SVM Optimization for Lattice Kernels

Cyril Allauzen
Google Research
76 Ninth Avenue
New York, NY 10011
allauzen@google.com

Corinna Cortes
Google Research
76 Ninth Avenue
New York, NY 10011
corinna@google.com

Mehryar Mohri
Courant Institute and Google
251 Mercer Street
New York, NY 10012
mohri@cs.nyu.edu

ABSTRACT

This paper presents general techniques for speeding up large-scale SVM training when using sequence kernels. Our techniques apply to the family of kernels commonly used in a variety of natural language processing applications, including speech recognition, speech synthesis, and machine translation. We report the results of large-scale experiments demonstrating dramatic reduction of the training time, typically by several orders of magnitude.

Categories and Subject Descriptors

G.1.6 [**Optimization**]: Metrics—*Constrained optimization, performance measures*; F.4.3 [**Formal Languages**]: Metrics—*Algebraic language theory, Classes defined by grammars or automata, Operations on languages*

General Terms

Algorithms,Theory

Keywords

SVMs, optimization, kernels, rational kernels, finite automata, weighted automata, weighted transducers

1. INTRODUCTION

Sequence kernels are similarity measures between sequences. When the kernels are positive semi-definite, PSD, they implicitly define an inner product in a Hilbert space where large-margin methods can be used for learning and estimation [22, 23]. These kernels can then be combined with algorithms such as support vector machines (SVMs) [3, 8, 25] or other kernel-based algorithms to form effective learning techniques.

Sequence kernels have been successfully used in a variety of applications in computational biology, natural language processing, and other sequence processing tasks, e.g., n-gram kernels, gappy n-gram kernels [19], mismatch kernels [17], locality-improved kernels [27], domain-based kernels [1], convolutions kernels for strings [13], and tree kernels [6].

However, scaling algorithms such as SVMs based on these kernels to large-scale problems remains a challenge. Both time and space complexity represent serious issues, which often make training impossible. One solution in such cases consists of using approximation techniques for the kernel matrix, e.g., [12, 2, 26, 16] or to use early stopping for optimization algorithms [24]. However, these approximations can of course result in some loss in accuracy, which, depending on the size of the training data and the difficulty of the task, can be significant.

This paper presents general techniques for speeding up large-scale SVM training when using sequence kernels, without resorting to such approximations. Our techniques apply to all *rational kernels*, that is sequence kernels that can be represented by weighted automata and transducers [7]. As pointed out by these authors, this family of kernels includes the sequence kernels commonly used in computational biology, natural language processing, or other sequence processing tasks, in particular all those already mentioned. Thus our techniques apply to all commonly used sequence kernels. We show, using the properties of rational kernels, that, remarkably, techniques similar to those used by [14] for the design of more efficient coordinate descent training algorithms for linear kernels can be used to design faster algorithms with significantly better computational complexity for SVMs combined with rational kernels.

These techniques were used by [14] to achieve a substantial speed-up of SVM training in the case of linear kernels, with very clear gains over the already optimized and widely used LIBSVM software library [5], and served as the basis for the design of the LIBLINEAR library [10]. We show experimentally that our techniques also lead to a substantial speed-up of training with sequence kernels. In most cases, we observe an improvement by several orders of magnitude.

The remainder of the paper is structured as follows. We start with a brief introduction of weighted transducers and rational kernels (Section 2), including definitions and prop-

MLG Washington D.C., USA

erties relevant to the following sections. Section 3 presents an overview of the coordinate descent solution by [14] for SVM optimization. Section 4 shows how a similar solution can be derived in the case of rational kernels. The analysis of the complexity and the implementation of this technique are described and discussed in Section 5. In section 6, we report the results of experiments with a large dataset and with several types of kernels demonstrating the substantial reduction of training time using our techniques.

2. PRELIMINARIES

This section briefly introduces the essential concepts and definitions related to weighted transducers and rational kernels. For the most part, we adopt the definitions and terminology of [7], but we also introduce a linear operator that will be needed for our analysis.

2.1 Weighted transducers and automata

Weighted transducers are finite-state transducers in which each transition carries some weight in addition to the input and output labels. The weight set has the structure of a semiring that is a ring that may lack negation. In this paper, we only consider weighted transducers over the *real semiring* $(\mathbb{R}_+, +, \times, 0, 1)$.

Figure 1(a) shows an example of a weighted finite-state transducer over the real semiring. In this figure, the input and output labels of a transition are separated by a colon delimiter and the weight is indicated after the slash separator. A weighted transducer has a set of initial states represented in the figure by a bold circle and a set of final states, represented by double circles. A path from an initial state to a final state is an accepting path. The input label of an accepting path is obtained by concatenating together the input symbols along the path from the initial to the final state. Similarly for the output label of an accepting path.

The weight of an accepting path is computed by multiplying the weights of its constituent transitions and multiplying this product by the weight of the initial state of the path (which equals one in our work) and by the weight of the final state of the path (displayed after the slash in the figure). The weight associated by a weighted transducer $\mathbf{U}$ to a pair of strings $(\mathbf{x}, \mathbf{y}) \in \Sigma^* \times \Sigma^*$ is denoted by $\mathbf{U}(\mathbf{x}, \mathbf{y})$ and is obtained by summing the weights of all accepting paths with input label $\mathbf{x}$ and output label $\mathbf{y}$.

