Encoding Sequential Structures using Kernels

ANDREA BAISERO

KTH Computer Science
and Communication

Master of Science Thesis
Stockholm, Sweden 2012

Encoding Sequential Structures using Kernels

A N D R E A B A I S E R O

DD221X, Master's Thesis in Computer Science (30 ECTS credits)
Degree Progr. in Information and Communication Technology 270 credits
Royal Institute of Technology year 2012
Supervisors at CSC were Carl Henrik Ek and Florian Pokorny
Examiner was Danica Kragic

TRITA-CSC-E 2012:094
ISRN-KTH/CSC/E--12/094--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc

Abstract

Sequential data-types represent a natural model for information in many fields, such as Time-Series Analysis and Computational Biology. Having a very dynamic nature, sequential data still represents a challenge to modern learning methods which struggle to fully integrate the underlying information into their mechanisms. Kernel Methods offer a practical and accessible framework for the integration of structured data into well-established Machine Learning algorithms, with the only requirement being the definition of an adequate kernel function which reflects the task at hand.

Mainstream kernel functions for strings, particular instances of sequences, are mostly based on explicitly constructed feature vectors which describe the input by its relationship with respect to a set of *references*, usually n-grams. More recently, new efforts have been made to develop more sophisticated and dedicated solutions: the current state-of-the-art uses *paths* to access *alignment*-based matchings between sequences.

This Master project aims at broadening the limited variety of existing kernels for sequences while taking into account considerations regarding structural locality of feature descriptors. Inspired by modern methods, and with consideration of the issues they suffer from, the resulting developments consist in three novel kernel functions for sequential data.

A set of experiments is set up to perform a qualitative evaluation of the proposed methods. For this purpose, univariate artificial sequential datasets are created with the insertion of various noise types to simulate real-world scenarios. The experiments include tasks of classification, evaluated through appropriate cross-validation, and regression, along with visual representations of the feature space induced by the kernels. Both qualitative and quantitative experiments are presented, whose results show substantial improvements in classification and generalisation properties, and in robustness to various levels of noise.

Referat

Att koda sekvensiella data med kärnfunktioner

Många modelleringsscenarier karakteriseras av sekvensiella data såsom tidsserier eller biologiska sekvenser, t.ex. proteiner och DNA. Utvecklingen inom maskininlärning har under de senaste decennierna inneburit stora framsteg inom många områden som datorseende och informationssökning. Denna utveckling har dock i huvudsak varit fokuserad på modellering av vektoriella data. På grund av sin dynamiska karaktär, är sekvensiella data fortfarande en utmaning för moderna inlärningsmetoder som strävar efter att integrera dessa sekvenser i sina mekanismer. Kärnmetoder erbjuder en praktisk och lättillgänglig inramning för integration av strukturerade data i välkända lärande algoritmer.

I denna uppsats presenterar vi en metod för att beskriva vektoriella representationer för sekvensiella data med kärnfunktioner. Detta innebär att vi tillgängliggör de verktyg som utvecklats inom maskininlärning för vektoriella data för att appliceras på sekvensiella data. Inspirerat av moderna metoder och med hänsyn tagen till deras problem, har vi utvecklat tre nya kärnfunktioner för sekvensiella data.

De föreslagna kärnorna utvärderas kvalitativt genom lämpliga experiment. Vi använder artificiella sekvensiella databaser där brus införts för att simulera verkliga scenarion. Både kvalitativa och kvantitativa experiment presenteras. Resultaten visar betydande förbättringar av klassificerings- och generaliseringsegenskaper och i robusthet mot olika brusnivåer.

Contents

Preface

This documents contains a description of the work I carried out to complete my Master of Science studies, which have been partially carried out at the Royal Institute of Technology in Sweden, and partially at the Polytechnic University of Turin, in Italy. These two universities, among the most recognised European institutes, have collaborated in a double-degree program which gifted me not only with the pleasure of studying in a dynamic, professional and multicultural environment, but also with the possibility of expanding my horizons for the immediate future after my graduation.

My thesis project consisted in an activity of research at the Computer Vision and Active Perception (CVAP) department of the Royal Institute of Technology. The research held at CVAP aims at tackling practical modern issues in machine learning and data analysis, with particular attention for applications in the field of robotics. I was supervised by Carl Henrik Ek, a passionate researcher at CVAP specialised in statistical methods for machine learning.

The project allowed me to carry out my own research profile. An initial period involved briefing, documentation and literature readings, which revolved around generic topics and out of which I was encouraged to pick a direction to follow based on what I perceived as interesting, challenging, and improvable; my topic finally focused on the development of new methodologies for sequential data analysis.

This *modus operandi* persisted throughout the whole project and, coupled with a close collaboration with other researchers and PhD students whose research topics are closely related to my own, allowed me to realistically experience several months of the work as a researcher. At the end of the experience I was able to write, with the generous help of Carl H. Ek and of Florian T. Pokorny, a conference publication with the contents of my research, which was sent for evaluation (and hopefully, acceptance) at the 2^{nd} International Conference on Pattern Recognition Applications and Methods (ICPRAM).

The results of research are so very often characterised by an unplannable nature. The contents of this document reflect this and are hardly able to describe the entirety of my experience. A good amount of documentation, readings, discussions and chains of thought resulted, in the end, not pertinent with the final products of the project, and have thus not found their way into this document.

Acknowledgements

My years as a student have taught me more than I could have hoped for, and I have many people to thank for that. In order of appearance:

To my father, mother and brother. I was raised with no constrictions whatsoever, and very few rules to follow: to think for myself and to do what I want as long as it is done well. To this day, I stand by these rules and am grateful of such an open-minded upbringing. Thanks to them, and to the many experiences I have lived around the world travelling with them, I have learned to be an independent and flexible man. I hope this makes them proud.

Most of my memories as a university student go to my friends in Turin, although there are far too many names to list them all. I will never forget my years at the 3^{rd} floor of residence hall Renato Einaudi, and all the people which have accompanied and shared with me good, foolish, and bad times. Never before was I able to experience that type of bond. Our time as corridor mates is something hardly comparable to anything I may imagine for my future, and something I will cherish indefinitely.

Coming to Sweden has been a great adventure which I will always envision as a "gateway" between my past as a student and my future as a researcher. Many people have assisted me during this (maybe) short journey, and I wish to thank them for allowing me to meet such a diverse group of human beings. Thanks to them I have learned to appreciate all experiences, no matter how short they may seem, and to actively pursue my goals.

Finally, I would be unbelievably ungrateful if I didn't mention and thank my supervisor Dr. Carl H. Ek for his incredible support. Even when I stop to question my own work, he is able to reassure the integrity and consistency of my thoughts and methods. Having almost no contact to refer to, I believe to have been incredibly lucky to meet and to work with him. In general, there are many thanks I should also distribute to Florian T. Pokorny for his rigorous assistance and to the rest of CVAP for providing me with such a pleasing environment to develop my work.

Chapter 1

Introduction

During the last few decades, Machine Learning methods have had an enormous impact on a large range of fields such as Computer Vision, Robotics and Computational Biology. These methods have allowed researchers to exploit evidence from data to learn models in a principled manner. One of the most important developments has been that of Kernel Methods [CSt00], which embed the input data in an induced vector space with the intention of achieving improved robustness of classification and regression techniques. The main benefit of Kernel Methods is that, rather than defining an explicit feature space having the desired properties, the embedding is characterized implicitly through the choice of a kernel function which models the inner product in the induced space. This creates a very natural paradigm for recovering the desired characteristics of a representation. Kernel functions based on a stationary distance measure in the observed space, such as radial basis functions, have been particularly successful in this context [Buh03]. However, for many application domains, the data does not naturally lend itself to a finite dimensional vectorial representation. Symbolic sequences and graphs, for example pose a problem for such kernels.

The techniques used for learning and inference relative to non-vectorial data are generally much less developed. A desirable approach is hence to first place the data in a vector space where the whole range of powerful machine learning algorithms can be applied. Simple approaches such as the Bag-of-Words model, which creates a vectorial representation based on occurrence counts of specific representative "words", have had a big impact on computer vision [SZ09]. These methods incorporate the fact that a distance in the observed space of image features does not necessarily reflect a similarity. Another approach where strings are transformed into a vectorial representation before a kernel methods is applied has been the development of String kernels [LSSt$^+$02, ST04]. Such kernels open up a whole range of powerful techniques for non-vectorial data and have been been applied successfully to robotics [LB11], computer vision [LZ06] and biology [LK04]. Finally, a recent and significant approach to define an inner product between sequences is to search for a space where similarity is reflected by "how well" sequences align [Wat99, CVBM07, Cut11].

Figure 1.1. Representation of how the *locality* of the descriptor influences the perception of structure as a whole. On the left, an ordinary object which we, as humans, are able to analyse both as a whole and in the detail. On the right, different levels of locality are imposed on the image, resulting in different levels of global knowledge on the object itself. When developing new internal representation mechanisms, the locality plays an important role in the resulting generalisation properties. The figure highlights how extreme local features, although easily accessible, do not lend themselves to globally sound representations.

1.1 Methodology

Given the objective of creating a system that can learn from data, a central issue concerns how the data should be represented. An ideal representation should be robust to noise, have a clearly defined similarity measure and be such that a majority of the variance in the descriptor corresponds to information relevant for the specific task at hand. Significant benefits can be achieved by focusing on extracting a good structure for the data combined with traditional learning algorithms.

Locality of Feature Descriptors

One of the aspects which should be carefully evaluated when developing structural representation is the amount of *locality* to be used in the application of the local *descriptors*. Learning systems often handle sensory data which is nowadays easily accessible in great quantities and with ever-increasing resolution. This availability has biased many current methods into using data encoded through extremely raw and local features. This trend has the disadvantage of requiring a considerable amount of data which is fed into the learning algorithms; the algorithms are then required to distinguish between the relevant and non-relevant features present in the huge amount of data.

A more refined approach is to adequately select the amount of variance in the data in such a way to adjust the amount of generalising and discriminative information pertinently with the underlying task [EBK12, EK11]. Additionally, data which encodes structured information rather than unstructured first-order statistics also

improves the general performance of already known algorithms [MPEDK12]. By working carefully on both aspects, learning algorithms benefit of requiring less data which, at the same time, contains more valuable and relevant information.

Experimentation

The products of the development stage are tested through a number of experimental tasks which allow a comparison against the state-of-the-art. For this purpose, artificially generated univariate sequences subject to different types and levels of noise are used. The experiments range over three different tasks, each of which is repeated a number of times and with different configurations to allow a general and robust evaluation. During the experimentation, both qualitative and quantitative results are taken into account for a complete evaluation.

1.2 Project Goals

The goal of this project is to explore and possibly develop novel techniques which are inspired by sequential representations of structured data. Sequences represent simple data structures which naturally arise in many fields of engineering in the form of time-series, spatial sequences, various types of biological sequences, textual data, and others. Despite the occurrence of sequences as a very common typology of data, current state-of-the-art kernels still have issues which affect their interpretation and performance. An accurate description of such problems is presented as the main motivation behind the development of new methods. A more refined and precise description of the project goals will be presented after the background and previous work is thoroughly presented.

1.3 Report Outline

The rest of the document is organized as follows. Chapter 2 is a background chapter with an introduction to classification and kernel methods. Chapter 3 contains a description of modern kernels for sequential data. Chapter 4 contains a series of reflections which can be made after having reviewed the previous work, and described more thoroughly the pursued goals. Chapter 5 presents the products of the research, whereas Chapter 6 describes the experiments through which such results are evaluated, with consideration of the-state-of-th

Complementing the contents of the main document, a supplementary appendix (Appendix A) contains theorem proofs which were removed from the main corpus for the sake of concision.

Chapter 2

Background

Kernel functions are mathematical constructs which, born in the field of Functional Analysis (FA), have found their way into Machine Learning (ML) where they are widely celebrated for their practicality and simplicity.

Although the study of kernel functions often requires a strong mathematical background, their use within learning algorithms is much simpler, requires much less knowledge and is overall more practical. This sort of *dualism* has created throughout the years a schism regarding how people view and use kernel functions; the FA perspective on one side, endowed with very strong mathematical foundations which give rigorous meaning to kernels; and the ML perspective on the other, which rather focuses on why kernels are so useful and on how to use them efficiently.

This chapter serves as an introduction to Kernel Methods (KMs), and is primarily based on the ML view-point of kernel functions.

2.1 Classification

One of the key cognitive acts humans are capable of is that of distinguishing between classes of objects or entities. Whenever we see a pen we recognize it is a pen; not a tree, not a book, not even a pencil, but a pen. Needless to say, this faculty is fundamental to achieve an harmonic interaction with our environment. How humans perform this seemingly simple task is still subject to modern research and is not within the scope of this document. Within the fields of AI and ML, the act of distinguishing between groups of entities is better known as *classification* and constitutes a point of great interest and ongoing research.

While some classification algorithms are able to directly perform multi-class classification (k-NN, LDA, QDA), others are intrinsically limited to binary classification only[1] (SVMs, NNs). The brief introduction on linear and non-linear classification that follows is limited, for simplicity, to the case of binary classification.

[1] However, schemes such as One-vs-One and One-vs-All make use of a group of binary classifiers to achieve multi-class classification.

For this purpose, ML methods typically make use of an abstract feature space: a vector space $\mathbb{R}^d$ in which objects of interest are represented as vectors. Each characteristic of the original object which is considered relevant for the underlying task is called feature and is represented as one of the dimensions in its corresponding vector. Under this setup, classification is generally achieved by estimating and thresholding a continuous *hypothesis* function $f : \mathbb{R}^d \to \mathbb{R}$.

$$\text{Class}\,(\boldsymbol{x}) = \begin{cases} 1 & f(\boldsymbol{x}) \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{2.1}$$

The set of data-points $\{\boldsymbol{x} | f(\boldsymbol{x}) = 0\}$ represents a surface called *decision boundary* which, although technically assigned to class 1, represents a *neutral* geometrical separation surface between the two classes.

For learning purposes, it often becomes convenient to follow the parametric paradigm [Vap98], through which a whole set of hypothesis functions $f(\boldsymbol{x}, \boldsymbol{\theta})$ becomes accessible by a small number of parameters $\boldsymbol{\theta}$. Practically, this implies the choice decision of a specific model for the separation boundary. Using this paradigm, learning reduces to a process through which the *best* parameters are inferred from a set of labeled data-points called *training set*, according to some criteria which by all means characterizes the learning algorithm.

2.1.1 Linear Classification

Linear classification refers to the use of a parametric hypothesis model which satisfies $f(\boldsymbol{x}, \boldsymbol{\theta}, b) = \langle \boldsymbol{x}, \boldsymbol{\theta} \rangle - b$. It follows that the decision boundary takes the semblance of an affine hyperplane $\{\boldsymbol{x} | \langle \boldsymbol{x}, \boldsymbol{\theta} \rangle = b\}$ which divides the feature space into two half-spaces.

$$\text{Class}\,(\boldsymbol{x}) = \begin{cases} 1 & \langle \boldsymbol{x}, \boldsymbol{\theta} \rangle \geq b \\ -1 & \text{otherwise} \end{cases}$$

Parameters $\boldsymbol{\theta}$ and b respectively control the hyperplane's orientation and offset from the origin, thus determining a linear separation of the space. There is extensive documentation on a wide variety of algorithms which infer these parameters by minimizing some error measure from a set of labeled training data-points; these include SVMs [CSt00], LDA, QDA and Perceptron Learning.

The performance of linear classifiers is sometimes limited by the input datasets themselves. The *linear separability* of two sets of data-points refers to the property by which the sets are separable by a flat surface or not, where the latter case represents an intrinsic upper bound imposed on the performance of linear classifiers. It is quite simple to visualize what is known as *hyperplane separation theorem*, which states that two datasets are linearly separable if and only if their convex hulls are disjoint [BV04] (Figure 2.1).

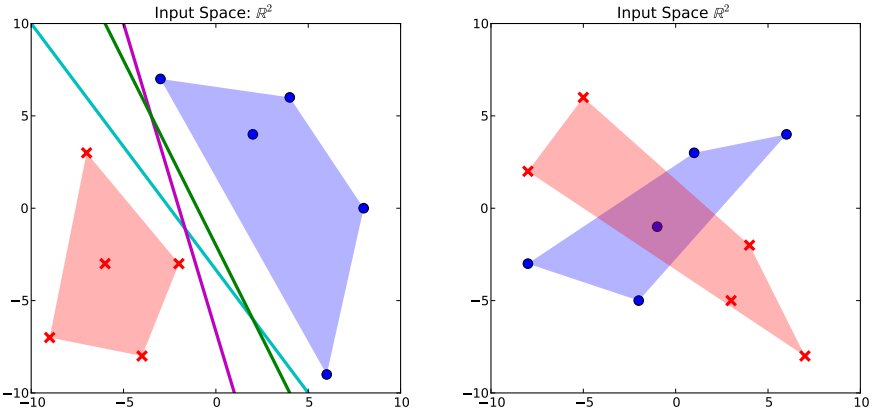

Figure 2.1. Different sets of 2D data-points. On the left, the data is linearly separable as shown by the multiple possible boundaries. On the right, the data, which resembles the structure of the well-known XOR problem, is not linearly separable in its original space. The convex hulls can be used to determine the data's separability.

2.1.2 Non-linear Classification

Linear classification algorithms can exploit a scheme which performs an enhancement in terms of non-linear classification through a projection of the data-points from the *input* feature space $\mathbb{R}^d$ into an *induced* feature space $\mathbb{R}^h$ by means of a non-linear transformation $\phi : \mathbb{R}^d \to \mathbb{R}^h$. The intended goal is to have a spatial relocation of the data-points into a more suitable disposition which allows to achieve, or otherwise improve, the linear separability of the classes. The described goal completely justifies the requirement of a non-linear ϕ. It is well worth the small effort to review the motivation for which a linear transformation ϕ results to be not useful or even harmful.

