.

All (hyper) parameters, such as the MMCL trade-off parameter λ, order p in the *AR* kernel, number of hidden states in the HMM based *Fisher* kernel, regularization parameter η for ridge regression etc. have been set by 5-fold cross-validation on the training set. We employ a well-known, widely accepted and used implementation of SVM – LIBSVM [19]. In LIBSVM, we use cross validation to tune the slack-weight regularization parameter C. After model selection using cross-validation on the training set, the selected model class representatives were retrained on the whole training set and were evaluated on the test set. Multi class classification is performed via the one-against-one strategy (default in LIBSVM). The SVM parameters, kernel width γ in eq. (5.32) and C, were tuned in the following ranges: $\gamma \in \{10^{-6}, 10^{-5}, \cdots, 10^{1}\}$, $C \in \{10^{-3}, 10^{-2}, \cdots, 10^{3}\}$. We also tested our MMCL method using a kNN classifier where $k \in \{1, 2, \cdots, 10\}$. We refer to SVM and kNN classifiers built within the MMCL framework as *MMCL-SVM* and *MMCL-kNN*, respectively. The search ranges for the parameters are detailed in Table 5.8.



(a) NARMA sequence



(b) Classification accuracy

Figure 5.2: (a) is the illustration of three NARMA sequences with different orders (10, 20 and 30). (b) describes the classification accuracies on the synthetic (test) data set.

Synthetic Data We employed three NARMA time series models of orders 10, 20 and 30, given by:

$$\begin{aligned} s(t+1) &= 0.3s(t) + 0.05s(t)\sum_{i=0}^{9} s(t-i) \\ &\quad +1.5u(t-9)u(t) + 0.1, \\ s(t+1) &= \tanh(0.3s(t) + 0.05s(t)\sum_{i=0}^{19} s(t-i) \\ &\quad +1.5u(t-19)u(t) + 0.01) + 0.2, \\ s(t+1) &= 0.2s(t) + 0.004s(t)\sum_{i=0}^{29} s(t-i) \\ &\quad +1.5u(t-29)u(t) + 0.201, \end{aligned}$$

where $s(t)$ is the output at time t, $u(t)$ is the input at time t. The inputs $u(t)$ form an i.i.d stream generated uniformly in the interval $[0, 0.5)$. We use the same input stream for generating the three long NARMA time series (60,000 items), one for each order. The time series are challenging due to non-linearity and long memory (see Figure 5.2).

For each order, the series of 60,000 numbers is partitioned into 200 non-overlapping time series of length 300. The first 100 time series for each order are used as a training set, and the other 100 time series form the test set. In order to study robustness of the kernels we also corrupt the time series with additive Gaussian noise (zero mean, standard derivation varies in [0.1,0.5]). Figure 5.2 shows the test set classification accuracy against the noise level. As a baseline, we also include results of SVM directly operating on the time series (300-dimensional inputs) - *NoKernel*. The *MMCL* based kernel method *MMCL-SVM* outperforms the baseline as well as all the other methods.

Univariate Benchmark Data We used 7 data sets from the UCR Time Series Repository [53]. Each data set has already been split into training and test sets (see Table 5.9). Table 5.9 reports performance of the studied methodologies on the benchmark data in terms of test set classification accuracy. The *MMCL* methods outperform the other algorithms on most of the data sets, except for being inferior to or on a par with *AR* kernel and *RV* kernel on Lightning2 and Coffee data sets, respectively, *MMCL-SVM* is superior to the simpler *MMCL-kNN* on 5 data sets.

Table 5.9: Description of the data sets (left) and the performances (right). The best performance (test set classification accuracy) for each data set is boldfaced. Note that in 4 out of the 7 datasets (OSULeaf,Oliveoil, Fish and Adiac) both MMCL techniques outperform others.

Dataset	Length	Classes	# Train	# Test	*DTW*	*AR*	*Fisher*	*RV*	*MMCL-kNN*	*MMCL-SVM*
OSULeaf	427	6	200	242	74.79	56.61	54.96	69.83	**88.02**	85.12
Oliveoil	570	4	30	30	83.33	73.33	56.67	86.67	86.67	**93.33**
Lightning2	637	2	60	61	64.10	**77.05**	64.10	**77.05**	70.49	75.41
Beef	470	6	30	30	66.67	78.69	58.00	80.00	63.33	**82.67**
Fish	463	8	175	175	69.86	60.61	57.14	79.00	**88.57**	87.43
Coffee	286	2	28	28	85.71	**100.00**	81.43	**100.00**	89.29	**100.00**
Adiac	176	37	390	391	65.47	64.45	68.03	72.63	72.30	**73.40**

As shown in Table 5.9, our methodology was slightly inferior to other algorithms on some datasets, e.g. Lightning2. The Lightning2 dataset contains time series of power spectra related lightning. The classes can be characterized by pronounced low-memory features of the power spectra series such as those corresponding to sharp turn-on of radiation, gradual increase in power series, or sudden spikes etc. Such time series features can be naturally captured by AR models and thus we hypothesize that for this particular data set, the AR kernels provide a strong distinguishing platform for lightning classification.

Multivariate Time Series The data sets used so far involved univariate time series. In this section, we perform classification on three multivariate time series - Brazilian sign language (*Libras*), *handwritten* characters and Australian language of signs (*AUSLAN*). Unlike the other data sets, the *handwritten* characters and *AUSLAN* data sets contain time series of variable length. Following [26] (previous *AR* kernel study) we split the data into training and test sets as detailed in Table 5.10.

The results are shown in Table 5.10. The *MMCL* based *MMCL-SVM* is superior on two data sets with slightly worse, but comparable performance to *Fisher* kernel on the Libras dataset.

Table 5.10: Summary of multivariate (variable length) time series classification problems (left) and the performances (right). The best performance (test set classification accuracy) for each data set is boldfaced. For the handwritten and AUSLAN datasets both MMCL variants outperform the other techniques.

Dataset	dim	length	classes	# train	# test	*DTW*	*AR*	*Fisher*	*RV*	*MMCL*-kNN	*MMCL*-SVM
Libras	2	45	15	360	585	94.02	91.79	**94.93**	93.25	94.19	94.87
handwritten	3	60-182	20	600	2258	88.67	73.30	87.52	89.41	91.31	**91.41**
AUSLAN	22	45-136	95	600	1865	97.00	76.53	94.74	96.00	97.05	**97.80**

In order to investigate MCML further, we performed model learning only (without metric learning) and (as expected) observed similar performance levels to those of RV (model parameters optimized by cross validation). We have also decoupled the model and metric learning. This naturally resulted in an inferior performance. Independent learning might ignore possible coupling between the provider of dynamical features (dynamical system) and the static readout models.

In terms of the learned parameters (r_i, r_c, r_j), we have checked their values on our datasets with and without metric learning. Interestingly enough, in general, metric learning lead to increased r_i and r_j. Cycle weights r_c differed by a comparatively smaller amount. The increased input loads r_i force the dynamical system to operate in saturation regimes of the tanh function (see eq. (5.16)). This can increase class separability of state space representations of sequences from different classes. Large jump weight r_j in effect broadens frequency spectrum of the non-autonomous dynamical system. While a cyclic coupling only [75] can capture slow dynamical regimes, introduction of jumps leads to broadening the sensitivity of the system to faster changes in the input stream.

