A General Regression Technique for Learning Transductions

Corinna Cortes CORINNA@GOOGLE.COM
Google Research
1440 Broadway, New York, NY 10018, USA

Mehryar Mohri MOHRI@CS.NYU.EDU
Courant Institute of Mathematical Sciences
719 Broadway, 12th Floor, New York, NY 10003, USA

Jason Weston JASONW@NEC-LABS.COM
NEC Research
4 Independence Way, Princeton, NJ 08540, USA

Abstract

The problem of learning a transduction, that is a string-to-string mapping, is a common problem arising in natural language processing and computational biology. Previous methods proposed for learning such mappings are based on *classification* techniques. This paper presents a new and general *regression* technique for learning transductions and reports the results of experiments showing its effectiveness. Our transduction learning consists of two phases: the estimation of a set of regression coefficients and the computation of the pre-image corresponding to this set of coefficients. A novel and conceptually cleaner formulation of kernel dependency estimation provides a simple framework for estimating the regression coefficients, and an efficient algorithm for computing the pre-image from the regression coefficients extends the applicability of kernel dependency estimation to output sequences. We report the results of a series of experiments illustrating the application of our regression technique for learning transductions.

1. Introduction

Machine learning has progressively extended its application domain by generalizing existing algorithms to cover more complex spaces. Originally, it was centered around classifying fixed-size vectors into two or more classes, or predicting one or several real-valued outputs. Kernel methods revolutionized machine learning by providing elegant extensions to structured input spaces: so long as a suitable kernel between two input elements can be defined, algorithms such as Support Vector Machines (SVMs) or ridge regression can be applied in a straight-forward manner. Still, the challenge remains as to how naturally extend learning algorithms to handle structured output spaces and accurately predict from structure to structure. This paper presents a new and general method for addressing a common instance of that problem, that of learning string-to-string mappings.

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, e.g., *data* $\rightarrow$ {*d ey t ax, d ey dx ax, d ae t ax, d ae dx ax*}. An accurate pronunciation model is crucial for the overall quality of such systems. Another major task in natural language processing is part-of-speech tagging, which consists of assigning a part-of-speech tag to each word of a sentence as in: *Mike spoke to the salesman* $\rightarrow$ *Mike N spoke V to P the D salesman N*, where *N* stands for a noun, *V* for a verb, *P* for a preposition, and *D* for a determiner. Similarly, parsing can be viewed as a string-to-string mapping where the target alphabet contains parentheses as in: *Mike spoke to the salesman* $\rightarrow$ *(((Mike N) NP) ((spoke V) (to P) ((the D) (salesman N) NP) VP) S)*.

A general framework for describing such problems is that of *learning transductions*. A transduction T from X^* to Y^* is a mapping from X^* to the powerset of Y^* (Berstel, 1979). Most transductions in natural language processing are either *algebraic* or *rational*. Algebraic transductions are transductions related to context-free languages (Salomaa & Soittola, 1978; Kuich & Salomaa, 1986). Rational transductions are those related to regular languages and can be represented by finite-state transducers. Similar transduction learning problems arise in computational biology (Durbin et al., 1998).

Several large-margin classification techniques have been recently proposed for learning such mappings, in particu-

Appearing in *Proceedings of the 22^{nd} International Conference on Machine Learning*, Bonn, Germany, 2005.

lar: Maximum Margin Markov Networks (M^{3}Ns) (Taskar et al., 2003; Bartlett et al., 2004) and SVMISOS (Tsochantaridis et al., 2004). These techniques all treat the learning problem just outlined as a *classification* problem over the pairs of input and output sequences (X, Y), imposing that the string Y matching X obtain a higher score than all other strings in Y^*. In contrast, this paper introduces a new and general technique, Regression for Learning Transductions, (RLT), that addresses the problem of learning transductions as a *regression* problem.

Seeking a regression solution is natural since one can define similarity measures between two possible sequence targets Y_1 and Y_2 associated with an input sequence X. Such similarities tend to be ignored by a classification loss that assigns to each pair (X, Y) the value one or zero regardless of the closeness between Ys.[1]

RLT consists of two phases: a regression of the input on an implicit or explicit feature space and the computation of the output or pre-image from that space. A reformulation of Kernel Dependency Estimation (KDE) (Weston et al., 2002) provides a clean framework for estimating the regression coefficients, and the pre-image is computed from these coefficients using a novel and efficient algorithm based on classical results from graph theory. A major computational advantage of KDE over the classification techniques mentioned is thus that it does not require an exhaustive pre-image search of Y^* during training. Our new pre-image algorithm extends the applicability of KDE to output sequences.

