International Journal of Foundations of Computer Science
© World Scientific Publishing Company

A Dual Coordinate Descent Algorithm for SVMs Combined with Rational 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 of Mathematical Sciences,
251 Mercer Street, New York, NY 10012, US,
and
Google Research,
76 Ninth Avenue, New York, NY 10011, US.
mohri@cs.nyu.edu

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 [23]. More recently, they have found other important applications in machine learning [7, 1]: they can be used to define a family of sequence kernels, *rational kernels* [7], 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) [9] to create powerful classification techniques, or with other learning algorithms to design regression, ranking, clustering, or dimensionality reduction solutions [25]. 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., [12, 2, 27, 18] or to use early stopping for optimization algorithms [26]. 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 [15] 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 [15] for linear SVMs. Section 5 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 6. In section 7, 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 [17]. In this paper, we only consider weighted transducers over the *real semiring* $(\mathbb{R}_+, +, \times, 0, 1)$. Figure 1(a) shows an example. 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 (resp. output) label of an accepting path is obtained by concatenating together the input (resp.

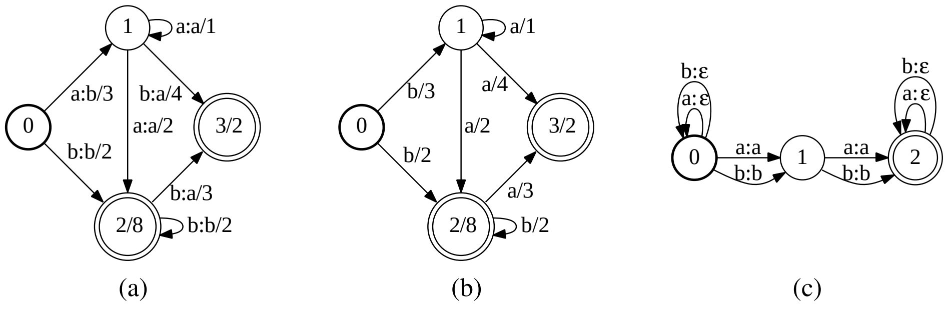

Fig. 1. (a) Example of weighted transducer $\mathbf{U}$. (b) Example of weighted automaton $\mathbf{A}$. In this example, $\mathbf{A}$ can be obtained from $\mathbf{U}$ by projection on the output and $\mathbf{U}(aab, baa) = \mathbf{A}(baa) = 3 \times 1 \times 4 \times 2 + 3 \times 2 \times 3 \times 2$. (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.

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 (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 $\mathbf{U}(\mathbf{x}, \mathbf{y})$. For example, the transducer of Figure 1(a) associates the weight 60 to the pair (aab, baa) since there are two accepting paths labeled with input aab and output baa: one with weight 24 and another one with weight 36.

A *weighted automaton* $\mathbf{A}$ can be defined as a weighted transducer with identical input and output labels. 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. Discarding the input labels of a weighted transducer $\mathbf{U}$ results in a weighted automaton $\mathbf{A}$, said to be the *output projection of* $\mathbf{U}$, and denoted by $\mathbf{A} = \Pi_2(\mathbf{U})$. 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 [24]: for any pair of strings $(\mathbf{x}, \mathbf{y})$ and real number γ,

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

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:

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

is well-defined and in $\mathbb{R}$ for all $\mathbf{x}, \mathbf{y}$ [24]. 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 $\mathbf{U}$. In the following, we shall use the distributivity of $+$ and multiplication by a real number, γ, over the composition of weighted transducers:

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

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

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

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, the following properties hold for all $\gamma \in \mathbb{R}$ and any weighted transducers $(\mathbf{U}_i)_{i \in [1,m]}$ and $\mathbf{U}$:

