1 A General Regression Framework for Learning String-to-String Mappings

Corinna Cortes
Google Research
1440 Broadway, New York, NY 10018
corinna@google.com

Mehryar Mohri
Courant Institute of Mathematical Sciences
251 Mercer Street, New York, NY 10012
mohri@cims.nyu.edu

Jason Weston
NEC Research
4 Independence Way, Princeton, NJ 08540
jasonw@nec-labs.com

1.1 Introduction

The problem of learning a mapping from strings to strings arises in many areas of text and speech processing. As an example, an important component of speech recognition or speech synthesis systems is a pronunciation model, which provides the possible phonemic transcriptions of a word, or a sequence of words. An accurate pronunciation model is crucial for the overall quality of such systems. Another typical task in natural language processing is part-of-speech tagging, which consists of assigning a part-of-speech tag, e.g., *noun*, *verb*, *preposition*, *determiner*, to each word of a sentence. Similarly, parsing can be viewed as a string-to-string mapping where the target alphabet contains additional symbols such as parentheses to equivalently represent the tree structure.

The problem of learning string-to-string mappings may seem similar to that of regression estimation where the task consists of learning a real-valued mapping. But, a key aspect of string-to-string mappings is that the target values, in this case strings, have some structure that can be exploited in learning. In particular, a similarity measure between target values can use the decomposition of the strings into their constituent symbols. More generally, since both input and output have some structure, one might wish to impose some constraints on the mappings based on prior knowledge about the task. A simple example is that of part-of-speech tagging where each tag is known to be associated with the word in the same position.

Several techniques have been described for learning string-to-string mappings, in particular Maximum Margin Markov Networks (M^{3}Ns) (Taskar et al., 2003; Bartlett et al., 2004) and SVMISOS (Tsochantaridis et al., 2004). These techniques treat the learning problem just outlined by learning a scoring function defined over the input-output pairs, imposing that the pair (x, y) with y matching x obtain a higher score than all other pairs (x, y'). This is done by using a binary loss function as in classification. This loss function ignores similarities between the output sequences. To correct for that effect, classification techniques such as SVMISOS (Tsochantaridis et al., 2004) craft an additional term in the loss function to help account for the closeness of the outputs y and y', but the resulting loss function is different from a regression loss function.

In contrast, this chapter introduces a general framework and algorithms that treat the problem of learning string-to-string mapping as a true *regression* problem. Seeking a regression-type solution is natural since it directly exploits the similarity measures between two possible target sequences y and y' associated with an input sequence x. Such similarity measures are improperly handled by the binary losses used in previous methods.

Our framework for learning string-to-string mappings can be viewed as a conceptually cleaner generalization of Kernel Dependency Estimation (KDE) (Weston et al., 2002). It decomposes the learning problem into two parts: a regression problem with a vector space image to learn a mapping from the input strings to an explicit or implicit feature space associated to the output strings, and a pre-image problem which consists of computing the output string from its feature space representation. We show that our framework can be generalized naturally to account for known constraints between the input and output sequences and that, remarkably, this generalization also leads to a closed form solution and to an efficient iterative algorithm, which provides a clean framework for estimating the regression coefficients. The pre-image is computed from these coefficients using a simple and efficient algorithm based on classical results from graph theory. A major computational advantage of our general regression framework over the binary loss learning techniques mentioned is that it does not require an exhaustive pre-image search of set of all output strings Y^* during training.

This chapter describes in detail our general regression framework and algorithms for string-to-string mappings and reports the results of experiments showing its effectiveness. The chapter is organized as follows. Section 1.2 presents a simple formulation of the learning problem and its decomposition into a regression problem with a vector space image and a pre-image problem. Section 1.3 presents several algorithmic solutions to the general regression problem, including the case of regression with prior constraints. Section 1.4 describes our pre-image algorithm for strings. Section 1.5 shows that several heuristic techniques can be used to substantially speed-up training. Section 1.6 compares our framework and algorithm with several other algorithms proposed for learning string-to-string mapping. Section 1.7 reports the results of our experiments in several tasks.

1.2 General Formulation

This section presents a general and simple regression formulation of the problem of learning string-to-string mappings.

Let X and Y be the alphabets of the input and output strings. Assume that a training sample of size m drawn according to some distribution D is given:

$$(x_1, y_1), \ldots, (x_m, y_m) \in X^* \times Y^*. \tag{1.1}$$

The learning problem that we consider consists of finding a hypothesis $f : X^* \to Y^*$ out of a hypothesis space H that predicts accurately the label $y \in Y^*$ of a string $x \in X^*$ drawn randomly according to D. In standard regression estimation problems, labels are real-valued numbers, or more generally elements of $\mathbb{R}^N$ with $N \geq 1$. Our learning problem can be formulated in a similar way after the introduction of a feature mapping $\Phi_Y : Y^* \to F_Y = \mathbb{R}^{N_2}$. Each string $y \in Y^*$ is thus mapped to an N_2-dimensional feature vector $\Phi_Y(y) \in F_Y$.

As shown by the diagram of Figure 1.1, our original learning problem is now decomposed into the following two problems:

- *Regression problem*: The introduction of Φ_Y leads us to the problem of learning a hypothesis $g : X^* \to F_Y$ predicting accurately the feature vector $\Phi_Y(y)$ for a string $x \in X^*$ with label $y \in Y^*$, drawn randomly according to D.
- *Pre-image problem*: to predict the output string $f(x) \in Y^*$ associated to $x \in X^*$, we must determine the pre-image of $g(x)$ by Φ_Y. We define $f(x)$ by:

$$f(x) = \operatorname*{argmin}_{y \in Y^*} \|g(x) - \Phi_Y(y)\|^2, \tag{1.2}$$

which provides an approximate pre-image when an exact pre-image does not exist ($\Phi_Y^{-1}(g(x)) = \emptyset$).

As with all regression problems, input strings in X^* can also be mapped to a Hilbert space F_X with $\dim(F_X) = N_1$, via a mapping $\Phi_X : X^* \to F_X$. Both mappings Φ_X and Φ_Y can be defined implicitly through the introduction of positive definite symmetric kernels K_X and K_Y such that for all $x, x' \in X^*,\ K_X(x, x') = \Phi_X(x) \cdot \Phi_X(x')$ and for all $y, y' \in Y^*,\ K_Y(y, y') = \Phi_Y(y) \cdot \Phi_Y(y')$.

This description of the problem can be viewed as a simpler formulation of the so-called Kernel Dependency Estimation (KDE) of (Weston et al., 2002). In the original presentation of KDE, the first step consisted of using K_Y and Kernel Principal Components Analysis to reduce the dimension of the feature space F_Y.

$$\begin{tikzcd}
X^* \arrow[r, "f"] \arrow[d, "\Phi_X"'] \arrow[dr, "g"] & Y^* \arrow[d, "\Phi_Y"] \\
F_X \arrow[r] & F_X \arrow[u, bend right, "\Phi_Y^{-1}"']
\end{tikzcd}$$

Figure 1.1 Decomposition of the string-to-string mapping learning problem into a regression problem (learning g) and a pre-image problem (computing Φ_Y^{-1} and using g to determine the string-to-string mapping f).

But, that extra step is not necessary and we will not require it, thereby simplifying that framework.

In the following two sections, we examine in more detail each of the two problems just mentioned (regression and pre-image problems) and present general algorithms for both.

1.3 Regression Problems and Algorithms

This section describes general methods for regression estimation when the dimension of the image vector space is greater than one. The objective functions and algorithms presented are not specific to the problem of learning string-to-string mapping and can be used in other contexts, but they constitute a key step in learning complex string-to-string mappings.

Different regression methods can be used to learn g, including kernel ridge regression (Saunders et al., 1998), Support Vector Regression (SVR) (Vapnik, 1995), or Kernel Matching Pursuit (KMP) (Vincent and Bengio, 2000). SVR and KMP offer the advantage of sparsity and fast training. But, a crucial advantage of kernel ridge regression in this context is, as we shall see, that it requires a single matrix inversion, independently of N_2, the number of features predicted. Thus, in the following we will consider a generalization of kernel ridge regression.

The hypothesis space we will assume is the set of all linear functions from F_X to F_Y. Thus, g is modeled as

$$\forall x \in X^*, g(x) = W(\Phi_X(x)), \tag{1.3}$$

where $W : F_X \to F_Y$ is a linear function admitting an $N_2 \times N_1$ real-valued matrix representation $\mathbf{W}$.

We start with a regression method generalizing kernel ridge regression to the case of vector space images. We will then further generalize this method to allow for the encoding of constraints between the input and output vectors.