5.4 Discussion and Conclusion

This chapter focuses on the *learning in the model space* approach where each data item (sequence) is represented by the corresponding model from a carefully designed model class retaining both reasonable flexibility to represent a wide variety of sequences and a relatively small number of free parameters. The model class used is a special form of state space model. The dynamic part of the model - state transition mapping - has only three free parameters that are learnt and then fixed for each data set. Individual sequences in the data set are then represented by the corresponding (static) linear readouts from the state space (fixed STM).

Given a particular data set, we learn both the appropriate state space model and metric in the readout model space. Since we consider sequence classification, learning is guided by two criteria: *representability* and *class separability* of the readout models.

On 11 sequence classification data sets the sequence kernels constructed using our methodology (and applied in SVM) were on a par or outperformed the other four studied methods - DTW based kernels, autoregressive kernel, the Fisher kernel based on hidden Markov model with Gaussian emissions, and the reservoir kernel with *manually* chosen state transition parameters by cross validation using grid search and *without* metric learning in the readout space.

Since in our framework each sequence is represented as a model in the readout model space with a learnt global metric tensor, any distance based classifier can be used. As a baseline we employed kNN. Except for the Coffee, Lightning2 and *Libras* data sets, kNN achieved comparable or better performance than the four alternatives.

Our results may seem surprising, especially given the rather simple structure of the state transition part. We emphasize that the STM employed does not necessarily have to provide a platform for the best sequence modelling. For that, the imposition of the same fixed STM for the whole data set may indeed be too restrictive. However, the task at hand is sequence classification and the requirements on the STM are of a different nature: allowing state space processing of sequences so that the corresponding linear readouts coincide as much as possible for sequences of the same class, while keeping the separation of the distinct classes. This, as demonstrated in our experiments, is indeed possible using a fixed STM. The optimal STM for the classification purposes is learnt alongside the metric tensor on linear readouts from the filter that enhances the within-class collapsed representations and between-class separation.

The superior performance of MMCL methodology comes at a price - relatively high computational complexity. In each iteration of gradient descent in MMCL, the complexity is $O(n^2M^2 + nML)$, where n is the number of sequences, M is the dimensionality of the time series, and L is the length of sequences (detailed information is given by C.3). However, this tolerable computational cost is well offset by the high classification accuracy and the saving of effort for tuning parameters in the state space model using cross validation with grid search.

CHAPTER 6

Conclusion and Future Work

This chapter summarizes the work presented in this thesis and discusses the potential future work.

6.1 Summary

How to utilize all available information within data during training to improve generalization performance of a learning algorithm is one of the main research questions in machine learning. This thesis presented different concepts of using available information within data to improve the generalization performance of kernel methods, particularly SVMs, in the application domain of time series data. The first contribution in this thesis utilized privileged information of the training examples, unavailable in the test regime, to improve generalization performance in SVM for ordinal regression via smooth modelling of slack variables. The second contribution constructed an efficient time series kernel based on reservoir model, capable of capturing the temporal dependence within the time series, to improve the generalization performance of the kernel machine. Moreover, we presented yet another novel model based kernel similarity by co-learning the dynamic reservoir and a global metric in the linear readout space, leading to improved generalization performance.

In many time series prediction problems, the task is to predict ordered categories of movements instead of the real values. In this context, ordinal regression which explicitly utilizes the ordering information exists in class labels can be applied. Within the ordinal regression framework, we proposed to exploit additional privileged information, which is only available during training stage but unavailable for test, to improve the generalization performance. Naturally, future events in the time series present in the training stage but unavailable in the test phase can form privileged information. We implemented the proposed algorithm by incorporating the LUPI framework [101] into SVORIM [24]. SVORIM exploits the order structure in class labels by constructing multiple parallel separating hyperplanes defined through ordered thresholds. The proposed algorithm utilizes privileged information by constructing correcting functions for each separating hyperplane. Consequently, the slack variables are computed by privileged information through correcting functions in our algorithm, opposing to original SVORIM where slack variables are obtained during the optimization procedure. Our work confirms the success of the LUPI framework and poses questions on the effect of smooth modelling of slack variables in SVM framework.

In time series classification, where the task is to predict a single label that applies to the whole sequence, we developed two novel methodologies of constructing time series kernels. We introduced the idea of constructing kernels in the deterministically constructed reservoir model space, instead of directly in the original data space. Each time series is represented as a reservoir model with a common dynamic part. The shared dynamic reservoir acts as a temporal filter providing a rich pool of dynamic features that can be selectively extracted by linear readout mappings. The linear readout model works as the final representation of the time series but the full underlying model is a non-linear dynamic system. Thus, our methodologies reduce the computational demands without the loss of the computational ability of non-linear models. Then, we defined our kernel similarity in the linear readout model space.

In our first kernel design, the distance between two time series is computed as the model distance between their corresponding linear readout mappings. The model distance is different from the Euclidean distance of the readout parameters, indicating that more importance is given to the "offset" than "orientation" of the readout mapping. We also estimated the model distance when the reservoir state distribution is non-uniform by approximating the probability density distribution of the states using a Gaussian mixture model. We also considered sampling methods in the case of non-uniformly distributed reservoir states. Moreover, We constructed Fisher kernel based on reservoir model. Furthermore, we realized a fast version of our kernel by training the readouts in on-line manner reservoir, in order to process extremely long time series efficiently.

The numerical experiment has been conducted to compare the proposed kernels with other competitors on synthetic and benchmark data sets. The results confirmed the efficiency of our proposed kernels. These kernels can achieve superior performance in terms of both generalization accuracy and computational time. Our proposed kernels favour very large data sets of long time series where the underlying parametric model is unknown. However, if the example time series in the data sets is very short, i.e. the length of the time series is less than 10, our kernels may fail to provide a good similarity measure for the time series. Certain number of observations of the time series are required to obtain a good model representation. However, this is not a problem for Sampling kernel since bootstrap techniques was used to stretch the time series. Similar trick can be used to avoid this problem for other kernels.

In our second kernel design, we proposed to learn both the shared dynamical reservoir and a global metric tensor in the linear readout space during the training process. Learning is guided by two criteria: representability and separability. Representability requires the model representation of the time series to faithfully represent the each time series while separability demands the sequences from the same class to have close model representations, while model representations of sequences from different classes to be well-separated. The two criteria are fulfilled simultaneously through optimizing an objective function of their weighted sum. This method treats the model parameters adaptation and model distance designing jointly rather than adapting the model parameters first and defining the model distance afterwards in two independent steps. Empirically, this method outperformed the reservoir kernel with *manually* chosen state transition parameters by cross validation using grid search and *without* metric learning in the readout space. Before presenting this kernel design, we demonstrated the initial investigation on learning the dynamical system of the deterministically constructed echo state network. Experimentally, we obtained competitive performance but better efficiency compared to using cross validation presented in [76]. This kernel design gives better generalization performance than the first one, but comes at a price - relatively high computational complexity. Since this method provides a distance measure between time series, we also applied the obtained distance in a distance based classifier, i.e. kNN. Experimental results shows very promising performance.