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

2.2. *Rational kernels*

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 $(K(\mathbf{x}_i, \mathbf{x}_j))_{1 \le i,j \le m}$ is symmetric and positive semi-definite (PSD) for any choice of m points in X [3]. A kernel between sequences $K\colon \Sigma^* \times \Sigma^* \to \mathbb{R}$ is *rational* [7] if there exists a weighted transducer $\mathbf{U}$ such that K coincides with the function defined by $\mathbf{U}$, that is

$$K(\mathbf{x}, \mathbf{y}) = \mathbf{U}(\mathbf{x}, \mathbf{y}) \tag{4}$$

for all $\mathbf{x}, \mathbf{y} \in \Sigma^*$. As shown by [7], when there exists a weighted transducer $\mathbf{T}$ such that $\mathbf{U}$ can be decomposed as $\mathbf{U} = \mathbf{T} \circ \mathbf{T}^{-1}$, K is PDS. All the sequence kernels seen in practice are precisely PDS rational kernels of this form [13, 19, 21, 28, 6, 8].

A standard family of rational kernels is n-gram kernels, see e.g. [21, 20]. 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).

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 $\mathbf{X}$ and $\mathbf{Y}$. We 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 apply similarly to large-scale training with kernels between lattices.

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 $2{\times}10^{15}$.

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:

$$\forall \mathbf{x}, \mathbf{y} \in X, \quad K(\mathbf{x}, \mathbf{y}) = \mathbf{\Phi}(\mathbf{x})^\top \mathbf{\Phi}(\mathbf{y}). \tag{6}$$

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 [9]. 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 $y_i \in \{1, -1\}$:

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

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:

$$\min_{\boldsymbol{\alpha}} \quad F(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{Q}\boldsymbol{\alpha} - \mathbf{1}^\top \boldsymbol{\alpha} \quad \text{s.t.} \quad \mathbf{0} \leq \boldsymbol{\alpha} \leq C, \tag{8}$$

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 $\mathbf{K}$:

$$\mathbf{Q}_{ij} = y_i y_j \mathbf{K}_{ij} = y_i y_j \mathbf{\Phi}(\mathbf{x}_i)^\top \mathbf{\Phi}(\mathbf{x}_j), \text{ for } i, j \in [1, m].$$

Expressed with the dual variables, the solution vector $\mathbf{w}$ can be written as

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{\Phi}(\mathbf{x}_i).$$

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 [15]. The optimal step size $\beta^\star$ corresponding to the update of α_i is obtained by solving

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

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 $\mathbf{Q}_{ii} = \mathbf{\Phi}(\mathbf{x}_i)^\top \mathbf{\Phi}(\mathbf{x}_i) = 0$, then $\mathbf{\Phi}(\mathbf{x}_i) = \mathbf{0}$ 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:

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

Fig. 2. Coordinate descent solution for SVM.

$\alpha_i \leftarrow 0$. 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

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

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 $O(mN)$, 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. [15] observed that when the kernel is linear, that is when $\mathbf{\Phi}(\mathbf{x}) = \mathbf{x}$, $\mathbf{Q}_i^\top \boldsymbol{\alpha}$ 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 $\nabla F(\boldsymbol{\alpha})$ is however still costly. The jth component of the gradient can be expressed as follows:

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

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 [15] or sequentially update the α_is. Hsieh et al. [15] also showed that it is possible to use the chunking method of [16] in conjunction with such heuristics. Using the results from [22], [15] showed that the coordinate descent algorithm with sequential update, SVMCoordinateDescent (Figure 2), converges to the optimal solution with a linear or faster convergence rate.

In the next section, we present an analysis of the convergence of the coordinate descent solution just discussed in terms of the properties of the kernel matrix $\mathbf{Q}$.

4. Convergence Guarantees for Coordinate Descent Algorithm

This section gives an explicit convergence guarantee for the coordinate descent algorithm SVMCoordinateDescent of Figure 2.

