

Using String Kernels to Identify Famous Performers from their Playing Style

Craig Saunders, David R. Hardoon, John Shawe-Taylor,
{cjs,drh,jst}@ecs.soton.ac.uk
School of Electronics & Computer Science,
ISIS Research Group, Building 1, Highfield,
University of Southampton,
Southampton, SO17 1BJ, UK

Gerhard Widmer
gerhard.widmer@jku.at
Department of Computational Perception
Johannes Kepler University of Linz Altenberger
Strasse 69 A-4040 Linz, Austria and
Austrian Research Institute for Artificial Intelligence
Freyung 6/6, A-1010 Vienna, Austria

June 6, 2006

Abstract

In this chapter we show a novel application of string kernels: that is to the problem of recognising famous pianists from their style of playing. The characteristics of performers playing the same piece are obtained from changes in beat-level tempo and beat-level loudness, which over the time of the piece form a *performance worm*. From such worms, general performance alphabets can be derived, and pianists' performances can then be represented as strings. We show that when using the string kernel on this data, both kernel partial least squares and Support Vector Machines outperform the current best results. Furthermore we suggest a new method of obtaining feature directions from the Kernel Partial Least Squares algorithm and show that this can deliver better performance than methods previously used in the literature when used in conjunction with a Support Vector Machine

1 Introduction

This chapter focuses on the problem of identifying famous pianists using only minimal information obtained from audio recordings of their playing. A tech-

nique called the performance worm which plots a real-time trajectory over 2D space is used to analyse changes in tempo and loudness at the beat level, and extract features for learning. Previous work on this data has compared a variety of machine learning techniques whilst using as features statistical quantities obtained from the performance worm. It is possible however to obtain a set of cluster prototypes from the worm trajectory which capture certain characteristics over a small time frame, say of two beats. These cluster prototypes form a 'performance alphabet' and there is evidence that they capture some aspects of individual playing style. For example a performer may consistently produce loudness/tempo changes unique to themselves at specific points in a piece, e.g. at the loudest sections of a piece. Once a performance alphabet is obtained, the prototypes can each be assigned a symbol and the audio recordings can then be represented as strings constructed from this alphabet. In this chapter we present a novel application of the string kernel by applying it to the problem of identifying famous pianists using only minimal information obtained from audio recordings of their playing. The musical pieces are represented by an alphabet of prototypical temp-loudness curves, which are obtained from a tool called the performance worm that is able to plot a 2-dimensional temp-loudness representation of a musical piece in real time. We show that using this representation delivers an improvement in performance over the current best results obtained using a feature-based approach.

The ability of the string kernel to include non-contiguous features is shown to be key in the performance of the algorithm. This is in contrast to results presented which used the string kernel on text, where the ability to handle non-contiguous substrings does not deliver significant performance over contiguous substrings (Lodhi, Saunders, Shawe-Taylor, Cristianini, & Watkins, 2002). Furthermore we examine the ability of dimension-reduction based methods such as Kernel Partial Least Squares (KPLS) and Kernel PCA to extract significant directions in the feature space in order to possibly gain performance benefits. One issue with dimensionality reduction methods such as the above, however, is that it introduces yet another parameter (the number of feature directions to extract). We therefore employ a semi-definite programming solution defined in (Lanckriet, Cristianini, Bartlett, Ghaoui, & Jordan, 2004) which optimises over a combination of orthogonal kernel matrices, and thus removes the need to select a parameter. This is then compared to the more traditional cross-validation approach.

The rest of this chapter is laid out as follows. In the following section we provide background details on the performance worm representation used for the music data. Section 3 outlines the Partial Least Squares (PLS) algorithm and string kernel function used to analyse the data. Section 4 then presents the Kernel variant of PLS algorithm and gives a formulation for extracting features which are then used in conjunction with support vector machines (SVMs). We then present experimental results in Section 5 and end with some analysis and suggestions for future research.

Table 1: Movements of Mozart piano sonatas selected for analysis.

Sonata	Movement	Key	Time sig.
K.279	1st mvt.	C major	4/4
K.279	2nd mvt.	C major	3/4
K.279	3rd mvt.	C major	2/4
K.280	1st mvt.	F major	3/4
K.280	2nd mvt.	F major	6/8
K.280	3rd mvt.	F major	3/8
K.281	1st mvt.	Bb major	2/4
K.282	1st mvt.	Eb major	4/4
K.282	2nd mvt.	Eb major	3/4
K.282	3rd mvt.	Eb major	2/4
K.330	3rd mvt.	C major	2/4
K.332	2nd mvt.	F major	4/4

2 A Musical Representation

The data used in this chapter, first described in (Zanon & Widmer, 2003), was obtained from recordings of sonatas by W.A. Mozart played by six famous concert pianists. In total the performances of 6 pianists were analysed across 12 different movements of Mozart sonatas. The movements represent a cross section of playing keys, tempi and time signatures, see Table 1 for details.

In many cases the only data available for different performances are standard audio recordings (as opposed to for example MIDI format data from which more detailed analysis is possible), which poses particular difficulties for the extraction of relevant performance information. A tool for analysing this type of data called the performance worm has recently been developed (Dixon, Goebel, & Widmer, 2002; Zanon & Widmer, 2003; Widmer, Dixon, Goebel, Pampalk, & Tobudic, 2003).