6.2 Future Work

This work could be further extended in several directions. In particular, we consider the following topics for future research:

- Our LUPI approach opens the door to application domains where ordinal regression with privileged information can be readily employed. For example, in automated trading where only ordered categorical information is often needed [84, 97, 98], the privileged information during training can be utilized (as suggested in [101]) in the form of the known future development of the time series. However, the proposed algorithm is not very efficient. Making it training more efficient is a matter for future work.

- We notice that in the time series prediction using ordinal regression, the class distribution are highly unbalanced i.e. 10% belongs to class 1, 40% belongs to class 2, 40% belongs to class 3 and 10% belongs to class 4. Extend our work to incorporate unbalancing learning techniques may potential improve the generalization performance of the learner.

- The proposed new time series kernels can obtain very promising generalization performance on both uni-variant and multi-variant time series data. However, the proposed kernel may produce inferior performance or even may fail to perform the learning task if the dimension of the sequence is too large, for example video sequence. Our future work is to extend our kernels for video sequence classification using some dimension reduction techniques such as random projection [30].

- In our MMCL model, we projected the matrix A in the distance metric obtained in each iteration onto the space of positive semi-definite matrices as in [40]. However, instead of performing the projection, we can alter our method by substituting $A = \Omega^T\Omega$, where Ω is an arbitrary real matrix of the same size as A [85]. Then instead of updating A, we can update Ω. The resulting A will be positive semi-definite, since $\mu^T A\mu = \mu^T\Omega^T\Omega\mu = (\Omega^T\mu)^2 \geqslant 0$ for all μ. And we can even push this further by limiting the rank of matrix Ω to reduce the computational time, as suggested in [12].

APPENDIX A

Model Distance Calculation

The model distance of f_1 and f_2 under the uniform distribution of activation states is computed as follows:

$$\begin{aligned} L_2(f_1, f_2) &= \left(\int_{\mathcal{Z}} ||W\boldsymbol{z} + \boldsymbol{a}||^2 d\boldsymbol{z}\right)^{1/2} = \left(\int_{\mathcal{Z}} (W\boldsymbol{z} + \boldsymbol{a})^T (W\boldsymbol{z} + \boldsymbol{a}) d\boldsymbol{z}\right)^{1/2} \\ &= \left(\int_{\mathcal{Z}} ||W\boldsymbol{z}||^2 + \boldsymbol{z}^T W^T \boldsymbol{a} + \boldsymbol{a}^T W\boldsymbol{z} + ||\boldsymbol{a}||^2 d\boldsymbol{z}\right)^{1/2}. \end{aligned}$$

Since $\boldsymbol{z}^T W^T \boldsymbol{a}$ is a number thus $\boldsymbol{z}^T W^T \boldsymbol{a} = \left(\boldsymbol{z}^T W^T \boldsymbol{a}\right)^T = a^T W\boldsymbol{z}$. Therefore,

$$L_2(f_1, f_2) = \left(\int_{\mathcal{Z}} ||W\boldsymbol{z}||^2 + 2\boldsymbol{a}^T W\boldsymbol{z} + ||\boldsymbol{a}||^2 d\boldsymbol{z}\right)^{1/2}.$$

Because $\mathcal{Z} \in [-1, 1]^N$, then

$$\int_{\mathcal{Z}} 2\boldsymbol{a}^T W\boldsymbol{z} d\boldsymbol{z} = 2\boldsymbol{a}^T W \int_{\mathcal{Z}} \boldsymbol{z} d\boldsymbol{z} = 0.$$

Thus

$$\begin{aligned} L_2(f_1, f_2) &= \left(\int_{\mathcal{Z}} ||W\boldsymbol{z}||^2 d\boldsymbol{z} + \int_{\mathcal{Z}} ||\boldsymbol{a}||^2 d\boldsymbol{z}\right)^{1/2} = \left(\int_{\mathcal{Z}} \sum_{i=1}^{M} \left(\sum_{j=1}^{N} w_{i,j} z_j\right)^2 d\boldsymbol{z} + ||\boldsymbol{a}||^2 \int_{\mathcal{Z}} d\boldsymbol{z}\right)^{1/2} \\ &= \left(\int_{\mathcal{Z}} \sum_{i=1}^{M} \sum_{j=1}^{N} w_{i,j}^2 z_j^2 d\boldsymbol{z} + \int_{\mathcal{Z}} \sum_{i=1}^{M} \sum_{j=1}^{N} \sum_{k \neq j} w_{i,j} w_{i,k} z_j z_k d\boldsymbol{z} + 2^N ||\boldsymbol{a}||\right)^{1/2} \\ &= \left(\int_{\mathcal{Z}} \sum_{i=1}^{M} \sum_{j=1}^{N} w_{i,j}^2 z_j^2 d\boldsymbol{z} + 2^N ||\boldsymbol{a}||\right)^{1/2} = \left(\frac{2^N}{3} \sum_{j=1}^{N} \sum_{i=1}^{M} w_{i,j}^2 + 2^N \|\boldsymbol{a}\|^2\right)^{1/2}. \end{aligned}$$

The model distance of f_1 and f_2 under the non-uniform state probability distribution where the probability density function of the states are modelled by K-component Gaussian mixture model, is computed as follows:

$$\begin{aligned}
L_2(f_1, f_2) &= \left(\int_{\mathcal{Z}} ||W\boldsymbol{z} + \boldsymbol{a}||^2 dP(\boldsymbol{z})\right)^{1/2} \\
&= \left(\int_{\mathcal{Z}} \left(\boldsymbol{z}^T W^T W \boldsymbol{z} + 2\boldsymbol{a}^T W \boldsymbol{z} + \boldsymbol{a}^T \boldsymbol{a}\right) p(\boldsymbol{z}) d\boldsymbol{z}\right)^{1/2} \\
&= \left(\int_{\mathcal{Z}} \left(\boldsymbol{z}^T W^T W \boldsymbol{z} + 2\boldsymbol{a}^T W \boldsymbol{z} + \boldsymbol{a}^T \boldsymbol{a}\right) \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_k, \Sigma_k) d\boldsymbol{z}\right)^{1/2} \\
&= \left(\sum_{k=1}^{K} \pi_k \int_{\mathcal{Z}} \left(\boldsymbol{z}^T W^T W \boldsymbol{z} + 2\boldsymbol{a}^T W \boldsymbol{z} + \boldsymbol{a}^T \boldsymbol{a}\right) \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_k, \Sigma_k) d\boldsymbol{z}\right)^{1/2} \\
&= \left(\sum_{k=1}^{K} \pi_k E_{\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_k, \Sigma_k)}(\boldsymbol{z}^T W^T W \boldsymbol{z} + 2\boldsymbol{a}^T W \boldsymbol{z} + \boldsymbol{a}^T \boldsymbol{a})\right)^{1/2} \\
&= \left(\sum_{k=1}^{K} \pi_k \left(E_{\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_k, \Sigma_k)}(\boldsymbol{z}^T W^T W \boldsymbol{z}) + 2\boldsymbol{a}^T W E_{\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_k, \Sigma_k)}(\boldsymbol{z}) + \boldsymbol{a}^T \boldsymbol{a}\right)\right)^{1/2}.
\end{aligned}$$