A *weighted automaton* $\mathbf{A}$ can be defined as a weighted transducer with identical input and output labels, for any transition. Since only pairs of the form $(\mathbf{x}, \mathbf{x})$ can have a non-zero weight, we denote the weight associated by $\mathbf{A}$ to $(\mathbf{x}, \mathbf{x})$ by $\mathbf{A}(\mathbf{x})$ and refer it as the weight associated by $\mathbf{A}$ to $\mathbf{x}$. Similarly, in the graph representation of weighted automata, the output (or input) label is omitted. Figure 1(b) shows an example of a weighted automaton. Omitting the input labels of a weighted transducer $\mathbf{U}$ results in a weighted automaton $\mathbf{A}$ which is said to be the *output projection of* $\mathbf{U}$, $\mathbf{A} = \Pi_2(\mathbf{U})$. The automaton in Figure 1(b) is the output

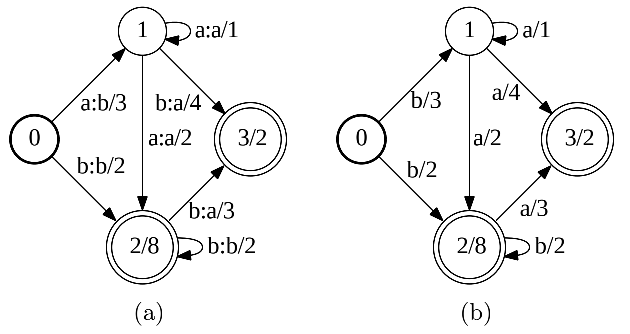

Figure 1: (a) Example of weighted transducer U. (b) Example of weighted automaton A. In this example, A can be obtained from U by projection on the output and $\mathbf{U}(aab, baa) = \mathbf{A}(baa) = 3\times1\times4\times2+3\times2\times3\times2$.

projection of the weighted transducer in Figure 1(a).

The standard operations of sum $+$, product or concatenation $\cdot$, and Kleene-closure * can be defined for weighted transducers [21]: for any pair of strings $(\mathbf{x}, \mathbf{y})$,

$$(\mathbf{U}_1 + \mathbf{U}_2)(\mathbf{x}, \mathbf{y}) = \mathbf{U}_1(\mathbf{x}, \mathbf{y}) + \mathbf{U}_2(\mathbf{x}, \mathbf{y})$$
$$(\mathbf{U}_1 \cdot \mathbf{U}_2)(\mathbf{x}, \mathbf{y}) = \sum_{\substack{\mathbf{x}_1\mathbf{x}_2=\mathbf{x}\\ \mathbf{y}_1\mathbf{y}_2=\mathbf{y}}} \mathbf{U}_1(\mathbf{x}_1, \mathbf{y}_1) \times \mathbf{U}_2(\mathbf{x}_2, \mathbf{y}_2)$$
$$(\mathbf{U}^*)(\mathbf{x}, \mathbf{y}) = \sum_{n\geq 0} (\mathbf{U}^n)(\mathbf{x}, \mathbf{y}).$$

For any transducer $\mathbf{U}$ and any real number γ, we denote by $\gamma\mathbf{U}$ a weighted transducer obtained from $\mathbf{U}$ by multiplying the final weights by γ. Thus, by definition, $(\gamma\mathbf{U})(\mathbf{x}, \mathbf{y}) = \gamma(\mathbf{U}(\mathbf{x}, \mathbf{y}))$ for any $\mathbf{x}, \mathbf{y} \in \Sigma^*$.

The *composition* of two weighted transducers $\mathbf{U}_1$ and $\mathbf{U}_2$ with matching input and output alphabets Σ, is a weighted transducer denoted by $\mathbf{U}_1 \circ \mathbf{U}_2$ when the semiring is commutative and the sum:

$$(\mathbf{U}_1 \circ \mathbf{U}_2)(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{z}\in\Sigma^*} \mathbf{U}_1(\mathbf{x}, \mathbf{z}) \times \mathbf{U}_2(\mathbf{z}, \mathbf{y})$$

is well-defined and in $\mathbb{R}$ for all $\mathbf{x}, \mathbf{y}$ [21]. It can be computed in time $O(|\mathbf{U}_1||\mathbf{U}_2|)$ where we denote by $|\mathbf{U}|$ the sum of the number of states and transitions of a transducer $\mathbf{U}$. In the following, we shall use the distributivity of $+$ and multiplication by a real number, γ, over the composition of weighted transducers:

$$(\mathbf{U}_1 \circ \mathbf{U}_3) + (\mathbf{U}_2 \circ \mathbf{U}_3) = (\mathbf{U}_1 + \mathbf{U}_2) \circ \mathbf{U}_3$$
$$\gamma(\mathbf{U}_1 \circ \mathbf{U}_2) = ((\gamma\mathbf{U}_1) \circ \mathbf{U}_2) = (\mathbf{U}_1 \circ (\gamma\mathbf{U}_2)).$$

For any transducer $\mathbf{U}$, $\mathbf{U}^{-1}$ denotes its *inverse*, that is the transducer obtained from $\mathbf{U}$ by swapping the input and output labels of each transition. For all $\mathbf{x}, \mathbf{y} \in \Sigma^*$, we have $\mathbf{U}^{-1}(\mathbf{x}, \mathbf{y}) = \mathbf{U}(\mathbf{y}, \mathbf{x})$.

We introduce a linear operator D over the set of weighted

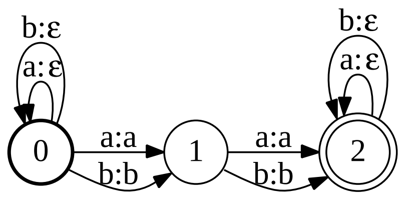

Figure 2: Counting transducer $\mathbf{T}_2$ for $\Sigma = \{a, b\}$.

transducers. For any transducer $\mathbf{U}$, we define $\mathrm{D}(\mathbf{U})$ as the sum of the weights of all accepting paths of the weighted transducer $\mathbf{U}$:

$$\mathrm{D}(\mathbf{U}) = \sum_{\pi \in \mathrm{Acc}(\mathbf{U})} w[\pi],$$

where $\mathrm{Acc}(\mathbf{U})$ denotes the accepting paths of $\mathbf{U}$ and $w[\pi]$ the weight of an accepting path π. By definition of D, we have the following properties for all $\gamma \in \mathbb{R}$ and any weighted transducers $\mathbf{U}_i$, $i \in [1, m]$ and $\mathbf{U}$:

$$\sum_{i=1}^{m} \mathrm{D}(\mathbf{U}_i) = \mathrm{D}\left(\sum_{i=1}^{m} \mathbf{U}_i\right) \qquad \gamma\,\mathrm{D}(\mathbf{U}) = \mathrm{D}(\gamma\mathbf{U}).$$