Theorem 1. *A linear transformation $\phi : \mathbb{R}^d \to \mathbb{R}^h$, represented as $\phi(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$ with $\boldsymbol{A} \in \mathbb{R}^{h \times d}$, can not improve the linear separability between classes. On the contrary, the separability degenerates or remains the same at best.*

Proof. Separation boundaries in the input and induced spaces are denoted by parameters $(\boldsymbol{\theta}_d, b_d)$ and $(\boldsymbol{\theta}_h, b_h)$. It is possible to verify that

$$\begin{aligned}\langle \boldsymbol{\theta}_h, \phi(\boldsymbol{x}) \rangle &= \boldsymbol{\theta}_h^\intercal \phi(\boldsymbol{x}) \\ &= \boldsymbol{\theta}_h^\intercal \boldsymbol{A}\boldsymbol{x} \\ &= (\boldsymbol{A}^\intercal \boldsymbol{\theta}_h)^\intercal \boldsymbol{x} \\ &= \langle \boldsymbol{A}^\intercal \boldsymbol{\theta}_h, \boldsymbol{x} \rangle.\end{aligned}$$

Thus, any boundary $(\boldsymbol{\theta}_h, b_h)$ in $\mathbb{R}^h$ achieves the same separation as $(\boldsymbol{A}^\intercal \boldsymbol{\theta}_h, b_h)$ in $\mathbb{R}^d$. Thus, all possible separations in the induced space can be achieved directly in the input space.

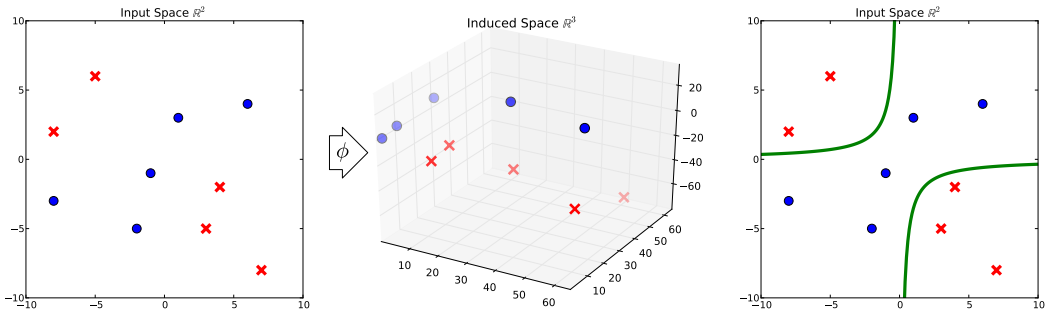

Figure 2.2. An example of kernel trick applied on the dataset from the non-linear dataset of Figure 2.1. On the center, the dataset is transformed using $\phi : \mathbb{R}^2 \to \mathbb{R}^3$, $\phi(\boldsymbol{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^\intercal$. The projected data-points become linearly separable in their induced space. On the right, one of the possible separations found in the induced space as perceived from the input space.

Perhaps counter-intuitively, this holds true even for $h > d$, when there are infinitely many more hyperplanes in $\mathbb{R}^h$ than there are in $\mathbb{R}^d$. The explanation to this is given by the fact that the image of ϕ is equivalent to the column space of $\boldsymbol{A}$, which has a dimensionality of $rank(\boldsymbol{A}) \leq d$. □

On the other hand, a non-linear ϕ is capable of restructuring the geometry of the data-points in such a way to alter and potentially improve their linear separability. Hyperplanes in $\mathbb{R}^h$ are virtually projected back into $\mathbb{R}^d$ and seen as more complex geometrical surfaces (Figure 2.2). The exact geometry of the resulting boundary depends extensively on ϕ.

About the Induced Space Dimensionality

When discussing this trick, the conviction that the induced space's dimensionality h is required to be higher than the input space's dimensionality d often arises. The motivation behind this claim is that infinitely many more hyperplanes exist in a higher dimensional space[2], each defining a different non-linear boundary in the input space. A higher dimensionality increases the number of available boundaries and their "freedom" to move around the space, supposedly making it a better setup for classification. On the other hand, these properties also encourage overfitting, the phenomenon through which an excessively complex solution is found rather than a smoother one which generalizes better when presented with new data.

The key point of a good ϕ is to shuffle data-points into more suitable dispositions, which can happen regardless of the resulting spaces dimensionality. In fact, it is advisable to limit the corresponding dimensionality to avoid overfitting and to yield simpler learning machines with fewer parameters. As a further argument against the need for high dimensional mappings, consider the ideal mapping which consists in $\tilde{\phi} : \mathbb{R}^d \to \mathbb{R}$ such that $\tilde{\phi}(\boldsymbol{x})$ directly represents the correct class of $\boldsymbol{x}$. Albeit being

[2]We have previously discussed the invalidity of this argument for the case of a linear ϕ only.

an utopian construct, extremely hard to find, and potentially overly-complicated, $\tilde{\phi}$ gives optimal performances while considerably limiting the dimensionality of the induced space.

2.2 The Kernel Trick

A wide range of ML algorithms can be rearranged, if necessary at all, in a dual form in which the data-points appear exclusively within inner products amongst themselves. *Kernel trick* is the name given to the process of embedding a non-linear transformation ϕ into these methods, which results in the data-points being used solely in expressions as $\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle$. For this reason, these methods are known as *Kernel Methods*. Whether for classification or regression, the kernel trick has gifted to the existing machine learning algorithms an extremely wide range of enhancements, all while maintaining a low computational profile.

2.2.1 Kernel Functions

A *kernel* is a function which takes advantage of such a representation to allow an efficient non-linear enhancement of the data. Formally, a kernel function is defined as follows.

Definition 1 (Kernel Function)**.** *A kernel function associated with ϕ is a real-valued function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ such that, for some mapping ϕ,*

$$k(\boldsymbol{x}, \boldsymbol{z}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle \tag{2.2}$$

A kernel can be interpreted as a generalization of the standard inner product in the input space: Although a kernel is generally not bilinear, it is symmetric, positive definite, and it obeys the Cauchy-Schwarz inequality,

$$k(\boldsymbol{x}, \boldsymbol{z})^2 \leq k(\boldsymbol{x}, \boldsymbol{x}) k(\boldsymbol{z}, \boldsymbol{z}) \tag{2.3}$$

The kernel trick consists in the definition of an "appropriate" kernel function to be embedded within the KMs in the place of its ordinary inner products. Such a procedure is nowise different from the enhancement of linear machines described in §2.1.2.

Implicit Kernels

Kernel functions have been introduced in a so called *explicit* manner: the kernel is determined by an explicit expression for ϕ. However, kernels can also be formulated in an *implicit* fashion, i.e. through an expression which is not (2.2), albeit it is equivalent to it. Implicit kernels transform data-points in some implied feature space accessible through a *surely-existing* but *not-necessarily-defined* mapping ϕ. Given proof of said equivalence, the use of an implicit kernel in the place of its explicit counterpart is perfectly justified.

Example 1. *Assume $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^2$, then $k(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{x}, \boldsymbol{z} \rangle^2$ represents a valid implicit kernel function. Specifically, k is associated with the same mapping illustrated in Figure 2.2, as shown in the following.*

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{z}) &= \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle \\ &= \langle (x_1^2, \sqrt{2}x_1x_2, x_2^2)^\intercal, (z_1^2, \sqrt{2}z_1z_2, z_2^2)^\intercal \rangle \\ &= x_1^2z_1^2 + 2x_1z_1x_2z_2 + x_2^2z_2^2 \\ &= (x_1z_1 + x_2z_2)^2 \\ &= \langle \boldsymbol{x}, \boldsymbol{z} \rangle^2 \end{aligned}$$

Implicit kernels always have an explicit counterpart[3]; an associated ϕ such that the equivalence with (2.2) is proven. On the other hand, it is hard to determine if an explicit kernel has one or more implicit counterparts. Once again, the use of an implicit kernel in place of its explicit counter part is completely justified by the equivalence between the two.

The importance of implicit kernels arises from the computational advantages of using a formulation which often proves to be much simpler than actually performing the mappings for each data-point. Moreover, implicit kernels allows to practically use mappings which would otherwise be impossible to use, such as mappings into infinite dimensional spaces; this is the case with the *Gaussian kernel*, one of the most notorious and widely used kernel functions.

The development of implicit kernels has become a determining factor for the popularity and development of KMs. Kernels have since become an efficient and elegant solution to deal with non-linear mappings into spaces of arbitrary dimension while maintaining a detachment from the mapping itself.

Kernel Validity

The use of implicit kernels leads to issues regarding what is known as the *validity* of a kernel. A function k is said to be *valid* as a kernel if the existence of a mapping ϕ such that (2.2) holds true is proven. It follows automatically that explicit kernels are valid by definition; implicit kernels typically need a more rigorous proof.

Definition 2 (Kernel matrix)**.** *Given a kernel k and a dataset $S = \{\boldsymbol{x}^{(i)}\}_{i=1}^N$, the kernel matrix of k associated with S is defined as $\boldsymbol{K} \in \mathbb{R}^{N \times N}$ such that*

$$[\boldsymbol{K}]_{i,j} = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}), \text{ i.e.} \tag{2.4}$$

$$\boldsymbol{K} = \begin{bmatrix} k(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(1)}) & k(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}) & \cdots & k(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(N)}) \\ k(\boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}) & k(\boldsymbol{x}^{(2)}, \boldsymbol{x}^{(2)}) & \cdots & k(\boldsymbol{x}^{(2)}, \boldsymbol{x}^{(N)}) \\ \vdots & \vdots & \ddots & \vdots \\ k(\boldsymbol{x}^{(N)}, \boldsymbol{x}^{(1)}) & k(\boldsymbol{x}^{(N)}, \boldsymbol{x}^{(2)}) & \cdots & k(\boldsymbol{x}^{(N)}, \boldsymbol{x}^{(N)}) \end{bmatrix} \tag{2.5}$$

[3]More precisely, they are infinite, albeit equivalent [SS01].

Expression	Conditions
$k(\boldsymbol{x}, \boldsymbol{z}) = c\, k_1(\boldsymbol{x}, \boldsymbol{z})$	c - any non negative real constant.
$k(\boldsymbol{x}, \boldsymbol{z}) = f(\boldsymbol{x}) k_1(\boldsymbol{x}, \boldsymbol{z}) f(\boldsymbol{z})$	f - any real-valued function.
$k(\boldsymbol{x}, \boldsymbol{z}) = q(k_1(\boldsymbol{x}, \boldsymbol{z}))$	q - any polynomial with non-negative coefficients.
$k(\boldsymbol{x}, \boldsymbol{z}) = \exp(k_1(\boldsymbol{x}, \boldsymbol{z}))$	
$k(\boldsymbol{x}, \boldsymbol{z}) = k_1(\boldsymbol{x}, \boldsymbol{z}) + k_2(\boldsymbol{x}, \boldsymbol{z})$	
$k(\boldsymbol{x}, \boldsymbol{z}) = k_1(\boldsymbol{x}, \boldsymbol{z}) k_2(\boldsymbol{x}, \boldsymbol{z})$	
$k(\boldsymbol{x}, \boldsymbol{z}) = k_3(\phi(\boldsymbol{x}), \phi(\boldsymbol{z}))$	k_3 - valid kernel in the space mapped by ϕ.
$k(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{A}\boldsymbol{x}, \boldsymbol{z} \rangle = \langle \boldsymbol{x}, \boldsymbol{A}\boldsymbol{z} \rangle$	$\boldsymbol{A}$ - symmetric psd matrix.
$k(\boldsymbol{x}, \boldsymbol{z}) = k_a(\boldsymbol{x}_a, \boldsymbol{z}_a) + k_b(\boldsymbol{x}_b, \boldsymbol{z}_b)$	$\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ - non-necessarily disjoint partitions of $\boldsymbol{x}$;
$k(\boldsymbol{x}, \boldsymbol{z}) = k_a(\boldsymbol{x}_a, \boldsymbol{z}_a) k_b(\boldsymbol{x}_b, \boldsymbol{z}_b)$	k_a and k_b - valid kernels on their respective spaces.

Table 2.1. Some operations which maintain the result's validity as a kernel. k_1 and k_2 must be valid kernels on the input space of $\boldsymbol{x}$, $\boldsymbol{z}$. The table was taken from [Bis06], pg. 296.

Definition 3 (Positive Semi-Definite Matrix)**.** *A real square matrix $\boldsymbol{M} \in \mathbb{R}^{n \times n}$ is said to be positive semi-definite (psd) if*

$$\boldsymbol{v}^\intercal \boldsymbol{M} \boldsymbol{v} \geq 0, \quad \forall \boldsymbol{v} \in \mathbb{R}^n \tag{2.6}$$

Definition 4 (Positive Semi-Definite Kernel)**.** *A kernel is said to be psd if its kernel matrix $\boldsymbol{K}$ is symmetric and psd for any dataset S.*

Theorem 2 (Mercer)**.** *A function k is valid, i.e. satisfies* (2.2) *for at least one mapping ϕ, if and only if it is psd. If such condition applies, then k is also known as* Mercer kernel*; thus the condition of positive semi-definiteness is also known as* Mercer's condition*.*

Theorem 2 is effectively establishes a strict correspondence equivalence between valid kernels, psd kernels, and Mercer kernels.

Constructing Kernels out of Kernels

Certain operations are proven to maintain a kernel's validity. This allows the use of psd kernels as building blocks to produce new psd kernels with different properties. Table 2.1 describes some of these simple operations.

2.2.2 Kernels for Structured Data

Kernel functions provide the means for another useful gimmick, which is the fact that they can be elegantly generalized to operate on structured data. Non-vectorial data typically requires an explicit conversion into a vectorial representation for it to be used with ML algorithms. However, it is possible to generalize a kernel function as a mapping between a pair of objects from a generic set $\mathcal{X}$ which is not required to

have any structure[4] and their related inner product in some Hilbert space embedding $\mathcal{H}$. For this purpose, it is sufficient to reformulate the following definitions:

$$\phi : \mathcal{X} \to \mathcal{H}$$
$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

This new adjustment introduces structured data into KMs in a transparent manner, while avoiding an explicit process of vectorization. What follows is a simple example of such a setup.

Example 2. *Assume $\mathcal{X}$ and $\mathcal{P}(\mathcal{X})$ to be a generic set of entities and its powerset respectively. The following is a valid kernel on subsets of $\mathcal{X}$, i.e. on $X, Z \in \mathcal{P}(\mathcal{X})$.*

$$k(X, Z) = 2^{|X \cap Z|}$$

This kernel computes the number of subsets of $\mathcal{X}$, or elements in $\mathcal{P}(\mathcal{X})$, that are also subsets of both X and Z. The corresponding feature space has dimensionality $|\mathcal{P}(\mathcal{X})|$ and can be indexed by elements of $\mathcal{P}(\mathcal{X})$.

$$\phi(X) = (\,\mathbb{I}[U \subseteq \mathcal{X}]\,)_{U \in \mathcal{P}(\mathcal{X})}$$

The creation of valid and efficient kernels for generic data-types has since become the focus of extensive and ongoing research in the fields of pattern analysis and learning machines development.

2.2.3 The Kernel as a Window into the Induced Space

The feature space induced by a kernel imposes a substantial influence on the performance of the learning algorithms. It is thus considered desirable to have a certain degree of knowledge regarding the space induced by a kernel. This implies knowledge on the mapping ϕ, which is not always easy to obtain, especially when dealing with implicit kernels. However, a good amount of information regarding projected data-points is directly accessible through the kernel itself. Being able to compute the inner product of a set of data-points amounts to being able to determine a whole set of geometrical constructs that can be formulated in terms of lengths, distances and angles:

$$\begin{aligned}
\|\boldsymbol{x}\|^2 &= \langle \boldsymbol{x}, \boldsymbol{x} \rangle \\
d(\boldsymbol{x}, \boldsymbol{z})^2 &= \|\boldsymbol{x} - \boldsymbol{z}\|^2 \\
&= \langle \boldsymbol{x} - \boldsymbol{z}, \boldsymbol{x} - \boldsymbol{z} \rangle \\
&= \langle \boldsymbol{x}, \boldsymbol{x} \rangle + \langle \boldsymbol{z}, \boldsymbol{z} \rangle - 2\langle \boldsymbol{x}, \boldsymbol{z} \rangle \\
\angle \boldsymbol{x}\boldsymbol{z} &= \arccos\left(\frac{\langle \boldsymbol{x}, \boldsymbol{z} \rangle}{\|\boldsymbol{x}\| \|\boldsymbol{z}\|} \right) \\
&= \arccos\left(\frac{\langle \boldsymbol{x}, \boldsymbol{z} \rangle}{\sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle \langle \boldsymbol{z}, \boldsymbol{z} \rangle}} \right)
\end{aligned}$$

[4] The geometrical interpretation of the data will be provided by the kernel itself.