1.3.1 Kernel Ridge Regression with Vector Space Images

For $i = 1, \ldots, m$, let $\mathbf{M}_{x_i} \in \mathbb{R}^{N_1 \times 1}$ denote the column matrix representing $\Phi_X(x_i)$ and $\mathbf{M}_{y_i} \in \mathbb{R}^{N_2 \times 1}$ the column matrix representing $\Phi_Y(y_i)$. We will denote by $\|\mathbf{A}\|_F^2 = \sum_{i=1}^p \sum_{j=1}^q \mathbf{A}_{ij}^2$ the Frobenius norm of a matrix $\mathbf{A} = (\mathbf{A}_{ij}) \in \mathbb{R}^{p \times q}$ and by $< \mathbf{A}, \mathbf{B} >_F = \sum_{i=1}^p \sum_{j=1}^q \mathbf{A}_{ij}\mathbf{B}_{ij}$ the Frobenius product of two matrices $\mathbf{A}$ and $\mathbf{B}$ in $\mathbb{R}^{p \times q}$. The following minimization problem:

$$\underset{\mathbf{W} \in \mathbb{R}^{N_2 \times N_1}}{\operatorname{argmin}} F(\mathbf{W}) = \sum_{i=1}^m \|\mathbf{W}\mathbf{M}_{x_i} - \mathbf{M}_{y_i}\|^2 + \gamma \|\mathbf{W}\|_F^2, \tag{1.4}$$

where $\gamma \geq 0$ is a regularization scalar coefficient, generalizes ridge regression to vector space images. The solution $\mathbf{W}$ defines the linear hypothesis g. Let $\mathbf{M}_X \in \mathbb{R}^{N_1 \times m}$ and $\mathbf{M}_Y \in \mathbb{R}^{N_2 \times m}$ be the matrices defined by:

$$\mathbf{M}_X = [\mathbf{M}_{x_1} \ldots \mathbf{M}_{x_m}] \qquad \mathbf{M}_Y = [\mathbf{M}_{y_1} \ldots \mathbf{M}_{y_m}]. \tag{1.5}$$

Then, the optimization problem 1.4 can be re-written as:

$$\underset{\mathbf{W} \in \mathbb{R}^{N_2 \times N_1}}{\operatorname{argmin}} F(\mathbf{W}) = \|\mathbf{W}\mathbf{M}_X - \mathbf{M}_Y\|_F^2 + \gamma \|\mathbf{W}\|_F^2. \tag{1.6}$$

Proposition 1 *The solution of the optimization problem 1.6 is unique and is given by either one of the following identities:*

$$\begin{aligned} \mathbf{W} &= \mathbf{M}_Y \mathbf{M}_X^\top (\mathbf{M}_X \mathbf{M}_X^\top + \gamma \mathbf{I})^{-1} && \text{(primal solution)} \\ \mathbf{W} &= \mathbf{M}_Y (\mathbf{K}_X + \gamma \mathbf{I})^{-1} \mathbf{M}_X^\top && \text{(dual solution).} \end{aligned} \tag{1.7}$$

where $\mathbf{K}_X \in \mathbb{R}^{m \times m}$ is the Gram matrix associated to the kernel K_X: $\mathbf{K}_{ij} = K_X(x_i, x_j)$.

Proof The function F is convex and differentiable, thus its solution is unique and given by $\nabla_{\mathbf{W}} F = 0$. Its gradient is given by:

$$\nabla_{\mathbf{W}} F = 2\left(\mathbf{W}\mathbf{M}_X - \mathbf{M}_Y\right)\mathbf{M}_X^\top + 2\gamma \mathbf{W}. \tag{1.8}$$

Thus,

$$\begin{aligned} \nabla_{\mathbf{W}} F = 0 \quad &\Leftrightarrow \quad 2(\mathbf{W}\mathbf{M}_X - \mathbf{M}_Y)\mathbf{M}_X^\top + 2\gamma \mathbf{W} = 0 \\ &\Leftrightarrow \quad \mathbf{W}(\mathbf{M}_X \mathbf{M}_X^\top + \gamma \mathbf{I}) = \mathbf{M}_Y \mathbf{M}_X^\top \\ &\Leftrightarrow \quad \mathbf{W} = \mathbf{M}_Y \mathbf{M}_X^\top (\mathbf{M}_X \mathbf{M}_X^\top + \gamma \mathbf{I})^{-1}, \end{aligned} \tag{1.9}$$

which gives the primal solution of the optimization problem. To derive the dual solution, observe that

$$\mathbf{M}_X^\top (\mathbf{M}_X \mathbf{M}_X^\top + \gamma \mathbf{I})^{-1} = (\mathbf{M}_X^\top \mathbf{M}_X + \gamma \mathbf{I})^{-1} \mathbf{M}_X^\top. \tag{1.10}$$

This can be derived without difficulty from a series expansion of $(\mathbf{M}_X\mathbf{M}_X^\top + \gamma\mathbf{I})^{-1}$. Since $K_X = \mathbf{M}_X^\top\mathbf{M}_X$,

$$\mathbf{W} = \mathbf{M}_Y \left(\mathbf{M}_X^\top\mathbf{M}_X + \gamma\mathbf{I}\right)^{-1}\mathbf{M}_X^\top = \mathbf{M}_Y \left(\mathbf{K}_X + \gamma\mathbf{I}\right)^{-1}\mathbf{M}_X^\top, \tag{1.11}$$

which is the second identity giving $\mathbf{W}$. ■

For both solutions, a single matrix inversion is needed. In the primal case, the complexity of that matrix inversion is in $O(N_1^3)$, or $O(N_1^{2+\alpha})$ with $\alpha < .376$, using the best known matrix inversion algorithms. When N_1, the dimension of the feature space F_X, is not large, this leads to an efficient computation of the solution. For large N_1 and relatively small m, the dual solution is more efficient since the complexity of the matrix inversion is then in $O(m^3)$, or $O(m^{2+\alpha})$.

Note that in the dual case, predictions can be made using kernel functions alone, as $\mathbf{W}$ does not have to be explicitly computed. For any $x \in X^*$, let $\mathbf{M}_x \in \mathbb{R}^{N_1\times 1}$ denote the column matrix representing $\Phi_X(x)$. Thus, $g(x) = \mathbf{W}\mathbf{M}_x$. For any $y \in Y^*$, let $\mathbf{M}_y \in \mathbb{R}^{N_2\times 1}$ denote the column matrix representing $\Phi_Y(y)$. Then, $f(x)$ is determined by solving the pre-image problem:

$$\begin{aligned}
f(x) &= \operatorname*{argmin}_{y\in Y^*} \|\mathbf{W}\mathbf{M}_x - \mathbf{M}_y\|^2 && (1.12)\\
&= \operatorname*{argmin}_{y\in Y^*} \left(\mathbf{M}_y^\top\mathbf{M}_y - 2\mathbf{M}_y^\top\mathbf{W}\mathbf{M}_x\right) && (1.13)\\
&= \operatorname*{argmin}_{y\in Y^*} \left(\mathbf{M}_y^\top\mathbf{M}_y - 2\mathbf{M}_y^\top\mathbf{M}_Y(\mathbf{K}_X + \gamma\mathbf{I})^{-1}\mathbf{M}_X^\top\mathbf{M}_x\right) && (1.14)\\
&= \operatorname*{argmin}_{y\in Y^*} \left(K_Y(y,y) - 2(\mathbf{K}_Y^y)^\top(\mathbf{K_X} + \gamma\mathbf{I})^{-1}\mathbf{K}_X^x\right), && (1.15)
\end{aligned}$$

where $\mathbf{K}_Y^y \in \mathbb{R}^{m\times 1}$ and $\mathbf{K}_X^x \in \mathbb{R}^{m\times 1}$ are the column matrices defined by:

$$\mathbf{K}_Y^y = \begin{bmatrix} K_Y(y,y_1)\\ \ldots \\ K_Y(y,y_m)\end{bmatrix} \quad\text{and}\quad \mathbf{K}_X^x = \begin{bmatrix} K_X(x,x_1)\\ \ldots \\ K_X(x,x_m)\end{bmatrix}. \tag{1.16}$$

1.3.2 Generalization to Regression with Constraints

In many string-to-string mapping learning tasks such as those appearing in natural language processing, there are some specific constraints relating the input and output sequences. For example, in part-of-speech tagging, a tag in the output sequence must match the word in the same position in the input sequence. More generally, one may wish to exploit the constraints known about the string-to-string mapping to restrict the hypothesis space and achieve a better result.

This section shows that our regression framework can be generalized in a natural way to impose some constraints on the regression matrix $\mathbf{W}$. Remarkably, this generalization also leads to a closed form solution and to an efficient iterative algorithm. Here again, the algorithm presented is not specific to string-to-string learning problems and can be used for other regression estimation problems.

Some natural constraints that one may wish to impose on $\mathbf{W}$ are linear constraints on its coefficients. To take these constraints into account, one can introduce additional terms in the objective function. For example, to impose that the coefficients with indices in some subset I_0 are null, or that two coefficients with indices in I_1 must be equal, the following terms can be added:

$$\beta_0 \sum_{(i,j) \in I_0} W_{ij}^2 + \beta_1 \sum_{(i,j,k,l) \in I_1} |W_{ij} - W_{kl}|^2, \tag{1.17}$$

with large values assigned to the regularization factors β_0 and β_1. More generally, a finite set of linear constraints on the coefficients of $\mathbf{W}$ can be accounted for in the objective function by the introduction of a quadratic form defined over W_{ij}, $(i,j) \in N_2 \times N_1$.