2.2 Rational kernels

A kernel between sequences $K : \Sigma^* \times \Sigma^* \to \mathbb{R}$ is *rational* [7] if there exists a weighted transducer $\mathbf{U}$ such that K coincides with the function defined by $\mathbf{U}$:

$$K(\mathbf{x}, \mathbf{y}) = \mathbf{U}(\mathbf{x}, \mathbf{y})$$

for all $\mathbf{x}, \mathbf{y} \in \Sigma^*$. When there exists a weighted transducer $\mathbf{T}$ such that $\mathbf{U}$ can be decomposed as $\mathbf{U} = \mathbf{T} \circ \mathbf{T}^{-1}$, then it was shown by [7] that K is symmetric and PSD. The sequence kernels commonly used in natural language processing and computational biology are precisely PSD rational kernels of this form.

A standard family of rational kernels is that of n-gram kernels, see [19, 18] for instance. The n-gram kernel K_n of order n is defined as

$$K_n(\mathbf{x}, \mathbf{y}) = \sum_{|\mathbf{z}|=n} c_{\mathbf{x}}(\mathbf{z}) c_{\mathbf{y}}(\mathbf{z}),$$

where $c_{\mathbf{x}}(\mathbf{z})$ is the number of occurrences of $\mathbf{z}$ in $\mathbf{x}$. K_n is a PSD rational kernel since it corresponds to the weighted transducer $\mathbf{T}_n \circ \mathbf{T}_n^{-1}$ where the transducer $\mathbf{T}_n$ is defined such that $\mathbf{T}_n(\mathbf{x}, \mathbf{z}) = c_{\mathbf{x}}(\mathbf{z})$ for all $\mathbf{x}, \mathbf{z} \in \Sigma^*$ with $|\mathbf{z}| = n$. The transducer $\mathbf{T}_2$ for $\Sigma = \{a, b\}$ is shown in Figure 2.

A key advantage of the rational kernel framework is that it can be straightforwardly extended to kernels between two sets of sequences, or distributions over sequences represented by weighted automata. Let $\mathbf{X}$ and $\mathbf{Y}$ be two weighted automata, we can then define $K(\mathbf{X}, \mathbf{Y})$ as follow:

$$\begin{aligned} K(\mathbf{X}, \mathbf{Y}) &= \sum_{\mathbf{x}, \mathbf{y} \in \Sigma^*} \mathbf{X}(\mathbf{x}) \times K(\mathbf{x}, \mathbf{y}) \times \mathbf{Y}(\mathbf{y}) \\ &= \sum_{\mathbf{x}, \mathbf{y} \in \Sigma^*} \mathbf{X}(\mathbf{x}) \times \mathbf{U}(\mathbf{x}, \mathbf{y}) \times \mathbf{Y}(\mathbf{y}) \\ &= \mathrm{D}(\mathbf{X} \circ \mathbf{U} \circ \mathbf{Y}). \end{aligned}$$

This extension is particularly important and relevant since it helps define kernels between the lattices output by information extraction, speech recognition, machine translation systems, and other natural language processing tasks. Our results for faster SVMs training with sequence kernels apply similarly to large-scale training with kernels between lattices.

3. COORDINATE DESCENT SOLUTION FOR SVM OPTIMIZATION

We first briefly discuss the coordinate descent solution for SVMs as in [14]. In the absence of the offset term b, where a constant feature is used instead, the standard dual optimization for SVMs for a sample of size m can be written as the convex optimization problem:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & F(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{Q}\boldsymbol{\alpha} - \mathbf{1}^\top \boldsymbol{\alpha} \\ \text{s.t.} \quad & \mathbf{0} \le \boldsymbol{\alpha} \le \mathbf{C}, \end{aligned}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^m$ is the vector of dual variables and the PSD matrix $\mathbf{Q}$ is defined in terms of the kernel matrix $\mathbf{K}$: $\mathbf{Q}_{ij} = y_i y_j \mathbf{K}_{ij}$, $i, j \in [1, m]$, and the labels $y_i \in \{-1, +1\}$.

A straightforward way to solve this convex problem is to use a coordinate descent method and at each iteration update just one coordinate α_i. The optimal step size $\beta^\star$ corresponding to the update of α_i is obtained by solving

$$\begin{aligned} \min_{\beta} \quad & \frac{1}{2}(\boldsymbol{\alpha} + \beta\boldsymbol{e}_i)^\top \mathbf{Q}(\boldsymbol{\alpha} + \beta\boldsymbol{e}_i) - \mathbf{1}^\top(\boldsymbol{\alpha} + \beta\boldsymbol{e}_i) \\ \text{s.t.} \quad & \mathbf{0} \le \boldsymbol{\alpha} + \beta\boldsymbol{e}_i \le \mathbf{C}, \end{aligned}$$

where $\boldsymbol{e}_i$ is an m-dimensional unit vector. Ignoring constant terms, the optimization problem can be written as

$$\begin{aligned} \min_{\beta} \quad & \frac{1}{2}\beta^2 \mathbf{Q}_{ii} + \beta\boldsymbol{e}_i^\top(\mathbf{Q}\boldsymbol{\alpha} - \mathbf{1}) \\ \text{s.t.} \quad & 0 \le \alpha_i + \beta \le C. \end{aligned}$$

If $\mathbf{Q}_{ii} = \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_i) = 0$, then $\Phi(\mathbf{x}_i) = 0$ and $\mathbf{Q}_i = \boldsymbol{e}_i^\top \mathbf{Q} = 0$. Hence the objective function reduces to $-\beta$, and the optimal step size is $\beta^\star = C - \alpha_i$, resulting in the update: $\alpha_i \leftarrow 0$. Otherwise $\mathbf{Q}_{ii} \neq 0$ and the objective function is a second-degree polynomial in β. Let $\beta_0 = -\frac{\mathbf{Q}_i^\top \boldsymbol{\alpha} - 1}{\mathbf{Q}_{ii}}$, then the optimal step size is given by

$$\beta^\star = \begin{cases} \beta_0 & \text{if } -\alpha_i \le \beta_0 \le C, \\ -\alpha_i & \text{if } \beta_0 \le -\alpha_i, \\ C - \alpha_i & \text{otherwise.} \end{cases}$$