The performance worm extracts data from audio recordings by examining tempo and general loudness of the audio when measured at the beat level. An interactive beat tracking program (Dixon, 2001) is used to find the beat from which changes in beat-level tempo and beat-level loudness can be calculated. These two types of changes can be integrated to form trajectories over tempo-loudness space that show the joint development of tempo and dynamics over time. As data is extracted from the audio the 2D plot of the performance curve can be constructed in real time to aid in visualisation of these dynamics, and this is called the performance worm. Figure 2 shows a screenshot of the worm in progress. Note that this is the only information used in the creation of the worm, more detailed information such as articulation, individual voicing or timing details of that below the level of a beat is not available.

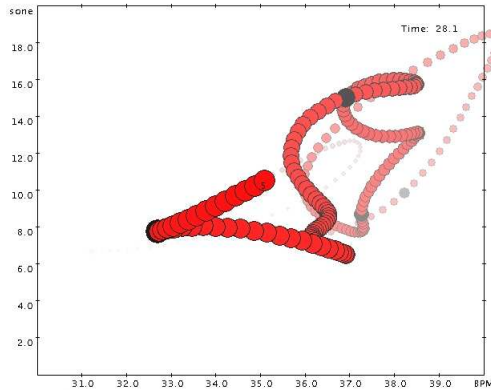


Figure 1: The performance worm: A 2D representation of changes in beat-level tempo and loudness can be plotted in realtime from an audio recording.

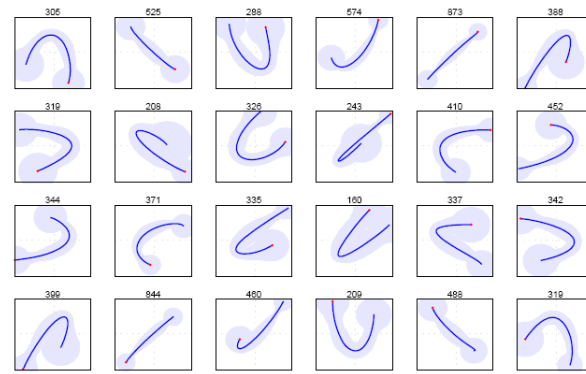


Figure 2: The performance alphabet: A set of cluster prototypes extracted from the performance worm.

2.1 A performance alphabet

From the performance worm, patterns can be observed which can help characterise the individual playing styles of some pianists. For example, in (Widmer et al., 2003) a set of tempo-loudness shapes typical of the performer Mitsuko Uchida were found. These shapes represented a particular way of combining a crescendo-decrescendo with a slowing down during a loudness maximum. These patterns were often repeated in Mozart performances by Mitsuko Uchida, but were rarely found when analysing the recordings of other performers. In order to try and capture more of these types of characterisations a ‘Mozart Performance Alphabet’ can be constructed in the following way. The trajectories of the performance worm are cut into short segments of a fixed length (e.g. 2 beats) and clustered into groups of similar patterns to form a series of prototypes (see Fig-

Table 2: List of pianists and recordings used

ID	Name	Recording
DB	Daniel Barenboim	EMI Classics CDZ 7 67295 2, 1984
RB	Roland Batik	Gramola 98701-705, 1990
GG	Glenn Gould	Sony Classical SM4K 52627, 1967
MP	Maria João Pires	DGG 431 761-2, 1991
AS	Andr�as Schiff	ADD (Decca) 443 720-2, 1980
MU	Mitsuko Uchida	Philips Classics 464 856-2, 1987

ure 2.1). Recordings of a performance can then be transcribed in terms of this alphabet which can then be compared using string matching techniques. The list of pianists and the recordings used to obtain the data can be found in Table 2. For more detailed information on the performance worm and constructing a performance alphabet of cluster prototypes, please refer to (Zanon & Widmer, 2003; Widmer et al., 2003).

Note that the only input to our algorithm will be the string representation of the piece, which is simply a sequence of curves from the performance alphabet. We are therefore using very little information (only loudness/temp at the beat level), which is known to be noisy (there may be some error in the clustering process, and the string of cluster prototypes certainly does not reproduce the original worm).

The task addressed in this chapter is to learn to recognise pianists solely from characteristics of their performance strings. The ability of kernel methods to operate over string-like structures using kernels such as the n-gram kernel and the string kernel will be evaluated on this task. In addition to simply applying an SVM to the data however, we will also examine the ability of dimension reduction methods such as Kernel PCA and Kernel Partial Least Squares (KPLS) to extract relevant features from the data before applying an SVM, which will hopefully lead to improved classification performance. KPCA is well known method and has often been used to extract features from data (see e.g. (Sch olkopf, Smola, & M uller, 1998)). Partial least squares and its kernel-based variant KPLS has recently gained popularity within the machine learning community (Rosipal & Trejo, 2001; Bennett & Embrechts, 2003; Rosipal, Trejo, & Matthews, 2003) and either can be used as a method for regression or classification, or as a method for dimension reduction. It is not always clear however, how to use the PLS-based methods to generate new input features for training and test data, so we shall briefly review the methods here.

3 Previous results

3.1 String kernels

The use of string kernels for analysing text documents was first studied by Lodhi et al. (Lodhi et al., 2002). We briefly review the approach to creating a feature

ϕ	ca	ct	at	ba	bt	cr	ar	br
cat	λ^2	λ^3	λ^2	0	0	0	0	0
car	λ^2	0	0	0	0	λ^3	λ^2	0
bat	0	0	λ^2	λ^2	λ^3	0	0	0
bar	0	0	0	λ^2	0	0	λ^2	λ^3

Table 3: Features and weights for the string kernel with $p = 2$ for the words "cat", "car", "bat", and "bar".

space and associated kernel.