Let $N = N_2 N_1$, and denote by $\bar{\mathbf{W}}$ the $N \times 1$ column matrix whose components are the coefficients of the matrix $\mathbf{W}$. The quadratic form representing the constraints can be written as $< \bar{\mathbf{W}}, \mathbf{R}\bar{\mathbf{W}} >$, where $\mathbf{R}$ is a positive semi-definite symmetric matrix. By Cholesky's decomposition theorem, there exists a triangular matrix $\bar{\mathbf{A}}$ such that $\mathbf{R} = \bar{\mathbf{A}}^\top \bar{\mathbf{A}}$. Denote by $\bar{\mathbf{A}}_i$ the transposition of the ith row of $\bar{\mathbf{A}}$, $\bar{\mathbf{A}}_i$ is an $N \times 1$ column matrix, then

$$< \bar{\mathbf{W}}, \mathbf{R}\bar{\mathbf{W}} >=< \bar{\mathbf{W}}, \bar{\mathbf{A}}^\top \bar{\mathbf{A}} \bar{\mathbf{W}} >= ||\bar{\mathbf{A}}\bar{\mathbf{W}}||^2 = \sum_{i=1}^{N} < \bar{\mathbf{A}}_i, \bar{\mathbf{W}} >^2 . \tag{1.18}$$

The matrix $\bar{\mathbf{A}}_i$ can be associated to an $N_2 \times N_1$ matrix $\mathbf{A}_i$, just as $\bar{\mathbf{W}}$ is associated to $\mathbf{W}$, and $< \bar{\mathbf{A}}_i, \bar{\mathbf{W}} >^2 =< \mathbf{A}_i, \mathbf{W} >_F^2$. Thus, the quadratic form representing the linear constraints can be re-written in terms of the Frobenius products of $\mathbf{W}$ with N matrices:

$$< \bar{\mathbf{W}}, \mathbf{R}\bar{\mathbf{W}} >= \sum_{i=1}^{N} < \mathbf{A}_i, \mathbf{W} >_F^2, \tag{1.19}$$

with each $\mathbf{A}_i$, $i = 1, \ldots, N$, being an $N_2 \times N_1$ matrix. In practice, the number of matrices needed to represent the constraints may be far less than N, we will denote by C the number of constraint-matrices of the type $\mathbf{A}_i$ used.

Thus, the general form of the optimization problem including input-output constraints becomes:

$$\operatorname*{argmin}_{\mathbf{W} \in \mathbb{R}^{N_2 \times N_1}} F(\mathbf{W}) = \|\mathbf{W}\mathbf{M}_X - \mathbf{M}_Y\|_F^2 + \gamma \|\mathbf{W}\|_F^2 + \sum_{i=1}^{C} \eta_i < \mathbf{A}_i, \mathbf{W} >_F^2, \tag{1.20}$$

where $\eta_i \geq 0$, $i = 1, \ldots, C$, are regularization parameters. Since they can be factored in the matrices A_i by replacing A_i with $\sqrt{\eta_i} A_i$, in what follows, we will assume without loss of generality that $\eta_1 = \ldots = \eta_C = 1$.

Proposition 2 *The solution of the optimization problem 1.20 is unique and is given by the following identity:*

$$\mathbf{W} = (\mathbf{M}_Y \mathbf{M}_X^\top - \sum_{i=1}^{C} a_i\, \mathbf{A}_i)\mathbf{U}^{-1}, \tag{1.21}$$

with $\mathbf{U} = \mathbf{M}_X \mathbf{M}_X^\top + \gamma \mathbf{I}$ *and*

$$\begin{bmatrix} a_1 \\ \dots \\ a_C \end{bmatrix} = \big((<\mathbf{A}_i, \mathbf{A}_j \mathbf{U}^{-1}>)_{ij} + \mathbf{I}\big)^{-1} \begin{bmatrix} <\mathbf{M}_Y \mathbf{M}_X^\top \mathbf{U}^{-1}, \mathbf{A}_1> \\ \dots \\ <\mathbf{M}_Y \mathbf{M}_X^\top \mathbf{U}^{-1}, \mathbf{A}_C> \end{bmatrix}. \tag{1.22}$$

Proof The new objective function F is convex and differentiable, thus, its solution is unique and given by $\nabla_{\mathbf{W}} F = 0$. Its gradient is given by:

$$\nabla_{\mathbf{W}} F = 2\,(\mathbf{W}\mathbf{M}_X - \mathbf{M}_Y)\,\mathbf{M}_X^\top + 2\gamma \mathbf{W} + 2\sum_{i=1}^{C} <\mathbf{A}_i, \mathbf{W}>_F\, \mathbf{A}_i. \tag{1.23}$$

Thus,

$$\nabla_{\mathbf{W}} F = 0 \Leftrightarrow 2(\mathbf{W}\mathbf{M}_X - \mathbf{M}_Y)\mathbf{M}_X^\top + 2\gamma \mathbf{W} + \sum_{i=1}^{C} <\mathbf{A}_i, \mathbf{W}>_F\, \mathbf{A}_i = 0 \tag{1.24}$$

$$\Leftrightarrow \mathbf{W}(\mathbf{M}_X \mathbf{M}_X^\top + \gamma \mathbf{I}) = \mathbf{M}_Y \mathbf{M}_X^\top - \sum_{i=1}^{C} <\mathbf{A}_i, \mathbf{W}>_F\, \mathbf{A}_i \tag{1.25}$$

$$\Leftrightarrow \mathbf{W} = (\mathbf{M}_Y \mathbf{M}_X^\top - \sum_{i=1}^{C} <\mathbf{A}_i, \mathbf{W}>_F\, \mathbf{A}_i)(\mathbf{M}_X \mathbf{M}_X^\top + \gamma \mathbf{I})^{-1}. \tag{1.26}$$

To determine the solution $\mathbf{W}$, we need to compute the coefficients $<\mathbf{A}_i, \mathbf{W}>_F$. Let $\mathbf{M} = \mathbf{M}_Y \mathbf{M}_X^\top$, $a_i =<\mathbf{A}_i, \mathbf{W}>_F$ and $\mathbf{U} = \mathbf{M}_X \mathbf{M}_X^\top + \gamma \mathbf{I}$, then the last equation can be re-written as:

$$\mathbf{W} = (\mathbf{M} - \sum_{i=1}^{C} a_i\, \mathbf{A}_i)\mathbf{U}^{-1}. \tag{1.27}$$

Thus, for $j = 1, \dots, C$,

$$a_j =<\mathbf{A}_j, \mathbf{W}>_F=<\mathbf{A}_j, \mathbf{M}\mathbf{U}^{-1}>_F - \sum_{i=1}^{C} a_i\ <\mathbf{A}_j, \mathbf{A}_i \mathbf{U}^{-1}>_F, \tag{1.28}$$

which defines the following system of linear equations with unknowns a_j:

$$\forall j,\ 1 \le j \le C, \quad a_j + \sum_{i=1}^{C} a_i\ <\mathbf{A}_j, \mathbf{A}_i \mathbf{U}^{-1}>_F=<\mathbf{A}_j, \mathbf{M}\mathbf{U}^{-1}>_F\ . \tag{1.29}$$

Since u is symmetric, for all i, j, $1 \le i, j \le C$,

$$<\mathbf{A}_j, \mathbf{A}_i \mathbf{U}^{-1}>_F= \mathbf{tr}(\mathbf{A}_j^\top \mathbf{A}_i \mathbf{U}^{-1}) = \mathbf{tr}(\mathbf{U}^{-1} \mathbf{A}_j^\top \mathbf{A}_i) =<\mathbf{A}_j \mathbf{U}^{-1}, \mathbf{A}_i>_F\ . \tag{1.30}$$

Thus, the matrix $(<\mathbf{A}_i, \mathbf{A}_j\mathbf{U}^{-1}>_F)_{ij}$ is symmetric and $(<\mathbf{A}_i, \mathbf{A}_j\mathbf{U}^{-1}>_F)_{ij} + \mathbf{I}$ is invertible. The statement of the proposition follows. ■

Proposition 1.20 shows that, as in the constrained case, the matrix $\mathbf{W}$ solution of the optimization problem is unique and admits a closed form solution. The computation of the solution requires inverting matrix $\mathbf{U}$, as in the unconstrained case, which can be done in time $O(N_1^3)$. But it also requires, in the general case, the inversion of matrix $(<\mathbf{A}_i, \mathbf{A}_j\mathbf{U}^{-1}>_F)_{ij} + \mathbf{I}$ which can be done in $O(C^3)$. For large C, C close to N, the space and time complexity of this matrix inversion may become prohibitive.

Instead, one can use an iterative method for computing $\mathbf{W}$, using Equation 1.31:

$$\mathbf{W} = (\mathbf{M}_Y\mathbf{M}_X^\top - \sum_{i=1}^{C} <\mathbf{A}_i, \mathbf{W}>_F \mathbf{A}_i)\mathbf{U}^{-1}, \tag{1.31}$$

and starting with the solution of the unconstrained solution:

$$\mathbf{W}_0 = \mathbf{M}_Y\mathbf{M}_X^\top\mathbf{U}^{-1}. \tag{1.32}$$

At iteration k, $\mathbf{W}_{k+1}$ is determined by interpolating its value at the previous iteration with the one given by Equation 1.31:

$$\mathbf{W}_{k+1} = (1-\alpha)\mathbf{W}_k + \alpha\,(\mathbf{M}_Y\mathbf{M}_X^\top - \sum_{i=1}^{C} <\mathbf{A}_i, \mathbf{W}_k>_F \mathbf{A}_i)\mathbf{U}^{-1}, \tag{1.33}$$

where $0 \leq \alpha \leq 1$. Let $\mathbf{P} \in \mathbb{R}^{(N_2\times N_1)\times(N_2\times N_1)}$ be the matrix such that

$$\mathbf{PW} = \sum_{i=1}^{C} <\mathbf{A}_i, \mathbf{W}>_F \mathbf{A}_i\mathbf{U}^{-1}. \tag{1.34}$$

The following theorem proves the convergence of this iterative method to the correct result when α is sufficiently small with respect to a quantity depending on the largest eigenvalue of $\mathbf{P}$. When the matrices $\mathbf{A}_i$ are sparse, as in many cases encountered in practice, the convergence of this method can be very fast.