The resulting update for α_i is

$$\alpha_i \leftarrow \min\left(\max\left(\alpha_i - \frac{\mathbf{Q}_i^\top \boldsymbol{\alpha} - 1}{\mathbf{Q}_{ii}}, 0\right), C\right).$$

Algorithm 1 Coordinate descent solution for SVM

```
TRAIN($(\mathbf{x}_i)_{i\in[1,m]}$)
1  $\boldsymbol{\alpha} \leftarrow \mathbf{0}$
2  while $\boldsymbol{\alpha}$ not optimal do
3    for $i \in [1, m]$ do
4      $g \leftarrow y_i \mathbf{x}_i^\top \mathbf{w} - 1$
5      $\alpha'_i \leftarrow \min(\max(\alpha_i - \frac{g}{\mathbf{Q}_{ii}}, 0), C)$
6      $\mathbf{w} \leftarrow \mathbf{w} + (\alpha'_i - \alpha_i)\mathbf{x}_i$
7      $\alpha_i \leftarrow \alpha'_i$
8  return w
```

When the matrix $\mathbf{Q}$ is too large to store in memory and $\mathbf{Q}_{ii} \neq 0$, the vector $\mathbf{Q}_i$ must be computed at each update of α_i. If the cost of the computation of each entry $\mathbf{K}_{ij}$ is in $O(N)$ where N is the dimension of the input space, computing $\mathbf{Q}_i$ is in the $O(mN)$, and hence the cost of each update is in $O(mN)$.

The selection of the coordinate α_i to update is based on the gradient. The gradient of the objective function is $\nabla F(\boldsymbol{\alpha}) = \mathbf{Q}\boldsymbol{\alpha} - \mathbf{1}$. It can be updated via

$$\nabla F(\boldsymbol{\alpha}) \leftarrow \nabla F(\boldsymbol{\alpha}) + \Delta(\alpha_i)\mathbf{Q}_i.$$

The cost of this update is also in $O(mN)$.

[14] observed that when the kernel is linear, $\mathbf{Q}_i^\top \boldsymbol{\alpha}$ can be expressed in terms of $\mathbf{w}$, the SVM weight vector solution, $\mathbf{w} = \sum_{j=1}^m y_j \alpha_j \mathbf{x}_j$:

$$\mathbf{Q}_i^\top \boldsymbol{\alpha} = \sum_{j=1}^m y_i y_j (\mathbf{x}_i^\top \mathbf{x}_j)\alpha_j = y_i \mathbf{x}_i^\top \mathbf{w}.$$

If the weight vector $\mathbf{w}$ is maintained throughout the iterations, then the cost of an update is only in $O(N)$ in this case. The weight vector $\mathbf{w}$ can be updated via

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta(\alpha_i) y_i \mathbf{x}_i.$$

Maintaining the gradient $\nabla F(\alpha)$ is however still costly. The jth component of the gradient can be expresses as:

$$\begin{aligned}[\nabla F(\alpha)]_j &= [\mathbf{Q}\boldsymbol{\alpha} - \mathbf{1}]_j \\ &= \sum_{i=1}^m y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \alpha_i - 1 = \mathbf{w}^\top (y_j \mathbf{x}_j) - 1.\end{aligned}$$

The update for the main term of component j of the gradient is thus given by:

$$\mathbf{w}^\top \mathbf{x}_j \leftarrow \mathbf{w}^\top \mathbf{x}_j + (\Delta \mathbf{w})^\top \mathbf{x}_j.$$

Each of these updates can be done in $O(N)$. The full update for the gradient can hence be done in $O(mN)$.

Several heuristics can be used to eliminate the cost of maintaining the gradient. For instance, one can choose a random α_i to update at each iteration [14] or sequentially update α_is. [14] also showed that it is possible to use the chunking method of [15] in conjunction with such heuristics.

Using the results from [20], [14] showed that the resulting coordinate descent algorithm, Algorithm 1, converges to the optimal solution with a convergence rate that is linear or faster.

4. COORDINATE DESCENT SOLUTION FOR RATIONAL KERNELS

This section shows that, remarkably, coordinate descent techniques similar to those described in the previous section can be used in the case of rational kernels.

For rational kernels, the input "vectors" $\mathbf{x}_i$ are sequences, or distributions over sequences, and the expression $\sum_{j=1}^m y_j \alpha_j \mathbf{x}_j$ can be interpreted as a weighted regular expression. Let $\mathbf{X}_i$ be a linear weighted automaton representing $\mathbf{x}_i$ for all $i \in [1, m]$, and let $\mathbf{W}$ denote a weighted automaton representing $\mathbf{w} = \sum_{j=1}^m y_j \alpha_j \mathbf{x}_j$.

Let $\mathbf{U}$ be the weighted transducer associated to the rational kernel K. Using the linearity of D and distributivity properties just presented, we can now write:

$$\begin{aligned}\mathbf{Q}_i^\top \boldsymbol{\alpha} &= \sum_{j=1}^m y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)\alpha_j \qquad (1)\\ &= \sum_{j=1}^m y_i y_j \, \mathrm{D}(\mathbf{X}_i \circ \mathbf{U} \circ \mathbf{X}_j)\alpha_j \\ &= \mathrm{D}(y_i \mathbf{X}_i \circ \mathbf{U} \circ \sum_{j=1}^m y_j \alpha_j \mathbf{X}_j) \\ &= \mathrm{D}(y_i \mathbf{X}_i \circ \mathbf{U} \circ \mathbf{W}).\end{aligned}$$