The key idea behind the gap-weighted subsequences kernel is to compare strings by means of the subsequences they contain – the more subsequences in common, the more similar they are – rather than only considering contiguous n-grams, the degree of contiguity of the subsequence in the input string s determines how much it will contribute to the comparison.

In order to deal with non-contiguous substrings, it is necessary to introduce a decay factor $\lambda \in (0, 1)$ that can be used to weight the presence of a certain feature in a string. For an index sequence $\mathbf{i} = (i_1, \dots, i_k)$ identifying the occurrence of a subsequence $u = s(\mathbf{i})$ in a string s , we use $l(\mathbf{i}) = i_k - i_1 + 1$ to denote the length of the string in s . In the gap-weighted kernel, we weight the occurrence of u with the exponentially decaying weight $\lambda^{l(\mathbf{i})}$.

Definition 1 (Gap-weighted subsequences kernel) *The feature space associated with the gap-weighted subsequences kernel of length p is indexed by $I = \Sigma^p$ (i.e. subsequences of length p from some alphabet Σ), with the embedding given by*

$$\phi_u^p(s) = \sum_{\mathbf{i}:u=s(\mathbf{i})} \lambda^{l(\mathbf{i})}, u \in \Sigma^p. \quad (1)$$

The associated kernel is defined as

$$\kappa_p(s, t) = \langle \phi^p(s), \phi^p(t) \rangle = \sum_{u \in \Sigma^p} \phi_u^p(s) \phi_u^p(t). \quad (2)$$

Consider the simple strings "cat", "car", "bat", and "bar". Fixing $p = 2$, the words are mapped as shown in table 3.

So the unnormalised kernel between "cat" and "car" is $\kappa(\text{"cat"}, \text{"car"}) = \lambda^4$, while the normalised version is obtained using

$$\kappa(\text{"cat"}, \text{"cat"}) = \kappa(\text{"car"}, \text{"car"}) = 2\lambda^4 + \lambda^6 \quad (3)$$

as $\hat{\kappa}(\text{"cat"}, \text{"car"}) = \lambda^4 / (2\lambda^4 + \lambda^6) = (2 + \lambda^2)^{-1}$.

We omit a description of the efficient dynamic programming algorithms for computing this kernel referring the reader to Lodhi et al. (Lodhi et al., 2002).

Algorithm 1 The PLS feature extraction algorithm

The PLS feature extraction algorithm is as follows:

process	$\mathbf{X}_1 = \mathbf{X}$
	for $j = 1, \dots, k$
	let \mathbf{u}_j, σ_j be the first singular vector/value of $\mathbf{X}_j' \mathbf{Y}$,
	$\mathbf{X}_{j+1} = \left(\mathbf{I} - \frac{\mathbf{X}_j \mathbf{u}_j \mathbf{u}_j' \mathbf{X}_j'}{\mathbf{u}_j' \mathbf{X}_j' \mathbf{X}_j \mathbf{u}_j} \right) \mathbf{X}_j = \mathbf{X}_j \left(\mathbf{I} - \frac{\mathbf{u}_j \mathbf{u}_j' \mathbf{X}_j' \mathbf{X}_j}{\mathbf{u}_j' \mathbf{X}_j' \mathbf{X}_j \mathbf{u}_j} \right)$
	end
output	Feature directions $\mathbf{u}_j, j = 1, \dots, k.$

3.2 Partial Least Squares

Partial Least Squares (PLS) was developed by Herman Wold during the 1960's in the field of econometrics (Wold, 1966). It offers an effective approach to solving problems with training data that has few points but high dimensionality, by first projecting the data into a lower-dimensional space and then utilising a Least Squares (LS) regression model. This problem is common in the field of Chemometrics where PLS is regularly used. PLS is a flexible algorithm that was designed for regression problems, though it can be used for classification by treating the labels $\{+1, -1\}$ as real outputs. Alternatively it can also be stopped after constructing the low-dimensional projection. The resulting features can then be used in a different classification or regression algorithm. We will also adopt this approach by applying an SVM in this feature space, an approach pioneered by Rosipal et al. (Rosipal et al., 2003). The procedure for PLS feature extraction is shown in Algorithm 1. The algorithmic procedure iteratively takes the first singular vector \mathbf{u}_i of the matrix $\mathbf{X}_i' \mathbf{Y}$, and then deflates the matrix \mathbf{X}_i to obtain \mathbf{X}_{i+1} . The deflation is done by projecting the columns of \mathbf{X}_i into the space orthogonal to $\mathbf{X}_i \mathbf{u}_i$. The difficulty with this simple description is that the feature directions \mathbf{u}_j are defined relative to the deflated matrix. We would like to be able to compute the PLS features directly from the original feature vector.