Being a kernel function equivalent to the inner product in the induced space, it becomes possible to evaluate distances, Euclidean norms, principal axes, and other properties of projected data-points. Essentially, a kernel is, just as the inner product, a fundamental operation through which other geometrical notions arise naturally:

$$\begin{aligned}
\|\boldsymbol{x}\|_\phi^2 &= \|\phi(\boldsymbol{x})\|^2 \\
&= \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}) \rangle \\
&= k(\boldsymbol{x}, \boldsymbol{x}) \\
d_\phi(\boldsymbol{x}, \boldsymbol{z})^2 &= d(\phi(\boldsymbol{x}), \phi(\boldsymbol{z}))^2 \\
&= \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}) \rangle + \langle \phi(\boldsymbol{z}), \phi(\boldsymbol{z}) \rangle - 2\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle \\
&= k(\boldsymbol{x}, \boldsymbol{x}) + k(\boldsymbol{z}, \boldsymbol{z}) - 2k(\boldsymbol{x}, \boldsymbol{z}) \\
\angle_\phi \boldsymbol{x}\boldsymbol{z} &= \angle \phi(\boldsymbol{x})\phi(\boldsymbol{z}) \\
&= \arccos\left(\frac{\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle}{\sqrt{\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}) \rangle \langle \phi(\boldsymbol{z}), \phi(\boldsymbol{z}) \rangle}} \right) \\
&= \arccos\left(\frac{k(\boldsymbol{x}, \boldsymbol{z})}{\sqrt{k(\boldsymbol{x}, \boldsymbol{x}) k(\boldsymbol{z}, \boldsymbol{z})}} \right)
\end{aligned}$$

Example 3. *This example serves as further validation of the notions described above. Assume a set of data-points* $S = \{\boldsymbol{x}^{(i)}\}_{i=1}^N$*, and an unknown mapping* ϕ *which determines the projected set* $\phi(S) = \{\phi(\boldsymbol{x}^{(i)})\}_{i=1}^N$*. The center of mass, or mean, of* S *is defined as* $\boldsymbol{\mu}_S = \frac{1}{N}\sum_{i=1}^N \boldsymbol{x}^{(i)}$*. A respective center of mass also exists for* $\phi(S)$ *as* $\boldsymbol{\mu}_{\phi(S)} = \frac{1}{N}\sum_{i=1}^N \phi(\boldsymbol{x}^{(i)})$*. Notice that the center of mass of the projected set is not the same as the projection of the center of mass of the set, i.e.* $\boldsymbol{\mu}_{\phi(S)} \neq \phi(\boldsymbol{\mu}_S)$*.*

A simple task is that of calculating the length of $\boldsymbol{\mu}_{\phi(S)}$*, which can be used to verify the degree to which the data is centered in the induced space.*

$$\begin{aligned}
\|\boldsymbol{\mu}_{\phi(S)}\|^2 &= \langle \boldsymbol{\mu}_{\phi(S)}, \boldsymbol{\mu}_{\phi(S)} \rangle \\
&= \langle \frac{1}{N}\sum_{i=1}^N \phi(\boldsymbol{x}^{(i)}), \frac{1}{N}\sum_{j=1}^N \phi(\boldsymbol{x}^{(j)}) \rangle \\
&= \frac{1}{N^2}\sum_{i,j=1}^N \langle \phi(\boldsymbol{x}^{(i)}), \phi(\boldsymbol{x}^{(j)}) \rangle \\
&= \frac{1}{N^2}\sum_{i,j=1}^N k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}).
\end{aligned}$$

A more elaborate task is to calculate the distance between $\boldsymbol{\mu}_{\phi(S)}$ *and* $\phi(\boldsymbol{\mu}_S)$*, which can be though of as a light-hearted measure for both the compactness of the induced*

data and the local smoothness of the projecting function in the area scattered by S.

$$
\begin{aligned}
d(\boldsymbol{\mu}_{\phi(S)}, \phi(\boldsymbol{\mu}_S))^2 &= \|\boldsymbol{\mu}_{\phi(S)} - \phi(\boldsymbol{\mu}_S)\|^2 \\
&= \langle \boldsymbol{\mu}_{\phi(S)} - \phi(\boldsymbol{\mu}_S), \boldsymbol{\mu}_{\phi(S)} - \phi(\boldsymbol{\mu}_S) \rangle \\
&= \langle \boldsymbol{\mu}_{\phi(S)}, \boldsymbol{\mu}_{\phi(S)} \rangle + \langle \phi(\boldsymbol{\mu}_S), \phi(\boldsymbol{\mu}_S) \rangle - 2\langle \boldsymbol{\mu}_{\phi(S)}, \phi(\boldsymbol{\mu}_S) \rangle \\
&= \langle \frac{1}{N} \sum_{i=1}^{N} \phi(\boldsymbol{x}^{(i)}), \frac{1}{N} \sum_{j=1}^{N} \phi(\boldsymbol{x}^{(j)}) \rangle + \langle \phi(\boldsymbol{\mu}_S), \phi(\boldsymbol{\mu}_S) \rangle - 2\langle \frac{1}{N} \sum_{i=1}^{N} \phi(\boldsymbol{x}^{(i)}), \phi(\boldsymbol{\mu}_S) \rangle \\
&= \frac{1}{N^2} \sum_{i,j=1}^{N} \langle \phi(\boldsymbol{x}^{(i)}), \phi(\boldsymbol{x}^{(j)}) \rangle + \langle \phi(\boldsymbol{\mu}_S), \phi(\boldsymbol{\mu}_S) \rangle - \frac{2}{N} \sum_{i=1}^{N} \langle \phi(\boldsymbol{x}^{(i)}), \phi(\boldsymbol{\mu}_S) \rangle \\
&= \frac{1}{N^2} \sum_{i,j=1}^{N} k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) + k(\boldsymbol{\mu}_S, \boldsymbol{\mu}_S) - \frac{2}{N} \sum_{i=1}^{N} k(\boldsymbol{x}^{(i)}, \boldsymbol{\mu}_S).
\end{aligned}
$$

2.2.4 Non Positive Semi-Definite Kernels

Positive semi-definiteness has been introduced as a basic requirement which justifies a kernel's geometrical interpretation. However, non-psd functions have also proven, throughout the last decades, to work considerably well under various circumstances [LL03]. This may verify, for example, if a non-psd function generates a psd kernel matrix for a specific dataset, or if the positive semi-definiteness is appropriately approximated. Non-psd kernels do not have a feature space representation, and thus lack of a complete geometrical interpretation which justify their use within well-known learning algorithms. However, a certain amount of ongoing research aims at finding such a geometrical interpretation in terms of separation spaces [OMCS04].

As a consequence, the term *kernel* has been used more broadly to indicate a similarity function between data-points, regardless of it being psd, almost psd, or simply "useful" as a kernel function. The rest of the document adjusts to this trend, and thus the term *kernel* is used hereafter to generically refer to a similarity function. When necessary, the validity of a kernel will be pointed out by explicitly referring to it as a valid kernel, a psd kernel or a Mercer kernel.

Chapter 3

Previous Work

This chapter contains descriptions of specific methods already developed for sequential data. This includes well established methods for textual data representation as the Bag-of-Words model, a number of String kernels, the innovative Dynamic Time Warping distance measure, and finally the most recent Global Alignment kernel. These models and kernels can be considered to have slightly different domains of action. What follows is a brief series of definitions, required for the full understanding of what follows.

3.1 Definitions

Throughout the document, we will refer to the following definitions and notation to describe sequences, strings, and other constructs useful for their study and comparison.

Definition 5 (Sequences)**.** *A sequence s is an ordered list of* symbols *drawn from an* alphabet *set Σ, i.e. $s = (s_1, s_2, \ldots, s_{|s|})$, where $s_i \in \Sigma$ and $|s|$ denotes the length of the sequence. We denote the set of all strings as Σ^*, and the set of strings with length $|s| = n$ as Σ^n. Contiguous subsequences of s are denoted using subscript indexing according to $s_{a:b} = (s_a, \ldots, s_b)$. We occasionally omit indexes a or b, in which case they implicitly refer to 1 or $|s|$ respectively. We define the* inverse *of s as* $\mathrm{inv}(s) = (s_{|s|}, \ldots, s_2, s_1)$.

Throughout this document, the difference between strings and sequences is attributed to the nature and dimension of the alphabet set: Strings are characterized by a finite alphabet, whereas sequences generalise Σ to be any finite or infinite set. Additionally, when dealing with sequences, we furthermore extend Σ with a meta-symbol ϵ which is intended exclusively as the value assumed by s_i for $i \leq 0$ and $i > |s|$. When the nature of the alphabet is unimportant, we may refer to sequences and strings alike.

A kernel function k_Σ describes the similarity between elements of the alphabet. We refer to k_Σ as the *alphabet* kernel, *symbol* kernel or *ground* kernel. Except when

PATHS			
γ_1	[(1,1)(2,1)(3,1)(4,1)(4,2)(4,3)(4,4)(4,5)(4,6)]	γ_2	[(1,1)(2,2)(3,3)(3,4)(3,5)(4,6)]
γ_3	[(1,1)(2,2)(2,3)(3,4)(3,5)(4,6)]	γ_4	[(1,1)(1,2)(1,3)(1,4)(1,5)(1,6)(2,6)(3,6)(4,6)]

ALIGNMENTS							
γ_1	A N N A A A A A A B B B B A N A N A	γ_2	A N N N N A B A N A N A	γ_3	A N N N N A B A N A N A	γ_4	A A A A A A N N A B A N A N A A A A

Table 3.1. On the top table, four different paths are defined over sequences with length 4 and 6. On the bottom table, the alignments relative to the same paths on $s =$"ANNA" and $t =$"BANANA". The underlines denote the repetition of symbols. Noteworthy is the fact that, although γ_2 and γ_3 produce the same alignments, they are different paths, and thus a difference may be imagined in "how" the alignments are determined. The same paths are illustrated alternatively in Figure 5.1.1.

otherwise stated, it is assumed that k_Σ may be any psd kernel function, and that it can thus be adapted to reflect the nature of the sequences and the task at hand.

We proceed to define a series of constructs which have already been exploited in the most recent literature regarding sequence and time-series analysis [CVBM07, Cut11]. We will, however, introduce our own notation to the following, without influencing the contents.

Definition 6 (Steps)**.** *We define the* null *step* $\delta_{00} \stackrel{\text{def}}{=} (0,0)$*, the* vertical *step* $\delta_{10} \stackrel{\text{def}}{=} (1,0)$*, the* horizontal *step* $\delta_{01} \stackrel{\text{def}}{=} (0,1)$*, and the* diagonal *step* $\delta_{11} \stackrel{\text{def}}{=} (1,1)$*.* $S \stackrel{\text{def}}{=} \{\delta_{10}, \delta_{01}, \delta_{11}\}$ *is the set of* admissible *steps.*

A sequence of admissible steps is used to define the notion of a path.

Definition 7 (Path)**.** *A* path *over two sequences with length* m *and* n *is a mapping* $\gamma : \{1, \ldots, |\gamma|\} \to \mathbb{Z}^+ \times \mathbb{Z}^+$ *such that*

$$\gamma(1) = (1,1), \tag{3.1}$$
$$\gamma(i+1) = \gamma(i) + \delta_i, \tag{3.2}$$
$$\text{for } 1 \leq i < |\gamma|\,, \text{ with } \delta_i \in S,$$
$$\gamma(|\gamma|) = (m,n). \tag{3.3}$$

$|\gamma|$ *and* δ_i *respectively denote the path's* length *and* i^{th} step*. We adopt the notation* $\gamma(i) = (\gamma_X(i), \gamma_Y(i))$*, which allows us to define the alignments (or* stretches*) of the input sequences according to* $s_{\gamma_X} = (s_{\gamma_X(1)}, \ldots, s_{\gamma_X(|\gamma|)})$ *and* $t_{\gamma_Y} = (t_{\gamma_Y(1)}, \ldots, t_{\gamma_Y(|\gamma|)})$*. Table 3.1 shows the alignments specified by four paths on the same input sequences.*

Given a $m \times n$ matrix $\boldsymbol{M}$, a path can be interpreted as a way of connecting the top-left and bottom-right corners[1], and allows to index specific elements of the matrix which appear in the *trail* accordingly to $\boldsymbol{M}_{\gamma(i)} = \boldsymbol{M}_{\gamma_X(i),\gamma_Y(i)}$. Given this perspective, we say that the path *travels* the matrix.

[1] Assuming $\boldsymbol{M}_{1,1}$ and $\boldsymbol{M}_{m,n}$ correspond to the top-left and bottom-right respectively.

Definition 8 (Path Derivative)**.** *We define the path derivative $\gamma\prime$ of γ as the mapping $\gamma\prime : \{1, \ldots, |\gamma|\} \to S \cup \delta_{00}$ such that*

$$\begin{aligned} \gamma\prime(1) &= \delta_{00} \\ \gamma\prime(i+1) &= \gamma(i+1) - \gamma(i) \quad 1 \leq i < |\gamma| \end{aligned}$$

We say that $\gamma\prime$ also has length $|\gamma|$. From (3.2)*, in the definition of path, it follows that*

$$\begin{aligned} \gamma\prime(i+1) &= \gamma(i+1) - \gamma(i) \\ &= \gamma(i) + \delta_i - \gamma(i) \\ &= \delta_i \end{aligned}$$

Thus $\gamma\prime$ represents the exact sequence of steps that γ uses, for example, to travel a matrix.

It is noticeable that there is a one-to-one correspondence between a path and its derivative: they merely constitute two different representations of the same entity. This allows us to treat a path and its derivative interchangeably and to describe them both as a concatenation of of steps $(\delta_1, \ldots, \delta_{l-1})$.

Definition 9 (Path Set)**.** *We denote the set of all paths over two sequences with lengths m and n as $\Gamma(m, n)$. Its cardinality can be obtained using the Delannoy numbers*[2]*, $|\Gamma(m, n)| = \mathrm{D}(m, n)$. Using the matrix interpretation of a path, this corresponds to the number of lattice paths which connect the two opposing corners using only vertical, horizontal and diagonal steps. We occasionally omit the lengths m and n when they are assumed to be obvious or unimportant.*

3.2 The Bag-of-Words Model

The Bag-of-Words model (BoW) is a simple model introduced for the task of textual categorization [Joa98]. The task of document analysis is more specific than the one of sequence analysis by means of how textual documents contain specific semantical information attributed to words and sentences. However, BoW defines a mapping from the space of textual documents to a vector space, and can thus be used to naturally define a kernel on strings k_{BoW}. BoW encodes documents into histograms by means of their term frequencies TF, which denote the occurrence frequency of a word in the document. Further refinements also consider equivalent words with the same stem (as expression with similar semantical information) and ignore common words (as expression with no semantical information). A dictionary set D can be used for the purpose of specifying the terms which are considered relevant. BoW thus represents the following vectorial representation of a document x.

$$\phi_{\mathrm{BoW}}(x) = (\mathrm{TF}(x, t))_{t \in D}. \tag{3.4}$$

[2] *Delannoy number $\mathrm{D}(m, n)$ denotes the number of distinct taxicab paths which connect opposing corners of an $m \times n$ grid.*

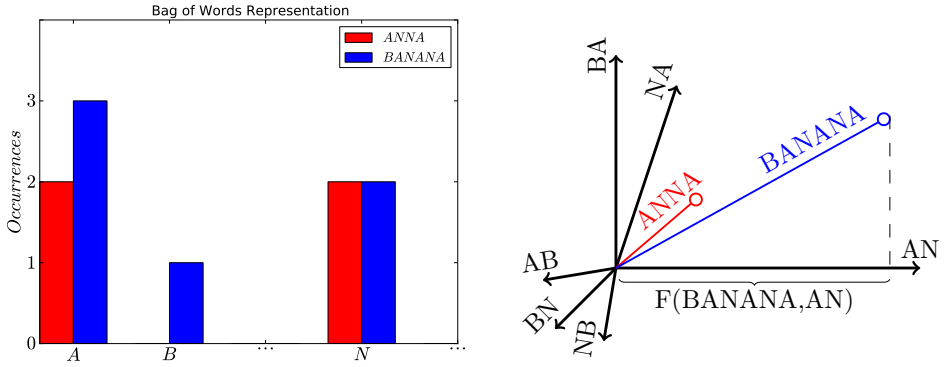

Figure 3.1. On the left, the BoW model. On the right, the generic mechanism of string kernels. An arbitrary number of axes can be constructed by manipulating the set of *reference* strings; On the right, a subset of all 2-grams consisting in "AN", "NA","BA","AB","BN" and "NB" is used. The projection of the inputs "ANNA" and "BANANA" onto each dimension, which is not depicted quantitatively, is determined by function F independently from the other dimensions. A wide variety of kernels with different properties are obtainable by changing the reference set and the projection function F.

Although the original model is designed to encode textual documents, it easily adapts to encode strings by substituting the dictionary set and terms with the alphabet set and symbols respectively.

BoW performs well the original task of document categorization, however it does also hint at not being able to generalize very well to strings. In textual documents, individual words maintain some semantical information even when taken out of the context of their sentences; the same can not be generically said for the symbols in a string, whose information is more likely to be contained in the relationship with the other symbols in the sequence rather than the presence of the symbol itself. Additionally, BoW does not distinguish between documents which are equal under an opportune permutation of words, i.e. BoW does not encode the sequentiality of the document, but rather only the first order statistics regarding its contents. Furthermore, BoW can not be applied to sequences because of the alphabet set's possibly infinite dimensionality[3].

3.3 String Kernels

There is a certain number of closely related string kernels developed in the last decade which have been successfully used in the fields of computational biology [LEN02, LK04] and activity recognition [EBK12, YGSK07]. All of these kernels can be said to generalise the coarse nature of BoW by taking into account the relative positions of the symbols under certain circumstances.

[3]Although a crude workaround would be to partition the alphabet set in a finite number of subsets which are then used as new "symbols" of a new "alphabet".

All string kernels are based on the following construction: An explicit vector space is defined by a set of *reference* strings. Each *reference* represents one of the space's dimensions, and the projection of a string s onto that dimension is determined by the result of an opportune *reference relationship* function. Altering the set of references and the notion of reference relationship function results, one can describe the following (and many more) string kernels. Figure 3.1 illustrates the basic principle behind all string kernels.

The n-Gram Kernel

The n-Gram Kernel $k_{n\mathrm{G}}$, also known as Spectrum Kernel, represents a generalisation to k_{BoW} [LSSt$^+$02, LEN02]. An n-gram consists in a string with length n. $k_{n\mathrm{G}}$ models a string as a histogram of the n-grams it contains[4], meaning that the dimensionality of the induced space is the number of distinct n-grams, i.e. $|\Sigma|^n$. The feature vectors easily reach high dimensions which are hard to handle. Nevertheless, because a string s can only contain up to $|s| - n + 1$ different n-grams, its resulting feature vector is most likely very sparse. An n-gram frequency function $n\mathrm{GF}$ counts the occurrences of each n-gram within the input string s.

$$\phi_{\mathrm{nG}}(s) = (n\mathrm{GF}(s,t))_{t\in\Sigma^n} \tag{3.5}$$

There is a certain number of variations of *ngk* which use the same working principle of creating a histogram over a set of n-grams, with the difference that the restrictions of the counting procedure are loosened. These include the following Mismatch kernel, the Restricted Gappy kernel, the Substitution kernel and the Wildcard kernel. The following are only but brief descriptions, I refer to the original documents for further information [LK04].

Variations to the n-Gram Kernel

The Mismatch kernel k_{M} is a variation of $k_{n\mathrm{G}}$ which counts matching n-grams as long as there are only up to m different symbols, where m represents an ulterior parameter. This kernel allows to soften the conditions allowing for slight "typo"-like mismatches to be recognised. When $m = 0$, the kernel degenerates back into $k_{n\mathrm{G}}$.

The Restricted Gappy kernel k_{RG} matches n-grams with sections of the input strings which may contain a limited number of unrelated symbols. This is achieved through the use of *gaps*, which are sections of the inputs which are conceptually ignored in case-by-case matchings.

The Substitution kernel is similar to k_{M} with the difference that only certain mismatching symbols are allowed. The possible mismatches are evaluated based on statistical relationships captured between sequences which are related by the task at hand.

Finally, the Wildcard kernel k_{W} makes use of a special *wildcard* symbol which, as a joker, matches positively with any of the other symbols.