Let $\boldsymbol{\alpha}^r$ denote the value of $\boldsymbol{\alpha}$ after r updates following the coordinate descent algorithm iterating sequentially over the training set. A full iteration of the algorithm over the full training set consists of m updates of $\boldsymbol{\alpha}$ hence $\boldsymbol{\alpha}^{km}$ is the value of $\boldsymbol{\alpha}$ after k iterations over the full training set. Let $\lambda_{\max}(\mathbf{Q})$ and $\lambda_{\min}^+(\mathbf{Q})$ denote the largest eigenvalue and the smallest non-zero eigenvalue of $\mathbf{Q}$. The following is the main result of this section.

Theorem 1. *There exists an optimal solution $\boldsymbol{\alpha}^*$ of (8), a constant $\eta > 1$ and $r_0 \in \mathbb{N}$ such that for all $r \geq r_0$,*

$$F(\boldsymbol{\alpha}^{r+m}) - F(\boldsymbol{\alpha}^*) \leq \left(1 - \frac{1}{\eta}\right)\big(F(\boldsymbol{\alpha}^r) - F(\boldsymbol{\alpha}^*)\big) \tag{20}$$

with

$$\eta = \frac{2m}{\min_{\mathbf{Q}_{ii} \neq 0} \mathbf{Q}_{ii}} \left(\sqrt{2} + \frac{1}{\sqrt{\lambda_{\min}^+(\mathbf{Q})}} + \sqrt{\lambda_{\max}(\mathbf{Q})}\right)^6. \tag{21}$$

Theorem 1 implies that an ϵ-accurate solution $\boldsymbol{\alpha}$, that is such that $F(\boldsymbol{\alpha}) \leq F(\boldsymbol{\alpha}^*) + \epsilon$, can be obtained after $O(\log(1/\epsilon))$ iterations over the training set. It further gives an explicit expression for the bound in terms of quantities depending on the kernel matrix. Observe that the closer η is to 1, the faster is the convergence of the algorithm. This implies that a large $\lambda_{\max}(\mathbf{Q})$, a large condition number $\mathrm{cond}(\mathbf{Q}) = \lambda_{\max}(\mathbf{Q})/\lambda_{\min}^+(\mathbf{Q})$ and a small $\min_{\mathbf{Q}_{ii} \neq 0} \mathbf{Q}_{ii}$ would result in a slow convergence.

When the kernel used is normalized, as in the case of the widely used Gaussian kernels, the expression of η given by (38) can be significantly simplified. Indeed, in that case, every non-zero diagonal entry $\mathbf{Q}_{ii}$ is equal to 1, $\lambda_{\max}(\mathbf{Q}) \geq 1$, and $\lambda^{+}_{\min}(\mathbf{Q}) \leq 1$. This implies that

$$\begin{aligned}
\eta &= \frac{2m\lambda_{\max}(\mathbf{Q})^3}{\min\limits_{\mathbf{Q}_{ii}\neq 0} \mathbf{Q}_{ii}} \left(\frac{\sqrt{2}}{\sqrt{\lambda_{\max}(\mathbf{Q})}} + \frac{1}{\sqrt{\lambda_{\max}(\mathbf{Q})\lambda^{+}_{\min}(\mathbf{Q})}} + 1 \right)^2 (1 + \frac{2}{\lambda_{\max}(\mathbf{Q})})^2 \\
&\leq 2m\lambda_{\max}(\mathbf{Q})^3 \left(\sqrt{2} + \frac{1}{\sqrt{\lambda^{+}_{\min}(\mathbf{Q})}} + 1 \right)^2 (1+2)^2 \\
&\leq 18m\lambda_{\max}(\mathbf{Q})^3 \left(\frac{\sqrt{2}+2}{\sqrt{\lambda^{+}_{\min}(\mathbf{Q})}} \right)^2 \leq 210\, m \frac{\lambda_{\max}(\mathbf{Q})^3}{\lambda^{+}_{\min}(\mathbf{Q})} = 210\, m\lambda_{\max}(\mathbf{Q})^2 \operatorname{cond}(\mathbf{Q}).
\end{aligned}$$

Thus, in that case, we can replace the expression of η in the statement of the theorem by $210\, m\lambda_{\max}(\mathbf{Q})^2 \operatorname{cond}(\mathbf{Q})$.