If we now consider a test point with feature vector $\phi(\mathbf{x})$ the transformations that we perform at each step should also be applied to $\phi_1(\mathbf{x}) = \phi(\mathbf{x})$ to create a series of feature vectors

$$\phi_{j+1}(\mathbf{x})' = \phi_j(\mathbf{x})' (\mathbf{I} - \mathbf{u}_j \mathbf{p}_j'), \quad (4)$$

where

$$\mathbf{p}_j = \frac{\mathbf{X}_j' \mathbf{X}_j \mathbf{u}_j}{\mathbf{u}_j' \mathbf{X}_j' \mathbf{X}_j \mathbf{u}_j}, \quad (5)$$

This is the same operation that is performed on the rows of \mathbf{X}_j in Algorithm 1. We can now write

$$\phi(\mathbf{x})' = \phi_{k+1}(\mathbf{x})' + \sum_{j=1}^k \phi_j(\mathbf{x})' \mathbf{u}_j \mathbf{p}_j'. \quad (6)$$

The feature vector that we need for the regression $\hat{\phi}(\mathbf{x})$ has components

$$\hat{\phi}(\mathbf{x}) = (\phi_j(\mathbf{x})' \mathbf{u}_j)_{j=1}^k, \quad (7)$$

since these are the projections of the residual vector at stage j onto the next feature vector \mathbf{u}_j . Rather than compute $\phi_j(\mathbf{x})'$ iteratively, consider using the inner products between the original $\phi(\mathbf{x})'$ and the feature vectors \mathbf{u}_j stored as the columns of the matrix \mathbf{U} :

$$\begin{aligned} \phi(\mathbf{x})' \mathbf{U} &= \phi_{k+1}(\mathbf{x})' \mathbf{U} + \sum_{j=1}^k \phi_j(\mathbf{x})' \mathbf{u}_j \mathbf{p}_j' \mathbf{U} \\ &= \phi_{k+1}(\mathbf{x})' \mathbf{U} + \hat{\phi}(\mathbf{x})' \mathbf{P}' \mathbf{U}, \end{aligned}$$

where \mathbf{P} is the matrix whose columns are \mathbf{p}_j , $j = 1, \dots, k$. Finally, it can be verified that

$$\mathbf{u}_i' \mathbf{p}_j = \delta_{ij} \quad \text{for } i \leq j. \quad (8)$$

Hence, for $s > j$, $(\mathbf{I} - \mathbf{u}_s \mathbf{p}_s') \mathbf{u}_j = \mathbf{u}_j$, while $(\mathbf{I} - \mathbf{u}_j \mathbf{p}_j') \mathbf{u}_j = 0$, so we can write

$$\phi_{k+1}(\mathbf{x})' \mathbf{u}_j = \phi_j(\mathbf{x})' \prod_{i=j}^k (\mathbf{I} - \mathbf{u}_i \mathbf{p}_i') \mathbf{u}_j = 0, \text{ for } j = 1, \dots, k. \quad (9)$$

It follows that the new feature vector can be expressed as

$$\hat{\phi}(\mathbf{x})' = \phi(\mathbf{x})' \mathbf{U} (\mathbf{P}' \mathbf{U})^{-1}. \quad (10)$$

If we consider the vectors of feature values across the training set $\mathbf{X}_j \mathbf{u}_j$, these are orthogonal since they are a linear combination of the columns of \mathbf{X}_j that have been repeatedly projected into the orthogonal complement of previous $\mathbf{X}_i \mathbf{u}_i$, for $i < j$. By the analysis above these vectors can be written as $\mathbf{X} \mathbf{U} (\mathbf{P}' \mathbf{U})^{-1}$, from which it follows that

$$(\mathbf{U}' \mathbf{P})^{-1} \mathbf{U}' \mathbf{X}' \mathbf{X} \mathbf{U} (\mathbf{P}' \mathbf{U})^{-1}$$

is a diagonal matrix. Since $(\mathbf{U}' \mathbf{P})$ is upper triangular it follows that the vectors $\mathbf{u}_1, \dots, \mathbf{u}_k$ are conjugate with respect to $\mathbf{X}' \mathbf{X}$.

These feature vectors can now be used in conjunction with a learning algorithm. If one wished to calculate the overall regression coefficients as in the full PLS algorithm, these can be computed as:

$$\mathbf{W} = \mathbf{U} (\mathbf{P}' \mathbf{U})^{-1} \mathbf{C}', \quad (11)$$

where \mathbf{C} is the matrix with columns

$$\mathbf{c}_j = \frac{\mathbf{Y}' \mathbf{X}_j \mathbf{u}_j}{\mathbf{u}_j' \mathbf{X}_j' \mathbf{X}_j \mathbf{u}_j}. \quad (12)$$

Algorithm 2 Pseudocode for kernel-PLS

Input: Data $S = x_1, \dots, x_l$ dimension k target outputs $Y \in \mathbb{R}^{l \times m}$

$$K_{ij} = \kappa(x_i, x_j)$$

$$K_1 = K$$

$$\hat{Y} = Y$$

for $i = 1, \dots, k$ **do**

$\beta_i =$ first column of \hat{Y}

 normalise β_i

repeat

$$\quad \beta_i = YY'K_i\beta_i$$

 normalise β_i

until convergence

$$\quad \tau_i = K_i\beta_i$$

$$\quad c_i = \hat{Y}'\tau_i/||\tau_i||^2$$

$$\quad \hat{Y} = \hat{Y} - \tau_i c_i'$$

$$\quad K_{i+1} = (I - \tau_i \tau_i' / ||\tau_i||^2) K_i (I - \tau_i \tau_i' / ||\tau_i||^2)$$

end for

$$B = [\beta_1, \dots, \beta_k]$$

$$T = [\tau_1, \dots, \tau_k]$$

$$\alpha = B(TKB)^{-1}T'Y$$

Output: Training outputs $Y - \hat{Y}$ and dual regression coefficients α

4 Kernel PLS

In this section we set out the kernel PLS algorithm and describe its feature extraction stage. The kernel PLS algorithm is given in Algorithm 2. The vector β_i is a rescaled dual representation of the primal vectors \mathbf{u}_i :