If $\boldsymbol{x} \sim N(\boldsymbol{\mu}, \Sigma)$, the expected value of the squared form is as follows [70] (page 42):

$$E(\boldsymbol{x}^T A \boldsymbol{x}) = \mathrm{Tr}(A\Sigma) + \boldsymbol{\mu}^T W^T W \boldsymbol{\mu}.$$

Thus,

$$E_{\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_k, \Sigma_k)}(\boldsymbol{z}^T W^T W \boldsymbol{z}) = \mathrm{Tr}(W^T W \Sigma_k) + \boldsymbol{\mu}_{\boldsymbol{k}}^T A \boldsymbol{\mu}_{\boldsymbol{k}}. \tag{A.1}$$

Therefore

$$L_2(f_1, f_2) = \sum_{k=1}^{K} \pi_k \left(\mathrm{Tr}(W^T W \Sigma_k) + \boldsymbol{\eta}_k^T W^T W \boldsymbol{\eta}_k + 2\boldsymbol{a}^T W \boldsymbol{\eta}_k + \boldsymbol{a}^T \boldsymbol{a}\right). \tag{A.2}$$

APPENDIX B

Derivation of Recursive Least Squares Updates for W

Recall that the linear readout mapping of echo state network is given as follows:

$$\boldsymbol{o}(t) = W\tilde{\boldsymbol{z}}(t), t = 1, ..., L \tag{B.1}$$

By denoting the true target $\boldsymbol{s}(t+1)$ as $\boldsymbol{y}(t)$, the recursive least squares (RLS) is to find the parameter W recursively in time to minimize the sum of the squared error given as follows:

$$\varepsilon(t) = \sum_{i=1}^{t} \varsigma^{t-i} ||\boldsymbol{e}(i)||^2 \tag{B.2}$$

where $\boldsymbol{e}(i) = \boldsymbol{y}(i) - \boldsymbol{o}(i) = \boldsymbol{y}(i) - W(t)\tilde{\boldsymbol{z}}(i)$, and $0 < \varsigma < 1$ is the forgetting factor that is used to reduce the influence of the old data. By denoting $\boldsymbol{y}'(i) = \sqrt{\varsigma^{t-i}}\boldsymbol{y}(i)$ and $\boldsymbol{z}'(i) = \sqrt{\varsigma^{t-i}}\tilde{\boldsymbol{z}}(i)$, $\varepsilon(t)$ can be rewritten as follows:

$$\varepsilon(t) = \sum_{i=1}^{t} \varsigma^{t-i} ||\boldsymbol{y}(i) - W(t)\tilde{\boldsymbol{z}}(i)||^2 = \sum_{i=1}^{t} ||\boldsymbol{y}'(i) - W(t)\boldsymbol{z}'(i)||^2$$

Through standard least squares method in the new variable $\boldsymbol{y}'(i)$ and $\boldsymbol{z}'(i)$, the following equation:

$$W(t) = \left(\sum_{i=1}^{t} \boldsymbol{y}'(i)[\boldsymbol{z}'(i)]^T\right) \left(\sum_{i=1} \boldsymbol{z}'(i)[\boldsymbol{z}'(i)]^T\right)^{-1} = \Psi(t) \cdot \Phi^{-1}(t) \tag{B.3}$$

where

$$\Phi(t) = \sum_{i=1}^{t} \varsigma^{t-i} \tilde{\boldsymbol{z}}(i)\tilde{\boldsymbol{z}}^T(i) \tag{B.4}$$

$$\Psi(t) = \sum_{i=1}^{t} \varsigma^{t-i} \boldsymbol{y}(i)\tilde{\boldsymbol{z}}^T(i) \tag{B.5}$$

can be obtained. Since $W(t) = \Psi(t)\cdot\Phi^{-1}(t)$ is required to be computed recursively in time using the information already available at time $t-1$, i.e. $W(t) = \Psi(t-1)\cdot\Phi^{-1}(t-1)$, the variable $\Psi(t)$ and $\Phi(t)$ need to be written as function of $\Psi(t-1)$ and $\Phi(t-1)$, respectively.

$$\begin{aligned}
\Phi(t) &= \sum_{i=1}^{t} \varsigma^{t-i}\tilde{\boldsymbol{z}}(i)\tilde{\boldsymbol{z}}^T(i) = \varsigma\sum_{i=1}^{t-1}\varsigma^{t-1-i}\tilde{\boldsymbol{z}}(i)\tilde{\boldsymbol{z}}^T i + \tilde{\boldsymbol{z}}(t)\tilde{\boldsymbol{z}}^T(t) = \varsigma\Phi(t-1) + \tilde{\boldsymbol{z}}(t)\tilde{\boldsymbol{z}}^T(t) \\
\Psi(t) &= \sum_{i=1}^{t} \varsigma^{t-i}\boldsymbol{y}(i)\tilde{\boldsymbol{z}}^T(i) = \varsigma\sum_{i=1}^{t-1}\varsigma^{t-1-i}\boldsymbol{y}(i)\tilde{\boldsymbol{z}}^T i + \boldsymbol{y}(t)\tilde{\boldsymbol{z}}^T(t) = \varsigma\Psi(t-1) + \boldsymbol{y}(t)\tilde{\boldsymbol{z}}^T(t)
\end{aligned}$$

According to the rule $(A+CBC^T)^{-1} = A^{-1} - A^{-1}C(B^{-1}+C^TA^{-1}C)^{-1}C^TA^{-1}$ [70], the inverse of $\Phi(t)$ can be computed as follows:

$$\begin{aligned}
\Phi^{-1}(t) &= [\varsigma\Phi(t-1) + \tilde{\boldsymbol{z}}(t)\tilde{\boldsymbol{z}}^T(t)]^{-1} = \varsigma^{-1}\Phi^{-1}(t-1) \\
&\quad -\varsigma^{-1}\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)\left(I_1 + \tilde{\boldsymbol{z}}^T(t)\varsigma^{-1}\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)\right)^{-1}\tilde{\boldsymbol{z}}^T(t)\varsigma^{-1}\Phi^{-1}(t-1) \\
&= \varsigma^{-1}\Phi^{-1}(t-1) - \frac{\varsigma^{-2}\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)\tilde{\boldsymbol{z}}^T(t)\Phi^{-1}(t-1)}{1+\varsigma^{-1}\tilde{\boldsymbol{z}}^T(t)\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)} \\
&= \varsigma^{-1}\Phi^{-1}(t-1) - \frac{\varsigma^{-1}\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)\tilde{\boldsymbol{z}}^T(t)\Phi^{-1}(t-1)}{\varsigma+\tilde{\boldsymbol{z}}^T(t)\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)}
\end{aligned}$$

Denoting $\Lambda(t) = \Phi^{-1}(t)$ and $\boldsymbol{k}(t) = \frac{\Lambda(t-1)\ \tilde{z}(t)}{\tilde{\boldsymbol{z}}^T(t)\ \Lambda(t-1)\ \tilde{z}(t)+\varsigma}$, we can have

$$\Lambda(t) = \varsigma^{-1}\Lambda(t-1) - \varsigma^{-1}\boldsymbol{k}(t)\tilde{\boldsymbol{z}}^T(t)\ \Lambda(t-1) \tag{B.6}$$