Our analysis of the convergence of Algorithm SVMCoordinateDescent is based on results by Luo and Tseng [22] on the convergence of the coordinate descent method. In [22], the authors considered the following convex optimization problem:

$$\min_{\boldsymbol{\alpha}} H(\boldsymbol{\alpha}) = G(\mathbf{E}\boldsymbol{\alpha}) + \mathbf{b}^{\top}\boldsymbol{\alpha} \quad \text{s.t.} \quad \boldsymbol{\alpha} \in \mathcal{A} \tag{22}$$

where (i) $\mathcal{A}$ is a possibly unbounded box of $\mathbb{R}^m$, (ii) H and G are proper closed convex functions respectively in $\mathbb{R}^m$ and $\mathbb{R}^N$, (iii) $\mathbf{E}$ is a $N \times m$ matrix with no zero column. The authors showed that assuming that (iv) the set $\mathcal{A}^*$ of optimal solutions in $\mathcal{A}$ is non empty, (v) the domain of G is open and G is strictly convex twice differentiable on its domain and (vi) $\nabla^2 G(\mathbf{E}\boldsymbol{\alpha}^*)$ is positive definite for all $\boldsymbol{\alpha}^* \in \mathcal{A}^*$, then the coordinate descent algorithm with sequential update converges to an optimal solution $\boldsymbol{\alpha}^* \in \mathcal{A}^*$ with a convergence rate at least linear. [22] showed that the sequence $(\boldsymbol{\alpha}^r)_{r\in\mathbb{N}}$ converges to an optimal solution $\boldsymbol{\alpha}^*$.

Theorem 2 ([22]) *There exists an optimal solution $\boldsymbol{\alpha}^*$ of (22), a constant $\eta > 1$ and $r_0 \in \mathbb{N}$ such that for all $r \geq r_0$,*

$$H(\boldsymbol{\alpha}^{r+m}) - H(\boldsymbol{\alpha}^*) \leq \left(1 - \frac{1}{\eta}\right)\big(H(\boldsymbol{\alpha}^r) - H(\boldsymbol{\alpha}^*)\big).$$

In [22], the authors showed that the constant η can be expressed as follows:

$$\eta = \rho\omega^2 m / (\sigma \min_i \|\mathbf{E}_i\|^2) \tag{24}$$

where σ and ρ only depend on G and $\omega = \kappa(2 + \|\mathbf{E}\|^2)$. [22] does not give an explicit expression of κ but we will show that it can be expressed as a function of $\|\mathbf{E}\|$, ρ and θ where θ is a constant depending only on $\mathbf{E}$.

The existence of the constants ρ and σ follows from the following observation. The assumptions made by [22] on G imply that there exists a closed ball $\mathcal{U}^*$ around $\mathbf{E}\boldsymbol{\alpha}^*$ and included in the domain of G and two constants σ and ρ such that for all $\mathbf{z}$ and $\mathbf{w}$ in $\mathcal{U}^*$:

$$(\nabla G(\mathbf{z}) - \nabla G(\mathbf{w}))^\top (\mathbf{z} - \mathbf{w}) \geq 2\sigma \|\mathbf{z} - \mathbf{w}\|^2, \tag{25}$$

$$\|\nabla G(\mathbf{z}) - \nabla G(\mathbf{w})\| \leq \rho \|\mathbf{z} - \mathbf{w}\|. \tag{26}$$

The existence of the constant θ comes from the following result from Hoffman [14].