$$a_i \mathbf{u}_i = \mathbf{X}'_i \beta_i, \quad (13)$$

the rescaling arising because of the different point at which the renormalising is performed in the dual. We can now express the primal matrix $\mathbf{P}'\mathbf{U}$ in terms of the dual variables as

$$\begin{aligned} \mathbf{P}'\mathbf{U} &= \text{diag}(\mathbf{a}) \text{diag}(\tau'_i \tau_i)^{-1} \mathbf{T}' \mathbf{X} \mathbf{X}' \mathbf{B} \text{diag}(\mathbf{a})^{-1} \\ &= \text{diag}(\mathbf{a}) \text{diag}(\tau'_i \tau_i)^{-1} \mathbf{T}' \mathbf{K} \mathbf{B} \text{diag}(\mathbf{a})^{-1}. \end{aligned}$$

Here $\text{diag}(\tau'_i \tau_i)$ is the diagonal matrix with entries $\text{diag}(\tau'_i \tau_i)_{ii} = \tau'_i \tau_i$, where $\tau_i = K_i \beta_i$. Finally, again using the orthogonality of $\mathbf{X}_j \mathbf{u}_j$ to τ_i , for $i < j$, we obtain

$$\mathbf{c}_j = \frac{\mathbf{Y}'_j \mathbf{X}_j \mathbf{u}_j}{\mathbf{u}'_j \mathbf{X}'_j \mathbf{X}_j \mathbf{u}_j} = \frac{\mathbf{Y}'_j \mathbf{X}_j \mathbf{u}_j}{\mathbf{u}'_j \mathbf{X}'_j \mathbf{X}_j \mathbf{u}_j} = a_j \frac{\mathbf{Y}'_j \tau_j}{\tau'_j \tau_j}, \quad (14)$$

making

$$\mathbf{C} = \mathbf{Y}' \mathbf{T} \text{diag}(\tau'_i \tau_i)^{-1} \text{diag}(\mathbf{a}). \quad (15)$$

Putting the pieces together we can compute the dual regression variables as

$$\alpha = \mathbf{B} (\mathbf{T}' \mathbf{K} \mathbf{B})^{-1} \mathbf{T}' \mathbf{Y}. \quad (16)$$

It is tempting to assume like (Rosipal et al., 2003) that a dual representation of the PLS features is then given by

$$\mathbf{B} (\mathbf{T}' \mathbf{K} \mathbf{B})^{-1}, \quad (17)$$

but in fact

$$\mathbf{U} (\mathbf{P}' \mathbf{U})^{-1} = \mathbf{X}' \mathbf{B} \text{diag}(\mathbf{a})^{-1} \left(\text{diag}(\mathbf{a}) \text{diag}(\tau'_i \tau_i)^{-1} \mathbf{T}' \mathbf{K} \mathbf{B} \text{diag}(\mathbf{a})^{-1} \right)^{-1}$$

so that the dual representation is

$$\mathbf{B} (\mathbf{T}' \mathbf{K} \mathbf{B})^{-1} \text{diag}(\mathbf{a})^{-1} \text{diag}(\tau'_i \tau_i) = \mathbf{B} (\mathbf{T}' \mathbf{K} \mathbf{B})^{-1} \text{diag}(\tau'_i \tau_i) \text{diag}(\mathbf{a})^{-1}. \quad (18)$$

The missing diagonal matrices perform a rescaling of the features extracted, which skews the geometry of the space and affects the performance of for example an SVM.

At first sight it seems as though the a_i are difficult to asses, and there is an argument that the values of a_i should not vary significantly over similar adjacent features since they will be related to the corresponding singular values. Recalling (13) however, we have the following

$$(a_i \mathbf{u}_i)^2 = (\mathbf{X}'_i \beta_i)^2 = \beta'_i \mathbf{K} \beta_i = a_i^2.$$

In our experiments we have compared the results that can be obtained ignoring both diagonal matrices with those obtained including the tau rescaling $\text{diag}(\tau'_i \tau_i)$ and the complete rescaling $\text{diag}(\tau'_i \tau_i) \text{diag}(\mathbf{a})^{-1}$.

4.1 Using Semi-Definite Programming

One disadvantage of using these subspace approaches is that we have introduced yet another parameter: the number of feature directions to use T . Also, we are forced into using just one particular value for T , rather than exploring combinations of directions which may yield better results (in practice of course we could try combinations, but this starts to explode the size of our parameter space even more).

An ideal scenario would be to generate several rank-one kernel matrices from the projections on to the directions $\mathbf{u}_1, \dots, \mathbf{u}_T$, and then optimise over the combination of these matrices in one step. This would both remove the need to choose a parameter T (apart from setting a limit for T in advance) and also allow combinations of features to be used. An approach outlined in (Lanckriet et al., 2004) shows that it is possible to use semi-definite programming to optimise over a combination over kernel matrices whilst solving the SVM maximisation problem. In this section we briefly review the method in (Lanckriet et al., 2004) and discuss how it can be applied to the problem of selecting the number of feature directions to use. In the paper it was shown that a if \hat{K} is a linear combination of kernel matrices $\hat{K} = \sum_{i=1}^k \mu_i K_i$ then it is possible to optimise the standard SVM formulation and the coefficients μ_i simultaneously using a semi-definite program. This is due to the problem being convex in \hat{K} . Furthermore, a generalisation bound is given which shows that the generalisation error, at least in part, depends on the trace of the matrix \hat{K} ; for further details we refer the reader to (Lanckriet et al., 2004).