This paper describes in detail the transduction regression technique RLT and reports the results of experiments showing its effectiveness. The paper is organized as follows. Section 2 describes a conceptually cleaner reformulation of the KDE method and Section 3 discusses several techniques for faster training within this framework. Section 4 presents our inverse pre-image function for strings. A comparison of RLT and other classification-based techniques is given in Section 5, which is followed by a series of experimental results reported in Section 6.

2. Reformulation of Kernel Dependency Estimation

Kernel Dependency Estimation (KDE) treats the problem of learning the mapping from X^* to Y^* as a regression problem (Figure 1). Input strings in X^* are mapped via Φ_X to a high-dimensional Hilbert space F_X and similarly output strings mapped to F_Y via the mapping Φ_Y. These mappings can be defined implicitly by the introduction of positive definite symmetric kernels K_X and K_Y associated with the mappings Φ_X and Φ_Y. A regression algorithm exploiting the kernel K_X, e.g., standard kernel ridge regression (Saunders et al., 1998), can be used to learn the mapping g from X^* to F_Y. For prediction, the pre-image of $g(x)$ by Φ_Y needs to be computed to determine $f(x) = \Phi_Y^{-1}(g(x))$.

[1] Classification techniques such as SVMISOS (Tsochantaridis et al., 2004) introduce a loss function to help correct the binary classification objective function, however.

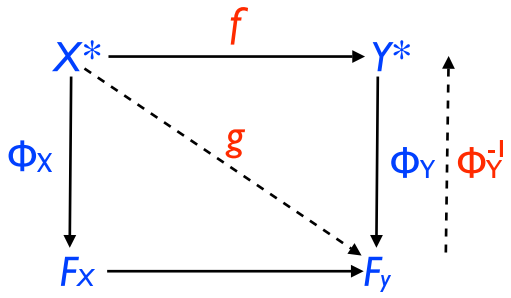

Figure 1. Diagram describing Kernel Dependency Estimation (KDE). f represents the transduction from X^* to Y^* learned by KDE.

We denote by N_1 the dimension of the feature space F_X and by N_2 that of F_Y. When there is no risk of confusion, we use the same notation for a vector, e.g., $\Phi_X(x)$ or $g(x)$, and the column matrix representing that vector.

In the original presentation of KDE, the first step consisted of using K_Y with Kernel Principal Components Analysis (KPCA) to reduce the dimension of the feature space F_Y (Weston et al., 2002). Here, we simplify the framework by not requiring this prior dimensionality reduction anymore. We now describe two general models for KDE.

Model 1. Here g is modeled as a linear function defined by

$$\forall x \in X^*, g(x) = \mathbf{W}\Phi_X(x), \tag{1}$$

where $\mathbf{W}$ is an $N_2 \times N_1$ real-valued matrix. Different regression algorithms can be used to learn g, or equivalently the matrix $\mathbf{W}$, including kernel ridge regression (Saunders et al., 1998), Support Vector Regression (SVR) (Vapnik, 1995), or Kernel Matching Pursuit (KMP) (Vincent & 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. Matrix $\mathbf{W}$ is then the solution of:

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

where $\|\cdot\|_F$ denotes the Frobenius norm and where $\gamma > 0$ is a regularization coefficient. m is the number of training examples.

Proposition 2.1. *The solution* $\mathbf{W}$ *is given by:*

$$\mathbf{W} = \mathbf{M}_Y(\mathbf{K_X} + \gamma\mathbf{I})^{-1}\mathbf{M}_X^\top. \tag{3}$$

where $\mathbf{M}_Y = [\Phi_Y(y_1), \ldots, \Phi_Y(y_m)]$ *is the* $N_2 \times m$ *matrix whose* j*th column vector is* $\Phi_Y(y_j)$*, similarly* $\mathbf{M}_Y = [\Phi_X(x_1), \ldots, \Phi_X(x_m)]$*, and where* $\mathbf{K}_X$ *is the Gram matrix associated to the kernel* K_X*:* $\mathbf{K}_X = (K_X(x_i, x_j)_{1\le i,j\le m})$.