Lemma 3 ([14]) *Let* $\mathbf{B}$ *be any* $k \times n$ *matrix. Then, there exists a constant* $\theta > 0$ *depending only on* $\mathbf{B}$ *such that, for any* $\bar{\boldsymbol{\alpha}} \in \mathcal{A}$ *and any* $\mathbf{d} \in \mathbb{R}^k$ *such that the linear system* $\mathbf{B}\boldsymbol{\beta} = \mathbf{d}$, $\boldsymbol{\beta} \in \mathcal{A}$ *is consistent, there exists a point* $\bar{\boldsymbol{\beta}}$ *satisfying* $\mathbf{B}\bar{\boldsymbol{\beta}} = \mathbf{d}$, $\bar{\boldsymbol{\beta}} \in \mathcal{A}$, *with*

$$\|\bar{\boldsymbol{\alpha}} - \bar{\boldsymbol{\beta}}\| \leq \theta \|\mathbf{B}\bar{\boldsymbol{\alpha}} - \mathbf{d}\|.$$

Let $\mathcal{A} = \{\boldsymbol{\alpha} \in \mathbb{R}^m \mid \mathbf{l} \leq \boldsymbol{\alpha} \leq \mathbf{u}\}$ with $\mathbf{l} \in [-\infty, +\infty)^m$ and $\mathbf{u} \in (-\infty, +\infty]^m$ and let $[\boldsymbol{\alpha}]^+$ denotes the vector in $\mathbb{R}^m$ defined by having for i-th coordinate $\max(l_i, \min(x_i, u_i))$, for all $i \in [1, m]$. Lemma 3 is used by Luo and Tseng [22] to establish the existence of the constant κ.

Lemma 4 ([22]) *There exists a constant* $\kappa > 0$ *such that*

$$\|\mathbf{E}\boldsymbol{\alpha}^r - \mathbf{E}\boldsymbol{\alpha}^*\| \leq \kappa \|\boldsymbol{\alpha}^r - [\boldsymbol{\alpha}^r - \nabla H(\boldsymbol{\alpha}^r)]^+\|, \qquad \textit{for all } r \geq r_1.$$

The full proof of Lemma 4 is given in [22] (Lemma 4.4). The following lemma gives an expression of κ as a function of θ, ρ, σ and $\|E\|$.

Lemma 5. *We have that:*

$$\kappa \leq \frac{\theta + \|\mathbf{E}\|\rho}{2\sigma} + \frac{1}{\sqrt{\sigma}} \tag{29}$$

Proof. Let $\mathbf{t}^* = \mathbf{E}\boldsymbol{\alpha}^*$ and $\boldsymbol{\gamma}^r = [\boldsymbol{\alpha}^r - \nabla H(\boldsymbol{\alpha}^r)]^+$. It follows from Lemma 3 that there exists $\boldsymbol{\beta}^r$ in $\mathcal{A}$ such that $\|\boldsymbol{\alpha}^r - \boldsymbol{\beta}^r\| \leq \theta \|\mathbf{E}\boldsymbol{\alpha}^r - \mathbf{t}^*\|$ for all $r \geq r_1$. Hence, we have:

$$\|\boldsymbol{\gamma}^r - \boldsymbol{\beta}^r\| \leq \|\boldsymbol{\gamma}^r - \boldsymbol{\alpha}^r\| + \|\boldsymbol{\alpha}^r - \boldsymbol{\beta}^r\| \tag{30}$$

$$\leq \|\boldsymbol{\gamma}^r - \boldsymbol{\alpha}^r\| + \theta \|\mathbf{E}\boldsymbol{\alpha}^r - \mathbf{t}^*\|. \tag{31}$$