In summary, the optimisation problem becomes

$$\min_{\hat{K}} \max_{\boldsymbol{\alpha}} 2\boldsymbol{\alpha}'\mathbf{e} - \boldsymbol{\alpha}'(G(\hat{K}) + \tau I)\boldsymbol{\alpha}$$

subject to

$$0 \leq \boldsymbol{\alpha} \leq C, \boldsymbol{\alpha}'\mathbf{y}, \tau > 0, \text{trace}(\hat{K}) = c,$$

where $G_{ij}(K) = k(\mathbf{x}_i, \mathbf{x}_j)y_i y_j$. For the case where we restrict $\mu_i \geq 0$ and consider the 1-norm SVM margin, the optimisation problem becomes:

$$\begin{aligned} \max_{\boldsymbol{\alpha}, t} \quad & 2\boldsymbol{\alpha}'\mathbf{e} - ct & (19) \\ \text{subject to} \quad & t \geq \frac{1}{r_i} \boldsymbol{\alpha}'G(K_i)\boldsymbol{\alpha}, \quad i = 1, \dots, k, \\ & \boldsymbol{\alpha}'\mathbf{y} = 0 \\ & 0 \leq \boldsymbol{\alpha} \leq C \end{aligned}$$

where $r_i = \text{trace}(K_i)$.

In this chapter we are considering the problem of choosing the number of feature projections to use when considering low-dimensional projections of feature-space data. In this case, the kernel matrices K_i are rank one matrices formed by the outer product of feature projections $\mathbf{v}_i \mathbf{v}_i'$, where \mathbf{v}_i is the dual representation of the data projected onto the i th extracted feature (e.g. the columns of

the matrix given by eq (18)). In general, the restriction of $\mu_i \geq 0$ may lead to sub-optimal combinations of the base kernel matrices, as allowing μ to have negative coefficients may still lead to a positive semi-definite matrix \hat{K} . Due to the orthogonality of \mathbf{v}_i (as discussed in Section 4), however, negative components μ_i would lead to negative eigenvalues; as eigenvalues of \hat{K} are equal to $\mu_i \|\mathbf{v}_i\|$ and therefore \hat{K} would not be positive definite. Note that most general SDP solvers use primal dual methods to solve problems such as (19). The primal variables μ_i are therefore easily covered once a solution is found.

5 Experiments

In our experiments we followed the setup given in (Zanon & Widmer, 2003). For each pair of performers a leave-one-out procedure was followed where on each iteration one movement played by each of a pair of performers was used for testing and the rest of the data was used for training. That is, for a given pair of performers, say Mitsuko Uchida and Daniel Barenboim (MU-DB), a total of 12 runs of an algorithm were performed (there are 12 movements and each time one movement by both performers was left out of the training set and tested upon). This was repeated for each of the possible 15 pairings of performers. Note that in all results the number reported is the number of *correct* classifications made by the algorithm.

5.1 Previous results

Previous results on the data (as described in (Zanon & Widmer, 2003)) used a feature-based representation and considered a range of machine learning techniques by using the well-known Waikato Environment for Knowledge Analysis (WEKA) software package (Witten & Frank, 1999) to compare bayesian, rule-based, tree-based and nearest-neighbour methods. The best results obtained previously on the data are for a classification via regression meta-learner. These results are reported as FB (feature-based) in the results table. The feature-based representation used in the experiments included the raw measures of tempo and loudness along with various statistics regarding the variance and standard deviation of these and additional information extracted from the worm such as the correlation of tempo and loudness values.

5.2 Results

Experiments were conducted using both the standard string kernel and the n-gram kernel and several algorithms. In both cases experiments were conducted using a standard SVM on the relevant kernel matrix. Kernel Partial Least Squares and Kernel Principal Component Regression were also used for comparison. Finally, an SVM was used in conjunction with the projected features obtained from the iterative KPLS deflation steps. For these features there were

Table 4: Total number of correct predictions across all splits against number of feature directions used (T) for both the feature projection methods described in this chapter (τ -R) and (τa -R) and that in previous work (ORIG). The parameters used in this were those optimal for the KPLS combination ($k = 5, \lambda = 0.9$).

Method/T	1	2	3	4	5	6	7	8	9	10
ORIG	284	246	230	236	249	240	235	239	244	240
τ -R	288	289	290	290	290	290	290	290	290	290
τa -R	286	289	290	287	287	290	289	290	291	289

three options; to use the features as described in (Rosipal et al., 2003), to include the extra reweighting factors $\text{diag}(\tau'_i \tau_i)$ or to also include the scalings produced by the a_i values as described above. We first performed a comparison of these two options by counting the total number of correct predictions across all splits for different feature dimensions (T) for the original weighting (ORIG), the *tau*-reweighted (τ -R), and the features with both τ and a rescaling (τa -R). Table 4 shows the results obtained. There is a clear advantage shown for the reweighting schemes, however the introduction of the $\text{diag}(\mathbf{a})^{-1}$ matrix seems to have little effect as expected. In this situation using the full reweighting actually suffered a slight performance loss against using τ only, however for completeness we will use the full reweighting in all future experiments.