[4]Thus, $k_{n\mathrm{G}}$ is equivalent to k_{BoW} when $n = 1$.

The Suffix Tree Kernel

The Suffix Tree kernel generalises $k_{n\mathrm{G}}$ by extending the feature vector to include the n-grams frequency for any length n [VS].

$$\phi_{\mathrm{ST}}(s) = (n\mathrm{GF}(s,t))_{t\in\Sigma^*} \tag{3.6}$$

Although the feature vectors belong to an infinite dimensionality space, because the dimension of a string is finite, the feature vectors are also very sparse. This allows for the inner product to be carried out explicitly. However, k_{ST} conveniently makes use of a suffix tree constructed on the input strings to elaborate the same result more efficiently.

The String Subsequence Kernel

The String Subsequence kernel defines a feature space constructed over all subsequences of length k, meaning the induced space has dimensionality Σ^k. A subsequence frequency function SSF elaborates a value which depends on how frequently and compactly a given subsequence appears in the input string.

$$\phi_{\mathrm{SS}}(s) = (\mathrm{SSF}(s,t))_{t\in\Sigma^k} \tag{3.7}$$

The novelty of this approach is that it recognises the presence of non-contiguous subsequence, albeit they are down-weighted to reflect the lower relevancy. Accordingly, a decay factor $\lambda \in (0,1)$ is used to penalize the contribution made by less compact subsequences. Further details are spared and can be reached in the original paper [LSSt$^+$02].

The Syllable Kernel

The Syllable kernel reinterprets the notion adopted by k_{BoW} and $k_{n\mathrm{G}}$ by using syllables instead of words or n-grams. The authors justify the use of syllables against single symbols for efficiency reasons, and against the use of n-grams by the argument that symbols contain more adequate semantical contents than n-grams, and is thus bound to improve performances [ST04]. Further extensions allows to weight the contributions based on relevance and to perform a *soft-matching*, through which similar syllables also influence the final result.

3.4 Dynamic Time Warping

Dynamic Time Warping (DTW) is a popular distance measure for sequences and time-series which was positively received as an innovation in the way strings and sequences are compared [SC78, GRS08]. It makes use of a distance metric (d_Σ) on symbols to find the minimal score alignment according to

$$d_{DTW}(s,t) = \min_{\gamma\in\Gamma} \frac{D_{s,t}(\gamma)}{|\gamma|}, \tag{3.8}$$

where $D_{s,t}$ represents the score of a shared path γ,

$$D_{s,t}(\gamma) = \sum_{i=1}^{|\gamma|} d_\Sigma(s_{\gamma_X(i)}, t_{\gamma_Y(i)}). \tag{3.9}$$

Although DTW is only a distance measure and not a kernel, edit-distances have recently been used to define kernel mappings into spaces which appropriately reflect the distance measure itself [NB06].

DTW presents further problems, the most important of which being that it does not necessarily respect the triangle inequality and thus does not correspond to a metric distance, i.e. DTW lacks of a strictly rigorous geometrical interpretation [CVBM07]. The same authors accuse a lack of robustness of DTW due to the limited use it makes of the input sequences, and propose kernel function similarly constructed but which tries to avoid the pointed issues.

The following non-psd kernel obtained by using the negative exponential[5] of the DTW distance will be regarded, for the purpose of this research, as the DTW kernel.

$$k_{DTW}(s,t) = e^{-d_{DTW}(s,t)} \tag{3.10}$$

3.5 The Global Alignment Kernel

The Global Alignment kernel k_{GA} is a very recent development which develops furthermore the concept of alignment introduced by DTW, making use of all possible alignments rather than only the one which yields the highest score [Cut11,CVBM07]. k_{GA} is defined as the exponentiated *soft-maximum*[6] of all alignment *scores*, i.e.

$$k_{\mathrm{GA}}(s,t) = \sum_{\gamma \in \Gamma} e^{-D_{s,t}(\gamma)}. \tag{3.11}$$

The authors provide an alternative formulation which follows from $k_\Sigma = e^{-d_\Sigma}$.

$$k_{\mathrm{GA}}(s,t) = \sum_{\gamma \in \Gamma} \prod_{i=1}^{|\gamma|} k_\Sigma(s_{\gamma_X(i)}, t_{\gamma_Y(i)}). \tag{3.12}$$

The leading principle in this approach is to combine scores obtained over all alignments between the input sequences. In their work, the authors show that incorporating all the paths into the final results can vastly improve performances, compared to the approach of only using the optimal-score path. Furthermore, k_{GA} is proven to be psd under the condition that both k_Σ and $\frac{k_\Sigma}{1+k_\Sigma}$ are psd [CVBM07], which gives formal foundation to its geometrical interpretation.

In successive work, the authors propose a generalization called the *Triangular Global Alignment Kernel* which achieves computational advantages at the cost of only taking consideration of a subset of all possible paths [Cut11].

[5] Exponentiation is suggested to provide traits of positive semi-definiteness [BHB].

[6] The soft-maximum of set of positive scalars z_i is defined by the authors as $\log \sum \exp(z_i)$.

Chapter 4

Preliminary Considerations

Sequentially structured data is characterised by a dynamicity which represents a challenge that modern research is still striving to solve.

We have seen that most string kernels address the problem by creating a well-defined feature space where each dimension is based on a specific *reference* string. The projection of an input string on a given dimension depends on the existing relation, which varies for each string kernel, incurring with its reference. Albeit this principle has proven to work for certain tasks, the approach relies on local descriptors which thus somehow limit the generative properties of the learning algorithms. Furthermore, the method is not specifically designed or justified for sequential data: the same approach may be used to construct kernels on arbitrary structures. This leads to imagine that certain properties and observations, which may be specific for sequences alone, may be overlooked.

On the other hand, DTW and the Global Alignment kernel exploit the path-based relationships occurring between the inputs. The framework presented by the use of paths is in nature more dynamic than that used by string kernels: it is specifically designed for sequential data, and it represents a novel approach which deserves to be thoroughly developed.

For this reason, Cuturi's Global Alignment kernel is used as main reference for the evaluation of theoretical properties and experimental results. We highlight a few issues which affect this kernel.

4.1 Issues with the State-of-the-Art

Equation (3.12) shows that k_{GA} makes wide use of productions over the path's elements, which are evaluate by k_Σ. This implies that the score for a whole path γ will be very small given a sufficiently large distance value $d_\Sigma(s_{\gamma_X(i)}, t_{\gamma_Y(i)})$ for any $i = 1, \ldots, |\gamma|$. This leads to the problem of yielding diagonally dominant kernel matrices, a documented issue for this kernel [CVBM07, GRS08, Cut11] which limits its performance. Diagonally dominant kernel matrices imply that the induced mapping tends to project data-points in almost orthogonal dispositions, which severely

limits the possibility of generalisation in the space. As an attempt to rescale the values, the authors suggest the use of its logarithm $\log(k_{\mathrm{GA}})$ which, however, does not maintain the positive definiteness.

A more accurate analysis shows that the issue becomes particularly troubling when occurring between symbols located at the start and the end of the input sequences. Such symbols are included in all possible paths defined over the sequences, and are thus the cause of the following undesirable property.

Theorem 3.

$$d_\Sigma(s_1, t_1) \gg 0 \Rightarrow k_{GA}(s, t) \approx 0 \tag{4.1}$$
$$d_\Sigma(s_{|s|}, t_{|t|}) \gg 0 \Rightarrow k_{GA}(s, t) \approx 0 \tag{4.2}$$

Proof. $d_\Sigma(s_1, t_1) \gg 0$ implies $k_\Sigma(s_1, t_1) \approx 0$. By the definition of path in (3.1), we take it that $\gamma(1)$ is always $(1, 1)$. This allows us to reformulate k_{GA} as follows.

$$k_{\mathrm{GA}}(s, t) = k_\Sigma(s_1, t_1) \sum_{\gamma \in \Gamma} \prod_{i=2}^{|\gamma|} k_\Sigma(s_{\gamma_X(i)}, t_{\gamma_Y(i)}). \tag{4.3}$$

Thus,

$$\begin{aligned} d_\Sigma(s_1, t_1) \gg 0 &\Rightarrow k_\Sigma(s_1, t_1) \approx 0 \\ &\Rightarrow k_{\mathrm{GA}}(s, t) \approx 0 \end{aligned}$$

An equivalent proof can be dealt for (4.2). □

Theorem 3 shows that, under certain conditions, the result of k_{GA} is unaffected by the majority of the information contained in the input sequences and is based only on a small portion of symbols extracted from the beginning and the end of the sequences. This represents a sub-optimal use of the data and of its structure, and is should thus be reconsidered for further development.

Furthermore, we have seen that in k_{GA} all the paths contribute evenly to the final result. Albeit we agree that taking consideration of all paths contributes to an increase of robustness, we also argue that not all paths are equally important. We argue that it may be beneficial to impose a preference on paths based on how much they "distort" the input sequences, i.e. a criteria which emphasises the importance of paths based on how much s_{γ_X} and t_{γ_Y} resemble s and t.

The Triangular Global Alignment kernel does somehow address this issue. Using an extra parameter, k_{TGA} generalises k_{GA} by only considers a subset of all paths. To do so, it applies a "crude" preference for paths which do not drift away from the main diagonal [Cut11]. However, this new alteration is justified exclusively by reasons related to computational efficiency, and not by a desire to assign a value to each path.

4.2 Problem Statement and Goals

The goal of my research is the development of new solutions to the problem of encoding sequential data for ML methods through the use of kernel functions. The proposed methods should attempt at addressing the issues which afflict the state-of-the-art and provide improvements in terms of performance and generalisation under experimental settings, all while taking into account the observations regarding the locality of descriptors provided in Chapter 1.

Paths represent flexible tools which are located in an intermediate point between extremely local and global feature descriptors, and thus represent a desirable medium through which to analyse sequences. For this reason, and supported by the fact that paths have yet to be fully exploited, the goal of this project is to continue, whenever reasonable, focusing on path-driven solutions.

Given the well-established tack of developing kernel functions for sequential data, the following characteristics are proposed as ideal properties of an optimal sequence kernel k:

Validity
k should be a valid kernel, with knowledge of its underlying mapping ϕ.

Similarity
k should be intuitively interpretable as a similarity measure.

Independence from length
k should accept sequences of arbitrary length.

Independence from type
k should be applicable to sequences of arbitrary data-type, and thus it should not be directly dependent on the nature of the alphabet set. The most efficient to achieve this is to handle symbols indirectly through the use of k_Σ.

Proportionality
k should compute, when its inputs are sequences of unit length, a value which is proportional to k_Σ by a strictly positive proportionality constant.

Invertibility (optional)
Depending on the nature of the sequences, it might result appropriate for k to be inversion-invariant, i.e. to satisfy $k(s,t) = k(\mathrm{inv}(s)\,, \mathrm{inv}(t))$. This may be more desirable, for instance, when dealing with sequences which arise from spatial contexts rather than temporal ones.

Chapter 5

Developed Kernels and their Properties

This chapter describes the products of my research and the process through which they were developed. These products are the *Path* kernel, the *Subsequence Sum* kernel, and the *Positional* kernel. Alongside with their definitions, an explanation of their characteristics and relative properties is presented. Finally, an algorithm which achieves high computational efficiency with there kernels is described.

5.1 Other Definitions

This section extends the definitions of §3.1 and provides the tools for the development and study of the proposed methods. In the following paragraphs, $\mathbb{I}[\cdot]$ refers to the indicator function, i.e.

$$\mathbb{I}[b] = \begin{cases} 1 & b = \text{true}, \\ 0 & \text{otherwise}. \end{cases} \tag{5.1}$$

5.1.1 The Path Matrix

The *Path matrix*, which we also refer to as *Crossover matrix* or *Ground Matrix*, is an important structure which harmonically unites with the concept of paths, and thus places itself at the center of the proposed methods.

Definition 10 (Path Matrix)**.** *Given any chosen ground kernel k_Σ, the* Path matrix *of two sequences s and t is defined as $\boldsymbol{G}(s,t) \in \mathbb{R}^{|s|\times|t|}$ such that $[\boldsymbol{G}(s,t)]_{ij} = k_\Sigma(s_i, t_j)$. When the inputs are assumed to be understood, obvious or unimportant, it is possible to omit them.*

The importance of such a structure arises from the realisation that its contents reflect both local and global information which can be efficiently used for the purpose of analysing the inputs. One of the main arguments which motivate my work is, in fact, that $\boldsymbol{G}$ contains all the necessary information to compute a solid kernel value. We refer to Figure 5.1 for a graphical illustration of the following ideas.

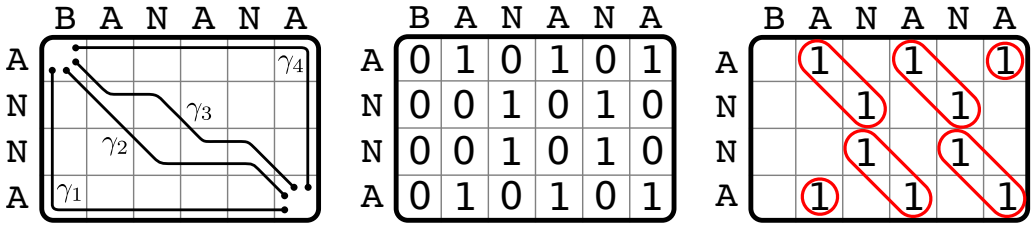

Figure 5.1. Illustration of the concepts behind the Path matrix. The examples are based on the contents of $\boldsymbol{G}(s,t)$ for $s =$"ANNA" and $t =$"BANANA". On the left, the illustration of a small amount of paths which travel $\boldsymbol{G}$. The Global Alignment Kernel is an example of the many kernels which may use these paths to collect data from the matrix and extract one single value. In the center, the contents of $\boldsymbol{G}(s,t)$, assuming $k_\Sigma(\alpha,\beta) = \delta_{\alpha\beta}$, i.e. Kroenecker's delta function (do not confuse with the notation for steps). On the right, the corresponding highlighted diagonals whose number, length and position relate with the similarity between subsequences of s and t.

The contents of $\boldsymbol{G}$ can be interpreted and combined using various levels of refinement. A crude method, justified by the fact that higher values imply a higher correspondence between the inputs, consists in taking the sum of the elements. Incidentally, this gives expression to a valid kernel function, the *Sum* kernel.

Definition 11 (Sum Kernel)**.** *The* Sum *kernel is defined as*

$$k_{SUM}(s,t) = \sum_{i,j} k_\Sigma(s_i, t_j). \tag{5.2}$$

Its associated mapping is

$$\phi_{SUM}(s) = \sum_{i=1}^{|s|} \phi_\Sigma(s_i). \tag{5.3}$$

However, this simplistic solution does not adequately take into account the inputs' structures. Sequences are handled using first order statistics which ignore the actual spatial disposition of the symbols, i.e. sequences are treated as sets of symbols rather than something more. The approach is thus suboptimal.

How to Use the Path Matrix

Instead, the contents of the Path matrix should be considered using more refined techniques. The main argument supported in this project is that the Path matrix contains all the information which is needed to precisely describe the relationship between two sequences. Its dimensionality, its values, the absolute position of each value and their relative positions are all sources of information which adequately describe the inputs.

Moreover, the following observation serves as main motivation of the proposed methods: A considerable amount of information is expressed by the presence of high-valued diagonals in the Path matrix. These diagonals are evidence of a strong

affinity between sections of the inputs. Their number, length and position are decisive characteristics of the Path matrix, and a kernel which focuses on these aspects is expected to achieve promising results.

5.1.2 Path Concatenation and other Path Sets

Definition 12 (d-Diagonal Path Subset)**.** *We define the* d-Diagonal Path Subset $\Gamma^{(d)}$ *as the subset of* Γ *containing paths which use exactly* d *diagonal steps, i.e.*

$$\Gamma^{(d)}(m,n) = \{\gamma \in \Gamma(m,n) \mid \sum_{i=2}^{|\gamma|} \mathbb{I}\left[\gamma\prime(i) = \delta_{11}\right] = d\}. \tag{5.4}$$

The set's cardinality can be expressed as a multinomial coefficient, which represents the number of distinct dispositions of $m-1-d$ *vertical,* $n-1-d$ *horizontal and* d *diagonal steps:*

$$\begin{aligned}\left|\Gamma^{(d)}(m,n)\right| &= (m-1-d, n-1-d, d)! \\ &= \binom{m+n-2-d}{m-1-d, n-1-d, d} \\ &= \frac{(m+n-2-d)!}{(m-1-d)!\,(n-1-d)!\,d!}\end{aligned}$$

The d-Diagonal Path Subsets characterised by the same input dimensions are disjoint and finite, and are related with Γ through the following equation.

$$\Gamma(m,n) = \bigcup_{d=0}^{\min(m,n)-1} \Gamma^{(d)}(m,n). \tag{5.5}$$

Definition 13 (Path Concatenation)**.** *We define as* concatenation *the operation* $(\cdot) : \Gamma(m,n) \times \Gamma(\tilde{m},\tilde{n}) \to \Gamma(m+\tilde{m}-1, n+\tilde{n}-1)$*.* $(\gamma,\tilde{\gamma})$ *is the path obtained by concatenating the steps used by* γ *and* $\tilde{\gamma}$*, preserving their sequential order, i.e.*

$$(\gamma,\tilde{\gamma})\prime = (\delta_{00}, \delta_2, \ldots, \delta_{|\gamma|}, \tilde{\delta}_2, \ldots, \tilde{\delta}_{|\tilde{\gamma}|}). \tag{5.6}$$

The resulting length is $|(\gamma,\tilde{\gamma})| = |\gamma| + |\tilde{\gamma}| - 1$*.*

We furthermore extend the concatenation to accept inputs in the form of either single steps, paths or path sets. The interpretation of a step is that of a path containing only that step, whereas the interpretation of a set as input is that of performing concatenation onto each of its paths, i.e. the result itself will be a path set.