Proof. Equation 2 can be rewritten as:

$$\underset{\mathbf{W}\in\mathbb{R}^{N_2\times N_1}}{\operatorname{argmin}} \|\mathbf{W}\mathbf{M}_X - \mathbf{M}_Y\|_F^2 + \gamma\|\mathbf{W}\|_F^2. \quad (4)$$

Differentiating the expression and setting it to zero yields:

$$\begin{array}{rl} & 2(\mathbf{W}\mathbf{M}_X - \mathbf{M}_Y)\mathbf{M}_X^\top + 2\gamma\mathbf{W} = 0 \\ \Leftrightarrow & \mathbf{W}(\mathbf{M}_X\mathbf{M}_X^\top + \gamma I) = \mathbf{M}_Y\mathbf{M}_X^\top. \end{array} \quad (5)$$

$$\begin{array}{rrcl} \text{Thus,} & \mathbf{W} & = & \mathbf{M}_Y\mathbf{M}_X^\top(\mathbf{M}_X\mathbf{M}_X^\top + \gamma I)^{-1} \\ & & = & \mathbf{M}_Y(\mathbf{M}_X^\top\mathbf{M}_X + \gamma I)^{-1}\mathbf{M}_X^\top \\ & & = & \mathbf{M}_Y(\mathbf{K}_\mathbf{X} + \gamma\mathbf{I})^{-1}\mathbf{M}_X^\top, \end{array} \quad (6)$$

where we have used the fact that $(\mathbf{M}_X^\top\mathbf{M}_X + \gamma\mathbf{I})^{-1}$ and $\mathbf{M}_X^\top$ commute as can be seen for example from a series expansion of $(\mathbf{M}_X^\top\mathbf{M}_X + \gamma\mathbf{I})^{-1}$. □

Predictions are made by solving the pre-image problem:

$$\begin{aligned} f(x) &= \operatorname*{argmin}_{y\in Y^*} \|\mathbf{W}\Phi_X(x) - \Phi_Y(y)\|^2 \\ &= \operatorname*{argmin}_{y\in Y^*} \Phi_Y(y)^\top\Phi_Y(y) - 2\Phi_Y(y)^\top\mathbf{W}\Phi_X(x) \\ &= \operatorname*{argmin}_{y\in Y^*} K_Y(y,y) - 2(\mathbf{K}_Y^y)^\top(\mathbf{K}_\mathbf{X} + \gamma\mathbf{I})^{-1}\mathbf{K}_X^x, \end{aligned}$$

where $\mathbf{K}_Y^y$ is the $m \times 1$ column matrix whose ith row is $K_Y(y, y_i)$, and $\mathbf{K}_X^x$ the column matrix whose ith row is $K_X(x_i, x)$. Thus, remarkably, $f(x)$ can be expressed using kernel functions alone.

Model 2. In some cases, it may be preferable to learn each component g_i of g, $i = 1, \ldots, N_2$, independently, in fact by using different input feature vectors $\Phi_{X,i}$ for each i. Each g_i is then modeled as a linear function defined by

$$\forall x \in X^*, 1 \le i \le N_2, g_i(x) = \mathbf{W}_i\Phi_{X,i}(x). \quad (7)$$

In fact, more generally, one may wish to define different subsets of the training data for learning each g_i. A full description of these generalizations will be given in a longer version of the paper. As described for the first model, the component functions g_i can be learned using different regression algorithms such as SVR or kernel ridge regression. In all cases, one still needs to solve the pre-image problem.

3. Speeding-up training

There are several ways of speeding up training when using kernel ridge regression. One solution is to apply incomplete Cholesky decomposition to the kernel matrix K_X (Bach & Jordan, 2002). To shorten this presentation, we do not detail the application of this technique here. Our experiments have shown a greedy technique to be far more effective (see Section 6.4). This consists, as with Kernel Matching Pursuit (KMP) (Vincent & 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:

$$g(x) = (\sum_{i\in S}\alpha_{i1}K_X(x_i, x), \ldots, \sum_{i\in S}\alpha_{iN_2}K_X(x_i, x)), \quad (8)$$

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:

$$j = \operatorname*{argmax}_{i\in\{1,\ldots,m\}\setminus S} ||\Phi_Y(y_i) - g(x_i)||^2; \quad (9)$$

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

$$\begin{array}{l} S \leftarrow S \cup \{x_j\}; \\ \alpha \leftarrow \operatorname*{argmin}_{\hat{\alpha}} \sum_{i=1}^m ||\Phi_Y(y_i) - g(x_i)||^2. \end{array} \quad (10)$$

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 & Bartlett, 2001), which results in a running time complexity of $O(nm^2N_2)$ as with KMP (Vincent & 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.