Since $\mathbf{U}$ is a constant, in view of the complexity of composition, the expression $y_i \mathbf{X}_i \circ \mathbf{U} \circ \mathbf{W}$ can be computed in time $O(|\mathbf{X}_i||\mathbf{W}|)$. When $y_i \mathbf{X}_i \circ \mathbf{U} \circ \mathbf{W}$ is acyclic, which is the case for example if $\mathbf{U}$ admits no input ϵ-cycle, then $\mathrm{D}(y_i \mathbf{X}_i \circ \mathbf{U} \circ \mathbf{W})$ can be computed in linear time in the size of $y_i \mathbf{X}_i \circ \mathbf{U} \circ \mathbf{W}$ using a shortest-distance algorithm, or forward-backward algorithm. For all of the rational kernels that we are aware of, $\mathbf{U}$ admits no input ϵ-cycle and this property holds. Thus, in that case, if we maintain a weighted automaton $\mathbf{W}$ representing $\mathbf{w}$, $\mathbf{Q}_i^\top \boldsymbol{\alpha}$ can be computed in $O(|\mathbf{X}_i||\mathbf{W}|)$. This complexity does not depend on m and the explicit computation of m kernel values $K(\mathbf{x}_i, \mathbf{x}_j)$, $j \in [1, m]$, is avoided.

The update rule for $\mathbf{W}$ consists of augmenting the weight of sequence $\mathbf{x}_i$ in the weighted automaton by $\Delta(\alpha_i) y_i$:

$$\mathbf{W} \leftarrow \mathbf{W} + \Delta(\alpha_i) y_i \mathbf{X}_i.$$

This update can be done very efficiently if $\mathbf{W}$ is deterministic, in particular if it is represented as a deterministic trie.

When the weighted transducer $\mathbf{U}$ can be decomposed as $\mathbf{T} \circ \mathbf{T}^{-1}$, as for all sequence kernels seen in practice, we can further improve the form of the updates. Let $\Pi_2(\mathbf{U})$ denote the weighted automaton obtained form $\mathbf{U}$ by projection over

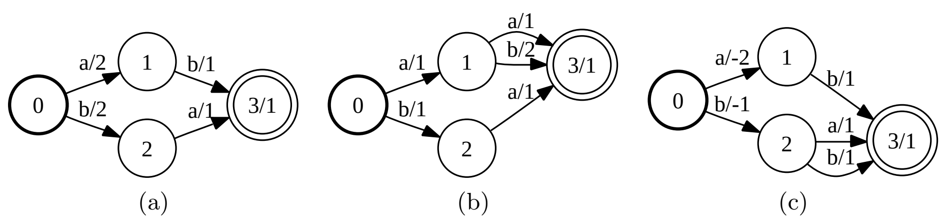

Figure 3: The automata $\mathbf{\Phi}'_i$ corresponding to the dataset of Table 1 when using a bigram kernel.

Algorithm 2 Coordinate descent solution for rational kernels

```
TRAIN($(\mathbf{\Phi}'_i)_{i\in[1,m]}$)
1  $\boldsymbol{\alpha} \leftarrow \mathbf{0}$
2  while $\boldsymbol{\alpha}$ not optimal do
3    for $i \in [1,m]$ do
4      $g \leftarrow \mathrm{D}(\mathbf{\Phi}'_i \circ \mathbf{W}') - 1$
5      $\alpha'_i \leftarrow \min(\max(\alpha_i - \frac{g}{\mathbf{Q}_{ii}}, 0), C)$
6      $\mathbf{W}' \leftarrow \mathbf{W}' + (\alpha'_i - \alpha_i)\mathbf{\Phi}'_i$
7      $\alpha_i \leftarrow \alpha'_i$
8  return $\mathbf{W}'$
```

the output labels as described in Section 2. Then

$$\begin{aligned}\mathbf{Q}_i^\top \boldsymbol{\alpha} &= \mathrm{D}(y_i\mathbf{X}_i \circ \mathbf{T} \circ \mathbf{T}^{-1} \circ \mathbf{W})\\ &= \mathrm{D}((y_i\mathbf{X}_i \circ \mathbf{T}) \circ (\mathbf{W} \circ \mathbf{T})^{-1})\\ &= \mathrm{D}(\Pi_2(y_i\mathbf{X}_i \circ \mathbf{T}) \circ \Pi_2(\mathbf{W} \circ \mathbf{T}))\\ &= \mathrm{D}(\mathbf{\Phi}'_i \circ \mathbf{W}'), \qquad (2)\end{aligned}$$

where $\mathbf{\Phi}'_i = \Pi_2(y_i\mathbf{X}_i \circ \mathbf{T})$ and $\mathbf{W}' = \Pi_2(\mathbf{W} \circ \mathbf{T})$. $\mathbf{\Phi}'_i$, $i \in [1,m]$ can be precomputed and instead of $\mathbf{W}$, we can equivalently maintain $\mathbf{W}'$, with the following simple update rule:

$$\mathbf{W}' \leftarrow \mathbf{W}' + \Delta(\alpha_i)\mathbf{\Phi}'_i. \qquad (3)$$

The gradient $\nabla(F)(\boldsymbol{\alpha}) = \mathbf{Q}\boldsymbol{\alpha} - \mathbf{1}$ can be expressed as follows

$$[\nabla(F)(\boldsymbol{\alpha})]_j = [\mathbf{Q}^\top\boldsymbol{\alpha} - \mathbf{1}]_j = \mathbf{Q}_j^\top\boldsymbol{\alpha} - 1 = \mathrm{D}(\mathbf{\Phi}'_j \circ \mathbf{W}') - 1.$$

The update rule for the main term $\mathrm{D}(\mathbf{\Phi}'_j \circ \mathbf{W}')$ can be written as

$$\mathrm{D}(\mathbf{\Phi}'_j \circ \mathbf{W}') \leftarrow \mathrm{D}(\mathbf{\Phi}'_j \circ \mathbf{W}') + \mathrm{D}(\mathbf{\Phi}'_j \circ \Delta\mathbf{W}').$$

Maintaining each of these terms explicitly could be costly.