Theorem 4 (Path Set Concatenation Cardinality)**.** *In the case of concatenation between sets of paths* P_1 *and* P_2*, the cardinality of the resulting set is obtained as*

$$|(P_1, P_2)| = |P_1|\,|P_2|\,. \tag{5.7}$$

Definition 14 (Pinned Path Set)**.** *The* Pinned Path Set $\Gamma^{(ij)}$ *is defined as the subset of* Γ *containing paths which pass through* (i,j)*, which is called the* pin*, i.e.*

$$\Gamma^{(ij)}(m,n) = \{\gamma \in \Gamma(m,n) \,|\, \exists k\, \gamma(k) = (i,j)\}. \tag{5.8}$$

Given a path $\gamma \in \Gamma^{(ij)}$*, the index* p *such that* $\gamma(p) = (i,j)$ *is called the* pin index*. Equivalently,* $\Gamma^{(ij)}$ *can also be defined using concatenation, as follows:*

$$\Gamma^{(ij)}(m,n) = (\Gamma(i,j), \Gamma(m-i+1, n-j+1)). \tag{5.9}$$

Definition 15 (Partially d-Diagonal Path Set)**.** *The* Partially d-Diagonal Path Set $\Gamma^{(ijd)}$ *is defined as the subset of* Γ *containing paths which pass through* (i,j) *and which use* d *diagonal steps* before *arriving there*[1]*, i.e.*

$$\Gamma^{(ijd)}(m,n) = \{\gamma \in \Gamma(m,n) \,|\, \exists k\, \gamma(k) = (i,j) \wedge \sum_{i=2}^{k} \mathbb{I}\left[\gamma\prime(i) = \delta_{11}\right] = d\}. \tag{5.10}$$

Equivalently, $\Gamma^{(ijd)}$ *can also be defined using concatenation, as follows:*

$$\Gamma^{(ijd)}(m,n) = (\Gamma^{(d)}(i,j), \Gamma(m-i+1, n-j+1)). \tag{5.11}$$

Similarly to the d-Diagonal path set which was a partition of the Complete path set, this set represents a partition of the Pinned path set:

$$\Gamma^{(ij)}(m,n) = \bigcup_{i=0}^{\min(i,j)-1} \Gamma^{(ijd)}(m,n). \tag{5.12}$$

Definition 16 (Elemental Path Decomposition)**.** *Assuming to exclude the "degenerate" case where* $m = 1$ *and* $n = 1$*, it is always possible to decompose any path set* Γ *in three partitions* Γ_{10}*,* Γ_{01} *and* Γ_{11} *which contain paths that start with* δ_{10}*,* δ_{01} *and* δ_{11} *respectively. This can be expressed using concatenation through the following equation:*

$$\Gamma = (\delta_{10}, \Gamma_{10}) \cup (\delta_{01}, \Gamma_{01}) \cup (\delta_{11}, \Gamma_{11}). \tag{5.13}$$

Because the concatenations in (5.13) *are disjoint, this also implies*

$$|\Gamma| = |\Gamma_{10}| + |\Gamma_{01}| + |\Gamma_{11}|\,. \tag{5.14}$$

[1] *The number of diagonal steps* after (i,j) *is unimportant.*

5.2 The Path Kernel

The objective of the *Path kernel* is to introduce a weighting of the paths based on the number of diagonal and off-diagonal steps taken by each path.

Definition 17 (Path Kernel)**.** *The* Path *kernel is defined as*

$$k_{PATH}(s,t) = \begin{cases} k_\Sigma(s_1,t_1) & \\ \quad + C_{HV} k_{PATH}(s_{2:},t) & \\ \quad + C_{HV} k_{PATH}(s,t_{2:}) & |s| \geq 1, |t| \geq 1, \\ \quad + C_D k_{PATH}(s_{2:},t_{2:}) & \\ 0 & \textit{otherwise}, \end{cases} \tag{5.15}$$

where C_{HV} and C_D represent weights assigned to vertical and horizontal steps and diagonal steps respectively. Constraints such that $C_{HV} > 0$ and $C_D > C_{HV}$ are aimed at enforcing the preference over paths with many diagonal steps. Occasionally, a different notation consisting of $C_{\delta_{10}}$, $C_{\delta_{01}}$ and $C_{\delta_{11}}$ might be used to refer to the same weights, where $C_{\delta_{10}} = C_{\delta_{01}} = C_{HV}$ and $C_{\delta_{11}} = C_D$.

Although the symmetry of k_{PATH} is easily verifiable, whether it is also psd is still uncertain as a rigorous proof is still missing. However, during the experimental stages, the kernel has shown to be experimentally psd, i.e. to always yield psd kernel matrices when used on the involved datasets. This is a good indication that, although potentially non psd, k_{PATH} may still be used as a useful kernel function.

5.2.1 Alternative Formulations

What follows are other alternative formulations of the kernel which will prove very useful for understanding how k_{PATH} actually makes use of the concept of paths, and for other reasons related with computational efficiency.

Theorem 5 (Degenerate Path Kernel)**.** *Whenever one of the inputs has unitary length, k_{PATH} reduces to a simpler formulation which will be useful to prove other theorems regarding it. The formulation, assuming $|t| = 1$, is the following.*

$$k_{PATH}(s,t) = \sum_{i=1}^{|s|} C_{HV}^{i-1} k_\Sigma(s_i,t_1). \tag{5.16}$$

The kernel is symmetric, so a similar expression applies for $|s| = 1$.

Proof.

$$\begin{aligned} k_{\text{PATH}}(s,t) &= k_\Sigma(s_1,t_1) + C_{HV}k(s_{2:},t) \\ &= k_\Sigma(s_1,t_1) + C_{HV}k_\Sigma(s_2,t_1) + C_{HV}^2 k(s_{3:},t) \\ &= k_\Sigma(s_1,t_1) + C_{HV}k_\Sigma(s_2,t_1) + C_{HV}^2 k_\Sigma(s_3,t) + \ldots + C_{HV}^{|s|-1}k_\Sigma(s_{|s|},t_1) \\ &= \sum_{i=1}^{|s|} C_{HV}^{i-1} k_\Sigma(s_i,t_1) \end{aligned}$$

□

Theorem 6 (Path Perspective)**.** *The value calculated by k_{PATH} for inputs s and t with lengths m and n can also be expressed using paths on $\boldsymbol{G}(s,t)$.*

$$k_{PATH}(s,t) = \sum_{\gamma\in\Gamma}\sum_{i=1}^{|\gamma|} \boldsymbol{G}(s,t)_{\gamma(i)} \frac{\prod_{k=2}^{i} C_{\gamma'(k)}}{\mathrm{D}(m-\gamma_X(i)+1, n-\gamma_Y(i)+1)} \tag{5.17}$$

Proof. The proof for this theorem spreads through multiple pages. For the sake of concision, the complete proof is omitted from the main corpus and can be found in Appendix A. What follows is the inductive procedure used, without the details on the single steps.

The proof makes use of the following inductive steps:

Base Case I The statement is valid for $|s| = m$ and $|t| = 1$.

Base Case II The statement is valid for $|s| = 1$ and $|t| = n$.

Inductive Step Assuming the statement to be valid for $|s| = m-1$, $|t| = n-1$; for $|s| = m-1$, $|t| = n$; and for $|s| = m$, $|t| = n-1$; then it is valid also for $|s| = m$ and $|t| = n$.

□

Expanding (5.15), it becomes obvious that k_{PATH} represents a weighted sum over all elements $k_\Sigma(s_i,t_j)$, i.e. $\boldsymbol{G}_{ij}$. The following theorem gives an answer to what are the weights associated with each element.

Theorem 7 (Grid Perspective)**.** *The value calculated by k_{PATH} for inputs s and t with lengths m and n can also be expressed using a weighted sum on $\boldsymbol{G}(s,t)$.*

$$k_{PATH}(s,t) = \sum_{i,j=1}^{m,n} \boldsymbol{G}(s,t)_{ij}\boldsymbol{\omega}_{PATHij} \tag{5.18}$$

where $\boldsymbol{\omega}_{PATH} \in \mathbb{R}^{m\times n}$ and

$$[\boldsymbol{\omega}_{PATH}]_{ij} = \sum_{d=0}^{\min(i,j)-1} C_{HV}^{i+j-2-2d} C_D^d \, (d, i-1-d, j-1-d)! \tag{5.19}$$

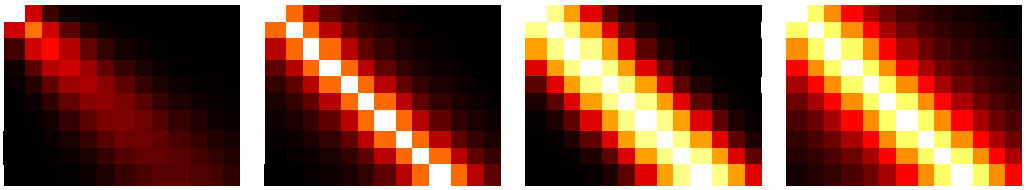

Figure 5.2. Weight matrices relative to inputs with lengths 10 and 12. From left to right: $\boldsymbol{\omega}_{\text{PATH}}$, $\boldsymbol{\omega}_{\text{SS}}$, $\boldsymbol{\omega}_{\text{POS}}$ using a Gaussian kernel as integer kernel, and $\boldsymbol{\omega}_{\text{POS}}$ using a Cauchy kernel as integer kernel. Along with the fact that all these kernels can be expressed by a weight matrix, the aspect of the weight matrices themselves are evidence of the similarity between k_{PATH}, k_{SS} and k_{POS}.

Proof. The proof is based on the result of Theorem 6, which is restrained to find the expression for the weight assigned to the generic element $\boldsymbol{G}_{ij}$. The complete proof can be found in Appendix A. □

Theorem 7 describes the path kernel in terms of the *weight* matrix $\boldsymbol{\omega}_{\text{PATH}}$. Shown in Figure 5.2, $\boldsymbol{\omega}_{\text{PATH}}$ can be used, as described in §5.7, as a mean to achieve particularly fast computations of the kernel on whole datasets. For now, the following section describes an efficient method to elaborate the kernel just once.

5.2.2 Computation of the Path Kernel

Albeit being an elegant mean to solve a problem, recursion is a notoriously non efficient process. Computing k_{PATH} through means of its original formulation in (5.15), which does involve recursion, represents a considerable deficiency with respect to both k_{DTW} and k_{GA}, both of which have computational complexities which scale as $O(|s|\,|t|)^2$.

However, it is possible to compute k_{PATH} using a dynamic programming algorithm which is computationally comparable to both k_{DTW} and k_{GA}. The algorithm is the following:

```
let M = G(s,t)
for i = 1 ... |s|
    for j = 1 ... |t|
        if i > 1
            M(i,j) = M(i,j) + CHV M(i-1,j)
        if j > 1
            M(i,j) = M(i,j) + CHV M(i,j-1)
        if i > 1 and j > 1
            M(i,j) = M(i,j) + CD M(i-1,j-1)
output = M(|s|,|t|)
```

Furthermore, an even faster computation can be achieved by adequately exploiting Theorem 7. The topic of a faster computation also interests the other

[2]The Triangular Global Alignment kernel k_{TGA} has a higher efficiency at the cost of being an approximation of k_{GA}, and is thus not taken into consideration.

kernels which are presented in the next sections, and will thus be discusses more thoroughly in §5.7. However, because the method is based on the pre-computation of the weight matrix $\boldsymbol{\omega}_{\text{PATH}}$, it is necessary to provide a feasible way to compute it up to an arbitrary dimension of $m \times n$. This can be achieved efficiently using a dynamic programming algorithm similar to the above:

```
let W = zeros(|s|,|t|)
let W(1,1) = 1
for i = 1 ... |s|
    for j = 1 ... |t|
        if i > 1
            W(i,j) = W(i,j) + CHV W(i-1,j)
        if j > 1
            W(i,j) = W(i,j) + CHV W(i,j-1)
        if i > 1 and j > 1
            W(i,j) = W(i,j) + CD W(i-1,j-1)
output = W
```

5.3 Subsequence Sum Kernel

Equation (5.2) introduced the *Sum* kernel, which was quickly set aside because of the scarce use it makes of the input structures. In this section, multiple instances of k_{SUM} are used onto the subsequences of the inputs. The result can be described in terms of a certain number of overlapping layers. The *Subsequence Sum* kernel takes into consideration the contribution of each of these layers as follows.

Definition 18 (Subsequence Sum Kernel)**.**

$$k_{SS}(s,t) = \sum_{l=0}^{\infty} \omega_l k_{SS}^{(l)}(s,t), \tag{5.20}$$

where

$$k_{SS}^{(l)}(s,t) = \begin{cases} k_{SUM}(s,t) & l = 0, \\ \sum_{p=-\infty}^{\infty} k_{SUM}(s_{p:p+l-1}, t_{p:p+l-1}) & 0 < l \leq \min(|s|,|t|), \\ 0 & l > \min(|s|,|t|). \end{cases} \tag{5.21}$$

The layers are numbered, and $k_{SS}^{(l)}$ represents the value associated with layer l. The weights ω_l represent an infinite number of parameters to k_{SS}. In practice, there is always an upper bound to the number of layers really used when evaluating the kernel, and that is $L_m = \min(|s|,|t|)$.

Each index l is associated with a *layer* which evaluates k_{SUM} on subsequences with length l. The exception to this rule is the 0-layer, which by default uses all the symbols in both inputs, regardless of their length. As depicted in Figure 5.3, these layers are easily interpreted as weights associated with the elements of $\boldsymbol{G}$, once again giving rise to the possibility of describing this kernel using a weight matrix.

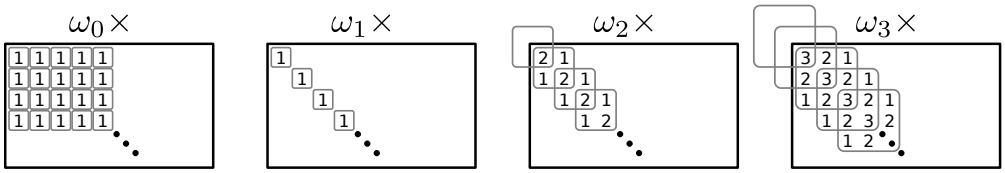

Figure 5.3. Illustration of the first four layers of k_{SS} onto the Path matrix. Each square depicts the effect of k_{SUM} on a different subsequence of the inputs. To ensure positive semi-definiteness, it is necessary for each subsequence to be coordinated such that starting position and length is the same for both inputs. On the left, the 0-layer ensures that all symbols contribute to the output. The other figures show the 1-layer, the 2-layer and the 3-layer, which consider subsequences with length 1, 2 and 3 respectively.

Theorem 8 (Subsequence Sum Kernel Validity)**.** *k_{SS} is psd if the symbol kernel k_Σ is also psd. The mapping associated with k_{SS} is the weighted* direct sum *($\oplus$) of the* Sum *kernel's mapping on different subsequences, as follows:*

$$\phi_{SS}(s) = \bigoplus_{l=0}^{\infty} \sqrt{\omega_l}\phi_{SS}^{(l)}(s), \tag{5.22}$$

where

$$\phi_{SS}^{(l)}(s) = \begin{cases} \phi_{SUM}(s) & l = 0 \\ \bigoplus_{p=-\infty}^{\infty} \phi_{SUM}(s_{p:p+l-1}) & 0 < l \leq |s| \\ 0 & l > |s| \end{cases} \tag{5.23}$$

Once again, the computation of the kernel results impractical if the original formulation were to be executed. Instead, one can use the following alternative formulation which, based on a weight matrix, yields the same result. The weight matrix can be visually constructed using the overlapping layers as depicted in Figure 5.3.

Theorem 9.

$$k_{SS}(s,t) = \sum_{ij} \boldsymbol{G}(s,t)_{ij}\, \boldsymbol{\omega}_{SS}^{(L_m)}{}_{ij} \tag{5.24}$$

where $L_m = \min(|s|, |t|)$ *and*

$$[\boldsymbol{\omega}_{SS}^{(L_m)}]_{ij} = \omega_0 + \sum_{l=1}^{L_m} \omega_l \max(0, l - |i-j|). \tag{5.25}$$

Taking into account that the weight matrix is symmetric and that the elements in each diagonal are equivalent, it is possible to compute any $\boldsymbol{\omega}_{\text{SS}}^{(L_m)}$ in $O(L_m)$. An exemplary weight matrix is shown in Figure 5.2.

Noise Issues

The construction of the weight matrix is based on the overlap of a number of squared contributions which, being centered on the main diagonal of the matrix, tend to create a steep cuspid-like barrier along the main diagonal. The steepness of such a weighting structure implies that small changes in the position of the inputs' symbols may result in a considerable change to the kernel's output. It thus becomes reasonable to hypothesise that k_{SS} may not be particularly robust against noise. Although not an effective measure against this issue, the weights ω_l can be manipulated to smoothen the steepness (without getting rid of it) of the weight matrix. For example, the following scheme may be used to decrease the weights for an increasing number of layer:

$$\omega_l = \begin{cases} 1 & l = 0, \\ l^{-\log(l)} & l > 0. \end{cases} \tag{5.26}$$

5.4 Positional Kernel

The *Positional* kernel is able to provide a higher resistance against noisy data using a method similarly used by Triangular Global Alignment kernel. The principle is to decouple each symbol s_i as two separate sources of information, namely its value and its position. The Positional kernel separates these sources into (s_i, i) and analyses them using a symbol kernel and an *Integer* kernel respectively.

Definition 19 (Positional Kernel)**.**

$$k_{POS}(s,t) = \sum_{ij} \boldsymbol{G}(s,t)_{ij} k_{INT}(i,j). \tag{5.27}$$

Theorem 10 (Positional Kernel Validity)**.** *k_{POS} is psd if both k_Σ and k_{INT} are psd. The mapping associated with k_{POS} is the* outer product *($\otimes$) between ϕ_Σ and ϕ_{INT}, summed over the sequence's symbols, i.e.*

$$\phi_{POS}(s) = \sum_i \phi_\Sigma(s_i) \otimes \phi_{INT}(i). \tag{5.28}$$

Proof.

$$\begin{aligned} k_{\mathrm{POS}}(s,t) &= \langle \phi_{\mathrm{POS}}(s), \phi_{\mathrm{POS}}(t) \rangle \\ &= \langle \sum_i \phi_\Sigma(s_i) \otimes \phi_{\mathrm{INT}}(i), \sum_j \phi_\Sigma(t_j) \otimes \phi_{\mathrm{INT}}(j) \rangle \\ &= \sum_{ij} \langle \phi_\Sigma(s_i) \otimes \phi_{\mathrm{INT}}(i), \phi_\Sigma(t_j) \otimes \phi_{\mathrm{INT}}(j) \rangle \\ &= \sum_{ij} \langle \phi_\Sigma(s_i), \phi_\Sigma(t_j) \rangle \langle \phi_{\mathrm{INT}}(i), \phi_{\mathrm{INT}}(j) \rangle \end{aligned}$$

$$= \sum_{ij} k_\Sigma(s_i, t_j) k_{\text{INT}}(i, j)$$
$$= \sum_{ij} \boldsymbol{G}(s, t)_{ij} k_{\text{INT}}(i, j). \qquad \square$$

The role of k_{INT} is that of encoding the similarities between the symbols' positions. For this reason, it is advised for it to be a radial basis kernel. To furthermore improve noise resistance, which was lacking from the previous *Subsequence Sum* kernel, it is advised for k_{INT} to maintain a certain degree of smoothness avoiding steep derivatives. The Gaussian kernel and the Cauchy kernel, used with an adequate amount of variance, are good practical examples which address both issues.

