Large-Scale Training of SVMs with Automata Kernels

Cyril Allauzen[1], Corinna Cortes[1], and Mehryar Mohri[2,1]

[1] Google Research, 76 Ninth Avenue, New York, NY 10011
[2] Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, NY 10012

Abstract. This paper presents a novel application of automata algorithms to machine learning. It introduces the first optimization solution for support vector machines used with sequence kernels that is purely based on weighted automata and transducer algorithms, without requiring any specific solver. The algorithms presented apply to a family of kernels covering all those commonly used in text and speech processing or computational biology. We show that these algorithms have significantly better computational complexity than previous ones and report the results of large-scale experiments demonstrating a dramatic reduction of the training time, typically by several orders of magnitude.

1 Introduction

Weighted automata and transducer algorithms have been used successfully in a variety of natural language processing applications, including speech recognition, speech synthesis, and machine translation [17]. More recently, they have found other important applications in machine learning [5, 1]: they can be used to define a family of sequence kernels, *rational kernels* [5], which covers all sequence kernels commonly used in machine learning applications in bioinformatics or text and speech processing.

Sequences kernels are similarity measures between sequences that are positive definite symmetric, which implies that their value coincides with an inner product in some Hilbert space. Kernels are combined with effective learning algorithms such as support vector machines (SVMs) [6] to create powerful classification techniques, or with other learning algorithms to design regression, ranking, clustering, or dimensionality reduction solutions [19]. These kernel methods are among the most widely used techniques in machine learning.

Scaling these algorithms to large-scale problems remains computationally challenging however, both in time and space. One solution consists of using approximation techniques for the kernel matrix, e.g., [9, 2, 21, 13] or to use early stopping for optimization algorithms [20]. 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 used with an arbitrary rational kernel, without resorting to such approximations. We show that coordinate descent approaches similar to those used by [10] for linear kernels can be extended to SVMs combined with rational kernels to design faster algorithms with significantly better computational complexity. Remarkably, our solution

techniques are purely based on weighted automata and transducer algorithms and require no specific optimization solver. To the best of our knowledge, they form the first automata-based optimization algorithm of SVMs, probably the most widely used algorithm in machine learning. Furthermore, we show experimentally that our techniques lead to a dramatic 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 to weighted transducers and rational kernels (Section 2), including definitions and properties relevant to the following sections. Section 3 provides a short introduction to kernel methods such as SVMs and presents an overview of the coordinate descent solution by [10] for linear SVMs. 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 introduces the essential concepts and definitions related to weighted transducers and rational kernels. Generally, we adopt the definitions and terminology of [5].

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 [12]. In this paper, we only consider weighted transducers over the *real semiring* $(\mathbb{R}_+, +, \times, 0, 1)$. Figure 1(a) shows an example. A path from an initial state to a final state is an accepting path. The input (resp. output) label of an accepting path is obtained by concatenating together the input (resp. output) symbols along the path from the initial to the final state. Its weight 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. 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 $\mathbf{U}(\mathbf{x}, \mathbf{y})$. For any transducer $\mathbf{U}$ we define the linear operator D as the sum of the weights of all accepting paths of $\mathbf{U}$.

A *weighted automaton* $\mathbf{A}$ can be defined as a weighted transducer with identical input and output labels. Discarding the input labels of a weighted transducer $\mathbf{U}$ results in a weighted automaton $\mathbf{A}$, said to be the *output projection of* $\mathbf{U}$, $\mathbf{A} = \Pi_2(\mathbf{U})$. The automaton in Figure 1(b) is the output projection of the transducer in Figure 1(a).

The standard operations of sum $+$, product or concatenation $\cdot$, multiplication by a real number and Kleene-closure * are defined for weighted transducers [18]. The *inverse* of a transducer $\mathbf{U}$, denoted by $\mathbf{U}^{-1}$, is obtained by swapping the input and output labels of each transition. For all pairs of strings $(\mathbf{x}, \mathbf{y})$, we have $\mathbf{U}^{-1}(\mathbf{x}, \mathbf{y}) = \mathbf{U}(\mathbf{y}, \mathbf{x})$. The *composition* of two weighted transducers $\mathbf{U}_1$ and $\mathbf{U}_2$ with matching output and input alphabets Σ, is a weighted transducer denoted by $\mathbf{U}_1 \circ \mathbf{U}_2$ when the sum:

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

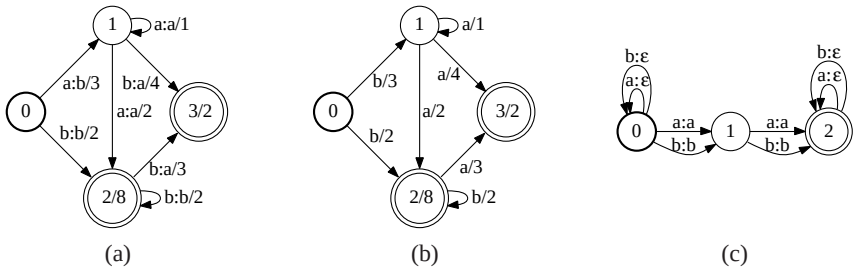


Fig. 1. (a) Example of weighted transducer $\mathbf{U}$. (b) Example of weighted automaton $\mathbf{A}$. In this example, $\mathbf{A}$ can be obtained from $\mathbf{U}$ by projection on the output and $\mathbf{U}(aab, baa) = \mathbf{A}(baa) = 3\times1\times4\times2+3\times2\times3\times2$. (c) Bigram counting transducer $\mathbf{T}_2$ for $\Sigma = \{a, b\}$. Initial states are represented by bold circles, final states by double circles and the weights of transitions and final states are indicated after the slash separator.

is well-defined and in $\mathbb{R}$ for all $\mathbf{x}, \mathbf{y}$ [18]. It can be computed in time $O(|\mathbf{U}_1||\mathbf{U}_2|))$ where $|\mathbf{U}|$ denotes the sum of the number of states and transitions of a transducer $\mathbf{U}$.

Given a non-empty set X, a function $K\colon X \times X \to \mathbb{R}$ is called a *kernel*. K is said to be *positive definite symmetric* (PDS) when the matrix $(K(\mathbf{x}_i, \mathbf{x}_j))_{1\leq i,j\leq m}$ is symmetric and positive semi-definite (PSD) for any choice of m points in X. A kernel between sequences $K\colon \Sigma^* \times \Sigma^* \to \mathbb{R}$ is *rational* [5] if there exists a weighted transducer $\mathbf{U}$ such that K coincides with the function defined by $\mathbf{U}$, that is $K(\mathbf{x}, \mathbf{y}) = \mathbf{U}(\mathbf{x}, \mathbf{y})$ 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 [5] that K is PDS. All the sequence kernels seen in practice are precisely PDS rational kernels of this form.

A standard family of rational kernels is n-gram kernels, see e.g. [15, 14]. Let $c_{\mathbf{x}}(\mathbf{z})$ be the number of occurrences of $\mathbf{z}$ in $\mathbf{x}$. The n-gram kernel K_n of order n is defined as $K_n(\mathbf{x}, \mathbf{y}) = \sum_{|\mathbf{z}|=n} c_{\mathbf{x}}(\mathbf{z}) c_{\mathbf{y}}(\mathbf{z})$. K_n is a PDS 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 $\mathbf{T}_n(\mathbf{x}, \mathbf{z}) = c_x(\mathbf{z})$ 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 1(c).

3 Kernel Methods and SVM Optimization

Kernel methods are widely used in machine learning. They have been successfully used in a variety of learning tasks including classification, regression, ranking, clustering, and dimensionality reduction. This section gives a brief overview of these methods, and discusses in more detail one of the most popular kernel learning algorithms, SVMs.

3.1 Overview of Kernel Methods

Complex learning tasks are often tackled using a large number of features. Each point of the input space X is mapped to a high-dimensional feature space F via a non-linear mapping $\mathbf{\Phi}$. This may be to seek a linear separation in a higher-dimensional space, which was not achievable in the original space, or to exploit other regression, ranking, clustering, or manifold properties that are easier to attain in that space. The dimension

of the feature space F can be very large. In document classification, the features may be the set of all trigrams. Thus, even for a vocabulary of just $200{,}000$ words, the dimension of F is 2×10^{15}.

The high dimensionality of F does not necessarily affect the generalization ability of large-margin algorithms such as SVMs: remarkably, these algorithms benefit from theoretical guarantees for good generalization that depend only on the number of training points and the separation *margin*, and not on the dimensionality of the feature space. But the high dimensionality of F can directly impact the efficiency and even the practicality of such learning algorithms, as well as their use in prediction. This is because to determine their output hypothesis or for prediction, these learning algorithms rely on the computation of a large number of dot products in the feature space F.