It is easy to prove that $\boldsymbol{k}(t) = \Lambda(t)\tilde{\boldsymbol{z}}(t)$. The prove is given as follows:

$$\begin{aligned}
\Lambda(t)\tilde{\boldsymbol{z}}(t) &= \Phi^{-1}(t)\tilde{\boldsymbol{z}}(t) \\
&= \varsigma^{-1}\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t) - \frac{\varsigma^{-1}\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)\tilde{\boldsymbol{z}}^T(t)\Phi^{-1}(t-1)}{\varsigma+\tilde{\boldsymbol{z}}^T(t)\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)}\tilde{\boldsymbol{z}}(t) \\
&= \frac{\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t) + \varsigma^{-1}\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)\tilde{\boldsymbol{z}}^T(t)\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t))}{\varsigma+\tilde{\boldsymbol{z}}^T(t)\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)} \\
&\quad - \frac{\varsigma^{-1}\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)\tilde{\boldsymbol{z}}^T(t)\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)}{\varsigma+\tilde{\boldsymbol{z}}^T(t)\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)} \\
&= \frac{\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)}{\varsigma+\tilde{\boldsymbol{z}}^T(t)\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)} = \frac{\Lambda(t-1)\tilde{\boldsymbol{z}}(t)}{\varsigma+\tilde{\boldsymbol{z}}^T(t)\Phi^{-1}(t-1)\tilde{\boldsymbol{z}}(t)} = \boldsymbol{k}(t)
\end{aligned}$$

Commonly $\boldsymbol{k}(t)$ is known as innovation vector, while $\Lambda(t)$ is called the error covariance matrix.

Since $W(t-1) = \Psi(t-1)\Phi^{-1}(t-1)$, then $\Psi(t-1) = W(t-1)\Phi(t-1)$. Consequently, the recursive updates of $W(t)$ can be obtained as follows:

$$\begin{aligned}
W(t) &= \Psi(t)\cdot\Phi^{-1}(t) = \Psi(t)\cdot\Lambda(t) = \left(\varsigma\Psi(t-1) + \boldsymbol{y}(t)\tilde{\boldsymbol{z}}^T(t)\right)\Lambda(t) \\
&= \left(\varsigma W(t-1)\Phi(t-1) + \boldsymbol{y}(t)\tilde{\boldsymbol{z}}^T(t)\right)\Lambda(t) \\
&= \left(W(t-1)(\Phi(t) - \tilde{\boldsymbol{z}}(t)\tilde{\boldsymbol{z}}^T(t)) + \boldsymbol{y}(t)\tilde{\boldsymbol{z}}^T(t)\right)\Lambda(t) \\
&= \left(W(t-1)\Phi(t) - W(t-1)\tilde{\boldsymbol{z}}(t)\tilde{\boldsymbol{z}}^T(t)) + \boldsymbol{y}(t)\tilde{\boldsymbol{z}}^T(t)\right)\Lambda(t) \\
&= \left(W(t-1)\Phi(t) + (\boldsymbol{y}(t) - W(t-1)\tilde{\boldsymbol{z}}(t))\,\tilde{\boldsymbol{z}}^T(t)\right)\Lambda(t) \\
&= W(t-1)\Phi(t)\Lambda(t) + (\boldsymbol{y}(t) - W(t-1)\tilde{\boldsymbol{z}}(t))\,\tilde{\boldsymbol{z}}^T(t)\Lambda(t) \\
&= W(t-1) + (\boldsymbol{y}(t) - \boldsymbol{o}(t))\boldsymbol{k}^T(t).
\end{aligned}$$

APPENDIX C

Derivatives Calculation in MMCL

C.1 Derivatives of Error Function

The error in (5.18) can be rewritten as follows:

$$E = \sum_{t=1}^{L}\sum_{i=1}^{M}(o_i(t) - y_i(t))^2 = \sum_{t=1}^{L}\sum_{i=1}^{M}\left(\sum_{j=1}^{N} w_{ij}z_j(t) - y_i(t)\right)^2 \tag{C.1}$$

The partial derivative of E with respect to w_{ij} can be computed as follows:

$$\frac{\partial E}{\partial w_{ij}} = 2\sum_{t=1}^{L}(o_i(t) - y_i(t))z_j(t) \tag{C.2}$$

To simply the presentation, we can rewrite the derivative in a vectorized way as follows:

$$\frac{\partial E}{\partial W} = 2\sum_{t=1}^{L}(\mathbf{o}(t) - \mathbf{y}(t))\mathbf{z}^T(t) \tag{C.3}$$

Collecting network outputs $\mathbf{o}(t)$, desired outputs $\mathbf{y}(t)$ and the state vectors $\boldsymbol{z}(t)$ where $t = 1, ..., L$ into matrices $O = [\mathbf{o}(1), ..., \mathbf{o}(L)]$, $Y = [\mathbf{y}(1), ..., \mathbf{y}(L)]$ and $Z = [\boldsymbol{z}(1), ..., \boldsymbol{z}(L)]$ respectively, the partial derivative of W becomes:

$$\frac{\partial E}{\partial W} = 2(O - Y)Z^T = 2(WZ - Y)Z^T \tag{C.4}$$

W is obtained by making the partial derivative of E with respect to W given by (C.4), equivalently (C.2), equal to 0 and is regarded as a function of $\boldsymbol{r} = [r_c, r_j, r_i]^T$. Thus, the partial derivative of E with respect to θ ($\theta = r_c$, r_j or r_i) can be computed as follows:

$$\begin{aligned}
\frac{\partial E}{\partial \theta} &= 2\sum_{t=1}^{L}\sum_{i=1}^{M}(o_i(t)-y_i(t))(\sum_{j=1}^{N} w_{ij}\frac{\partial z_j(t)}{\partial \theta} + \sum_{j=1}^{N}\frac{\partial w_{ij}}{\partial \theta} z_j(t)) \\
&= 2\sum_{t=1}^{L}\sum_{i=1}^{M}(o_i(t)-y_i(t))\sum_{j=1}^{N} w_{ij}\frac{\partial z_j(t)}{\partial \theta} + 2\sum_{t=1}^{L}\sum_{i=1}^{M}(o_i(t)-y_i(t))\sum_{j=1}^{N}\frac{\partial w_{ij}}{\partial \theta} z_j(t) \\
&= 2\sum_{t=1}^{L}\sum_{i=1}^{M}\sum_{j=1}^{N}(o_i(t)-y_i(t))\, w_{ij}\frac{\partial z_j(t)}{\partial \theta} + 2\sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{t=1}^{T}(o_i(t)-y_i(t)) z_j(t)\frac{\partial w_{ij}}{\partial \theta} \\
&= 2\sum_{t=1}^{L}\sum_{i=1}^{M}\sum_{j=1}^{N}(o_i(t)-y_i(t))\, w_{ij}\frac{\partial z_j(t)}{\partial \theta}
\end{aligned}$$

Note the coefficients before $\frac{\partial w_{ij}}{\partial \theta}$, i.e. $\sum_{t=1}^{L}(o_i(t)-y_i(t))z_j(t)$ is essentially equal to zero because we compute W by making it so. Again, we can rewrite the partial derivative in a vectorized form:

$$\frac{\partial E}{\partial \theta} = 2\sum_{t=1}^{L}(\mathbf{o}(t)-\mathbf{y}(t))^T W \frac{\partial \mathbf{z}(t)}{\partial \theta} \tag{C.5}$$