Equation (5.27) opens a direct interpretation of the values of k_{INT} as the values of a weight matrix $[\boldsymbol{\omega}_{\text{POS}}]_{ij} = k_{\text{INT}}(i, j)$ through which, similarly to the previous kernels, k_{POS} can be computed as a weighted sum over $\boldsymbol{G}$. Figure 5.2 shows the positional kernel's weight matrices when using a Gaussian or Cauchy kernel. Although developed following completely different approaches, a striking similarity can be noticed between k_{SS} and k_{POS}, with the main difference being the steepness of the elements in the respective weight matrices.

5.5 Ground Kernel Choice

The idea behind k_Σ is that it can be adjusted to fit the task at hand. Apart from it being a valid kernel function, which is required to guarantee that both k_{SS} and k_{POS} are psd, there have been no further constraints. However, an arbitrarily selected k_Σ may lead to undesired behaviours.

Example 4. *Assume an alphabet Σ containing (at least) two symbols α and β. Given the input sequences $s = (\alpha, \beta, \ldots, \alpha, \beta)$ and $t = (\beta, \alpha, \ldots, \beta, \alpha)$, is it reasonable to expect a good sequence kernel to find a high similarity between s and t. However, consider the use of a ground kernel k_Σ such that $k_\Sigma(\alpha, \alpha) = 1$, $k_\Sigma(\beta, \beta) = 1$ and $k_\Sigma(\alpha, \beta) = -1$. The resulting Path matrix shows a collection of ones and negative ones organized in a chessboard-like structure, i.e.*

$$\boldsymbol{G}(s, t) = \begin{bmatrix} -1 & 1 & \ldots & -1 & 1 \\ 1 & -1 & \ldots & 1 & -1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & 1 & \ldots & -1 & 1 \\ 1 & -1 & \ldots & 1 & -1 \end{bmatrix} \tag{5.29}$$

Consider any of the proposed kernels: The effect of such a ground kernel used on these inputs is that most of the values in the Path matrix, even after being weights, will erase each other out determining at the same time a slight dissimilarity and the semi-orthogonality between the inputs. Both of there are considered undesirable under the initial assumption that the inputs are, in fact, similar. Furthermore, the

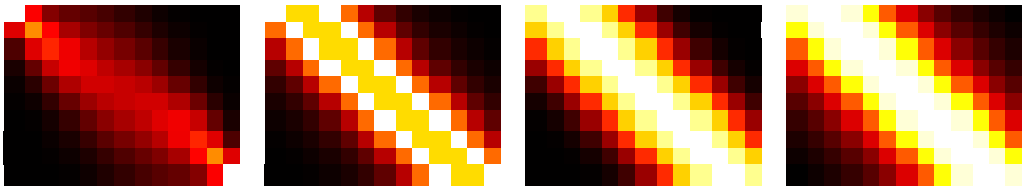

Figure 5.4. Inversion invariant weight matrices relative to Figure 5.2.

semi-orthogonality is also encountered in the values of $k(s,s)$ *and* $k(t,t)$*, which may be considered a more important effect.*

This particular issue is easily dismissed by enforcing a constraint on the ground kernel such that it may only yield non-negative results. However, the purpose of this brief section is to point out that there may also be other undesired behaviours which arise for particular instances of k_Σ apart from the one described.

5.6 Inversion-Invariant Kernels

In the list of the ideal characteristics of a sequential kernel, in §4.2, the last point considered the behaviour of the kernel when faced with inverted sequences. A kernel was described to be *inversion invariant* if $k(s,t) = k(\mathrm{inv}(s), \mathrm{inv}(t))$. Originally, neither k_{PATH}, k_{SS} nor k_{POS} are inversion-invariant. However, it is easily possible to achieve an inversion-invariant kernel which preserves positive definiteness and characteristics of the input kernel.

Definition 20 (Inversion-Invariant Kernel)**.** *The* inversion-invariant *version of* k *is defined as*

$$\mathrm{inv}(k)\,(s,t) = \frac{k(s,t) + k(\mathrm{inv}(s), \mathrm{inv}(t))}{2}. \tag{5.30}$$

The resulting function $\mathrm{inv}(k)$ *is inversion-invariant. If* k *is already inversion-invariant, the operation does not alter the result.*

The inversion-invariant copies of the proposed kernels can also be achieved by modifying the weight matrices accordingly through an operation which is referred to as *flipping*. Given a weight matrix $\boldsymbol{\omega}$ with the correct dimensions, flipping resolves to the following operation.

$$\boldsymbol{\omega}_{ij} = \frac{\boldsymbol{\omega}_{ij} + \boldsymbol{\omega}_{|s|-i+1,|t|-j+1}}{2} \tag{5.31}$$

Using the presented kernels with a *flipped* weight matrix in place of the original one achieves the same result as the inversion-invariant version of the kernel itself. Using this trick, it is possible to avoid computing the kernel twice. Figure 5.4 illustrates the flipped weight matrices previously encountered in Figure 5.2.

The experiments carried out in Chapter 6 only focus on the inversion-invariant versions of the proposed kernels.

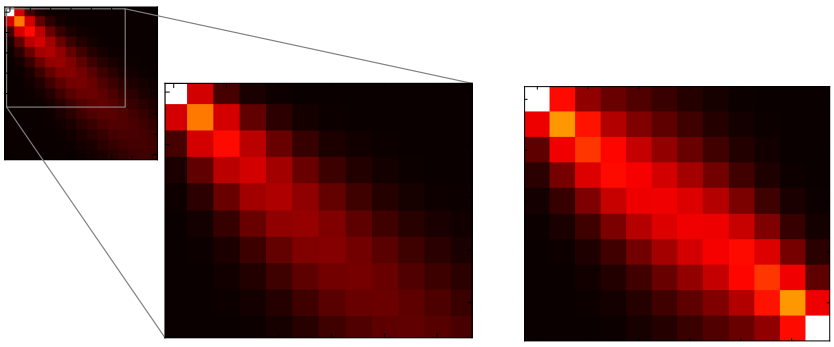

Figure 5.5. Example weight matrix $\boldsymbol{\omega}_{\text{PATH}}$ construction when using the fast computation method. On the left, a precomputed 15×15 weight matrix is used to select a 10×12 weight sub-matrix which can practically be used to elaborate $k_{\text{PATH}}(s,t)$ for input sizes $|s| = 10$ and $|t| = 12$. On the right, the inversion invariant $\tilde{\boldsymbol{\omega}}_{PATH}$ correspondent to $\tilde{k}_{PATH}$ for the same input sizes.

5.7 Fast Computation on Batches of Sequences

Kernel Methods typically require the computation of the kernel function on all pair-wise combinations of sequences from a dataset S. This implies that the number of kernel evaluations grows quadratically with the size of S, and thus requires particular care and optimisation to avoid unnecessary efficiency leaks.

All of the proposed kernels have been expressed, at some point, as a weighted sum over the Path matrix through means of a weight matrix $\boldsymbol{\omega}$. Using this as a starting base, it is possible to ulteriorly change the expression to one which uses the *trace* of the multiplication between $\boldsymbol{G}$ and $\boldsymbol{\omega}$:

$$\begin{aligned} k(s,t) &= \sum_{ij} \boldsymbol{G}(s,t)_{ij} \boldsymbol{\omega}_{ij} \\ &= \operatorname{Tr}\left[\boldsymbol{G}(s,t)^{\intercal} \boldsymbol{\omega}\right]. \end{aligned} \tag{5.32}$$

Equation (5.32) allows the implementation of a *trick* which considerably enhances the efficiency of the kernels' evaluation. Being the elements in the weight matrix $\boldsymbol{\omega}$ not depend on the input sequences themselves[3], it becomes possible to perform a pre-computation stage where the matrix is computed up to an adequate size, which depends on the length of the sequences in the dataset S. The enhancement is implemented as follows:

Preprocessing Stage

Pre-compute $\boldsymbol{\omega}$ up to a sufficient dimension:

```
let max_l = max(|s| for s in S)
W = precompute_weights(max_l)
```

[3] $\boldsymbol{\omega}_{\text{SS}}^{(L_m)}$, being dependent on the inputs' lengths is an exception. This is not, we will see, an unsurmountable obstacle.

Computation Stage

To perform evaluation for input sequences s and t, crop the appropriate sub-matrix from the preprocessed matrix:

```
W_temp = W(1:|s|,1:|t|)
output = Trace(G(s,t)'W_temp)
```

Given the pre-existing weight matrix, it is possible to perform kernel evaluations at the same computational cost of matrix multiplication[4]. Coupling . Furthermore, it is possible to embed the use of inversion-invariant kernels in this fast computation enhancement by *flipping* the cropped sub-matrix before using it in the matrix multiplication. Figure 5.5 illustrates the trick and the use of matrix flipping.

Fast Computation for the Subsequence Sum Kernel

Because its weight matrix actually depends on the lengths of the inputs, the Subsequence Sum kernel is not able to use the trick as it is. Instead, to make use of the same enhancement, it becomes necessary to store multiple pre-processed weight matrices (as many as there are sequences with different lengths, to be precise). At the moment of computation, the length of the input sequences can be used to determine which of the preprocessed matrices should be used. Assuming there is a relatively small variance in the length of the input sequences with respect to the size of the dataset, the procedure is still expected to improve computational times.

[4]The computational complexity of $\mathrm{Tr}\,[\cdot]$ is linear and thus insignificant compared to matrix multiplication.

Chapter 6

Experiments and Results

This section describes the process through which the proposed kernels' performance and characteristics are evaluated qualitatively and quantitatively.

Ideally, this section would contain experiments involving the DTW kernel, the global alignment kernel and the proposed kernels. However, the DTW kernel is only used in one of the tasks because of the issues its non positive semi-definiteness causes with the SVM algorithm. Furthermore, out of the proposed kernels, only the path kernel is experimented due to time-related constraints.

The experiments are divided into three categories: Principal Component analysis (PCA) is used as a sneak-peek into the induced spaces, classification assesses the kernels' discriminatory abilities, whereas regression is used to evaluate the ability to generalise. The data used in the experiments is artificially generated, which allows greater control on the tests' execution. Various datasets, which differ by the number of sequences and amount of noise, are created.

Kernel Parameters

The path kernel has various parameters to choose from: The ground kernel k_Σ and the weights associated with the steps C_{HV}, C_D. Because a complete analysis on all settings would have been impractical, the parameters maintain the same value throughout all the experiments, namely k_Σ is a Gaussian kernel with $\sigma = 0.1$, and the step weights are chosen[1] to be $C_{HV} = 0.3$ and $C_D = 0.34$. The behaviour of the path kernel is most likely to change with the value of these parameters, however, a complete analysis of said change is beyond the time constraints of the project. The consequences are that the path kernel's behaviour might not be optimised during the tests. However, as show in the next sections, this does not compromise the results of the experiments.

[1] C_{HV} and C_D have been chosen qualitatively by observing the contents of the resulting weight matrices. The motivation is that the following experiments do not aim at determining the optimal use of the path kernel, but rather at demonstrating the potentiality of such kernel.

6.1 Artificial Data Generation

All sequences originate from an univariate periodic waveform g with period $P = 50$ bounded by $[-1, 1]$. Three types of noise are considered during the generation: The *length* noise, the *input* noise and the *output* noise, respectively modeled as Gaussians with std σ_l, σ_i and σ_o, and used in the generating process of a sequence s as follows:

i. The *length* noise determines a sequence's length according to $|s| \sim \mathcal{N}(100, \sigma_l^2)$.

ii. An *input* sequence of $|s|$ equidistant numbers between 0 and 100 ($2P$) is created and summed with the *input* noise $\mathcal{N}(0, \sigma_i^2)$, which increases the uncertainty of the inputs to function g.

iii. The input sequence is given as input in g to obtain the *output* sequence, to which the *output* noise $\mathcal{N}(0, \sigma_o^2)$ is summed to yield s.

The PCA and classification tasks considers eight different originating waveforms, namely the *sine*, *cosine*, *sawtooth*, *square* waveforms and their negations. The regression task only considers different phase shifts of the *sine* waveform. Figures 6.1, 6.2 and 6.3 illustrate three datasets, each with 20 sequences per waveform.

6.2 Principal Component Analysis

The PCA task represents a qualitative visualisation of the data embedded in their induced feature spaces. The $2D$ projections of the datasets, shown in Figure 6.4, are obtained by selecting the 2 principal components of their space, i.e. the directions through which most variance is measured. The variance shown on the images is a measure of how much of the datasets' total variance is actually captured by their $2D$ representations. Specifically, k_{PATH}'s $2D$ projections contain around half as much information than the projections of $\log(k_{\text{GA}})$, which means that the space induced by k_{PATH} is more articulated than what is visible through the plots.

6.3 Classification

The classification experiments aim at verifying the discriminability of the sequences in the induced feature spaces. The task uses of SVM classifier by means of a 2-fold cross-validation to separate training and test sets. Each experiment is repeated 50 times and the results averaged to achieve a more robust measure of the performance. Due to issues related with non-psd kernel matrices, the classification fails for k_{DTW}, and thus only the results relative to $\log(k_{\text{GA}})$ and k_{PATH} are reported.

Figure 6.5 shows how the obtained performances scale with respect to the noise levels. The kernels result indistinctive from each other for moderate noise levels (up to $\sigma_{\{l,i,o\}} = 5$). However, increasing the noise produces a rapid decline in the performance of $\log(k_{\text{GA}})$, whereas the effects are less radical for k_{PATH}. In particular, the testcase with 50 sequences per class and a noise level of $\sigma_{\{l,i,o\}} = 9$

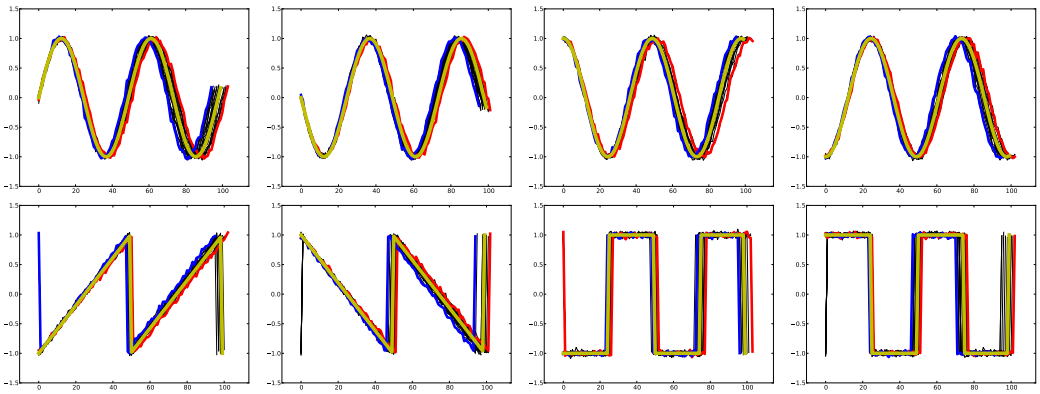

Figure 6.1. Dataset created with noise levels $\sigma_{\{l,i,o\}} = .5$. The yellow sequence represents the generating waveform, whereas the blue and red ones are respectively the shortest and longest sequences in the dataset.

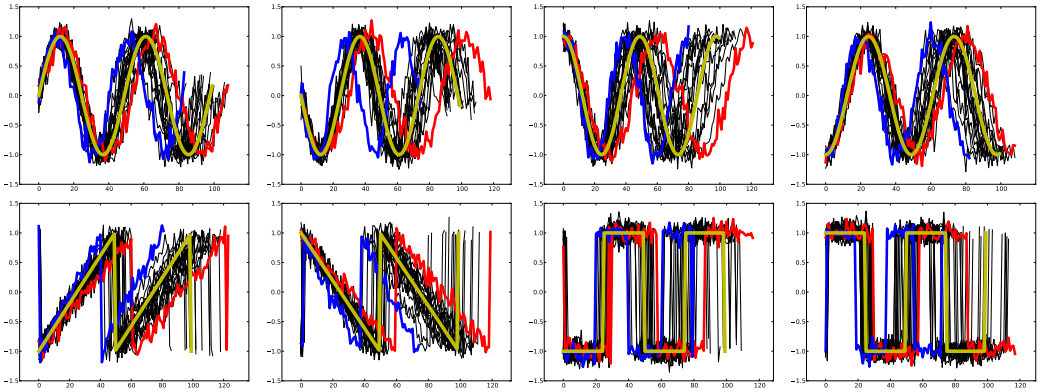

Figure 6.2. Dataset created with noise levels $\sigma_{\{l,i,o\}} = 2$. The yellow sequence represents the generating waveform, whereas the blue and red ones are respectively the shortest and longest sequences in the dataset.

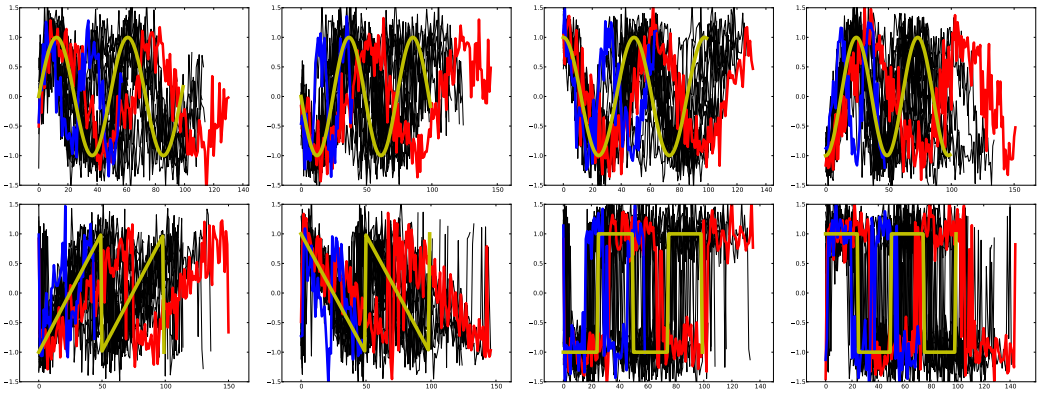

Figure 6.3. Dataset created with noise levels $\sigma_{\{l,i,o\}} = 5$. The yellow sequence represents the generating waveform, whereas the blue and red ones are respectively the shortest and longest sequences in the dataset.