A solution to this problem is the so-called *kernel method*. This consists of defining a function $K\colon X\times X\to\mathbb{R}$ called a *kernel*, such that the value it associates to two examples $\mathbf{x}$ and $\mathbf{y}$ in input space, $K(\mathbf{x},\mathbf{y})$, coincides with the dot product of their images $\mathbf{\Phi}(\mathbf{x})$ and $\mathbf{\Phi}(\mathbf{y})$ in feature space. K is often viewed as a similarity measure:

$$\forall \mathbf{x},\mathbf{y}\in X,\quad K(\mathbf{x},\mathbf{y})=\mathbf{\Phi}(\mathbf{x})^\top\mathbf{\Phi}(\mathbf{y}). \tag{1}$$

A crucial advantage of K is efficiency: there is no need anymore to define and explicitly compute $\mathbf{\Phi}(\mathbf{x})$, $\mathbf{\Phi}(\mathbf{y})$, and $\mathbf{\Phi}(\mathbf{x})^\top\mathbf{\Phi}(\mathbf{y})$. Another benefit of K is flexibility: K can be arbitrarily chosen so long as the existence of $\mathbf{\Phi}$ is guaranteed, a condition that holds when K verifies Mercer's condition. This condition is important to guarantee the convergence of training for algorithms such as SVMs. In the discrete case, it is equivalent to K being PDS.

One of the most widely used two-group classification algorithm is SVMs [6]. The version of SVMs without offsets is defined via the following convex optimization problem for a training sample of m points $\mathbf{x}_i\in X$ with labels $y_i\in\{1,-1\}$:

$$\min_{\mathbf{w},\boldsymbol{\xi}}\frac{1}{2}\mathbf{w}^2+C\sum_{i=1}^m\xi_i\quad\text{s.t.}\quad y_i\mathbf{w}^\top\mathbf{\Phi}(\mathbf{x}_i)\geq1-\xi_i\quad\forall i\in[1,m],$$

where the vector $\mathbf{w}$ defines a hyperplane in the feature space, $\boldsymbol{\xi}$ is the m-dimensional vector of slack variables, and $C\in\mathbb{R}_+$ is a trade-off parameter. The problem is typically solved by introducing Lagrange multipliers $\boldsymbol{\alpha}\in\mathbb{R}^m$ for the set of constraints. The standard dual optimization for SVMs can be written as the convex optimization problem:

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

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 $\mathbf{K}$: $\mathbf{Q}_{ij}=y_iy_j\mathbf{K}_{ij}=y_iy_j\mathbf{\Phi}(\mathbf{x}_i)^\top\mathbf{\Phi}(\mathbf{x}_j)$, $i,j\in[1,m]$. Expressed with the dual variables, the solution vector $\mathbf{w}$ can be written as $\mathbf{w}=\sum_{i=1}^m\alpha_iy_i\mathbf{\Phi}(\mathbf{x}_i)$.

3.2 Coordinate Descent Solution for SVM Optimization

A straightforward way to solve the convex dual SVM problem is to use a coordinate descent method and to update only one coordinate α_i at each iteration, see [10]. The

SVMCOORDINATEDESCENT$((\mathbf{x}_i)_{i\in[1,m]})$

$\boldsymbol{\alpha} \leftarrow \mathbf{0}$
while $\boldsymbol{\alpha}$ not optimal **do**
  for $i \in [1, m]$ **do**
    $g \leftarrow y_i \mathbf{x}_i^\top \mathbf{w} - 1$ and $\alpha_i' \leftarrow \min(\max(\alpha_i - \frac{g}{\mathbf{Q}_{ii}}, 0), C)$
    $\mathbf{w} \leftarrow \mathbf{w} + (\alpha_i' - \alpha_i)\mathbf{x}_i$ and $\alpha_i \leftarrow \alpha_i'$
return $\mathbf{w}$

Fig. 2. Coordinate descent solution for SVM.

optimal step size $\beta^\star$ corresponding to the update of α_i is obtained by solving

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

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

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

If $\mathbf{Q}_{ii} = \boldsymbol{\Phi}(\mathbf{x}_i)^\top \boldsymbol{\Phi}(\mathbf{x}_i) = 0$, then $\boldsymbol{\Phi}(\mathbf{x}_i) = \mathbf{0}$ and $\mathbf{Q}_i = \boldsymbol{e}_i^\top \mathbf{Q} = \mathbf{0}$. 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 and update is given by

$$\beta^\star = \begin{cases} \beta_0, & \text{if } -\alpha_i \leq \beta_0 \leq C - \alpha_i, \\ -\alpha_i, & \text{if } \beta_0 \leq -\alpha_i, \\ C - \alpha_i, & \text{otherwise} \end{cases} \quad \text{and} \quad \alpha_i \leftarrow \min\left(\max\left(\alpha_i - \frac{\mathbf{Q}_i^\top \boldsymbol{\alpha} - 1}{\mathbf{Q}_{ii}}, 0\right), C\right).$$

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 feature space, computing $\mathbf{Q}_i$ is in the $O(mN)$, and hence the cost of each update is in $O(mN)$.