4. Pre-image solution for strings

4.1. A general problem: finding pre-images

A critical component of RLT 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. 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 5).

Several instances of the pre-image problem have been studied in the past in cases where the pre-images are fixed-size vectors (Schölkopf & 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.

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 x and y in Y^*, $n \geq 1$, is defined by:

$$k_n(x, y) = \sum_{|u|=n} |x|_u \, |y|_u, \tag{11}$$

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).

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 $y = (y_1, \ldots, y_l)$, where $l = |\Sigma|^n$ and y_k is the count for an n-gram sequence u_k, find string x such that for $k = 1, \ldots, l$, $|x|_{u_k} = y_k$. Several standard problems arise in this context: the existence of x given y, its uniqueness when it exists, and the need for an efficient algorithm to determine x when it exists. We will address all these questions in the following sections.

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 associated with n and the vector y. Indeed, let $G_{y,n}$ be 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 $H_{y,n}$ be the resulting unweighted graph.

The problem of finding the string x is then equivalent to that of finding an Euler circuit of $H_{y,n}$, that is a circuit on the graph in which each edge is traversed exactly one. 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 2 illustrates the construction of the graphs $G_{y,n}$ and $H_{y,n}$ in a special case.

4.5. Existence

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

Theorem 4.6. *The vector y admits a pre-image iff for any vertex q of $H_{y,n}$, $in\text{-}degree(q) = out\text{-}degree(q)$.*

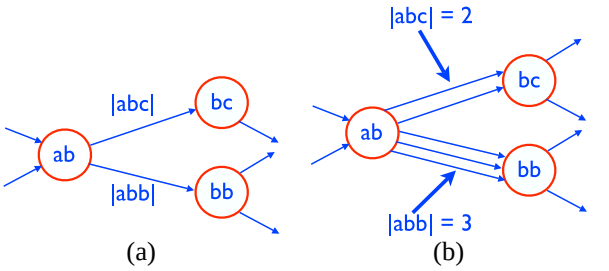

Figure 2. (a) The De Bruijn graph $G_{y,3}$ associated with the vector y in the case of trigrams ($n = 3$). The weight carried by the edge from vertex ab to vertex bc is the number of occurrences of the trigram abc as specified by the vector y. (b) The expanded graph $H_{y,3}$ associated with $G_{y,3}$. An edge in $G_{y,3}$ is repeated as many times as there were occurrences of the corresponding trigram.

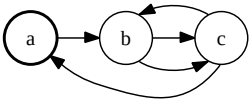

Figure 3. Example of a pre-image computation. The graph is associated with the vector $y = (0, 1, 0, 0, 0, 2, 1, 1, 0)$ whose coordinates indicate the counts of the bigrams $aa, ab, ac, ba, bb, bc, ca, cb, cc$. The graph verifies the conditions of theorem 4.6, thus it admits an Eulerian circuit, which in this case corresponds to the pre-image $x = bcbca$ 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). □

4.7. Compact algorithm

There exists a linear-time algorithm for determining an Eulerian circuit of a graph verifying the conditions of theorem 4.6 (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:

$$O(|H_{y,n}|) = O(\sum_{i=1}^{l} y_i) = O(|x|). \tag{12}$$

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

EULER(q)
1 path $\leftarrow \epsilon$
2 **for** each unmarked edge e leaving q **do**
3 MARK(e)
4 path $\leftarrow e$ 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

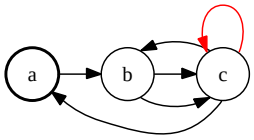

Figure 4. Case of non-unique pre-images. Both $bcbcca$ and $bccbca$ are possible pre-images. Our EULER algorithm can produce both solutions, depending on the order in which outgoing edges of the vertex c are examined. The graph differs from that of Figure 3 only by self-loop at the vertex identified with c.