Theorem 3 *Let $\lambda_{\max}$ be the largest eigenvalue of $\mathbf{P}$. Then, $\lambda_{\max} \geq 0$ and for $0 < \alpha < \min\left\{\frac{2}{\lambda_{\max}+1}, 1\right\}$, the iterative algorithm just presented converges to the unique solution of the optimization problem 1.20.*

Proof We first show that the eigenvalues of $\mathbf{P}$ are all non-negative. Let $\mathbf{X}$ be an eigenvector associated to an eigenvalue λ of $\mathbf{P}$. By definition,

$$\sum_{i=1}^{C} <\mathbf{A}_i, \mathbf{X}>_F \mathbf{A}_i\mathbf{U}^{-1} = \lambda\mathbf{X}. \tag{1.35}$$

Taking the dot product of each side with $\mathbf{XU}$ yields:

$$\sum_{i=1}^{C} < \mathbf{A}_i, \mathbf{X} >_F < \mathbf{A}_i \mathbf{U}^{-1}, \mathbf{XU} >_F = \lambda < \mathbf{X}, \mathbf{XU} >_F . \tag{1.36}$$

Since $\mathbf{U} = \mathbf{M}_X \mathbf{M}_X^\top + \gamma \mathbf{I}$, it is symmetric and positive and by Cholesky's decomposition theorem, there exists a matrix $\mathbf{V}$ such that $\mathbf{U} = \mathbf{V}\mathbf{V}^\top$. Thus,

$$\begin{aligned} < \mathbf{X}, \mathbf{XU} >_F &= < \mathbf{X}, \mathbf{XVV}^\top >_F = \mathbf{tr}(\mathbf{X}^\top \mathbf{XVV}^\top) = \mathbf{tr}(\mathbf{V}^\top \mathbf{X}^\top \mathbf{XV}) \\ &= \mathbf{tr}((\mathbf{XV})^\top \mathbf{XV}) = \|\mathbf{XV}\|_F^2, \end{aligned} \tag{1.37}$$

where we used the property $\mathbf{tr}(\mathbf{AB}) = \mathbf{tr}(\mathbf{BA})$, which holds for all matrices $\mathbf{A}$ and $\mathbf{B}$. Now, using this same property and the fact that $\mathbf{U}$ is symmetric, for $i = 1, \ldots, C$,

$$< \mathbf{A}_i \mathbf{U}^{-1}, \mathbf{XU} >_F = \mathbf{tr}(\mathbf{U}^{-1} \mathbf{A}_i^\top \mathbf{XU}) = \mathbf{tr}(\mathbf{A}_i^\top \mathbf{XUU}^{-1}) = < \mathbf{A}_i, \mathbf{X} >_F . \tag{1.38}$$

Thus, Equation 1.36 can be re-written as:

$$\sum_{i=1}^{C} < \mathbf{A}_i, \mathbf{X} >_F^2 = \lambda \|\mathbf{XV}\|_F^2. \tag{1.39}$$

If $\|\mathbf{XV}\|_F^2 = 0$, then for $i = 1, \ldots, C$, $< \mathbf{A}_i, \mathbf{X} >_F = 0$, which implies $\mathbf{PX} = 0 = \lambda \mathbf{X}$ and $\lambda = 0$. Thus, if $\lambda \neq 0$, $\|\mathbf{XV}\|_F^2 \neq 0$, and in view of Equation 1.39,

$$\lambda = \frac{\sum_{i=1}^{C} < \mathbf{A}_i, \mathbf{X} >_F^2}{\|\mathbf{XV}\|_F^2} \geq 0. \tag{1.40}$$

Thus, all eigenvalues of $\mathbf{P}$ are non-negative. This implies in particular that $\mathbf{I} + \mathbf{P}$ is invertible.

Equation 1.31 giving $\mathbf{W}$ can be rewritten as

$$\mathbf{W} = \mathbf{W}_0 - \mathbf{PW} \Longleftrightarrow \mathbf{W} = (\mathbf{I} + \mathbf{P})^{-1} \mathbf{W}_0. \tag{1.41}$$

with $\mathbf{W}_0 = \mathbf{M}_Y \mathbf{M}_X^\top \mathbf{U}^{-1}$, which gives the unique solution of the optimization problem 1.20. Using the same notation, the iterative Definition 1.33 can written for all $n \geq 0$ as

$$\mathbf{W}_{n+1} = (1 - \alpha)\mathbf{W}_n + \alpha\,(\mathbf{W}_0 - \mathbf{PW}_n). \tag{1.42}$$

For $n \geq 0$, define $\mathbf{V}_{n+1}$ by $\mathbf{V}_{n+1} = \mathbf{W}_{n+1} - \mathbf{W}_n$. Then,

$$\forall n \geq 0, \quad \mathbf{V}_{n+1} = -\alpha((\mathbf{I} + \mathbf{P})\mathbf{W}_n - \mathbf{W}_0). \tag{1.43}$$

Thus, for all $n \geq 0$, $\mathbf{V}_{n+1} - \mathbf{V}_n = -\alpha(\mathbf{I} + \mathbf{P})\mathbf{V}_n$. Summing these equalities gives:

$$\forall n \geq 0, \quad \mathbf{V}_{n+1} = [\mathbf{I} - \alpha(\mathbf{I} + \mathbf{P})]^n \mathbf{V}_1. \tag{1.44}$$

Assume that $0 < \alpha < 1$. Let μ be an eigenvalue of $\mathbf{I} - \alpha(\mathbf{I} + \mathbf{P})$, then, there exists $\mathbf{X} \neq 0$ such that

$$\mathbf{I} - \alpha(\mathbf{I} + \mathbf{P})X = \mu X \Longleftrightarrow \mathbf{PX} = \frac{(1-\alpha)-\mu}{\alpha}\mathbf{X}. \tag{1.45}$$

Thus, $\frac{(1-\alpha)-\mu}{\alpha}$ must be an eigenvalue of $\mathbf{P}$. Since the eigenvalues of $\mathbf{P}$ are non-negative,

$$0 \leq \frac{(1-\alpha)-\mu}{\alpha} \Longrightarrow \mu \leq 1 - \alpha < 1. \tag{1.46}$$

By definition of $\lambda_{\max}$,

$$\frac{(1-\alpha)-\mu}{\alpha} \leq \lambda_{\max} \Longrightarrow \mu \geq (1-\alpha) - \alpha\lambda_{\max} = 1 - \alpha(\lambda_{\max} + 1). \tag{1.47}$$

Thus, for $\alpha < \frac{2}{\lambda_{\max}+1}$,

$$\mu \geq 1 - \alpha(\lambda_{\max} + 1) > 1 - 2 = -1. \tag{1.48}$$

By Equations 1.46 and 1.48, any eigenvalue μ of $\mathbf{I} - \alpha(\mathbf{I} + \mathbf{P})$ verifies $|\mu| < 1$. Thus, in view of Equation 1.48,

$$\lim_{n\to\infty} \mathbf{V}_{n+1} = \lim_{n\to\infty} [\mathbf{I} - \alpha(\mathbf{I} + \mathbf{P})]^n \mathbf{V}_1 = 0. \tag{1.49}$$

By Equation 1.43 and the continuity of $(\mathbf{I} + \mathbf{P})^{-1}$, this implies that

$$\lim_{n\to\infty} (\mathbf{I} + \mathbf{P})\mathbf{W}_n = \mathbf{W}_0 \Longrightarrow \lim_{n\to\infty} \mathbf{W}_n = (\mathbf{I} + \mathbf{P})^{-1}\mathbf{W}_0, \tag{1.50}$$

which proves the convergence of the iterative algorithm to the unique solution of the optimization problem 1.20. ■

Corollary 4 *Assume that* $0 < \alpha < \min\left\{\frac{2}{\|P\|+1}, 1\right\}$*, then the iterative algorithm presented converges to the unique solution of the optimization problem 1.20.*

Proof Since $\lambda_{\max} \leq \|P\|$, the results follows directly Theorem 3. ■

Thus, for smaller values of α, the iterative algorithm converges to the unique solution of the optimization problem 1.20. In view of the proof of the theorem, the algorithm converges at least as fast as in

$$O(\max\{|1-\alpha|^n, |(1-\alpha) - \alpha\lambda_{\max}|^n\}) = O(|1-\alpha|^n). \tag{1.51}$$

The extra terms we introduced in the optimization function to account for known constraints on the coefficients of $\mathbf{W}$ (Equation 1.20), together with the existing term $\|\mathbf{W}\|_F^2$, can be viewed as a new and more general regularization term for $\mathbf{W}$. This idea can be used in other contexts. In particular, in some cases, it could be beneficial to use more general regularization terms for the weight vector in support vector machines. We leave the specific analysis of such general regularization terms to a future study.

1.4 Pre-Image Solution for Strings

1.4.1 A General Problem: Finding Pre-Images

A critical component of our general regression framework is the pre-image computation. This consists of determining the predicted output: given $z \in F_Y$, the problem consists of finding $y \in Y^*$ such that $\Phi_Y(y) = z$, see Figure 1.1. Note that this is a general problem, common to all kernel-based structured output problems, including Maximum Margin Markov Networks Taskar et al. (2003) and SVMISOS Tsochantaridis et al. (2004) although it is not explicitly described and discussed by the authors (see Section 1.6).