Using (2) to compute the gradient and (3) to update $\mathbf{W}'$, we can generalize Algorithm 1 and obtain Algorithm 2. It follows from [20] that Algorithm 2 converges at least linearly towards a global optimal solution. Moreover, the heuristics used by [14] and mentioned in the previous section can also be applied here to empirically improve the convergence rate of the algorithm.

Table 2 shows the first iteration of Algorithm 2 on the dataset given by Table 1 when using a bigram kernel.

Table 1: Example dataset, the given $\mathbf{\Phi}'_i$ and $\mathbf{Q}_{ii}$'s assume the use of a bigram kernel.

i	1	2	3
$\mathbf{x}_i$	*ababa*	*abbab*	*abbab*
y_i	$+1$	$+1$	-1
$\mathbf{\Phi}'_i$	Fig. 3(a)	Fig. 3(b)	Fig. 3(c)
$\mathbf{Q}_{ii}$	8	6	4

Table 2: First iteration of Algorithm 2 on the dataset given Table 1. The last line gives the values of $\boldsymbol{\alpha}$ and $\mathbf{W}'$ at the end of the iteration.

i	$\boldsymbol{\alpha}$	$\mathbf{W}'$	$\mathbf{\Phi}'_i \circ \mathbf{W}'$	$\mathrm{D}(\mathbf{\Phi}'_i \circ \mathbf{W}')$	α'_i
1	$(0,0,0)$	Fig. 4(a)	Fig. 5(a)	0	$\frac{1}{8}$
2	$(\frac{1}{8},0,0)$	Fig. 4(b)	Fig. 5(b)	$\frac{3}{4}$	$\frac{1}{24}$
3	$(\frac{1}{8},\frac{1}{24},0)$	Fig. 4(c)	Fig. 5(c)	$-\frac{5}{8}$	$\frac{13}{32}$
	$(\frac{1}{8},\frac{1}{24},\frac{13}{32})$	Fig. 4(d)			

5. IMPLEMENTATION AND ANALYSIS

We now proceed with the analysis of the complexity of each iteration of Algorithm 2. Clearly, this complexity depends on several implementation choices, but also on the kernel used and on the structural properties of the problem considered. A key factor is the choice of the data structure used to represent $\mathbf{W}'$.

In order to simplify the analysis, we assume that the $\mathbf{\Phi}'_i$s, and thus $\mathbf{W}'$, are acyclic. This assumption holds for all rational kernels used in practice, however, it is not a requirement for the correctness of Algorithm 2.

Given an acyclic weighted automaton $\mathbf{A}$, we denote by $l(\mathbf{A})$ the maximal length of an accepting path in $\mathbf{A}$ and by $n(\mathbf{A})$ the number of accepting paths in $\mathbf{A}$.

5.1 Naive representation of $\mathbf{W}'$

A straightforward choice consists of following directly the definition of $\mathbf{W}'$:

$$\mathbf{W}' = \sum_{i=1}^{m} \alpha_i \mathbf{\Phi}'_i,$$

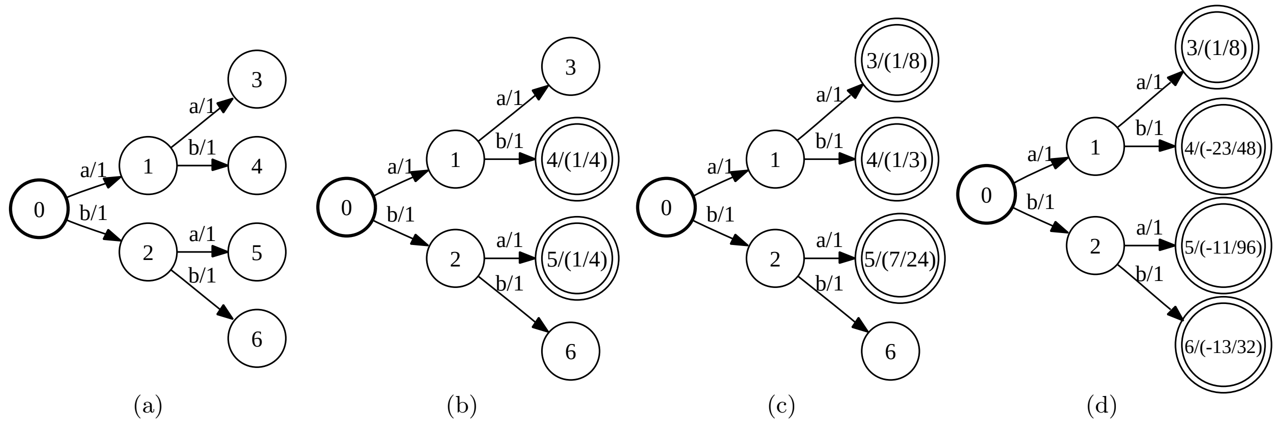

Figure 4: Evolution of $\mathbf{W}'$ through the first iteration of Algorithm 2 on the dataset from Table 1.

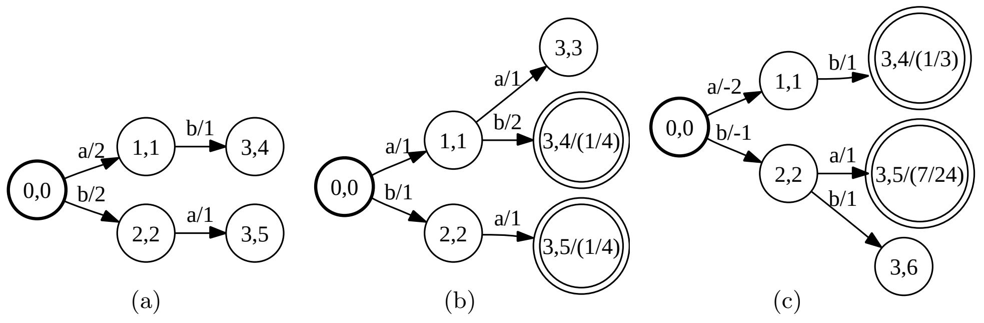

Figure 5: The automata $\mathbf{\Phi}_i' \circ \mathbf{W}'$ computed during the first iteration of Algorithm 2 on the dataset from Table 1.

and define $\mathbf{W}'$ as a non-deterministic weighted automaton with a single initial state and m outgoing ϵ-transitions, where the weight of the ith transition is α_i and its destination state the initial state of $\mathbf{\Phi}_i'$. The size of this choice of $\mathbf{W}'$ is $|\mathbf{W}'| = m + \sum_{i=1}^m |\mathbf{\Phi}_i'|$.