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 non-trivial and we leave it to a longer version of the paper. However, its compact form makes it easy to modify and analyze the effect of the modifications.

4.8. Uniqueness

In general, when it exists, the pre-image sequence is not unique. Figure 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, $\Phi_n(x)$ is the vector whose components are the counts of the n-grams appearing in x.

Theorem 4.9 ((Kontorovich, 2004)). *The set of strings x such that $\Phi(x)$ 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 y, 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.

4.10. Generalized algorithm

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

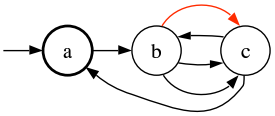

Figure 5. Illustration of the application of the generalized algorithm to the case of a graph that does not admit the Euler property. The graph differs from that of Figure 3 by just one edge (edge in red). The possible pre-images returned by the algorithm are $bccbca$ and $bcbcca$.

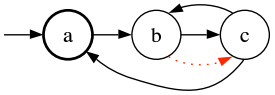

Figure 6. Further illustration of the application of the generalized algorithm to the case of a graph that does not admit the Euler property. The graph differs from that of Figure 3 by just one edge (the missing edge in red). The pre-image returned by the algorithm is $bcba$.

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 x whose length is $|x| = \sum_{i=1}^{l} y_i$ since each edge of the graph $H_{y,n}$ is visited exactly once. Clearly, the result is a pre-image when y admits one. But, how different is the output string x from the original pre-image when we modify the count of one of the components of y by one, either by increasing or decreasing it? Figures 3 and 4 can serve to illustrate that in a special case since the graph of Figure 4 differs from that of Figure 3 by just one edge, which corresponds to the existence or not of the bigram cc.[2] 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 5 illustrates another case where the graph differs from that of Figure 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 6 illustrates yet another case where the graph differs from that of Figure 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 $bcba$.

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

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 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.

5. Relation to other algorithms

Classification-based learning A kernel perceptron-based learning approach for structured outputs was described by (Collins & Duffy, 2001). Since then, several similar classification-based techniques (Tsochantaridis et al., 2004; Taskar et al., 2003) have been described for the same problem. In all these techniques, given an input x, the output is calculated via:

$$\hat{y}(x) = \underset{y \in Y}{\operatorname{argmax}}\, w \cdot \Phi_{XY}(x, y), \tag{13}$$

with $w = \sum_{i=1}^{m} \sum_{y \in Y} \alpha_{iy} \Phi_{XY}(x_i, y)$. Here, a joint embedding space for both *inputs and outputs* is employed, rather than two separate spaces. Nevertheless, in essence, this is similar to the pre-image problem (Equation 7) that arises in KDE.[3]

The structured output perceptron is trained by iteratively calculating $\hat{y}(x_i)$ for the each element of the training data. When the prediction is incorrect ($\hat{y}(x_i) \neq y_i$) the model is updated as follows: $\alpha_{i\hat{y}} \leftarrow \alpha_{i\hat{y}} - 1$, $\alpha_{iy} \leftarrow \alpha_{iy} + 1$. Thus, in contrast to KDE, a pre-image search is required at every iteration. KDE requires pre-image calculations only in the test phase. Also, in the case of the perceptron, α is a matrix of size $m \times |Y|$, while in KDE, even without dimensionality reduction, it is of size $m \times m$. In practice however, the perceptron is sparse, whereas kernel ridge regression may not be. The greedy incremental approach described in Section 3 addresses this problem. On the other hand, the use of a joint embedding space allows one to encode prior knowledge about the relationship between input and output sequences, which is not possible in our method unless we directly encode the input in the output space. See (Weston et al., 2004) for an empirical analysis of the benefits of joint feature spaces.