Several instances of the pre-image problem have been studied in the past in cases where the pre-images are fixed-size vectors Schölkopf and Smola (2002). The pre-image problem is trivial when the feature mapping Φ_Y corresponds to polynomial kernels of odd degree since Φ_Y is then invertible. There also exists a fixed-point iteration approach for RBF kernels. In the next section, we describe a new pre-image technique for strings that works with a rather general class of string kernels.

1.4.2 n-gram Kernels

n-gram kernels form a general family of kernels between strings, or more generally weighted automata, that measure the similarity between two strings using the counts of their common n-gram sequences. Let $|x|_u$ denote the number of occurrences of u in a string x, then, the n-gram kernel k_n between two strings y_1 and y_2 in Y^*, $n \geq 1$, is defined by:

$$k_n(y_1, y_2) = \sum_{|u|=n} |y_1|_u \, |y_2|_u, \tag{1.52}$$

where the sum runs over all strings u of length n. These kernels are instances of *rational kernels* and have been used successfully in a variety of difficult prediction tasks in text and speech processing (Cortes et al., 2004).

1.4.3 Pre-Image Problem for n-gram Kernels

The pre-image problem for n-gram kernels can be formulated as follows. Let Σ be the alphabet of the strings considered. Given $z = (z_1, \ldots, z_l)$, where $l = |\Sigma|^n$ and z_k is the count for an n-gram sequence u_k, find string y such that for $k = 1, \ldots, l$, $|y|_{u_k} = z_k$. Several standard problems arise in this context: the existence of y given z, its uniqueness when it exists, and the need for an efficient algorithm to determine y when it exists. We will address all these questions in the following sections.

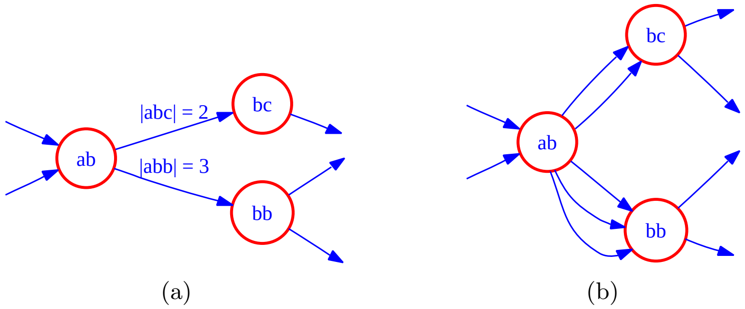

Figure 1.2 (a) The De Bruijn graph $G_{z,3}$ associated with the vector z in the case of trigrams ($n = 3$). The weight carried by the edge from vertex ab to vertex bc is the number of occurrences of the trigram abc as specified by the vector z. (b) The expanded graph $H_{z,3}$ associated with $G_{z,3}$. An edge in $G_{z,3}$ is repeated as many times as there were occurrences of the corresponding trigram.

1.4.4 Equivalent Graph-Theoretical Formulation of the Problem

The pre-image problem for n-gram kernels can be formulated as a graph problem by considering the *De Bruijn graph* $G_{z,n}$ associated with n and the vector z (van Lint and Wilson, 1992). $G_{z,n}$ is the graph constructed in the following way: associate a vertex to each (n-1)-gram sequence and add an edge from the vertex identified with $a_1 a_2 \ldots a_{n-1}$ to the vertex identified with $a_2 a_3 \ldots a_n$ weighted with the count of the n-gram $a_1 a_2 \ldots a_n$. The De Bruijn graph can be expanded by replacing each edge carrying weight c with c identical unweighted edges with the same original and destination vertices. Let $H_{z,n}$ be the resulting unweighted graph.

The problem of finding the string y is then equivalent to that of finding an *Euler circuit* of $H_{z,n}$, that is a circuit on the graph in which each edge is traversed exactly one (van Lint and Wilson, 1992). Each traversal of an edge between $a_1 a_2 \ldots a_{n-1}$ and $a_1 a_2 \ldots a_n$ corresponds to the consumption of one instance of the n-gram $a_1 a_2 \ldots a_n$. Figure 1.2 illustrates the construction of the graphs $G_{z,n}$ and $H_{z,n}$ in a special case.

1.4.5 Existence

The problem of finding an Eulerian circuit of a graph is a classical problem. Let *in-degree*(q) denote the number of incoming edges of vertex q and *out-degree*(q) the number of outgoing edges. The following theorem characterizes the cases where the pre-image y exists.

Theorem 5 *The vector z admits a pre-image iff for any vertex q of $H_{z,n}$, in-degree(q) = out-degree(q).*

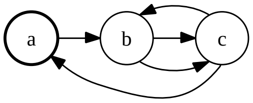

Figure 1.3 Example of a pre-image computation. The graph is associated with the vector $z = (0, 1, 0, 0, 0, 2, 1, 1, 0)$ whose coordinates indicate the counts of the bigrams $aa, ab, ac, ba, bb, bc, ca, cb, cc$. The graph verifies the conditions of theorem 5, thus it admits an Eulerian circuit, which in this case corresponds to the pre-image $y = bcbca$ if we start from the vertex a which can serve here as both the start and end symbol.

Proof The proof is a direct consequence of the graph formulation of the problem and a classical result related to the problem of Euler (1736) Wilson, R. J. (1979).

■

1.4.6 Compact Algorithm

There exists a linear-time algorithm for determining an Eulerian circuit of a graph verifying the conditions of theorem 5 Wilson, R. J. (1979). Here, we give a simple, compact, and recursive algorithm that produces the same result as that algorithm with the same linear complexity:

$$O(|H_{z,n}|) = O(\sum_{i=1}^{l} z_i) = O(|y|). \tag{1.53}$$

Note that the complexity of the algorithm is optimal since writing the output sequence y takes the same time $(O(|y|))$. The following is the pseudocode of our algorithm.

```
EULER($q$)
1  path ← $\epsilon$
2  for each unmarked edge $e$ leaving $q$ do
3        MARK($e$)
4        path ← $e$ EULER($dest(e)$) path
5  return path
```

A call to the function EULER with argument q returns a path corresponding to an Eulerian circuit from q. Line 1 initializes the path to the empty path. Then, each time through the loop of lines 2-4, a new outgoing edge of q is examined. If it has not been previously marked (line 3), then *path* is set to the concatenation of the edge e with the path returned by a call to EULER with the destination vertex of e and the old value of *path*.

While this is a very simple algorithm for generating an Eulerian circuit, the proof of its correctness is in fact not as trivial as that of the standard Euler algorithm. However, its compact form makes it easy to modify and analyze the effect of the modifications.

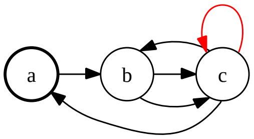

Figure 1.4 Case of non-unique pre-images. Both *bcbcca* and *bccbca* are possible pre-images. Our Euler algorithm can produce both solutions, depending on the order in which outgoing edges of the vertex c are examined. The graph differs from that of Figure 1.3 only by self-loop at the vertex identified with c.

1.4.7 Uniqueness

In general, when it exists, the pre-image sequence is not unique. Figure 1.4 gives a simple example of a graph with two distinct Eulerian circuits and distinct pre-image sequences. A recent result of Kontorovich (2004) gives a characterization of the set of strings that are unique pre-images. Let Φ_n be the feature mapping corresponding to n-gram sequences, that is, $\Phi_n(y)$ is the vector whose components are the counts of the n-grams appearing in y.

Theorem 6 (Kontorovich (2004)) *The set of strings y such that $\Phi(y)$ admits a unique pre-image is a regular language.*

In all cases, our algorithm can generate all possible pre-images starting from a given (n-1)-gram. Indeed, different pre-images simply correspond to different orders of examining outgoing edges. In practice, for a given vector z, the number of outgoing edges at each vertex is small (often 1, rarely 2). Thus, the extra cost of generating all possible pre-images is very limited.

1.4.8 Generalized Algorithm

The algorithm we presented can be used to generate efficiently all possible pre-images corresponding to a vector z when it admits a pre-image. However, due to regression errors, the vector z might not admit a pre-image. Also, as a result of regression, the components of z may be non-integer.

One solution to this problem is to round the components to obtain integer counts. As we shall see, incrementing or decrementing a component by one only leads to the local insertion or deletion of one symbol.

To deal with regression errors and the fact that y might not admit a pre-image, we can simply use the same algorithm. To allow for cases where the graph is not connected, the function Euler is called at each vertex q whose outgoing edges are not all marked. The resulting path is the concatenation of the paths returned by different calls to this function.

The algorithm is guaranteed to return a string y whose length is $|y| = \sum_{i=1}^{l} z_i$ since each edge of the graph $H_{z,n}$ is visited exactly once. Clearly, the result is a pre-image when z admits one. But, how different is the output string y from the

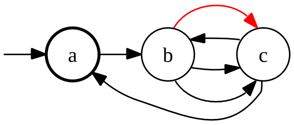

Figure 1.5 Illustration of the application of the generalized algorithm to the case of a graph that does not admit the Euler property. The graph differs from that of Figure 1.3 by just one edge (edge in red). The possible pre-images returned by the algorithm are *bccbca* and *bcbcca*.