The benefit of this representation is that the update of $\boldsymbol{\alpha}$ using (3) can be performed in constant time since it requires modifying only the weight of one of the ϵ-transitions out of the initial state. However, the complexity of computing the gradient using (2) is in $O(|\mathbf{\Phi}_i'||\mathbf{W}'|) = O(|\mathbf{\Phi}_i'| \sum_{i=1}^m |\mathbf{\Phi}_i'|)$.

From an algorithmic point of view, using this naive representation of $\mathbf{W}'$ is equivalent to using (1) with $y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) = \mathrm{D}(\mathbf{\Phi}_i' \circ \mathbf{\Phi}_j')$ to compute the gradient.

5.2 Representing $\mathbf{W}'$ as a trie

Representing $\mathbf{W}'$ as a deterministic weighted trie is another approach that can lead to a simple update using (3). A *weighted trie* is a rooted tree where each edge is labeled and each node is weighted.

During composition, each accepting path in $\mathbf{\Phi}_i'$ is matched with a distinct node in $\mathbf{W}'$. Thus, $n(\mathbf{\Phi}_i')$ paths of $\mathbf{W}'$ are explored during composition. Since the length of each of these paths is at most $l(\mathbf{\Phi}_i')$, this leads to a complexity in $O(n(\mathbf{\Phi}_i')l(\mathbf{\Phi}_i'))$ for computing $\mathbf{\Phi}_i' \circ \mathbf{W}'$ and thus for computing the gradient using (2).

Since each accepting path in $\mathbf{\Phi}_i'$ corresponds to a distinct node in $\mathbf{W}'$, the weights of at most $n(\mathbf{\Phi}_i')$ nodes of $\mathbf{W}'$ need to be updated. Thus, the complexity of an update of $\mathbf{W}'$ is then in $O(n(\mathbf{\Phi}_i'))$.

5.3 Representing $\mathbf{W}'$ as a minimal automaton

The drawback of a trie representation of $\mathbf{W}'$ is that it does not provide all of the sparsity benefits of a fully automata-based approach. A more space-efficient approach consists of representing $\mathbf{W}'$ as a minimal deterministic weighted automaton which can be substantially smaller, exponentially smaller in some cases, than the corresponding trie.

The complexity of computing the gradient using (2) is then in $O(|\mathbf{\Phi}_i' \circ \mathbf{W}'|)$ which is significantly less than the $O(n(\mathbf{\Phi}_i')l(\mathbf{\Phi}_i'))$ complexity of the trie representation. Performing the update of $\mathbf{W}'$ using (3) can be more costly though. With the straightforward approach of using the general union, weighted determinization and minimization algorithms [7], the complexity depends on the size of $\mathbf{W}'$. The cost of an update can thus sometimes become large. However, it is perhaps possible to design more efficient algorithms for augmenting a weighted automaton with a single string or even

Table 3: The time complexity of each gradient computation and of each update of $\mathbf{W}'$ and the space complexity required for representing $\mathbf{W}'$ given for each type of representation of $\mathbf{W}'$.

Representation of $\mathbf{W}'$	Time complexity (gradient)	Time complexity (update)	Space complexity (for storing $\mathbf{W}'$)
naive ($\mathbf{W}'_n$)	$O(\lvert\mathbf{\Phi}'_i\rvert \sum_{i=1}^{m} \lvert\mathbf{\Phi}'_i\rvert)$	$O(1)$	$O(m)$
trie ($\mathbf{W}'_t$)	$O(n(\mathbf{\Phi}'_i)l(\mathbf{\Phi}'_i))$	$O(n(\mathbf{\Phi}'_i))$	$O(\lvert\mathbf{W}'_t\rvert)$
minimal automaton ($\mathbf{W}'_m$)	$O(\lvert\mathbf{\Phi}'_i \circ \mathbf{W}'_m\rvert)$	open	$O(\lvert\mathbf{W}'_m\rvert)$

Table 4: Time (in minutes and seconds) for training an SVM classifier using an SMO-like algorithm and Algorithm 2 using a trie representation for $\mathbf{W}'$.

Dataset	Kernel	SMO-like	Alg. 2
Reuters	4-gram	2m 18s	25s
(subset)	5-gram	3m 56s	30s
	6-gram	6m 16s	41s
	7-gram	9m 24s	1m 01s
	10-gram	25m 22s	1m 53s
	gappy 3-gram	10m 40s	1m 23s
	gappy 4-gram	58m 08s	7m 42s
Reuters	4-gram	618m 43s	16m 30s
(full)	5-gram	$>$ 2000m	23m 17s
	6-gram	$>$ 2000m	31m 22s
	7-gram	$>$ 2000m	37m 23s

a set of strings represented by a deterministic automaton, while preserving determinism and minimality. The approach just described forms a strong motivation for the study and analysis of such non-trivial and probably sophisticated automata algorithms since it could lead to even more efficient updates of $\mathbf{W}'$ and overall speed-up of the SVMs training with rational kernels. We leave the study of this open question to the future. We note, however, that that analysis could benefit from existing algorithms in the unweighted case. Indeed, in the unweighted case, a number of efficient algorithms have been designed for incrementally adding a string to a minimal deterministic automaton while keeping the result minimal and deterministic [9, 4], and the complexity of each addition of a string using these algorithms is only linear in the length of the string added.