C.2 Derivatives of Regularized Error Function

The regularized cost function (5.12) can be rewritten as follows:

$$E_r = \sum_{t=1}^{L}\sum_{i=1}^{M}\left(\sum_{j=1}^{N} w_{ij} z_j(t) - y_i(t)\right)^2 + \lambda\sum_{i=1}^{M}\sum_{j=1}^{N} w_{ij}^2 \tag{C.6}$$

The partial derivative of E_r with respect to w_{ij} will be computed as follows:

$$\frac{\partial E_r}{\partial w_{ij}} = 2\sum_{t=1}^{L}(o_i(t)-y_i(t))z_j(t) + 2\lambda w_{ij} \tag{C.7}$$

As (C.4), this equation can be rewritten as follows:

$$\frac{\partial E_r}{\partial W} = 2(WZ-Y)Z^T + 2\lambda W \tag{C.8}$$

Similarly, W can be obtained by making equation (C.8) or equivalently (C.7) equal to 0. Thus, the partial derivative of E with respect to θ can be computed as follows:

$$\begin{aligned}
\frac{\partial E_r}{\partial \theta} &= 2\sum_{t=1}^{L}\sum_{i=1}^{M}(o_i(t)-y_i(t))(\sum_{j=1}^{N} w_{ij}\frac{\partial z_j(t)}{\partial \theta}+\sum_{j=1}^{N}\frac{\partial w_{ij}}{\partial \theta}z_j(t))+2\lambda\sum_{i=1}^{M}\sum_{j=1}^{N} w_{ij}\frac{\partial w_{ij}}{\partial \theta} \\
&= 2\sum_{t=1}^{L}\sum_{i=1}^{M}(o_i(t)-y_i(t))\sum_{j=1}^{N} w_{ij}\frac{\partial z_j(t)}{\partial \theta}+2\sum_{t=1}^{L}\sum_{i=1}^{M}(o_i(t)-y_i(t))\sum_{j=1}^{N}\frac{\partial w_{ij}}{\partial \theta}z_j(t) \\
&\quad +2\lambda\sum_{i=1}^{M}\sum_{j=1}^{N} w_{ij}\frac{\partial w_{ij}}{\partial \theta} \\
&= 2\sum_{t=1}^{L}\sum_{i=1}^{M}\sum_{j=1}^{N}(o_i(t)-y_i(t))\, w_{ij}\frac{\partial z_j(t)}{\partial \theta} \\
&\quad +2\sum_{i=1}^{M}\sum_{j=1}^{N}\left(\sum_{t=1}^{T}(o_i(t)-y_i(t))z_j(t)+\lambda w_{ij}\right)\frac{\partial w_{ij}}{\partial \theta} \\
&= 2\sum_{t=1}^{L}\sum_{i=1}^{M}\sum_{j=1}^{N}(o_i(t)-y_i(t))\, w_{ij}\frac{\partial z_j(t)}{\partial \theta}
\end{aligned}$$

Thus, we can see that regularization will not change the $\frac{\partial E_r}{\partial \theta}$ and remains the same as (5.7) given as follows:

$$\frac{\partial E_r}{\partial \theta} = 2\sum_{t=1}^{L}(\mathbf{o}(t)-\mathbf{y}(t))^T W\frac{\partial \mathbf{z}(t)}{\partial \theta} \tag{C.9}$$

C.3 Time Complexity Analysis

Before we give the time complexity of our MMCL model, we list the following preliminaries: the time complexity of the product between $n \times p$ matrix and $p \times m$ matrix is $O(npm)$; the time complexity of the product between $n \times p$ matrix and p dimensional vector is $O(np)$; and the time complexity of the product between two p dimensional vectors is $O(p)$[1].

In each iteration of gradient descent in MMCL, we need to calculate $\frac{\partial Q}{\partial A}$ and $\frac{\partial Q}{\partial \theta}$, where $\theta = r_i, r_c$ or r_j. The calculation of $\frac{\partial Q}{\partial A}$ need $O(n^2M^2N^2)$ steps, where n is the number of time series in the data sets, M is the dimensionality of the time series and N is the dimensionality of the reservoir state vector. Since the reservoir structure is fixed in our methodologies, i.e. N is a constant, the time complexity of $\frac{\partial Q}{\partial A}$ can be simplified as $O(n^2M^2)$. Similarly, the time complexity of $\frac{\partial Q}{\partial \theta}$ is $O(n^2M^2 + nML)$, where L is the length of the time series. Thus, the overall time complexity of calculating the required gradients is $O(n^2M^2 + nML)$.

[1] http://www.seas.ucla.edu/~vandenbe/103/lectures/flops.pdf(16/11/2014)

List of References

[1] E. Alpaydin. *Introduction to machine learning (adaptive computation and machine learning)*. MIT Press, 2nd edition, 2010.

[2] C. Bahlmann, B. Haasdonk, and H. Burkhardt. On-line handwriting recognition with support vector machines-a kernel approach. In *Proceedings of the 8th international conference on Frontiers in Handwriting Recognition*, pages 49–54, 2002.

[3] Y. Bengio, P. Simard, and P. Frasconi. Learning long–term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[4] D Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370, 1994.

[5] C. M. Bishop. *Neural networks for pattern recognition*. Clarendon Press, Oxford, 1995.

[6] C. M. Bishop. *Pattern Recognition and machine learning*. Springer, 2006.

[7] B. Boots. Learning stable linear dynamical systems. Master's thesis, Carnegie Mellon University, May 2009.

[8] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

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

[10] S. Brahim-Belhouari and J. Vesin. Bayesian learning using gaussian process for time series prediction. In *Proceedings of the 11th IEEE Signal Processing Workshop on Statistical Signal Processing*, pages 433–436, 2001.

[11] K.H. Brodersen, T.M. Schofield, A.P. Leff, C.S. Ong, E.I. Lomakina, J.M. Buhmann, and K.E. Stephan. Generative embedding for model-based classification of fMRI data. *PLoS Computational Biology*, 7(6):1–19, 2011.

[12] K. Bunte, P. Schneider, B. Hammer, F.-M. Schleif, T. Villmann, and M. Biehl. Limited rank matrix learning, discriminative dimension reduction and visualization. *Neural Networks*, 26(0):159 – 173, 2012.

[13] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[14] F. Cai and V. Cherkassky. SVM+ regression and multi-task learning. In *In International Joint Conference on Neural Networks*, pages 418–424, 2009.

[15] F. Cai and V. Cherkassky. Generalized SMO algorithm for SVM-based multitasks learning. *IEEE Transactions on Neural Networks and Learning Systems*, 23(6):997–1003, 2012.

[16] J. S. Cardoso and J. F. Pinto da Costa. Learning to classify ordinal data: The data replication method. *Journal of Machine Learning Research*, 8:1393–1429, 2007.

[17] Antoni B. Chan and Nuno Vasconcelos. Probabilistic kernels for the classification of auto-regressive visual process. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'05)*, pages 846–851, 2005.