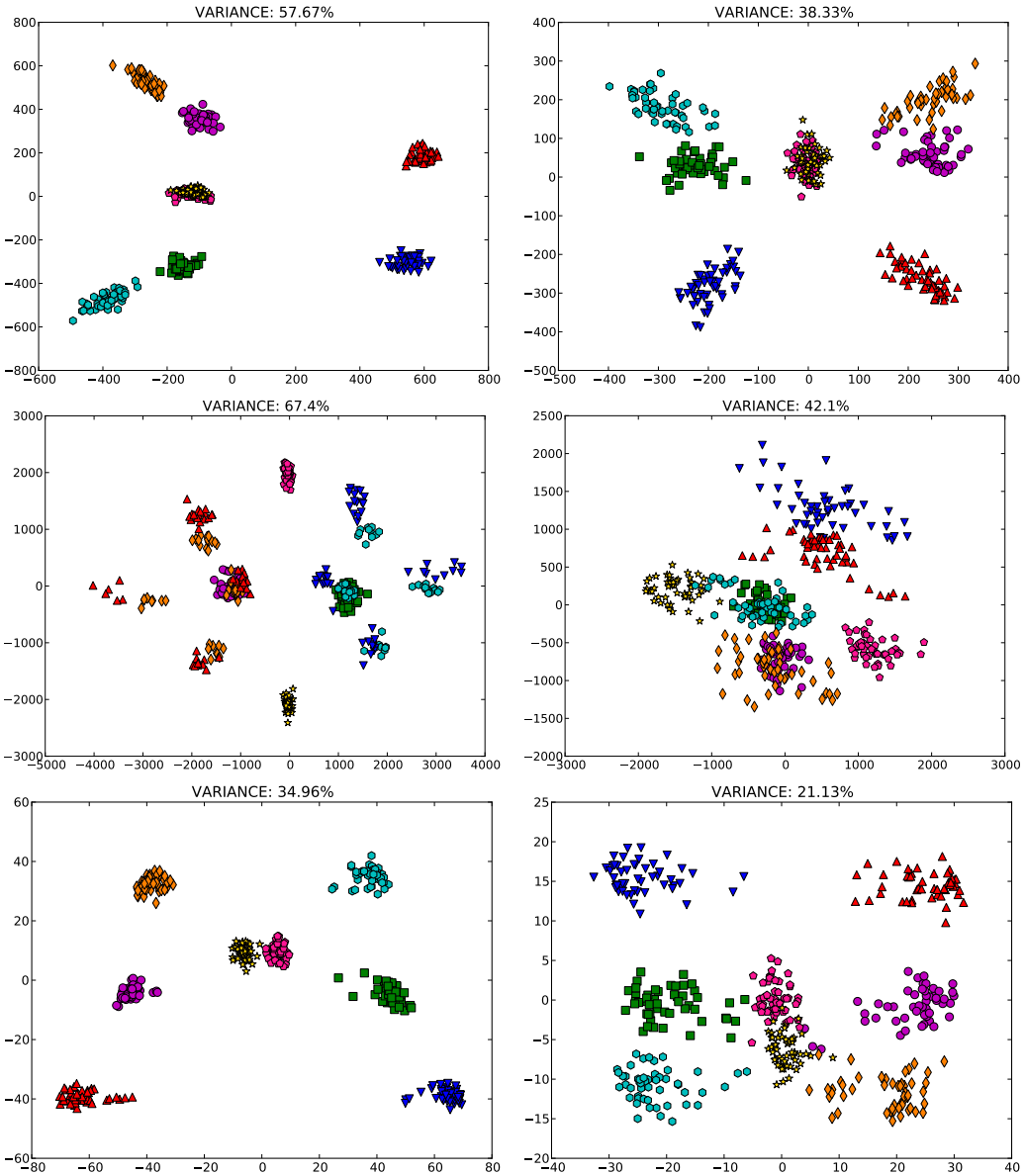

Figure 6.4. The above figure illustrates the 2-dimensional principal component subspace for k_{DTW} (top), $\log(k_{\text{GA}})$ (center) and k_{PATH} (bottom). The data on the left column is characterised by a generating noise of $\sigma_{\{l,i,o\}} = 2$, whereas on the right column the noise is increased to $\sigma_{\{l,i,o\}} = 5$.

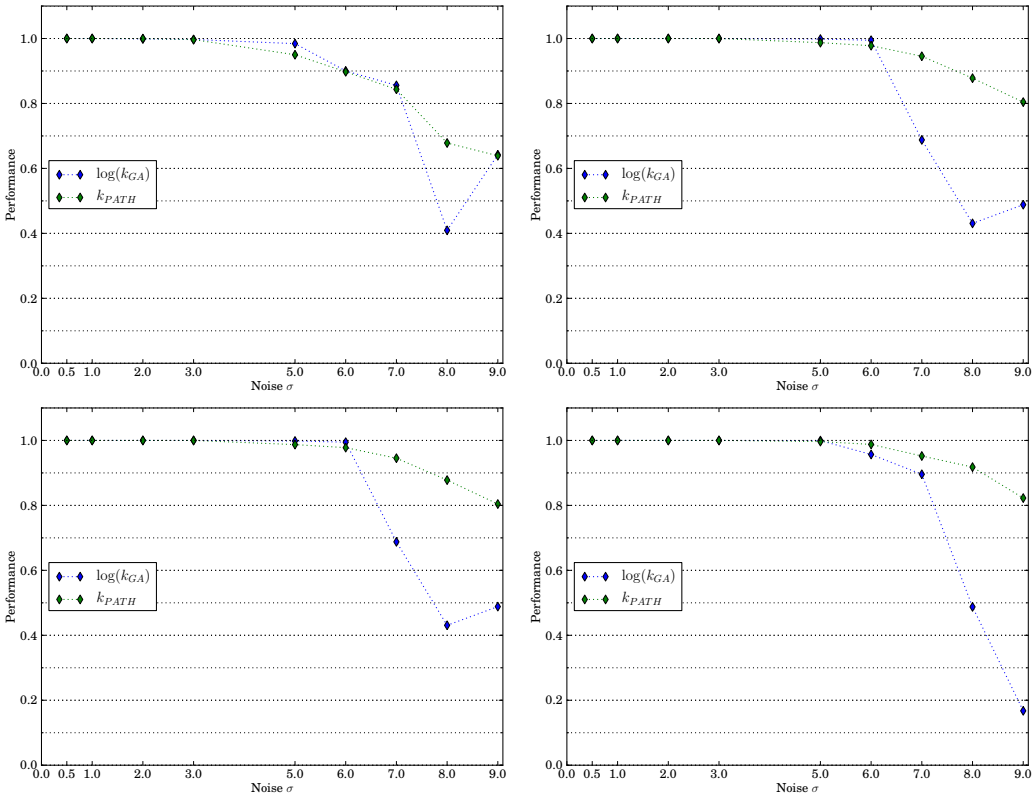

Figure 6.5. Classification rates for the prediction of the generating waveform using a SVM classifier enhanced with the kernels. From top-left to bottom-right, the datasets contain respectively 10, 20, 30 and 50 instances per class.

shows that the global alignment kernel's performance is about chance ($\sim 1/8$ are correctly guessed), whereas the path kernel still achieves over 80% classification rate.

The classification experiments show that the path kernel significantly outperforms the global alignment kernel when noise becomes significant. Referring to the induced spaces depicted in Figure 6.4, the path kernel encodes a feature space characterised by more clearly defined clusters corresponding to the different waveforms, and also has a simpler structure. This should tend to a better generalisation, where it is beneficial to have a large continuous region of support which gracefully describes the variations in the data rather than a space that barely separates the classes, which is generally ideal for classification.

6.4 Regression

The regression task makes use of a new data-set consisting of 100 noisy sine-waves shifted in phase between 0 and 25 ($P/2$). The data is split uniformly into two sets

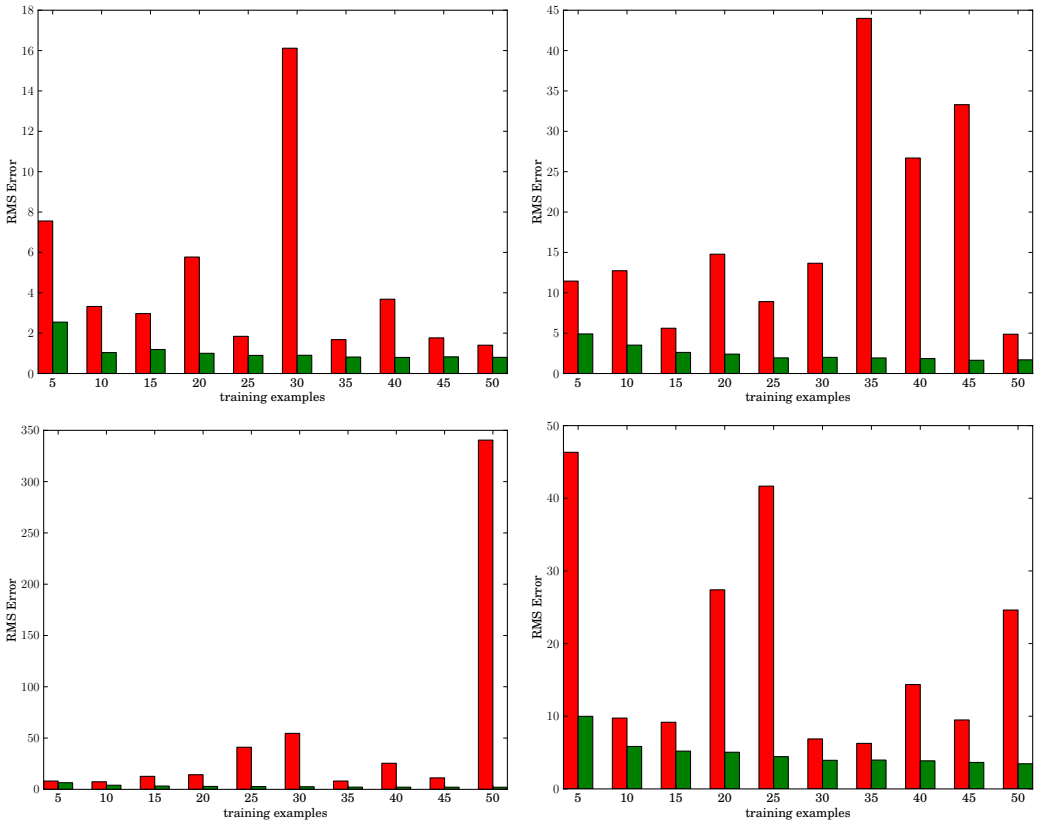

Figure 6.6. The above figure depicts the RMS error when predicting the phase shift from a noisy sine waveform by a regression over the feature space induced by the kernels. From top-left to bottom-right, the plots show the results when dealing with an increasing amount of noise, i.e. $\sigma_{\{l,i,o\}} = 1, 2, 3, 5$. The red bars correspond to the global alignment kernel while the green relates to the path kernel. The y-axis shows the error in percentage of phase while the x-axis indicates the size of the training data-set.

where the first one is used for training and the second for testing. Each experiment differs in the amount of noise used to generate the sequences and the number of actual training sequences used from the training set.

The task is to predict the phase of the sequences in the test set using training sets of different sizes. The prediction is perfomed using simple least-square regression over the kernel induced feature space and the performance is measured in terms of the Root Mean Squared (RMS). In Figure 6.6, the results are shown using different sizes of the training data. The DTW kernel is omitted from the results because of its huge RMS errors, which obstructs the view of the other columns.

The path kernel performs significantly better compared to the global alignment kernel, with results which improve increasingly with the size of the training data-set. Interestingly, the global alignment kernel produces very different results depending on the size of the training data-set, which is a clear indication of severe overfitting.

Chapter 7

Conclusions

Although Kernel Methods have already proven in the last decades to be considerably useful and successful, much of their potentiality is still hidden and unveiled little by little as new kernels are formulated and studied. The family of kernels for sequential data is, in particular, mostly unexplored and thus is open to a great range of potential improvements. The current state-of-the-art still suffers from a series of issues which strictly limit their capabilities. This project aimed at creating novel kernels which, inspired by recent methods, was able to overcome such issues.

So What?

The project resulted in the formulation of a new family of kernels based on the contents of a *Path* matrix. An important characteristic of the developed kernels, which has a considerable impact on their usability and properties, is that they can all be described as various forms of weighted summations over the Path matrix. This allowed the formulation of an efficient computation which achieves speed rates comparable to those of a matrix multiplication when used on datasets of sequences.

A number of experimental results confirm that the proposed methods are characterised by higher levels of robustness and descriptiveness: The path kernel achieves superior performance in terms of discrimination and generalisation capabilities when tested against the state-of-the-art. Although no experiments were carried out for the subsequence sum kernel and the positional kernel, the similarity between their associated weight matrices is evidence of a certain degree of similarity between the kernels themselves, the main difference being that the last two kernels are demonstrably psd, thus giving a solid geometrical interpretation to the kernels' projections.

But..

Nonetheless, the proposed kernels are afflicted by issues of their own: The path kernel gathers most of its information from the start and the end of sequences, and is thus inclined at underrating possible matches which appear elsewhere; Similarly, the subsequence sum kernel focuses mainly on the contents aligned with the main

diagonal of the Path matrix, and is thus likely to underrate possible matches which are even slightly misaligned; The positional kernel is not afflicted as much by this issue, although it can not be said that it is immune to it either[1]; Finally, the proposed kernels show[2] a deficiency when trying to correctly match input sequences with substantially different lengths.

An argument developed near the end of the project is that these issues are likely related to a common property of the kernels: The possibility of being formulated in terms of a weight matrix. Although this same property gives rise to the trick for fast computations on data batches, it is also evidence of the kernels' "stationarity". The presence of a weight matrix which weights the elements of the path matrix solely based on their position rather than the surroundings is a contradiction to the intuition through which the high-valued *diagonals* are the key entities to focus on. Being a *diagonal* composed of several elements, it is unlikely that a method which considers each element individually can achieve the intended goal.

What Now?

The Path matrix contains a lot of knowledge which describe the relationship between two sequences. The matrix itself is, in that sense, considerably dynamic: Its dimensions; its elements; their absolute and relative disposition; all these allows for innumerable ways to extract relevant information. Overall, the products of my project have only scratched the surface of what can be achieved using such a structure. A considerable amount of improvements can still be devised and studied. The following is a list of plausible directions which might be considered for future developments:

i) More dynamic and refined kernels can be developed, which consider the relations between the elements in the Path matrix rather than computing a weighted summation over it.

ii) The Path matrix may be used as an intermediary mean to define a secondary structure from which the kernels actually depend.

iii) A different structure which makes a different use of the symbols in the input sequences may be developed and adopted to substitute the Path matrix.

iv) Some form of normalisation may be developed to deal with the kernels' issues regarding the inputs' lengths.

v) New experiments should evaluate all the proposed kernels' performance using non-artificial but rather real-world data.

[1] Moreover, even using a Gaussian kernel or a Cauchy kernel as integer kernel, tweaking the variance requires a lot of attention.

[2] This was not represented in the experiments, but is rather an expected behaviour given the considered weight matrices.

Appendices

Appendix A

Additional Proofs

This appendix hosts the complete proofs for Theorem 6 and Theorem 7, which were omitted from the main document to maintain concision and simplicity. For reasons of clarity, the statement of each theorem and the details concerning the inductive process will be briefly repeated before the development.

A.1 Proof of the Path Perspective Theorem

The statement made by (5.17) is that, for all strings s and t where m and n represent their respective length $|s|$ and $|t|$,

$$k_{\text{PATH}}(s,t) = \sum_{\gamma\in\Gamma}\sum_{i=1}^{|\gamma|} \boldsymbol{G}(s,t)_{\gamma(i)} \frac{\prod_{k=2}^{i} C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1, n-\gamma_Y(i)+1)} \tag{A.1}$$

The Proof

The proof consists in an induction process, as already described in the main corpus of the document. Following, the proof of each inductive step.

Base Case I $|s| = m$ and $|t| = 1$.

In this case Γ contains only one path which can be expressed as $\gamma(i) = (i, 1)$ and $\gamma\prime(i) = \delta_{10}$ with $1 \leq i \leq m$. Thus, the following holds:

$$\begin{aligned}
&\sum_{\gamma\in\Gamma}\sum_{i=1}^{|\gamma|} \boldsymbol{G}(s,t)_{\gamma(i)} \frac{\prod_{k=2}^{i} C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1, n-\gamma_Y(i)+1)} \\
=&\sum_{i=1}^{|\gamma|} \boldsymbol{G}(s,t)_{\gamma(i)} \frac{\prod_{k=2}^{i} C_{\delta_{10}}}{\mathrm{D}(m-\gamma_X(i)+1, n-\gamma_Y(i)+1)} \\
=&\sum_{i=1}^{m} k_\Sigma(s_i, t_1) \frac{\prod_{k=2}^{i} C_{HV}}{\mathrm{D}(m-i+1, 1-1+1)}
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=1}^{m} k_\Sigma(s_i, t_1) \frac{C_{HV}^{i-1}}{\mathrm{D}(m-i+1, 1)} \\
&= \sum_{i=1}^{m} C_{HV}^{i-1} k_\Sigma(s_i, t_1)
\end{aligned}
$$

which is, according to Theorem 5, equivalent to $k_{\mathrm{PATH}}(s,t)$ for this specific case.

Base Case II $|s| = 1$ and $|t| = n$.

This case is symmetric to the previous one, and can thus be proven either using the same process or by making use of the symmetry of k_{PATH}.

Inductive Step Assuming the statement to be true for $|s| = m-1$ and $|t| = n-1$; for $|s| = m-1$ and $|t| = n$; and for $|s| = m$ and $|t| = n-1$; then it is valid also for $|s| = m$ and $|t| = n$.