The next step in the proof of Lemma 4 is as follows. For any subset $I \subseteq [1, m]$, the authors define a set $\mathcal{R}_I$ such that for all $r \in \mathcal{R}_I$ such that $r \geq r_1$, the following inequalities hold:

$$2\sigma \|\mathbf{E}\boldsymbol{\alpha}^r - \mathbf{t}^*\|^2 \leq \|\boldsymbol{\alpha}^r - \boldsymbol{\gamma}^r\| \, \|\boldsymbol{\alpha}^r - \boldsymbol{\beta}^r\| + \|\nabla H(\boldsymbol{\alpha}^r) - \nabla H(\boldsymbol{\beta}^r)\| \, \|\boldsymbol{\alpha}^r - \boldsymbol{\gamma}^r\| \tag{32}$$

$$\leq (\|\boldsymbol{\alpha}^r - \boldsymbol{\beta}^r\| + \|\mathbf{E}\|\rho \|\mathbf{E}\boldsymbol{\alpha}^r - \mathbf{t}^*\|) \|\boldsymbol{\alpha}^r - \boldsymbol{\gamma}^r\| \tag{33}$$

$$\leq (\|\boldsymbol{\alpha}^r - \boldsymbol{\gamma}^r\| + \|\boldsymbol{\gamma}^r - \boldsymbol{\beta}^r\| + \|\mathbf{E}\|\rho \|\mathbf{E}\boldsymbol{\alpha}^r - \mathbf{t}^*\|) \|\boldsymbol{\alpha}^r - \boldsymbol{\gamma}^r\| \tag{34}$$

$$\leq (2\|\boldsymbol{\alpha}^r - \boldsymbol{\gamma}^r\| + (\theta + \|\mathbf{E}\|\rho) \|\mathbf{E}\boldsymbol{\alpha}^r - \mathbf{t}^*\|) \|\boldsymbol{\alpha}^r - \boldsymbol{\gamma}^r\| \tag{35}$$

Thus,

$$\|\mathbf{E}\boldsymbol{\alpha}^r - \mathbf{t}^*\|^2 \leq \frac{1}{\sigma} \|\boldsymbol{\alpha}^r - \boldsymbol{\gamma}^r\|^2 + \frac{\theta + \|\mathbf{E}\|\rho}{2\sigma} \|\mathbf{E}\boldsymbol{\alpha}^r - \mathbf{t}^*\| \, \|\boldsymbol{\alpha}^r - \boldsymbol{\gamma}^r\| \tag{36}$$

In [22], the expression of the constants are not explicitly derived for the last three inequalities. Also, their proof goes through a few extra steps that are not required. Indeed, we have obtained with (36) a second-degree polynomial inequality of the form $x^2 - axy - by^2 \leq 0$ with $x, y, a, b > 0$ which implies that $x \leq (ay + \sqrt{a^2y^2 + 4by^2})/2 \leq y(a + \sqrt{b})$. This leads us to:

$$\|\mathbf{E}\boldsymbol{\alpha}^r - \mathbf{t}^*\| \leq \left(\frac{\theta + \|\mathbf{E}\|\rho}{2\sigma} + \frac{1}{\sqrt{\sigma}}\right) \|\boldsymbol{\alpha}^r - \boldsymbol{\gamma}^r\| \tag{37}$$

Since the disjoint union of the $\mathcal{R}_I$'s is equal to $\mathbb{N}$ (see [22]), the inequality above holds for all $r \geq r_1$ and Lemma 5 follows. □

The previous lemmas can be used to give the proof of Theorem 1.

Proof of Theorem 1. The SVM objective function F coincides with H when $G(\boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{\beta}^\top\boldsymbol{\beta}$ for $\boldsymbol{\beta} \in \mathbb{R}^N$, $\mathbf{b} = \mathbf{1}$, $\mathbf{E}$ is the $N \times m$ matrix defined by $\mathbf{E} = (y_1\boldsymbol{\Phi}(\mathbf{x}_1), \ldots, y_m\boldsymbol{\Phi}(\mathbf{x}_m))$ and $\mathcal{A} = \{\boldsymbol{\alpha} \in \mathbb{R}^m | \ \mathbf{0} \leq \boldsymbol{\alpha} \leq C\}$. It is then clear that assumptions (i), (ii) and (v) hold. Assumption (iv) follows from Weierstrass' Theorem and assumption (vi) follow from the fact that $\mathbf{E}^\top\mathbf{E} = \mathbf{Q}$ is a PSD matrix.