[18] C. C. Chang and C. J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions Intelligent System Technology*, 2:1–27, 2010.

[19] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27, 2011.

[20] C. Chatfield. *The analysis of time series : an introduction.* CRC Press, 6th edition, 2004.

[21] H. Chen, P. Tiňo, A. Rodan, and X. Yao. Learning in the model space for cognitive fault diagnosis. *IEEE Transactions on Neural Networks and Learning Systems*, 25:124–136, 2013.

[22] W. Chu and Z. Ghahramani. Gaussian processes for ordinal regression. *Journal of Machine Learning Research*, 6:1019–1041, 2005.

[23] W. Chu and S. S. Keerthi. New approaches to support vector ordinal regression. In *Proceedings of the 22nd International Conference on Machine learning*, (ICML'05), pages 145–152, 2005.

[24] W. Chu and S. S. Keerthi. Support vector ordinal regression. *Neural Computation*, 19(3):792–815, 2007.

[25] K. Crammer and Y. Singer. Pranking with ranking. In *Advances in Neural Information Processing Systems 14*, pages 641–647. MIT Press, 2001.

[26] M. Curturi and A. Doucet. Autoregressive kernels for time series. *ArXiv:1101.0673*, 2011.

[27] M. Cuturi. Fast global alignment kernels. In *Proceedings of the 28th International Conference on Machine Learning (ICML'11)*, pages 929–936, 2011.

[28] M. Cuturi, J-P Vert, O. Birkenes, and T. Matsui. A kernel for time series based on global alignments. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'07)*, volume 2, pages 413–416, 2007.

[29] G. Doretto, A. Chiuso, Y. Wu, and S. Soatto. Dynamic textures. *International Journal of Computer Vision*, 51(2):91–109, 2003.

[30] R.J. Durrant and A. Kaban. Random projections as regularizers: Learning a linear discriminant ensemble from fewer observations than dimensions. In *The 5th Asian Conference on Machine Learning (ACML 2013), Journal of Machine Learning Research-Proceedings Track*, volume 29, pages 17–32, 2013.

[31] J. Faraway and C. Chatfield. Time series forecasting with neural networks: a comparative study using the airline data. *Applied Statistics*, 47(2):231–250, 1998.

[32] J. Feyereisl and U. Aickelin. Privileged information for data clustering. *Information Science*, 194(1):4–23, 2012.

[33] M. A. T. Figueiredo and A. K. Jain. Unsupervised learning of finite mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):381–396, 2002.

[34] E. Frank and M. Hall. A simple approach to ordinal classification. In *Proceedings of the European Conference on Machine Learning*, volume 2167, pages 145–165, 2001.

[35] A. Gaidon, Z. Harchaoui, C. Schmid, et al. A time series kernel for action recognition. In *British Machine Vision Conference*, pages 63.1–63.11, 2011.

[36] M. G. Genton. Classes of kernels for machine learning : A statistic perspective. *Journal of Machine Learning Research*, 2:299–312, 2001.

[37] Z. Ghahramani and G. E. Hinton. Parameter estimation for linear dynamical systems. Technical report, Department of Computer Science, University of Toronto, 1996.

[38] Z. Ghahrammani. An introduction to hidden markov models and bayesian network. *Journal of Pattern Recognition and Artificial Intelligence*, 15(1):9–42, 2001.

[39] B. Ghanem and N. Ahuja. Maximum margin distance learning for dynamic texture recognition. In *Proceedings of the 11th European Conference on Computer Vision (ECCV'10)*, pages 223–236. Springer Berlin Heidelberg, 2010.

[40] A. Globerson and S. Roweis. Metric learning by collapsing classes. In *Advances in neural information processing systems (NIPS)*, volume 18, pages 451–458, 2006.

[41] A. C. Harvey. *Time series models.* Financial Times Prentice Hall, 2nd edition, 1993.

[42] S. Hausler, H. Markram, and W. Maass. Perspectives of the high-dimensional dynamics of neural microcircuits from the point of view of low-dimensional readouts. *Special Issume on Complex Adaptive Systems*, 8(4):39–50, 2003.

[43] M. M. Hein, M. Hein, and O. Bousquet. Hilbertian metrics and positive definite kernels on probability. In *Proceedings of International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 136–143, 2005.

[44] R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *International Conference on Artificial Neural Networks*, number 470, pages 97–102, 1999.

[45] R. Herbrich, T. Graepel, and K. Obermayer. *Large Margin Rank Boundaries for Ordinal regression*, chapter 7, pages 115–132. MIT Press, Cambridge, MA, 2000.

[46] A. Ilin and T. Raiko. Practical approaches to principal component analysis in the presence of missing values. *Journal of Machine Learning Research*, 11:1957–2000, 2010.

[47] T. Jaakkola, M. Diekhans, and D. Haussler. Using the fisher kernel method to detect remote protein homologies. In *Proceedings of the International Conference on Intelligent Systems for Molecular Biology*, pages 149–158, 1999.

[48] T.S. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In *NIPS*, pages 487–493, 1999.

[49] H. Jaeger. The echo state approach to analysing and training recurrent neural networks. Technical report, German National Research Center for Information Technology, 2001.

[50] H. Jaeger. Tutorial on training recurrent neural networks. Technical Report GMD-Report 159, German National Research Institute for Computer Science, 2002.

[51] H. Jaeger. Adaptive nonlinear system identification with echo state networks. In *Advances in Neural Information Processing Systems*, pages 593–600. MIT Press, Cambridge, MA„ 2003.

[52] T. Jebara, R. Kondor, and A. Howard. Probability product kernels. *The Journal of Machine Learning Research*, 5:819–844, 2004.

[53] E. Keogh, Q. Zhu, B. Hu, Y. Hao, X. Xi, L. Wei, and C. A. Ratanamahatana. The ucr time series classification/clustering, 2011. http://www.cs.ucr.edu/~eamonn/time˙series˙data/.

[54] E. J. Keogh and M. J. Pazzani. Derivative dynamic time warping. In *SIAM International Conference on Data Mining (SDM)*, 2001.

[55] K.-J. Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55:307–319, 2003.

[56] M. Kim and V. Pavlovic. Discriminative learning of dynamical systems for motion tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[57] K. Kumara, R. Agrawal, and C. Bhattacharyya. A large margin approach for writer independent online handwriting classification. *Pattern Recognition Letters*, 29(7):933–937, 2008.

[58] S. N. Lahiri. Theoretical comparisons of block bootstrap methods. *The Annals of Statistics*, 27(1):386–404, 1999.

[59] L. Li and H. Lin. Ordinal regression by extended binary classification. In *Advances in Neural Information Processing System*, pages 865–872, 2006.

[60] L. Liang and V. Cherkassky. Learning using structured data : Application to fMRI data analysis. In *International Joint Conference on Neural Networks*, pages 495–499, 2007.

[61] H.-T. Lin and L. Li. Reduction from cost-sensitive ordinal ranking to weighted binary classification. *Neural Computation*, 24(5):1329–1367, 2012.

[62] M. Lukoševičius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[63] W. Maass, T. Natschlager, and H. Markram. Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2513 – 2560, 2002.

[64] T. M. Mitchell. *Machine learning*. McGraw Hill, 1997.