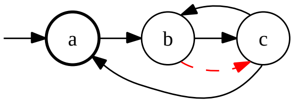

Figure 1.6 Further illustration of the application of the generalized algorithm to the case of a graph that does not admit the Euler property. The graph differs from that of Figure 1.3 by just one edge (the missing edge in red). The pre-image returned by the algorithm is *bcba*.

original pre-image when we modify the count of one of the components of z by one, either by increasing or decreasing it? Figures 1.3 and 1.4 can serve to illustrate that in a special case since the graph of Figure 1.4 differs from that of Figure 1.3 by just one edge, which corresponds to the existence or not of the bigram *cc*.[1] The possible pre-images output by the algorithm given the presence of the bigram *cc* only differ from the pre-image in the absence of the bigram *cc* by one letter, *c*. Their edit-distance is one. Furthermore, the additional symbol *c* cannot appear at any position in the string, its insertion is only *locally* possible.

Figure 1.5 illustrates another case where the graph differs from that of Figure 1.3 by one edge corresponding to the bigram *bc*. As in the case just discussed, the potential pre-images can only contain one additional symbol, *c*, which is inserted locally.

Figure 1.6 illustrates yet another case where the graph differs from that of Figure 1.3 by one edge missing which corresponds to the bigram *bc*. The graph does not have the Euler property. Yet, our algorithm can be applied and outputs the pre-image *bcba*.

Thus, in summary, the algorithm we presented provides a simple and efficient solution to the pre-image problem for strings for the family of n-gram kernels. It also has the nice property that changing a coordinate in feature space has minimal impact on the actual pre-image found.

One can use additional information to further enhance the accuracy of the pre-image algorithm. For example, if a large number of sequences over the target

[1] We can impose the same start and stop symbol, a, for all sequences.

alphabet is available, we can create a statistical model such as an n-gram model based on those sequences. When the algorithm generates several pre-images, we can use that statistical model to rank these different pre-images by exploiting output symbol correlations. In the case of n-gram models, this can be done in linear time in the sum of the lengths of the pre-image sequences output by the algorithm.

1.5 Speeding-up Training

This section examines two techniques for speeding up training when using our general regression framework.

1.5.1 Incomplete Cholesky Decomposition

One solution is to apply incomplete Cholesky decomposition to the kernel matrix $\mathbf{K}_X \in \mathbb{R}^{m \times m}$ (Bach and Jordan, 2002). This consists of finding a matrix $\mathbf{L} \in \mathbb{R}^{m \times n}$, with $n \ll m$, such that:

$$\mathbf{K}_X = \mathbf{L}\mathbf{L}^\top. \tag{1.54}$$

Matrix $\mathbf{L}$ can be found in time $O(mn^2)$ using an incomplete Cholesky decomposition which takes $O(mn^2)$ operations (see e.g., (Bach and Jordan, 2002), for the use of this technique in the context of kernel independent component analysis). To invert $\mathbf{K}_X + \gamma\mathbf{I}$ one can use the so-called inversion lemma or Woodbury formula:

$$(\mathbf{A} + \mathbf{B}\mathbf{C})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}(\mathbf{I} + \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}\mathbf{A}^{-1}, \tag{1.55}$$

which here leads to:

$$(\gamma\mathbf{I} + \mathbf{L}\mathbf{L}^\top)^{-1} = \frac{1}{\gamma}[\mathbf{I} - \mathbf{L}\ (\gamma\mathbf{I} + \mathbf{L}^\top\mathbf{L})^{-1}\ \mathbf{L}^\top]. \tag{1.56}$$

Since $(\gamma\mathbf{I}+\mathbf{L}^\top\mathbf{L}) \in \mathbb{R}^{n \times n}$, the cost of the matrix inversion in the dual computation is reduced from $O(m^3)$ to $O(n^3)$. This method can thus be quite effective at reducing the computational cost. Our experiments (Section 1.7.4) have shown however that the simple greedy technique described in the next section is often far more effective.

1.5.2 Greedy Technique

This consists, as with Kernel Matching Pursuit (KMP) (Vincent and Bengio, 2000), of defining a subset $n \ll m$ of kernel functions in an incremental fashion. This subset is then used to define the expansion. Consider the case of a finite-dimensional output space:

$$g(x) = (\sum_{i \in S} \alpha_{i1} K_X(x_i, x), \ldots, \sum_{i \in S} \alpha_{iN_2} K_X(x_i, x)), \tag{1.57}$$

where S is the set of indices of the kernel functions used in the expansion, initialized to $\emptyset$. The algorithm then consists of repeating the following steps so long as $|S| < n$:

1. Determine the training point x_j with the largest residual:

$$j = \operatorname*{argmax}_{i \in \{1,\ldots,m\} \setminus S} ||\Phi_Y(y_i) - g(x_i)||^2; \tag{1.58}$$

2. Add x_j to the set of "support vectors" and update α:

$$\begin{aligned} & S \leftarrow S \cup \{x_j\}; \\ & \alpha \leftarrow \operatorname*{argmin}_{\hat{\alpha}} \textstyle\sum_{i=1}^m ||\Phi_Y(y_i) - g(x_i)||^2. \end{aligned} \tag{1.59}$$

The matrix inversion required in step 2 is done with $\alpha = {\mathbf{K}_S}^{-1} \mathbf{K}_{S,*} Y$ where $\mathbf{K}_S$ is the kernel matrix between input examples indexed by S only, and $\mathbf{K}_{S,*}$ is the kernel matrix between S and all other examples.

In practice, this can be computed incrementally via rank one updates (Smola and Bartlett, 2001), which results in a running time complexity of $O(nm^2N_2)$ as with KMP (Vincent and Bengio, 2000) (but there, $N_2 = 1$). A further linear speed-up is possible by restricting the subset of the data points in step 1. Note that this approach differs from KMP in that we select basis functions that approximate all the output dimensions at once, resulting in faster evaluation times. The union of the support vectors over all output dimensions is indeed smaller.

One could also apply this procedure to the iterative regression algorithm incorporating constraints of Section 1.3.2. One would need to add a "support vector" greedily as above, run several iterations of update rule given in equation 1.27, and then repeat n times.

1.6 Comparison with Other Algorithms

This section compares our regression framework for learning string-to-string mappings with other algorithms described in the literature for the same task. It describes the objective function optimized by these other algorithms, which can all be viewed as classification-based, and points out important differences in the computational complexity of these algorithms.

1.6.1 Problem Formulation

Other existing algorithms for learning a string-to-string mapping (Collins and Duffy, 2001; Tsochantaridis et al., 2004; Taskar et al., 2003) formulate the problem as that of learning a function $f : X \times Y \to \mathbb{R}$ defined over the pairs of input-output strings, such that the output $\hat{y}(x)$ associated to the input x is given by

$$\hat{y}(x) = \operatorname*{argmax}_{y \in Y} f(x, y). \tag{1.60}$$

The hypothesis space considered is typically that of linear functions. Pairs of input-output strings are mapped to a feature space F via a mapping $\Phi_{XY} : X \times Y \rightarrow F$. Thus, the function f learned is defined as

$$\forall (x, y) \in X \times Y, \quad f(x, y) = w \cdot \Phi_{XY}(x, y), \tag{1.61}$$

where $w \in F$. Thus, a joint embedding Φ_{XY} is used, unlike the separate embeddings Φ_X and Φ_Y adopted in our framework. The learning problem can be decomposed into the following two problems, as in our case:

- learning the weight vector w to determine the linear function f, which is similar to our problem of determining W;
- given x, computing $\hat{y}(x)$ following Equation 1.60, which is also a pre-image problem. Indeed, let g_x be the function defined by $g_x(y) = f(x, y)$, then

$$\hat{y}(x) = \operatorname*{argmax}_{y \in Y} g_x(y). \tag{1.62}$$

When the joint embedding Φ_{XY} can be decomposed as

$$\Phi_{XY}(x, y) = \Phi_X(x) \otimes \Phi_Y(y), \tag{1.63}$$

where $\otimes$ indicates the tensor product of the vectors, the hypothesis functions sought coincide with those assumed in our regression framework. In both cases, the problem then consists of learning a matrix $\mathbf{W}$ between the same two spaces as in our approach. The only difference lies is the choice of the loss function.

Other joint embeddings Φ_{XY} may help encode conveniently some prior knowledge about the relationship between input and output sequences (see Weston et al. (2004) for an empirical analysis of the benefits of joint feature spaces). For example, with

$$\langle \Phi_{XY}(x, y), \Phi_{XY}(x', y') \rangle = \langle \Phi_X(x) \otimes \Phi_Y(y), \mathbf{R}\ (\Phi_X(x') \otimes \Phi_Y(y')) \rangle \tag{1.64}$$

the matrix $\mathbf{R}$ can be used to favor some terms of $\mathbf{W}$. In our method, such relationships can also be accounted for by imposing some constraints on the matrix $\mathbf{W}$ and solving the generalized regression problem with constraints described in Section 1.3.2.