The main difference between the perceptron-based learning and the more recent classification-based methods (Tsochantaridis et al., 2004; Taskar et al., 2003) is that, inspired by support vector machines (Vapnik, 1995) they perform classification with a large margin, and introduce soft margins to deal with the linearly inseparable case. Classification loss seems to be a poor choice for structured learning as there is rarely one correct output, with all others being equally incorrect. This is why a modification of the classification loss function was proposed for these algorithms:

$$f(x_i, y_i) > f(x_i, y) + L(y_i, y), \;\; \forall y \in Y - \{y_i\}, \tag{14}$$

where $f(x, y) = w \cdot \Phi_{XY}(x, y)$ and $L(\cdot, \cdot)$ is a suitable loss function. The is similar to a ranking constraint, seeking to rank the correct output higher than all other outputs for computation of the argmax. However, here the margin plays the role of controlling how relatively "wrong" an example is. This makes the loss function similar, though not equivalent, to the objective function of a regression problem. We argue that minimizing the distance in the chosen output feature space, i.e., learning regression, is a more natural way of learning the relevant structure.

Limitations and pre-image computation Our algorithm works with any problem where the outputs are strings. the pre-image computation for strings that we presented in previous sections can be used with all n-gram kernels. We are exploring several extensions of the algorithm presented to deal with other families of kernels, but, in general, other string kernels lead to intractable pre-image problems. The complexity of our pre-image algorithm is linear, $O(|y|)$, in the length of the pre-image string y it generates. It imposes no restriction on the type of regression technique used in the first phase of RLT, nor does it constrain the choice of the features over the input X. To our knowledge, it is the first pre-image solution for a general family of string kernels that does not require an exhaustive search.

The computation of the pre-image in several of the classification-based 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 applicability of such pre-image computations often hinges on some specific constrains on the type of features used. As an example, in the experiments described by (Taskar et al., 2003), a Markovian assumption is made on Y, and furthermore X and Y are assumed to be dependent only at the same position in each sequence.

6. Experiments

6.1. Description of the dataset

To test the effectiveness of RLT, 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

[3] In fact, the equations are identical if the joint feature space is defined via $K_{XY}((x, y), (x', y')) = K_X(x, x') K_Y(y, y')$ when the output feature space data has norm one (Weston et al., 2004).

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 [4]. 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.

6.2. Perfect segmentation

Our first experimental setting is exactly the one used by Taskar et al. (2003), where the image segmentation is known to be perfect and where there is a one-to-one mapping of images to characters. Image segment x_i corresponds exactly to one word character, the character of y in position i, and there is no correlation between input segment images.

To use the prior knowledge about the one-to-one mapping in this task, we applied our most general modeling by predicting the character associated to each image individually and using that to predict the final output word sequence. This resulted in a regression problem with a 26-dimensional output space, and $m \approx 5,000$ examples in each fold. For the input images, we used a polynomial kernel of third degree. The best empirical value for the ridge regression coefficient γ was $\gamma = 0.01$. Since here the position of each character predicted was already known, the use of the pre-image algorithm based on the Euler circuits was not needed. We only needed, as mentioned in Section 4.10, to apply an n-gram statistical model based on the words of the training data to help discriminate between different word sequence hypotheses. In this case we used the Viterbi algorithm to compute the pre-image solution, as in M^3Ns. Where the two algorithms differ, however, is the way the model itself is learned. Table 1 reports the results of our experiments with $n = 2$ and $n = 3$ and compares them with the best result reported by Taskar et al. (2003) for the same problem and dataset. The accuracy is measured as the percentage of the total number of word characters correctly predicted.

The high accuracy achieved with this setting can be viewed as reflecting the simplicity of this task. The experiment allowed us to compare these results with those obtained using M^{3}Ns. But we are interested in more complex string-to-string prediction problems with no prior knowledge such as a one-to-one mapping. Our second set of experiments corresponds to a more realistic and challenging setting.

[4] This decision was not made by us. We simply kept the dataset unchanged to make the experiments comparable.

Technique	Accuracy	
RLT ($n = 2$)	86.1%	$\pm.7\%$
RLT ($n = 3$)	98.2%	$\pm.3\%$
M^{3}Ns (cubic kernel)	87.0%	$\pm.4\%$

Table 1. Experimental results with the perfect segmentation setting. The M^{3}N results are read of the graph in (Taskar et al., 2003)

6.3. 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:

$$K_X(x_1, x_2) = \sum_{x_{1,i}, x_{2,j}} (1 + x_{1,i}\, x_{2,j})^d, \qquad (15)$$

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.