In the remaining experiments we chose one pair of composers (RB-DB) to select the various parameters. These included the number of characters used by the string kernel, the decay parameter and the number of PLS features extracted where appropriate. Substring lengths of $k = 1, \dots, 10$ were tried for both the n-gram and string kernels, $\lambda = \{0.2, 0.5, 0.9\}$ decay parameters were used for the string kernel and for both KPLS and KPCR methods the number of feature directions (T) ranged from 1 to 10. All kernel matrices were normalised and whenever an SVM was used, the parameter C was set to one. In each case the parameters that delivered the best performance on the RB-DB data were chosen. Once selected the settings of these parameters were fixed for the remaining test experiments for all of the results reported in Tables 5 and 6 – note that the RB-DB row is deliberately left blank to emphasise this.

The results obtained from using these methods and kernels show an improvement over the previous best results using statistical features extracted from the performance worm. We use the following shorthand to refer to the relevant algorithm/kernel combinations; **FB**: Previous best method using statistical features, **KPLS**: Kernel Partial Least Squares, **SVM**: Support Vector Machine, **KP-SV**: SVM using KPLS features, **KPCR**: Kernel Principal Components regression. If an n-gram kernel is used rather than a string kernel we append '-n' to the method name.

The use of the methods in this chapter in conjunction with the n-gram kernel (results shown in table 5) offer a clear performance advantage over the feature-based approach. This may suggest that for this kernel, projecting into a lower subspace is beneficial. The ability of KPLS however to correlate the feature

Table 5: Comparison of algorithms across each pairwise coupling of performers for the ngram kernel. Note that in all case the figures given are the number of movements correctly identified out of a maximum of 24. FB represents the previous best results using a feature-based representation rather than the 'performance alphabet' used for the other approaches.

Pairing	FB	KPLS-n	SVM-n	KP-SV-n	KPCR-n
RB - DB	–	–	–	–	–
GG - DB	15	18	21	22	14
GG - RB	17	21	20	20	11
MP - DB	17	16	18	18	17
MP - RB	21	18	18	17	12
MP - GG	17	20	22	22	18
AS - DB	15	15	17	15	10
AS - RB	16	17	19	20	14
AS - GG	17	18	18	19	13
AS - MP	20	20	22	22	14
MU - DB	17	13	9	14	12
MU - RB	16	18	15	17	12
MU - GG	16	16	20	20	14
MU - MP	15	17	18	21	16
MU - AS	17	16	17	16	17
Total	236	243	254	263	194
Average (%)	70.2	71.4	75.6	78.2	57.7

Table 6: Comparison of algorithms across each pairwise coupling of performers for the string kernel.

Pairing	FB	KPLS	SVM	KP-SV	KPCR
RB - DB	–	–	–	–	–
GG - DB	15	18	17	21	14
GG - RB	17	23	22	23	10
MP - DB	17	22	21	21	12
MP - RB	21	21	17	21	16
MP - GG	17	24	22	24	14
AS - DB	15	17	17	20	18
AS - RB	16	21	23	21	14
AS - GG	17	16	15	16	16
AS - MP	20	24	23	24	9
MU - DB	17	15	11	13	12
MU - RB	16	15	18	18	15
MU - GG	16	19	17	21	13
MU - MP	15	16	16	17	19
MU - AS	17	19	18	19	16
Total	236	270	257	279	198
Average (%)	70.2	80.4	76.5	83.0	58.9

directions it selects with the output variable gives it a clear advantage over KPCR and as expected from previous results on other data (Rosipal & Trejo, 2001; Bennett & Embrechts, 2003), a performance gain is achieved. When using the SVM in conjunction with the features obtained from the KPLS deflation steps, the performance improves further which has also been the case on other data sets (Rosipal et al., 2003).

In all cases short substrings (with substring lengths of only 1 or 2 characters) achieved the best performance, which would perhaps indicate that complex features are not used. Indeed when running the ngram kernel, any string length above 5 has very few non-zero off-diagonal entries and therefore contains almost no information. It is interesting to note that in the experiments KPCR requires more feature directions to achieve good performance, whereas KPLS consistently requires fewer directions to perform well.

The string kernel operating over the performance alphabet (results shown in Table 6) provides significantly better classification performance than the feature-based method and in every case also outperforms the n-gram kernel. This would indicate that the ability of the string kernel to allow gaps in matching subsequences is a key benefit for this data, and that complex features are indeed needed to obtain good performance. This is in contrast to results reported using the string kernel for text, where the classification rate of n-gram kernels using contiguous sequences is equal to that of the string kernel if not superior (Lodhi et al., 2002). For the string kernel, Interestingly, KPLS outperforms an SVM when using this kernel. As with the n-gram kernel we are able to observe that

Table 7: Performance of SDP using KPLS features in conjunction with the n-gram and string kernels. The KP-SV-n and KP-SV columns show the best performance achieved using ngram and string kernels respectively with the SVM an KPLS features when T was selected by cross-validation

Pairing	KP-SV-n	SDP-n	KP-SV	SDP
RB - DB	-	-	-	-
GG - DB	22	21	21	20
GG - RB	20	20	23	22
MP - DB	18	18	21	20
MP - RB	17	17	21	21
MP - GG	22	22	24	24
AS - DB	15	15	20	17
AS - RB	20	20	21	20
AS - GG	19	19	16	16
AS - MP	22	22	24	23
MU - DB	14	14	13	16
MU - RB	17	17	18	15
MU - GG	20	20	21	18
MU - MP	21	21	17	18
MU - AS	16	16	19	16
Total	263	262	279	266
Average (%)	78.2	78.0	83.0	79.0

using the SVM in conjunction with the features from the KPLS deflation further improve the performance.