The weight vector w in previous techniques is learned using the kernel perceptron algorithm (Collins and Duffy, 2001), or a large-margin maximization algorithm (Tsochantaridis et al., 2004; Taskar et al., 2003). These techniques treat the learning problem outlined above by imposing that the pair (x, y) with y matching x obtain a higher score than all other pairs (x, y'). This is done by using a binary loss function as in classification which ignores the similarities between the output sequences. To correct for that effect, classification techniques such as SVMISOS (Tsochantaridis et al., 2004) modify the binary loss function to impose the following condition for $i = 1, \ldots, m$ and any $y \in Y - \{y_i\}$

$$f(x_i, y_i) > f(x_i, y) + L(y_i, y), \tag{1.65}$$

where L is a loss function based on the output strings. This makes the loss function similar, though not equivalent, to the objective function of a regression problem. To further point out this similarity, consider the case of the joint embedding (1.63). The inequality can then be re-written as

$$\Phi_Y(y_i)^\top \mathbf{W}\Phi_X(x_i) - \Phi_Y(y)^\top \mathbf{W}\Phi_X(x_i) \geq L(y_i, y). \tag{1.66}$$

In our general regression framework, using Equation 1.4, a solution with zero empirical error, i.e., $\mathbf{W}\Phi_X(x_i) = \Phi_Y(y_i)$ for $i = 1, \ldots, m$, verifies the following equality:

$$||\mathbf{W}\Phi_X(x_i) - \Phi_Y(y)||^2 = ||\mathbf{W}\Phi_X(x_i) - \Phi_Y(y_i)||^2 + L(y_i, y), \tag{1.67}$$

where $L(y_i, y) = ||\Phi_Y(y_i) - \Phi_Y(y)||^2$. Assuming that the outputs are normalized, i.e., $||\Phi_Y(y)|| = 1$ for all y, this equation is equivalent to:

$$\Phi_Y(y_i)^\top \mathbf{W}\Phi_X(x_i) - \Phi_Y(y)^\top \mathbf{W}\Phi_X(x_i) = \frac{1}{2}L(y_i, y), \tag{1.68}$$

which is similar to the zero-one loss constraint of Equation 1.66 with the inequality replaced with an equality here.

We argue that in structured output prediction problems, it is often natural to introduce a similarity measure capturing the closeness of the outputs. In view of that, minimizing the corresponding distance is fundamentally a regression problem.

1.6.2 Computational Cost

The computational cost of other techniques for learning string-to-string mapping significantly differs from that of our general regression framework.

In the case of other techniques, a pre-image computation is required at every iteration during training, and the algorithms can be shown to converge in polynomial time if the pre-image computation itself can be computed in polynomial time. In our case, pre-image calculations are needed only at testing time, and do not affect training time. Since the pre-image computation may be often very costly, this can represent a substantial difference in practice.

The complexity of our Euler circuit string pre-image algorithm is linear in the length of the pre-image string y it generates. It imposes no restriction on the type of regression technique used, nor does it constrain the choice of the features over the input X.

The computation of the pre-image in several of the other techniques consists of applying the Viterbi algorithm, possibly combined with a heuristic pruning, to a dynamically expanded graph representing the set of possible candidate pre-images. The complexity of the algorithm is then $O(|y||G|)$ where y is the string for which a pre-image is sought and G the graph expanded. The practicality of such pre-image computations often relies on some rather restrictive constraints on the type of features used, which may impact the quality of the prediction. As an example, in the experiments described by Taskar et al. (2003), a Markovian assumption is made about the output sequences, and furthermore the dependency between the input and output symbols is strongly restricted: y_i, the output symbol at position i, depends only on x_i, the symbol at the same position in the input.

The two approaches differ significantly in terms of the number of variables to estimate in the dual case. With the kernel perceptron algorithm (Collins and Duffy, 2001) and the large-margin maximization algorithms of Tsochantaridis et al. (2004) and Taskar et al. (2003), the number of variables to estimate is at most $m|Y|$, where $|Y|$, the number of possible labels, could be potentially very large. On the positive side, this problem is partially alleviated thanks to the sparsity of the solution. Note that the speed of training is also proportional to the number of non-zero coefficients.

In the case of our general regression framework with no constraints, the number of dual variables is m^2, and is therefore independent of the number of output labels. On the negative side, the solution is in general not sparse. The greedy incremental technique described in Section 1.5 helps overcome this problem however.

1.7 Experiments

1.7.1 Description of the Dataset

To test the effectiveness of our algorithms, we used exactly the same dataset as the one used in the experiments reported by Taskar et al. (2003) with the same specific cross-validation process and the same folds: the data is partitioned into ten folds, and ten times one fold is used for training, and the remaining nine are used for testing.

The dataset, including the partitioning, is available for download from `http://ai.stanford.edu/~btaskar/ocr/`. It is a subset of the handwritten words collected by Rob Kassel at the MIT Spoken Language Systems Group for an optical character recognition (OCR) task. It contains 6,877 word instances with a total of 52,152 characters. The first character of each word has been removed to keep only lowercase characters (this decision was not made by us. We simply kept the dataset unchanged to make the experiments comparable). The image of each character has been rasterized and normalized into a $16 \times 8 = 128$ binary-pixel representation.

The general handwriting recognition problem associated with this dataset is to determine a word y given the sequence of pixel-based images of its handwritten segmented characters $x = x_1 \cdots x_k$. We report our experimental results in this task with two different settings.

1.7.2 Perfect Segmentation

Our first experimental setting matches exactly that of Taskar et al. (2003), where a perfect image segmentation is given with a one-to-one mapping of images to characters. Image segment x_i corresponds exactly to one word character, the character, y_i, of y in position i.

To exploit these conditions in learning, we will use the general regression framework with constraints described in Section 1.3.2. The input and output feature mappings Φ_X and Φ_Y are defined as follows.

Let v_i, $i = 1, \ldots, N_1$, denote all the image segments in our training set and let k_X be a positive definite symmetric kernel defined over such image segments. We denote by l the maximum length of a sequence of images in the training sample.

The feature vector $\Phi_X(x)$ associated to an input sequence of images $x = x_1 \cdots x_q$, $q \leq l$, is defined by:

$$\Phi_X(x) = [k'_X(v_1, x_{p(v_1)}), \ldots, k'_X(v_{N_1}, x_{p(v_{N_1})})]^\top. \tag{1.69}$$

where for $i = 1, \ldots, N_1$, $k'_X(v_i, x_{p(v_i)}) = k_X(v_i, x_{p(v_i)})$ if $p(v_i) \leq q$, $k'_X(v_i, x_{p(v_i)}) = 0$ otherwise. Thus, we use a so-called empirical kernel map to embed the input strings in the feature space F_X of dimension N_1.

The feature vector $\Phi_Y(y)$ associated to the output string $y = y_1 \cdots y_q$, $q \leq l$, is a $26l$-dimensional vector defined by

$$\Phi_Y(y) = [\phi_Y(y_1), \ldots, \phi_Y(y_q), 0, \ldots, 0]^\top, \tag{1.70}$$

where $\phi_Y(y_i)$, $1 \leq i \leq q$, is a 26-dimensional vector whose components are all zero except from the entry of index y_i, which is equal to one. With this feature mapping, the pre-image problem is straightforward since each position can be treated separately. For each position i, $1 \leq i \leq l$, the alphabet symbol with the largest weight is selected.

Given the embeddings Φ_X and Φ_Y, the matrix $\mathbf{W}$ learned by our algorithm is in $\mathbb{R}^{N_2 \times N_1}$, with $N_1 \approx 5,000$ and $N_2 = 26l$. Since the problem setting is very restricted, we can impose some constraints on $\mathbf{W}$. For a given position i, $1 \leq i \leq l$, we can assume that input features corresponding to other positions, that is input features v_j for which $p(v_j) \neq i$, are (somewhat) irrelevant for predicting the character in position i. This translates into imposing that the coefficients of $\mathbf{W}$ corresponding to such pairs be small. For each position i, there are $26(N_1 - |\{v_j : p(v_j) = i\}|)$ such constraints, resulting in a total of

$$C = \sum_{i=1}^{l} 26(N_1 - |\{v_j : p(v_j) = i\}|) = 26N_1(l-1) \tag{1.71}$$

constraints. These constraints can be easily encoded following the scheme outlined in Section 1.3.2. To impose a constraint on the coefficient $\mathbf{W}_{rs}$ of $\mathbf{W}$, it suffices to introduce a matrix $\mathbf{A}$ whose entries are all zero except from the coefficient of row index r and column index s, which is set to one. Thus, $< \mathbf{W}, \mathbf{A} >= \mathbf{W}_{rs}$. To impose all C constraints, C matrices $\mathbf{A}_i$, $i = 1, \ldots, C$, of this type can be defined. In our experiments, we used the same regularization parameter η for all of these constraints.

Technique	Accuracy	
REG-constraints $\eta = 0$	84.1%	±.8%
REG-constraints $\eta = 1$	88.5%	±.9%
REG	79.5%	±.4%
REG-Viterbi ($n = 2$)	86.1%	±.7%
REG-Viterbi ($n = 3$)	98.2%	±.3%
SVMS (cubic kernel)	80.9%	±.5%
M^{3}Ns (cubic kernel)	87.0%	±.4%

Table 1.1 Experimental results with the perfect segmentation setting. The M^{3}N and SVM results are read from the graph in Taskar et al. (2003)

In our experiments, we used the efficient iterative method outlined in Section 1.3.2 to compute $\mathbf{W}$. However, it is not hard to see that in this case, thanks to the simplicity of the constraint matrices $\mathbf{A}_i$, $i = 1, \ldots, C$, the resulting matrix $(< \mathbf{A}_i, \mathbf{A}_j \mathbf{U}^{-1} >_F)_{ij} + \mathbf{I}$ can be given a simple block structure that makes it easier to invert. Indeed, it can be decomposed into l blocks in $\mathbb{R}^{N_1 \times N_1}$ that can be inverted independently. Thus, the overall complexity of matrix inversion, which dominates the cost of the algorithm, is only $O(lN_1^3)$ here.

Table 1.1 reports the results of our experiments using a polynomial kernel of third degree for k_X, and the best empirical value for the ridge regression coefficient γ which was $\gamma = 0.01$. The accuracy is measured as the percentage of the total number of word characters correctly predicted. REG refers to our general regression technique, REG-constraints to our general regression with constraints. The results obtained with the regularization parameter η set to 1 are compared to those with no constraint, i.e., $\eta = 0$. When $\eta = 1$, the constraints are active and we observe a significant improvement of the generalization performance.