The choice of the coordinate α_i to update is based on the gradient. The gradient of the objective function is $\nabla F(\boldsymbol{\alpha}) = \mathbf{Q}\boldsymbol{\alpha} - \mathbf{1}$. At a cost in $O(mN)$ it can be updated via

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

Hsieh et al. [10] 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

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

SVMRATIONALKERNELS$((\mathbf{\Phi}'_i)_{i\in[1,m]})$

$\boldsymbol{\alpha} \leftarrow \mathbf{0}$
while $\boldsymbol{\alpha}$ not optimal **do**
  for $i \in [1, m]$ **do**
    $g \leftarrow \mathrm{D}(\mathbf{\Phi}'_i \circ \mathbf{W}') - 1$ and $\alpha'_i \leftarrow \min(\max(\alpha_i - \frac{g}{\mathbf{Q}_{ii}}, 0), C)$
    $\mathbf{W}' \leftarrow \mathbf{W}' + (\alpha'_i - \alpha_i)\mathbf{\Phi}'_i$ and $\alpha_i \leftarrow \alpha'_i$
return $\mathbf{W}'$

Fig. 3. Coordinate descent solution for rational kernels.

Maintaining the gradient $\nabla F(\boldsymbol{\alpha})$ is however still costly. The jth component of the gradient can be expressed as follows:

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

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 [10] or sequentially update the α_is. Hsieh et al. [10] also showed that it is possible to use the chunking method of [11] in conjunction with such heuristics. Using the results from [16], [10] showed that the resulting coordinate descent algorithm, SVMCOORDINATEDESCENT (Figure 2) converges to the optimal solution with a linear or faster convergence rate.

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.

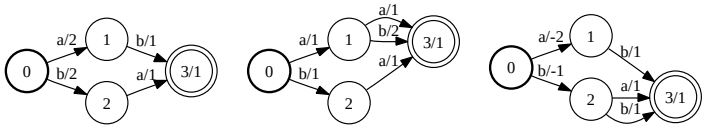
For rational kernels, the input "vectors" $\mathbf{x}_i$ are sequences, or distributions over sequences, and the expression $\sum_{j=1}^{m} y_j \alpha_j \mathbf{x}_j$ can be interpreted as a weighted regular expression. For any $i \in [1, m]$, let $\mathbf{X}_i$ be a simple weighted automaton representing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

$$\begin{aligned} \mathbf{Q}_i^\top \boldsymbol{\alpha} &= \sum_{j=1}^{m} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)\alpha_j = \sum_{j=1}^{m} y_i y_j \, \mathrm{D}(\mathbf{X}_i \circ \mathbf{U} \circ \mathbf{X}_j)\alpha_j \\ &= \mathrm{D}(y_i \mathbf{X}_i \circ \mathbf{U} \circ \sum_{j=1}^{m} y_j \alpha_j \mathbf{X}_j) = \mathrm{D}(y_i \mathbf{X}_i \circ \mathbf{U} \circ \mathbf{W}). \end{aligned} \tag{2}$$

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

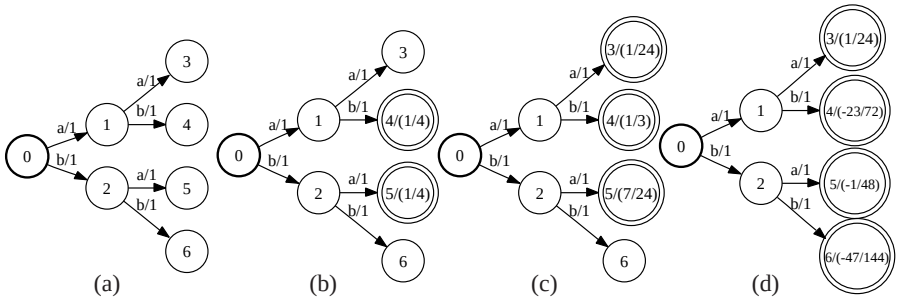
i	$\mathbf{x}_i$	y_i	$\mathbf{Q}_{ii}$
1	$ababa$	$+1$	8
2	$abaab$	$+1$	6
3	$abbab$	-1	6

(a)



(b) (c) (d)

Fig. 4. (a) Example dataset. (b-d) The automata $\mathbf{\Phi}'_i$ corresponding to the dataset of (a) when using a bigram kernel. The given $\mathbf{\Phi}'_i$ and $\mathbf{Q}_{ii}$'s assume the use of a bigram kernel.