When using the semi-definite programming approach outlined in Section 4.1, we no longer have to use cross-validation to select the parameter T ; however the subsequence length k and for the string kernel the gap penalty λ must still be chosen via cross-validation. The results for choosing k and λ based on the performance of the first pairing, then running the SDP optimisation for a combination of all feature directions are shown in table 7. For the case of the ngram kernel, the SDP solution performed well, however it suffered some performance loss when working with the string kernel. In the latter case the best performance on the first pairing was with a very low λ and k value ($\lambda = 0.2, k = 2$) - which is in contrast to the parameters selected for KP-SV. Interestingly in nearly all cases for all parameter settings, the SDP optimisation problem chose to weight the features beyond $T = 5$ with very little or no weight. This result is in keeping with observations from running the full experiments with all parameter settings; choosing $T > 5$ tends to lead to poor performance, hence most information is contained within the first few projections.

Overall the techniques used in this chapter offer a clear performance advantage over the feature based method which delivered the previous state of the art performance on this data set. As expected KPCR does not perform as well as KPLS, the ability to use label information when selecting feature directions

gives a clear performance benefit. Using KPLS features with an SVM improved performance greatly using the n-gram kernel, although this was not the case for the string kernel. This suggests that the feature space induced by the string kernel is able to capture more informative structure in the training examples, and is therefore particularly suited to this type of application.

6 Conclusions

In this chapter we have presented a novel application of the string kernel: to classify pianists by examining their playing style. This is an extremely complex task and has previously been attempted by analysing statistical features obtained from audio recordings. Here we have taken a different approach and have examined using feature-projection methods in conjunction with kernels which operate on text. These can be applied to the performer recognition problem by representing the performance as a string of characteristic tempo-loudness curves, which are obtained by analysing a performance worm. We have reviewed the Kernel Partial Least Squares method and shown how this can be successfully used to generate new features which can then be used in conjunction with learning methods such as an SVM. We have also shown a reweighting scheme for obtaining feature directions from KPLS that performs better than the technique used in current literature. All algorithms tested in this chapter provided higher performance than the previous state of the art results on the data. We have also shown that the ability of the string kernel to consider and match non-contiguous substrings of input sequence has a real performance benefit over only considering contiguous substrings. This is in contrast to many applications of the string kernel to text, where the relative performance of the string kernel to the n-gram kernel tends to be very close or even slightly worse. We also shown that it is feasible to use SemiDefinite programming approaches in practice in order to remove the need to select the number of feature directions via cross-validation. It is an open problem to determine in what circumstances using KPLS to obtain features will result in an improvement in generalisation performance and this will be researched in future. One other aspect that is perhaps unsatisfactory with these techniques is that these projections can still be quite time-consuming and are not sparse, therefore we would like to investigate the use of greedy methods within the KPLS feature extraction framework.

7 Acknowledgements

This work was supported in part by EPSRC grant no GR/S22301/01 ("Development and Application of String-Type Kernels"), the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778 and by the Austrian Fonds zur Förderung der Wissenschaftlichen Forschung (FWF) under grant Y99-INF. The Austrian Research Institute for Artificial Intelligence is supported by the Austrian Federal Ministry for Educa-

tion, Science, and Culture, and by the Austrian Federal Ministry for Transport, Innovation, and Technology.

References

- Bennett, K. P., & Embrechts, M. J. (2003). An optimization perspective on kernel partial least squares regression. *Advances in Learning Theory: Methods, Models and Applications. NATO Science Series III: Computer & Systems Science*, 190, 227–250.
- Dixon, S. (2001). Automatic extraction of tempo and beat from expressive performances. *Journal of New Music Research*, 30(1), 39–58.
- Dixon, S., Goebel, W., & Widmer, G. (2002). The performance worm: Real time visualisation of expression based on langner’s tempo-loudness animation. In *Proceedings of the International Computer Music Conference (ICMC 2002)*.
- Lanckriet, G., Cristianini, N., Bartlett, P., Ghaoui, L. E., & Jordan, M. I. (2004, Jan). Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5, 27–72.
- Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*(2), 419–444.
- Rosipal, R., & Trejo, L. (2001). Kernel partial least squares regression in reproducing kernel hilbert space. *Journal of Machine Learning Research*, 2, 97–123.
- Rosipal, R., Trejo, L., & Matthews, B. (2003). Kernel pls-svc for linear and nonlinear classification. In *Proceedings of the twentieth International Conference on Machine Learning (ICML-2003)*.
- Schölkopf, B., Smola, A., & Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*.
- Widmer, G., Dixon, S., Goebel, W., Pampalk, E., & Tobudic, A. (2003). In search of the Horowitz factor. *AI Magazine*, 3(24), 111–130.
- Witten, I., & Frank, E. (1999). *Data mining*. San Francisco, CA: Morgan Kaufmann.
- Wold, H. (1966). Estimation of principal components and related models by iterative least squares. *Multivariate Analysis*, 391–420.
- Zanon, P., & Widmer, G. (2003, August). Learning to recognise famous pianists with machine learning techniques. In *Proceedings of the Stockholm Music Acoustics Conference (SMAC '03)*.