Equation (A.1) is decomposed into 4 parts which are studies separately:

$$
\begin{aligned}
&\sum_{\gamma\in\Gamma} \sum_{i=1}^{|\gamma|} \boldsymbol{G}(s,t)_{\gamma(i)} \frac{\prod_{k=2}^{i} C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1, n-\gamma_Y(i)+1)} \\
&= \sum_{\gamma\in\Gamma} \boldsymbol{G}(s,t)_{\gamma(i)} \frac{\prod_{k=2}^{i} C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1, n-\gamma_Y(i)+1)}\bigg|_{i=1} \\
&+ \sum_{\gamma\in\Gamma} \sum_{i=2}^{|\gamma|} \boldsymbol{G}(s,t)_{\gamma(i)} \frac{\prod_{k=2}^{i} C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1, n-\gamma_Y(i)+1)} \\
&= \sum_{\gamma\in\Gamma} \boldsymbol{G}(s,t)_{\gamma(i)} \frac{\prod_{k=2}^{i} C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1, n-\gamma_Y(i)+1)}\bigg|_{i=1} && \text{(A.2)} \\
&+ \sum_{\gamma\in(\delta_{10},\Gamma_{10})} \sum_{i=2}^{|\gamma|} \boldsymbol{G}(s,t)_{\gamma(i)} \frac{\prod_{k=2}^{i} C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1, n-\gamma_Y(i)+1)} && \text{(A.3)} \\
&+ \sum_{\gamma\in(\delta_{01},\Gamma_{01})} \sum_{i=2}^{|\gamma|} \boldsymbol{G}(s,t)_{\gamma(i)} \frac{\prod_{k=2}^{i} C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1, n-\gamma_Y(i)+1)} && \text{(A.4)} \\
&+ \sum_{\gamma\in(\delta_{11},\Gamma_{11})} \sum_{i=2}^{|\gamma|} \boldsymbol{G}(s,t)_{\gamma(i)} \frac{\prod_{k=2}^{i} C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1, n-\gamma_Y(i)+1)} && \text{(A.5)}
\end{aligned}
$$

Next, these parts are individually studied and the following will be proven:

a) (A.2) is equivalent to $k_\Sigma(s_1, t_1)$.

b) (A.3) is equivalent to $C_{HV} k_{\mathrm{PATH}}(s_{2:}, t)$.

c) (A.4) is equivalent to $C_{HV} k_{\mathrm{PATH}}(s, t_{2:})$.

d) (A.5) is equivalent to $C_D\, k_{\mathrm{PATH}}(s_{2:}, t_{2:})$.

If the previous points are proven, they can be substituted to yield

$$
\begin{aligned}
&\sum_{\gamma\in\Gamma}\sum_{i=1}^{|\gamma|}\boldsymbol{G}(s,t)_{\gamma(i)}\frac{\prod_{k=2}^{i}C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1,n-\gamma_Y(i)+1)}\\
=&\sum_{\gamma\in\Gamma}\boldsymbol{G}(s,t)_{\gamma(i)}\frac{\prod_{k=2}^{i}C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1,n-\gamma_Y(i)+1)}\bigg|_{i=1}\\
+&\sum_{\gamma\in(\delta_{10},\Gamma_{10})}\sum_{i=2}^{|\gamma|}\boldsymbol{G}(s,t)_{\gamma(i)}\frac{\prod_{k=2}^{i}C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1,n-\gamma_Y(i)+1)}\\
+&\sum_{\gamma\in(\delta_{01},\Gamma_{01})}\sum_{i=2}^{|\gamma|}\boldsymbol{G}(s,t)_{\gamma(i)}\frac{\prod_{k=2}^{i}C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1,n-\gamma_Y(i)+1)}\\
+&\sum_{\gamma\in(\delta_{11},\Gamma_{11})}\sum_{i=2}^{|\gamma|}\boldsymbol{G}(s,t)_{\gamma(i)}\frac{\prod_{k=2}^{i}C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1,n-\gamma_Y(i)+1)}\\
=&k_\Sigma(s_1,t_1)+C_{HV}k_{\mathrm{PATH}}(s_{2:},t)+C_{HV}k_{\mathrm{PATH}}(s,t_{2:})+C_Dk_{\mathrm{PATH}}(s_{2:},t_{2:})\\
=&k_{\mathrm{PATH}}(s,t)
\end{aligned}
$$

Next, the procedures through which (a), (b), (c) and (d) are proven:

a) According to the definition of a path we can make use of $\gamma(1)=(1,1)$, which holds for any path, to show that

$$
\begin{aligned}
&\sum_{\gamma\in\Gamma}\boldsymbol{G}(s,t)_{\gamma(i)}\frac{\prod_{k=2}^{i}C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1,n-\gamma_Y(i)+1)}\bigg|_{i=1}\\
=&\sum_{\gamma\in\Gamma}k_\Sigma(s_1,t_1)\frac{1}{\mathrm{D}(m,n)}\\
=&k_\Sigma(s_1,t_1)\sum_{\gamma\in\Gamma}\frac{1}{\mathrm{D}(m,n)}\\
=&k_\Sigma(s_1,t_1)\frac{|\Gamma|}{\mathrm{D}(m,n)}\\
=&k_\Sigma(s_1,t_1)
\end{aligned}
$$

b) This part requires a series of observations. First of all is the fact that, for all the paths $\gamma\in(\delta_{10},\Gamma_{10})$, it is always true that $\gamma(2)=(2,1)$ and that $\gamma\prime(2)=\delta_{10}$. Secondly, it becomes necessary for the proof to point out that each path $\gamma\in(\delta_{10},\Gamma_{10})$ can be associated with a path $\tilde{\gamma}\in\Gamma_{10}$ by the fact that the two are equivalent with the exception of the first step of γ which is missing in $\tilde{\gamma}$. The two paths are related by the following

properties:

$$\begin{aligned}\gamma(i) &= \tilde{\gamma}(i-1) + \delta_{10}, && 2 \le i \le |\gamma|\,,\\ \gamma\prime(i) &= \tilde{\gamma\prime}(i-1), && 3 \le i \le |\gamma|\,,\\ |\gamma| &= |\tilde{\gamma}| + 1.\end{aligned}$$

This allows us to embed the substitution within (A.2) to show that

$$\begin{aligned}&\sum_{\gamma\in(\delta_{10},\Gamma_{10})}\sum_{i=2}^{|\gamma|}\boldsymbol{G}(s,t)_{\gamma(i)}\frac{\prod_{k=2}^{i}C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1,n-\gamma_Y(i)+1)}\\ &=\sum_{\gamma\in(\delta_{10},\Gamma_{10})}\sum_{i=2}^{|\gamma|}\boldsymbol{G}(s,t)_{\gamma(i)}\frac{C_{\gamma\prime(2)}\prod_{k=3}^{i}C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1,n-\gamma_Y(i)+1)}\\ &=\sum_{\gamma\in(\delta_{10},\Gamma_{10})}\sum_{i=2}^{|\gamma|}\boldsymbol{G}(s,t)_{\gamma(i)}\frac{C_{HV}\prod_{k=3}^{i}C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(i)+1,n-\gamma_Y(i)+1)}\end{aligned}$$

We now substitute γ with $\tilde{\gamma}$. Following that, the indexes are changed without changing the result.

$$\begin{aligned}&= C_{HV}\sum_{\tilde{\gamma}\in\Gamma_{10}}\sum_{i=2}^{|\tilde{\gamma}|+1}\boldsymbol{G}(s,t)_{\tilde{\gamma}(i-1)+\delta_{10}}\frac{\prod_{k=3}^{i}C_{\tilde{\gamma}\prime(k-1)}}{\mathrm{D}(m-\tilde{\gamma}_X(i-1)-1+1,n-\tilde{\gamma}_Y(i-1)+1)}\\ &= C_{HV}\sum_{\tilde{\gamma}\in\Gamma_{10}}\sum_{i=1}^{|\tilde{\gamma}|}\boldsymbol{G}(s,t)_{\tilde{\gamma}(i)+\delta_{10}}\frac{\prod_{k=2}^{i}C_{\tilde{\gamma}\prime(k)}}{\mathrm{D}(m-\tilde{\gamma}_X(i),n-\tilde{\gamma}_Y(i)+1)}\\ &= C_{HV}\sum_{\tilde{\gamma}\in\Gamma_{10}}\sum_{i=1}^{|\tilde{\gamma}|}k_{\Sigma}(s_{\tilde{\gamma}_X(i)+1},t_{\tilde{\gamma}_Y(i)})\frac{\prod_{k=2}^{i}C_{\tilde{\gamma}\prime(k)}}{\mathrm{D}((m-1)-\tilde{\gamma}_X(i)+1,n-\tilde{\gamma}_Y(i)+1)}\\ &= C_{HV}\sum_{\tilde{\gamma}\in\Gamma_{10}}\sum_{i=1}^{|\tilde{\gamma}|}k_{\Sigma}([s_{2:}]_{\tilde{\gamma}_X(i)},t_{\tilde{\gamma}_Y(i)})\frac{\prod_{k=2}^{i}C_{\tilde{\gamma}\prime(k)}}{\mathrm{D}((m-1)-\tilde{\gamma}_X(i)+1,n-\tilde{\gamma}_Y(i)+1)}\\ &= C_{HV}\sum_{\tilde{\gamma}\in\Gamma_{10}}\sum_{i=1}^{|\tilde{\gamma}|}\boldsymbol{G}(s_{2:},t)_{\tilde{\gamma}(i)}\frac{\prod_{k=2}^{i}C_{\tilde{\gamma}\prime(k)}}{\mathrm{D}((m-1)-\tilde{\gamma}_X(i)+1,n-\tilde{\gamma}_Y(i)+1)}\\ &= C_{HV}k(s_{2:},t)\end{aligned}$$

c) Item (c) can be proven in an analogous way to the second. For the purpose of brevity, an explicit step-by-step process will be omitted.

d) Item (d) too can be proven in an analogous way which will be omitted.

□

A.2 Proof of the Grid Perspective Theorem

The statement made by (5.18) is that, for all strings s and t where m and n represent their respective length $|s|$ and $|t|$,

$$k_{\text{PATH}}(s,t) = \sum_{ij} \boldsymbol{G}(s,t)_{ij} \boldsymbol{\omega}_{\text{PATH}ij} \tag{A.6}$$

$$\boldsymbol{\omega}_{\text{PATH}ij} = \sum_{d=0}^{\min(i,j)-1} C_{HV}^{i+j-2-2d} C_D^d (d, i-1-d, j-1-d)! \tag{A.7}$$

The Proof

This proof makes use of the Pinned and Partially d-Diagonal path sets. Additionally, it is based on the following observations:

- If p is the pin index of $\gamma \in \Gamma^{(ijd)}$, then the first $p-1$ steps used by γ consist of $i-1-d$ vertical, $j-1-d$ horizontal and d diagonal steps, which implies

$$\prod_{k=2}^{p} C_{\gamma\prime(k)} = C_{HV}^{i+j-2-2d} C_D^d.$$

- Equation (5.12) supports the decomposition of $\Gamma^{(ij)}$, and can be used to substitute the summation over the relative elements, i.e.

$$\sum_{\gamma \in \Gamma^{(ij)}} = \sum_{d=0}^{\min(i,j)-1} \sum_{\gamma \in \Gamma^{(ijd)}}.$$

- Applying Theorem 4 onto (5.11), it is obtained that

$$\begin{aligned} \left|\Gamma^{(ijd)}(m,n)\right| &= \left|\Gamma^{(d)}(i,j)\right| \left|\Gamma(m-i+1, n-j+1)\right| \\ &= (d, i-1-d, j-1-d)!\, \mathrm{D}(m-i+1, n-j+1). \end{aligned}$$

The proof is based on the path perspective, (5.17). The idea is to extrapolate a generic expression for $[\boldsymbol{\omega}_{\text{PATH}}]_{ij}$ by taking only account of the contributions to $\boldsymbol{G}_{ij}$, which can be done by making use of the Pinned path set $\Gamma^{(ij)}$. Referring to p as the *pin index*, we have that

$$\begin{aligned} \boldsymbol{G}_{ij} \boldsymbol{\omega}_{\text{PATH}ij} &= \sum_{\gamma \in \Gamma^{(ij)}} \boldsymbol{G}_{ij} \frac{\prod_{k=2}^{p} C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(p)+1, n-\gamma_Y(p)+1)} \\ &= \boldsymbol{G}_{ij} \sum_{\gamma \in \Gamma^{(ij)}} \frac{\prod_{k=2}^{p} C_{\gamma\prime(k)}}{\mathrm{D}(m-\gamma_X(p)+1, n-\gamma_Y(p)+1)}. \end{aligned}$$

We can thus eliminate $\boldsymbol{G}_{ij}$ and find the expression for $[\boldsymbol{\omega}_{\text{PATH}}]_{ij}$:

$$
\begin{aligned}
[\boldsymbol{\omega}_{\text{PATH}}]_{ij} &= \sum_{\gamma \in \Gamma^{(ij)}} \frac{\prod_{k=2}^{p} C_{\gamma'(k)}}{\mathrm{D}(m-\gamma_X(p)+1, n-\gamma_Y(p)+1)} \\
&= \sum_{d=0}^{\min(i,j)-1} \sum_{\gamma \in \Gamma^{(ijd)}} \frac{\prod_{k=2}^{p} C_{\gamma'(k)}}{\mathrm{D}(m-\gamma_X(p)+1, n-\gamma_Y(p)+1)} \\
&= \sum_{d=0}^{\min(i,j)-1} \sum_{\gamma \in \Gamma^{(ijd)}} \frac{C_{HV}^{i+j-2-2d} C_D^d}{\mathrm{D}(m-i+1, n-j+1)} \\
&= \sum_{d=0}^{\min(i,j)-1} C_{HV}^{i+j-2-2d} C_D^d \frac{\left|\Gamma^{(ijd)}\right|}{\mathrm{D}(m-i+1, n-j+1)} \\
&= \sum_{d=0}^{\min(i,j)-1} C_{HV}^{i+j-2-2d} C_D^d \frac{(d, i-1-d, j-1-d)!\,\mathrm{D}(m-i+1, n-j+1)}{\mathrm{D}(m-i+1, n-j+1)} \\
&= \sum_{d=0}^{\min(i,j)-1} C_{HV}^{i+j-2-2d} C_D^d (d, i-1-d, j-1-d)!.
\end{aligned}
$$

□

Bibliography

[BHB] C. Bahlmann, B. Haasdonk, and H. Burkhardt. Online handwriting recognition with support vector machines - a kernel approach. In *Proceedings Eighth International Workshop on Frontiers in Handwriting Recognition*, pages 49–54. IEEE Comput. Soc.

[Bis06] Christopher Bishop. *Pattern recognition and machine learning*. Springer, 1st ed. 20 edition, 2006.

[Buh03] MD Buhmann. Radial basis functions: theory and implementations. 2003.

[BV04] S Boyd and L Vandenberghe. Convex optimization. 2004.

[CSt00] Nello Cristianini and John Shawe-taylor. An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, 2000.

[Cut11] Marco Cuturi. Fast global alignment kernels. 2011.

[CVBM07] Marco Cuturi, Jean-Philippe Vert, Oystein Birkenes, and Tomoko Matsui. A Kernel for Time Series Based on Global Alignments. *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, 1(i):II–413–II–416, 2007.

[EBK12] Carl Henrik Ek, Niklas Bergström, and Danica Kragic. Abstraction by Structure: Learning Representations of Objects and Actions. 2012.

[EK11] Carl Henrik Ek and Danica Kragic. The importance of structure. 2011.

[GRS08] Steinn Gudmundsson, Thomas Philip Runarsson, and Sven Sigurdsson. Support vector machines and dynamic time warping for time series. *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, (x):2772–2776, June 2008.

[Joa98] T Joachims. Text categorization with support vector machines: Learning with many relevant features. *Machine learning: ECML-98*, 1998.

[LB11] G Luo and N Bergstrom. Representing actions with kernels. *Intelligent Robots and ...*, 2011.

[LEN02] Christina Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: a string kernel for SVM protein classification. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, 575:564–75, January 2002.

[LK04] Christina Leslie and Rui Kuang. Fast String Kernels using Inexact Matching for Protein Sequences. 5:1435–1455, 2004.

[LL03] HT Lin and CJ Lin. A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods. *submitted to Neural Computation*, 2003.

[LSSt+02] Huma Lodhi, Craig Saunders, John Shawe-taylor, Nello Cristianini, and Chris Watkins. Text Classification using String Kernels. 2:419–444, 2002.

[LZ06] M Li and Y Zhu. Image classification via LZ78 based string kernel: a comparative study. *Advances in Knowledge Discovery and Data Mining*, 2006.

[MPEDK12] Marianna Madry-Pronobis, Carl Henrik Ek, Renaud Detry, and Danica Kragic. Global Structure Histogram: where less is more. 2012.

[NB06] Michel Neuhaus and Horst Bunke. Edit distance-based kernel functions for structural pattern classification. *Pattern Recognition*, 39(10):1852–1863, October 2006.

[OMCS04] CS Ong, Xavier Mary, S Canu, and AJ Smola. Learning with non-positive kernels. *... conference on Machine learning*, 2004.

[SC78] H Sakoe and S Chiba. Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech and Signal Processing, ...*, 1978.

[SS01] Bernhard Scholkopf and Alexander J. Smola. Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. December 2001.

[ST04] C Saunders and Hauke Tschach. Syllables and other string kernel extensions. (October), 2004.

[SZ09] J Sivic and A Zisserman. Efficient visual search of videos cast as text retrieval. *Pattern Analysis and Machine ...*, 2009.

[Vap98] Vladimir Vapnik. *Statistical Learning Theory*. Wiley, 1 edition, 1998.

[VS] S V N Vishwanathan and Alexander J. Smola. Fast Kernels for String and Tree Matching.

[Wat99] C Watkins. Dynamic alignment kernels. *Advances in Neural Information Processing Systems*, 1999.

[YGSK07] Changjiang Yang, Yanlin Guo, Harpreet Sawhney, and Rakesh Kumar. Learning Actions Using Robust String Kernels. pages 313–327, 2007.

TRITA-CSC-E 2012:094
ISRN-KTH/CSC/E--12/094-SE
ISSN-1653-5715

www.kth.se