(a) (b) (c) (d)

Fig. 5. Evolution of $\mathbf{W}'$ through the first iteration of SVMRATIONALKERNELS on the dataset from Figure 4.

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 the output labels as described in Section 2. Then

$$\begin{aligned}\mathbf{Q}_i^\top \boldsymbol{\alpha} &= \mathrm{D}\left(y_i\mathbf{X}_i \circ \mathbf{T} \circ \mathbf{T}^{-1} \circ \mathbf{W}\right) = \mathrm{D}((y_i\mathbf{X}_i \circ \mathbf{T}) \circ (\mathbf{W} \circ \mathbf{T})^{-1}) \\ &= \mathrm{D}\left(\Pi_2(y_i\mathbf{X}_i \circ \mathbf{T}) \circ \Pi_2(\mathbf{W} \circ \mathbf{T})\right) = \mathrm{D}(\mathbf{\Phi}'_i \circ \mathbf{W}'), \end{aligned} \tag{3}$$

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 update rule:

$$\mathbf{W}' \leftarrow \mathbf{W}' + \Delta(\alpha_i)\mathbf{\Phi}'_i. \tag{4}$$

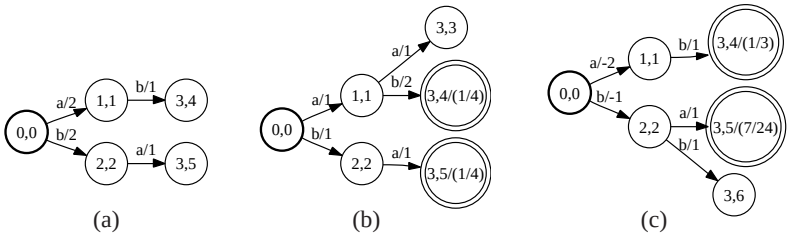


Fig. 6. The automata $\mathbf{\Phi}'_i \circ \mathbf{W}'$ during the first iteration of SVMRATIONALKERNELS on the data in Figure 4.

Table 1. First iteration of SVMRATIONALKERNELS on the dataset given Figure 4. 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. 5(a)	Fig. 6(a)	0	$\frac{1}{8}$
2	$(\frac{1}{8},0,0)$	Fig. 5(b)	Fig. 6(b)	$\frac{3}{4}$	$\frac{1}{24}$
3	$(\frac{1}{8},\frac{1}{24},0)$	Fig. 5(c)	Fig. 6(c)	$-\frac{23}{24}$	$\frac{47}{144}$
	$(\frac{1}{8},\frac{1}{24},\frac{47}{144})$	Fig. 5(d)			

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

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

Using (3) to compute the gradient and (4) to update $\mathbf{W}'$, we can generalize Algorithm SVMCOORDINATEDESCENT of Figure 2 and obtain Algorithm SVMRATIONALKERNELS of Figure 3. It follows from [16] that this algorithm converges at least linearly towards a global optimal solution. Moreover, the heuristics used by [10] and mentioned in the previous section can also be applied here to empirically improve the convergence rate of the algorithm. Table 1 shows the first iteration of SVMRATIONALKERNELS on the dataset given by Figure 4 when using a bigram kernel.

5 Implementation and Analysis

A key factor in analyzing the complexity of SVMRATIONALKERNELS 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 SVMRATIONALKERNELS. 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 $\mathbf{A}$.

A straightforward choice follows directly from the definition of $\mathbf{W}'$. $\mathbf{W}'$ is represented as a non-deterministic weighted automaton, $\mathbf{W}' = \sum_{i=1}^{m} \alpha_i \mathbf{\Phi}'_i$, 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

Table 2. 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)	(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	$O(\lvert\mathbf{W}'_m\rvert)$

$|\mathbf{W}'| = m + \sum_{i=1}^{m} |\mathbf{\Phi}'_i|$. The benefit of this representation is that the update of $\boldsymbol{\alpha}$ using (4) 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 (3) is in $O(|\mathbf{\Phi}'_j||\mathbf{W}'|) = O(|\mathbf{\Phi}'_j| \sum_{i=1}^{m} |\mathbf{\Phi}'_i|)$.