Table 3 summarizes the time and space requirements for each type of representation for $\mathbf{W}'$. In the case of an n-gram kernel of order k, $l(\mathbf{\Phi}'_i)$ is a constant k, $n(\mathbf{\Phi}'_i)$ is the number of distinct k-grams occurring in $\mathbf{x}_i$, $n(\mathbf{W}'_t)$ $(= n(\mathbf{W}'_m))$ the number of distinct k-grams occurring in the dataset, and $\lvert\mathbf{W}'_t\rvert$ the number of distinct n-grams of order less or equal to k in the dataset.

6. EXPERIMENTS

We used the Reuters-21578 dataset, a large data set convenient for our analysis and commonly used in experimental analyses of string kernels.[1] We shall refer by *full dataset* to the 12,902 news stories part of the ModeApte split.[2] We also considered a subset of that dataset consisting of 466 news stories. We experimented both with n-gram kernels and gappy n-gram kernels with different n-gram orders. We trained binary SVM classification for the `acq` class using the following two algorithms: (a) the SMO-like algorithm of [11] implemented using LIBSVM [5] and modified to handle the on-demand computation of rational kernels; and (b) Algorithm 2 implemented using a trie representation for $\mathbf{W}'$. We chose a dataset of moderate size in order to be able to run the SMO-like algorithm. Table 4 reports the training time observed,[3] excluding the pre-processing step which consists of computing $\mathbf{\Phi}'_i$ for each data point and that is common to both algorithms.

To estimate the benefits of representing $\mathbf{W}'$ as a minimal automaton as described in Section 5.3, we applied the weighted minimization algorithm to the tries output by Algorithm 2 (after shifting the weights to the non-negative domain) and observed the resulting reduction in size. The results are reported in Table 5. They show that representing $\mathbf{W}'$ by a minimal deterministic automaton can lead to very significant savings in space and point out the substantial benefits of the representation discussed in Section 5.3, and further substantial reduction of the training time with respect to the trie representation with an incremental addition of strings to $\mathbf{W}'$.

7. CONCLUSION

We presented novel techniques for large-scale training of SVMs when used with sequence kernels. We gave a detailed description of our algorithms and discussed different implementation choices, and presented an analysis of the resulting complexity. Our empirical results with large-scale data sets demonstrate dramatic reductions of the training time. We plan to make our software publicly available through an open-source project. From the algorithmic point of view, it is interesting to note that our training algorithm for SVMs is entirely based on automata algorithms and requires no specific solver.

[1]Available at: `http://www.daviddlewis.com/resources/`.

[2]Since we are not interested in classification accuracy, we actually train on the training and test sets combined.

[3]Experiments were performed on dual-core 2.2 GHz AMD Opteron workstation with 16GB of RAM.

Table 5: Size of $\mathbf{W}'$ (number of transitions) when representing $\mathbf{W}'$ as a deterministic weighted trie and a minimal deterministic weighted automaton.

Dataset	Kernel	Number of transitions in $\mathbf{W}'$ (trie)	(minimal automaton)
Reuters	4-gram	66,331	34,785
(subset)	5-gram	154,460	63,643
	6-gram	283,856	103,459
	7-gram	452,881	157,390
	10-gram	1,151,217	413,878
	gappy 3-gram	103,353	66,650
	gappy 4-gram	1,213,281	411,939
	gappy 5-gram	6,423,447	1,403,744
Reuters	4-gram	242,570	106,640
(full)	5-gram	787,514	237,783
	6-gram	1,852,634	441,242
	7-gram	3,570,741	727,743

References

[1] C. Allauzen, M. Mohri, and A. Talwalkar. Sequence kernels for predicting protein essentiality. In *ICML 2008*, 2008.

[2] F. R. Bach and M. I. Jordan. Kernel independent component analysis. *JMLR*, 3:1–48, 2002.

[3] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *COLT*, volume 5, 1992.

[4] R. C. Carrosco and M. L. Forcada. Incremental construction and maintenance of minimal finite-state automata. *Computational Linguistics*, 28(2):207–216, 2002.

[5] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001.

[6] M. Collins and N. Duffy. Convolution kernels for natural language. In *NIPS*. MIT Press, 2002.

[7] C. Cortes, P. Haffner, and M. Mohri. Rational Kernels: Theory and Algorithms. *JMLR*, 5, 2004.

[8] C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3), 1995.

[9] J. Daciuk, S. Mihov, B. W. Watson, and R. Watson. Incremental construction of minimal acyclic finite state automata. *Computational Linguistics*, 26(1):3–16, 2000.

[10] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 9, 2008.

[11] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training SVM. *JMLR*, 6:1889–1918, 2005.

[12] S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2002.

[13] D. Haussler. Convolution Kernels on Discrete Structures. Technical Report UCSC-CRL-99-10, University of California at Santa Cruz, 1999.

[14] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *ICML*, pages 408–415, 2008.

[15] T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods: Support Vector Learning*. The MIT Press, 1998.

[16] S. Kumar, M. Mohri, and A. Talwalkar. On sampling-based approximate spectral decomposition. In *ICML*, 2009.

[17] C. S. Leslie, E. Eskin, A. Cohen, J. Weston, and W. S. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4), 2004.

[18] C. S. Leslie, E. Eskin, and W. S. Noble. The Spectrum Kernel: A String Kernel for SVM Protein Classification. In *Pacific Symposium on Biocomputing*, pages 566–575, 2002.

[19] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *JMLR*, 2, 2002.

[20] Z. Q. Luo and P. Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. *J. of Optim. Theor. and Appl.*, 72(1):7–35, 1992.

[21] A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag, 1978.

[22] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press: Cambridge, MA, 2002.

[23] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge Univ. Press, 2004.

[24] I. W. Tsang, J. T. Kwok, and P.-M. Cheung. Core vector machines: Fast SVM training on very large data sets. *JMLR*, 6:363–392, 2005.

[25] V. N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

[26] C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *NIPS*, pages 682–688, 2000.

[27] A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer, and K.-R. Müller. Engineering support vector machine kernels that recognize translation initiation sites. *Bioinformatics*, 16(9), 2000.