[65] P. J. Moreno, P. P.Ho, and N. Vasconcelos. A Kullback-Leibler divergence based kernel for SVM classification in multimedia application. In *Advances in Neural Information Processing Systems*, 2004.

[66] K. Müller, A. Smola, G. Ratsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In *Proceedings of the International Conference on Artificial Neural Networks*. Springer, 1997.

[67] D. Niu, L. Ji, Xing M., and J. Wang. Multi-variable echo state network optimization by bayesian regulation for daily peak load forecasting. *Journal of Networks*, 7(11):1790–1795, 2012.

[68] M. C. Ozturk, D. Xu, and J.C. Principe. Analysis and design of echo state network. *Neural Computation*, 19(1):111–138, 2007.

[69] S.-H. Park, J.-H. Lee, J.-W. Song, and T.-S. Park. Forecasting change directions for financial time series using hidden markov model. In P. Wen, Y. Li, L. Polkowski, Y. Yao, S. Tsumoto, and G. Wang, editors, *Rough Sets and Knowledge Technology*, volume 5589 of *Lecture Notes in Computer Science*, pages 184–191. Springer Berlin Heidelberg, 2009.

[70] K.B. Petersen and M.S. Pedersen. The matrix cookbook. Technical report, Technical University of Denmark, 2008.

[71] J. C. Platt. Sequential minimal optimization : A fast algorithm for training support vector machines. Technical report, Microsoft Research, 1998.

[72] D. N Politis and H. White. Automatic block-length selection for the dependent bootstrap. *Econometric Reviews*, 23(1):53–70, 2004.

[73] J. Quinonero-Candela and L.K. Hansen. Time series prediction based on the relevance vector machine with adaptive kernels. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pages 985–988, 2002.

[74] B. Ribeiro, C. Silva, A. Vieira, A. Gaspar-Cunha, and J. C. das Neves. Financial distress model prediction using svm+. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2010.

[75] A. Rodan and P. Tiňo. Minimum complexity echo state network. *IEEE Transactions on Neural Networks*, 22(1):131–144, 2011.

[76] A. Rodan and P. Tiňo. Simple deterministically constructed cycle reservoirs with regular jumps. *Neural Computation*, 24(7):1822–1852, 2012.

[77] A. Rodan and P. Tiňo. Simple deterministically constructed recurrent neural networks. In *Proceedings of the 11th international conference on Intelligent data engineering and automated learning*, IDEAL'10, pages 267–274, Berlin, Heidelberg, 2010. Springer-Verlag.

[78] A. Rodan and P. Tiňo. Minimum complexity echo state network. *IEEE Transactions on Neural Networks*, 22(1):131–144, 2011.

[79] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel distributed processing, explorations in the microstructure of cognition*, volume 1, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[80] Stefan Rüping. Svm kernels for time series analysis, 2001.

[81] P. Saisan, G. Doretto, Y. Wu, and S. Soatto. Dynamic texture recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 58–63, 2001.

[82] A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal*, 3(3):210–229, 1959.

[83] J. Sánchez-Monedero, P. A. Gutiérrez, P. Tiňo, and C. Hervás-Martínez. Exploitation of pairwise class distances for ordinal classification. *Neural Computation*, 25(9):2450–2485, 2013.

[84] C. Schittenkopf, P. Tiňo, and G. Dorffner. The benefit of information reduction for trading strategies. *Applied Financial Economics*, 34(7):917–930, 2002.

[85] P. Schneider, P. Biehl, and B. Hammer. Adaptive relevance matrices in learning vector quantization. *Neural Computation*, 21(12):3532–3561, 2009.

[86] C.-W. Seah, I. W. Tsang, and Y.-S. Ong. Transductive ordinal regression. *IEEE Transactionss on Neural Networks and Learning Systems*, 23(7):1074–1086, 2012.

[87] M. Sewell. The fisher kernel: A brief review. Research note, UCL, 2011.

[88] S.Fouad and P. Tiňo. Adaptive metric learning vector quantization for ordinal classification. *Neural Computation*, 24(11):2825–2851, 2012.

[89] F. Sha and L. K. Saul. Large margin hidden markov models for automatic speech recognition. In *Advances in neural information processing systems (NIPS)*, 2007.

[90] A. Shashua and A. Levin. Ranking with large margin principle: Two approaches. In *Advances in Neural Information Processing System 15*, pages 937–944, 2003.

[91] J Shawe-Taylor and N. Cristinanini. *Kernel methods for patten analysis*. Cambridge University press, 2004.

[92] Z. Shi and M. Han. Ridge regression learning in esn for chaotic time series prediction. *Control and Decision*, 22(3):258–267, 2007.

[93] H Shimodaira, K.-I Noma, M. Nakai, and S. Sagayama. Dynamic time-alignment kernel in support vector machine. In *Advances in Neural Information Processing System 14*, volume 2, pages 921–928, 2002.

[94] R. H. Shumway and D. S. Stoffer. An approach to time series smoothing and forecasting using EM algorithm. *Journal of Time Series Analysis*, 3(4):253–264, 1982.

[95] M. A. Stephens. Edf statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 67(347):730–737, 1974.

[96] P. Tiňo, I. Farkaš, and J. van Mourik. Dynamics and topographic organization of recursive self-organizing maps. *Neural Computation*, 18(10):2529–2567, 2006.

[97] P. Tiňo, C. Schittenkopf, and G. Dorffner. Financial volatility trading using recurrent neural networks. *IEEE Transactions on Neural Networks*, 12(4):865–874, 2001.

[98] P. Tiňo, C. Schittenkopf, and G. Dorffner. Volatility trading via temporal pattern recognition in quantized financial time series. *Pattern Analysis and Application*, 4(4):283–299, 2001.

[99] L. Van der Maaten. Learning discriminative fisher kernels. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[100] V. Vapnik. *Statistical Learning Theory.* New York ; Chichester : Wiley, 1998.

[101] V. Vapnik and A. Vashist. A new paradigm: Learning using privileged information. *Neural Networks*, 22(5-6):544–557, 2009.

[102] V. Vapnik, A. Vashist, and N. Pavlovitch. Learning using hidden information (learning with teacher). In *Proceeding of International Joint Conference on Neural Networks*, pages 3188–3195, 2009.

[103] A. R. Webb and D. Lowe. A hybrid optimization strategy for adaptive feed-forward layered networks. Technical report, RSRE Memorandum 4193, Royal Signals and Radar Establishment, St Andrews Road, Malvern, UK, 1988.

[104] M. West and J. Harrison. *Bayesian forecasting and dynamic models.* Springer-Verlag New York, Inc, 2nd edition, 1997.

[105] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[106] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[107] E. P. Xing, M. I. Jordan, S. Russell, and A. Ng. Distance metric learning with application to clustering with side-information. In *Advances in neural information processing systems*, pages 505–512, 2002.

[108] Z. Xing, J. Pei, and E. Keogh. A brief survey on sequence classification. *ACM SIGKDD Explorations Newsletter*, 12(1):40–48, 2010.

[109] Donald W. Zimmerman. A note on interpretation of the paired-samples t test. *Journal of Educational and Behavioral Statistics*, 22(3):349–360, 1997.