We obtained the best results using unigrams and second-degree polynomials in the input space and bigrams in the output space. For each of the 26^n predicted features, the output of the ridge regression is discretized so that, averaged over all the training data, the correct number of n-grams is predicted. (i.e., this amounts to choosing for each output dimension the best threshold for rounding). For these experiments, we obtained a test accuracy of 75.6 ± 1.5 by combining predictions from several n-gram order predictions and computing an Euler tour from the pool of predicted n-grams. Note that the word sequences are predicted do not always have the same length as the target sequences. Extra or missing characters are counted as errors in our evaluation of the accuracy.

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.

6.4. Faster training

As pointed out in Section 3, 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 segmenta-

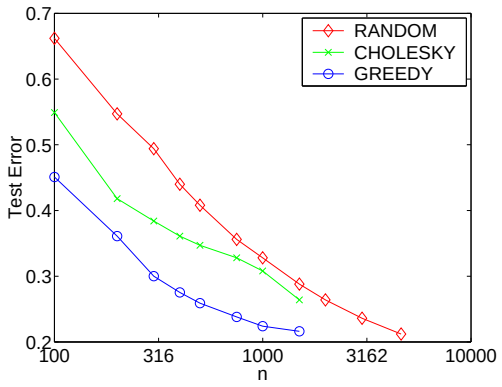

Figure 7. Comparison of random sub-sampling of n points from the OCR dataset, incomplete Cholesky decomposition after n iterations and greedy incremental learning with n basis functions. The main bottleneck for all of these algorithms is the matrix inversion where the size of the matrix is $n \times n$, 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.

tion 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 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 RLT.

7. Conclusion

We presented a novel and general regression technique for learning transductions. Several methods can be explored to further speed-up training and the applicability of RLT to very large tasks. We also described an efficient pre-image algorithm for strings that can be used for a general family of string kernels in other contexts.

References

Bach, F. R., & Jordan, M. I. (2002). Kernel independent component analysis. *Journal of Machine Learning Research (JMLR), 3*, 1–48.

Bartlett, P. L., Collins, M., Taskar, B., & McAllester, D. (2004). Exponentiated Gradient Algorithms for Large-margin Structured Classification. *Neural Information Processing Systems 17*.

Berstel, J. (1979). *Transductions and context-free languages*. Teubner Studienbucher: Stuttgart.

Collins, M., & Duffy, N. (2001). Convolution kernels for natural language. *Neural Processing Information Systems 14*.

Cortes, C., Haffner, P., & Mohri, M. (2004). Rational Kernels: Theory and Algorithms. *Journal of Machine Learning Research (JMLR), 5*, 1035–1062.

Durbin, R., Eddy, S., Krogh, A., & Mitchison, G. (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge UK.

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

Kuich, W., & Salomaa, A. (1986). *Semirings, automata, languages*. No. 5 in EATCS Monographs on Theoretical Computer Science. Berlin, Germany: Springer-Verlag.

Salomaa, A., & Soittola, M. (1978). *Automata-theoretic aspects of formal power series*. Springer-Verlag: New York.

Saunders, C., Gammerman, A., & Vovk, V. (1998). Ridge Regression Learning Algorithm in Dual Variables. *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 515–521). Morgan Kaufmann Publishers Inc.

Schölkopf, B., & Smola, A. (2002). *Learning with Kernels*. MIT Press: Cambridge, MA.

Smola, A. J., & Bartlett, P. L. (2001). Sparse greedy Gaussian process regression. *Neural Processing Information Systems 13*, 619–625.

Taskar, B., Guestrin, C., & Koller, D. (2003). Max-Margin Markov Networks. *Neural Information Processing Systems 16*.

Tsochantaridis, I., Hofmann, T., Joachims, T., & Altun, Y. (2004). Support Vector Machine Learning for Interdependent and Structured Output Spaces. *ICML*.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. New York: Springer Verlag.

Vincent, P., & Bengio, Y. (2000). *Kernel Matching Pursuit* (Technical Report 1179). Département d'Informatique et Recherche Opérationnelle, Université de Montréal. Presented at Snowbird'00.

Weston, J., Chapelle, O., Elisseeff, A., Schölkopf, B., & Vapnik, V. (2002). Kernel Dependency Estimation. *Neural Processing Information Systems 15*.

Weston, J., Schölkopf, B., Bousquet, O., Mann, T., & Noble, W. S. (2004). *Joint kernel maps* (Technical Report). Max Planck Institute for Biological Cybernetics.

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