Representing $\mathbf{W}'$ as a deterministic weighted trie can lead to a simple update using (4). 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 $O\left(n(\mathbf{\Phi}'_i) l(\mathbf{\Phi}'_i)\right)$ for computing $\mathbf{\Phi}'_i \circ \mathbf{W}'$ and thus for computing the gradient using (3). 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 $O\left(n(\mathbf{\Phi}'_i)\right)$.

The drawback of a trie representation 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 (3) is then in $O(|\mathbf{\Phi}'_i \circ \mathbf{W}'|)$ which is significantly less than the $O\left(n(\mathbf{\Phi}'_i) l(\mathbf{\Phi}'_i)\right)$ complexity of the trie representation. Performing the update of $\mathbf{W}'$ using (4) can be more costly though. With the straightforward approach of using the general union, weighted determinization and minimization algorithms [5], 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 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 [7, 3], and the complexity of each addition of a string using these algorithms is only linear in the length of the string added.

Table 2 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 $|\mathbf{W}'_t|$ the number of distinct n-grams of order less than or equal to k in the dataset.

Table 3. Time for training an SVM classifier using an SMO-like algorithm and SVMRATIONALKERNELS using a trie representation for $\mathbf{W}'$, and size of $\mathbf{W}'$ (number of transitions) when representing $\mathbf{W}'$ as a deterministic weighted trie and a minimal deterministic weighted automaton.

Dataset	Kernel	SMO-like	New Algo.	trie	min. aut.
Reuters	4-gram	2m 18s	25s	66,331	34,785
(subset)	5-gram	3m 56s	30s	154,460	63,643
	6-gram	6m 16s	41s	283,856	103,459
	7-gram	9m 24s	1m 01s	452,881	157,390
	10-gram	25m 22s	1m 53s	1,151,217	413,878
	gappy 3-gram	10m 40s	1m 23s	103,353	66,650
	gappy 4-gram	58m 08s	7m 42s	1,213,281	411,939
Reuters	4-gram	618m 43s	16m 30s	242,570	106,640
(full)	5-gram	>2000m	23m 17s	787,514	237,783
	6-gram	>2000m	31m 22s	1,852,634	441,242
	7-gram	>2000m	37m 23s	3,570,741	727,743

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 (`http://www.daviddlewis.com/resources/`). We refer by *full dataset* to the $12{,}902$ news stories part of the ModeApte split. Since our goal is only to test speed (and not accuracy), we train on training and test sets combined. 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 [8] implemented using LIBSVM [4] and modified to handle the on-demand computation of rational kernels; and (b) SVMRATIONALKERNELS implemented using a trie representation for $\mathbf{W}'$. Table 3 reports the training time observed using a dual-core 2.2 GHz AMD Opteron workstation with 16GB of RAM, 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, we applied the weighted minimization algorithm to the tries output by SVMRATIONALKERNELS (after shifting the weights to the non-negative domain) and observed the resulting reduction in size. The results reported in Table 3 show that representing $\mathbf{W}'$ by a minimal deterministic automaton can lead to very significant savings in space and a substantial reduction of the training time with respect to the trie representation using 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. Our software will be made publicly available through an open-source project. Remarkably, our training algorithm for SVMs is entirely based on weighted automata algorithms and requires no specific solver.

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. R. C. Carrosco and M. L. Forcada. Incremental construction and maintenance of minimal finite-state automata. *Computational Linguistics*, 28(2):207–216, 2002.
4. C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001.
5. C. Cortes, P. Haffner, and M. Mohri. Rational Kernels: Theory and Algorithms. *JMLR*, 2004.
6. C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3), 1995.
7. 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.
8. 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.
9. S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2002.
10. 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.
11. T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods: Support Vector Learning*. The MIT Press, 1998.
12. Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*. Number 5 in EATCS Monographs on Theoretical Computer Science. Springer, New York, 1986.
13. S. Kumar, M. Mohri, and A. Talwalkar. On sampling-based approximate spectral decomposition. In *ICML*, 2009.
14. 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.
15. H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *JMLR*, 2, 2002.
16. 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.
17. Mehryar Mohri. Weighted automata algorithms. In *Handbook of Weighted Automata*, pages 213–254. Springer, 2009.
18. A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer, 1978.
19. John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge Univ. Press, 2004.
20. 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.
21. C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *NIPS*, pages 682–688, 2000.