If there exists some zero columns in $\mathbf{E}$, then the first iteration of the algorithm will set the corresponding α_is to 0 and subsequent iterations will leave these values unchanged, solving the sub-problem restricted to $\{i | \mathbf{E}_i \neq \mathbf{0}\}$. Hence we can assume without loss of generality that assumption (iii) holds.

Therefore, we can apply Theorem 2 to our problem. Since $\nabla G(\boldsymbol{\beta}) = \boldsymbol{\beta}$, it follows that $\mathcal{U}^* = \mathbb{R}^N$, $\sigma = 1/2$ and $\rho = 1$. Moreover, $\|\mathbf{E}_i\|^2 = \mathbf{E}_i^\top\mathbf{E}_i = \mathbf{Q}_{ii} = \mathbf{K}_{ii}$. Finally, we have that $\|\mathbf{E}\|^2 = \lambda_{\max}(\mathbf{E}^\top\mathbf{E}) = \lambda_{\max}(\mathbf{Q})$ and $\theta \leq 1/\sqrt{\lambda^+_{\min}(\mathbf{Q})}$. This leads to $\kappa \leq \sqrt{2} + 1/\sqrt{\lambda^+_{\min}(\mathbf{Q})} + \sqrt{\lambda_{\max}(\mathbf{Q})}$ and

$$\eta = \frac{2m(2 + \lambda_{\max}(\mathbf{Q}))^2}{\min\limits_{\mathbf{Q}_{ii} \neq 0} \mathbf{Q}_{ii}} \left(\sqrt{2} + \frac{1}{\sqrt{\lambda^+_{\min}(\mathbf{Q})}} + \sqrt{\lambda_{\max}(\mathbf{Q})}\right)^2. \tag{38}$$

Since the trace of $\mathbf{Q}$ is the sum of its eigenvalues, we have that $m\lambda_{\max}(\mathbf{Q}) \geq \mathrm{Tr}(\mathbf{Q}) = \sum_{i=1}^m \mathbf{Q}_{ii} \geq \min_i \mathbf{Q}_{ii}$ and hence

$$\frac{m\lambda_{\max}(\mathbf{Q})}{\min_i \mathbf{Q}_{ii}} \geq 1. \tag{39}$$

We also have

$$\lambda_{\max}(\mathbf{Q}) \left(\sqrt{2} + \frac{1}{\sqrt{\lambda^+_{\min}(\mathbf{Q})}} + \sqrt{\lambda_{\max}(\mathbf{Q})}\right)^2 > \frac{\lambda_{\max}(\mathbf{Q})}{\lambda^+_{\min}(\mathbf{Q})} \geq 1. \tag{40}$$

From (39) and (40) it follows that $\eta > 1$ and (20) then implies that $(\boldsymbol{\alpha}^r)_{r\in\mathbb{N}}$ converges to $\boldsymbol{\alpha}^*$. The simpler but less favorable expression of η given by (21) can be obtained as follows

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

Fig. 3. Coordinate descent solution for rational kernels.

from (38):

$$\begin{aligned}\eta &= \frac{2m(2+\lambda_{\max}(\mathbf{Q}))^2}{\min\limits_{\mathbf{Q}_{ii}\neq 0} \mathbf{Q}_{ii}} \left(\sqrt{2} + \frac{1}{\sqrt{\lambda^+_{\min}(\mathbf{Q})}} + \sqrt{\lambda_{\max}(\mathbf{Q})}\right)^2 \\ &\leq \frac{2m(\sqrt{2}+\sqrt{\lambda_{\max}(\mathbf{Q})})^4}{\min\limits_{\mathbf{Q}_{ii}\neq 0} \mathbf{Q}_{ii}} \left(\sqrt{2} + \frac{1}{\sqrt{\lambda^+_{\min}(\mathbf{Q})}} + \sqrt{\lambda_{\max}(\mathbf{Q})}\right)^2 \\ &\leq \frac{2m}{\min\limits_{\mathbf{Q}_{ii}\neq 0} \mathbf{Q}_{ii}} \left(\sqrt{2} + \frac{1}{\sqrt{\lambda^+_{\min}(\mathbf{Q})}} + \sqrt{\lambda_{\max}(\mathbf{Q})}\right)^6 . \end{aligned}$$