For comparison, we also trained a single predictor over all images of our training sample, regardless of their positions. This resulted in a regression problem with a 26-dimensional output space, and $m \approx 5,000$ examples in each training set. This effectively corresponds to a hard weight-sharing of the coefficients corresponding to different positions within matrix $\mathbf{W}$, as described in Section 1.3.2. The first 26 lines of $\mathbf{W}$ are repeated $(l - 1)$ times. That predictor can be applied independently to each image segment x_i of a sequence $x = x_1 \cdots x_q$. Here too, we used a polynomial kernel of third degree and $\gamma = 0.01$ for the ridge regression parameter. We also experimented with the use of an n-gram statistical model based on the words of the training data to help discriminate between different word sequence hypotheses, as mentioned in Section 1.4.8.

Table 1.1 reports the results of our experiments within this setting. REG refers to the hard weight-sharing regression without using a statistical language model and is directly comparable to the results obtained using Support Vector Machines (SVMs). REG-Viterbi with $n = 2$ or $n = 3$ corresponds to the results obtained within this setting using different n-gram order statistical language models. In this case, we used the Viterbi algorithm to compute the pre-image solution, as in M^3Ns. The results shows that coupling a simple predictor with a more sophisticated pre-image algorithm can significantly improve the performance. The high accuracy achieved in this setting can be viewed as reflecting the simplicity of this task. The dataset contains only 55 unique words, and the same words appear in both the training and the test set.

We also compared all these results with the best result reported by Taskar et al. (2003) for the same problem and dataset. The experiment allowed us to compare these results with those obtained using M^3Ns. But, we are interested in more complex string-to-string prediction problems where restrictive prior knowledge such as a one-to-one mapping is not available. Our second set of experiments corresponds to a more realistic and challenging setting.

1.7.3 String-to-String Prediction

Our method generalizes indeed to the much harder and more realistic problem where the input and output strings may be of different length and where no prior segmentation or one-to-one mapping is given. For this setting, we directly estimate the counts of all the n-grams of the output sequence from one set of input features and use our pre-image algorithm to predict the output sequence.

In our experiment, we chose the following polynomial kernel K_X between two image sequences:

$$K_X(x_1, x_2) = \sum_{x_{1,i}, x_{2,j}} (1 + x_{1,i}\, x_{2,j})^d, \tag{1.72}$$

where the sum runs over all n-grams $x_{1,i}$ and $x_{2,j}$ of input sequences x_1 and x_2. The n-gram order and the degree d are both parameters of the kernel. For the kernel K_Y we used n-gram kernels.

As a result of the regression, the output values, that is the estimated counts of the individual n-grams in an output string are non-integers and need to be discretized for the Euler circuit computation, see Section 1.4.8. The output words are in general short, so we did not anticipate counts higher than one. Thus, for each output feature, we determined just one threshold above which we set the count to one. Otherwise, we set the count to zero. These thresholds were determined by examining each feature at a time and imposing that averaged over all the strings in the training set, the correct count of the n-gram be predicted.

Note that, as a consequence of this thresholding, the predicted strings do not always have the same length as the target strings. Extra and missing characters are counted as errors in our evaluation of the accuracy.

We obtained the best results using unigrams and second-degree polynomials in the input space and bigrams in the output space. For this setting, we obtained an accuracy of 65.3 ± 2.3.

A significant higher accuracy can be obtained by combining the predicted integer counts from several different input and output kernel regressors, and computing an

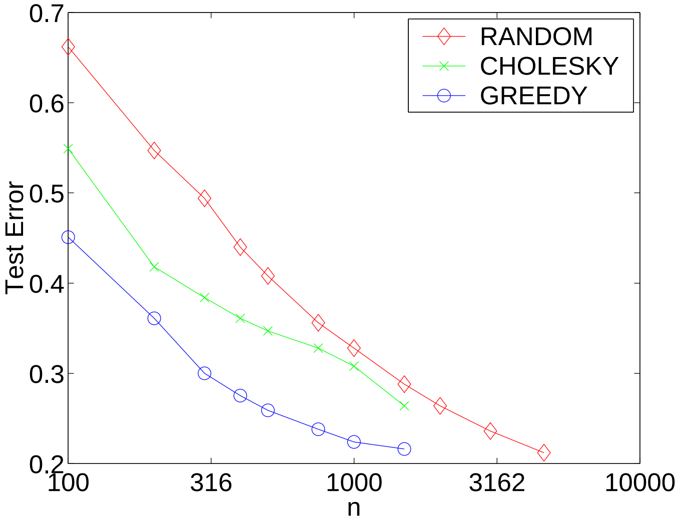

Figure 1.7 Comparison of random sub-sampling of n points from the OCR dataset, incomplete Cholesky decomposition after n iterations and greedy incremental learning with n basis functions. The main bottleneck for all of these algorithms is the matrix inversion where the size of the matrix is $n \times n$, we therefore plot test error against n. The furthest right point is the test error rate of training on the full training set of $n = 4,617$ examples.

Euler circuit using only the n-grams predicted by the majority of the regressors. Combining 5 such regressors, we obtained a test accuracy of 75.6 ± 1.5.

A performance degradation for this setting was naturally expected, but we view it as relatively minor given the increased difficulty of the task. Furthermore, our results can be improved by combining a statistical model with our pre-image algorithm.

1.7.4 Faster Training

As pointed out in Section 1.5, faster training is needed when the size of the training data increases significantly. This section compares the greedy incremental technique described in that section with the partial Cholesky decomposition technique and the baseline of randomly sub-sampling n points from the data, which of course also results in reduced complexity, giving only a $n \times n$ matrix to invert. The different techniques are compared in the perfect segmentation setting on the first fold of the data. The results should be indicative of the performance gain in other folds.

In both partial Cholesky decomposition and greedy incremental learning, n iterations are run and then an $n \times n$ matrix is inverted, which may be viewed as the bottleneck. Thus, to determine the learning speed we plot the test error for the regressions problem versus n. The results are shown in Figure 1.7. The greedy learning technique leads to a considerable reduction in the number of kernel computations required and the matrix inversion size for the same error rate as the full dataset. Furthermore, in greedy incremental learning we are left with only n kernels to compute for a given test point, independently of the number of outputs. These reasons combined make the greedy incremental method an attractive approximation technique for our regression framework.

1.8 Conclusion

We presented a general regression framework for learning string-to-string mappings and illustrated its use in several experiments. Several paths remained to be explored to further extend the applicability of this framework.

The pre-image algorithm for strings that we described is general and can be used in other contexts. But, the problem of pre-image algorithms for strings may have other efficient solutions that need to be examined.

Efficiency of training is another key aspect of all string-to-string mapping algorithms. We presented several heuristics and approximations that can be used to substantially speed up training in practice. Other techniques could be studied to further increase speed and extend the application of our framework to very large tasks without sacrificing accuracy.

Much of the framework and algorithms presented can be used in a similar way for other prediction problems with structured outputs. A new pre-image algorithm needs to be introduced however for other learning problems with structured outputs.

References

Francis R. Bach and Michael I. Jordan. Kernel independent component analysis. *Journal of Machine Learning Research (JMLR)*, 3:1–48, 2002.

Peter L. Bartlett, Michael Collins, Ben Taskar, and David McAllester. Exponentiated Gradient Algorithms for Large-margin Structured Classification. *Neural Information Processing Systems 17*, 2004.

Michael Collins and Nigel Duffy. Convolution kernels for natural language. *Neural Processing Information Systems 14*, 2001.

Corinna Cortes, Patrick Haffner, and Mehryar Mohri. Rational Kernels: Theory and Algorithms. *Journal of Machine Learning Research (JMLR)*, 5:1035–1062, 2004.

Leo Kontorovich. Uniquely Decodable n-gram Embeddings. *Theoretical Computer Science*, 329/1-3:271–284, 2004.

Craig Saunders, Alexander Gammerman, and Volodya Vovk. Ridge Regression Learning Algorithm in Dual Variables. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 515–521. Morgan Kaufmann Publishers Inc., 1998. ISBN 1-55860-556-8.

Bernhard Schölkopf and Alex Smola. *Learning with Kernels*. MIT Press: Cambridge, MA, 2002.

Alex J. Smola and Peter L. Bartlett. Sparse greedy Gaussian process regression. *Neural Processing Information Systems 13*, pages 619–625, 2001.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-Margin Markov Networks. *Neural Information Processing Systems 16*, 2003.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support Vector Machine Learning for Interdependent and Structured Output Spaces. *ICML*, 2004.

Jacobus H. van Lint and Richard M. Wilson. *A Course in Combinatorics*. Cambridge University Press, 1992.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.

Pascal Vincent and Yoshua Bengio. Kernel Matching Pursuit. Technical Report 1179, Département d'Informatique et Recherche Opérationnelle, Université de Montréal, 2000.

Jason Weston, Olivier Chapelle, André Elisseeff, Bernhard Schölkopf, and Vladimir Vapnik. Kernel Dependency Estimation. *Neural Processing Information Systems 15*, 2002.

Jason Weston, Bernhard Schölkopf, Olivier Bousquet, Tobias Mann, and William Stafford Noble. Joint kernel maps. Technical report, Max Planck Institute for Biological Cybernetics, 2004.

Wilson, R. J. *Introduction to Graph Theory*. Longman, 1979.