□

5. 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:

$$\begin{aligned}\mathbf{Q}_i^\top \boldsymbol{\alpha} &= \sum_{j=1}^m y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)\alpha_j = \sum_{j=1}^m y_i y_j \, \mathrm{D}(\mathbf{X}_i \circ \mathbf{U} \circ \mathbf{X}_j)\alpha_j \qquad (41) \\ &= \mathrm{D}(y_i \mathbf{X}_i \circ \mathbf{U} \circ \sum_{j=1}^m y_j \alpha_j \mathbf{X}_j) = \mathrm{D}(y_i \mathbf{X}_i \circ \mathbf{U} \circ \mathbf{W}).\end{aligned}$$

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

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

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

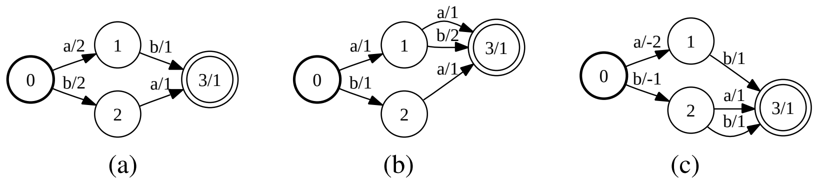

Fig. 4. The automata $\mathbf{\Phi}'_i$ corresponding to the dataset from Table 1 when using a bigram kernel.

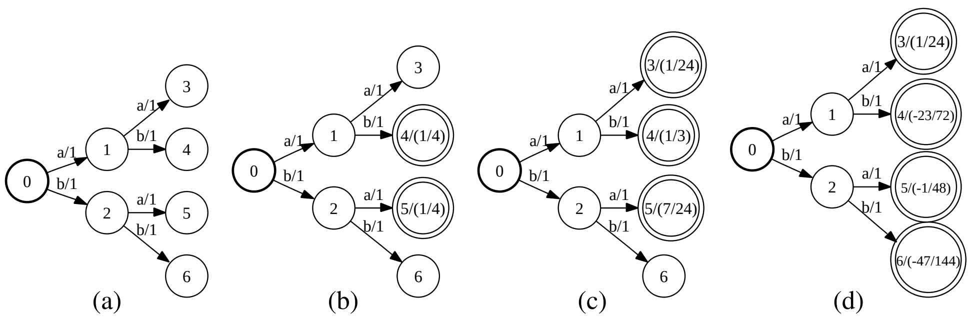

Fig. 5. Evolution of $\mathbf{W}'$ through the first iteration of SVMRationalKernels on the dataset from Table 1.

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

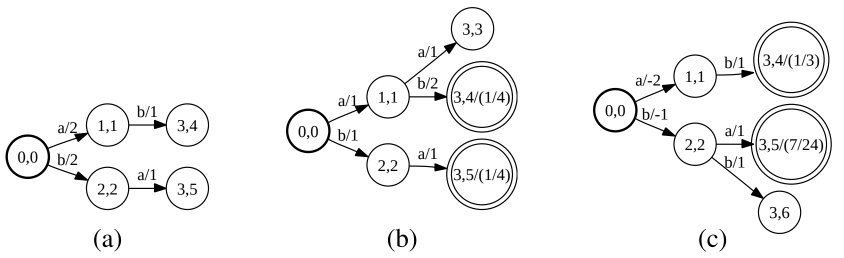

Fig. 6. The automata $\mathbf{\Phi}'_i \circ \mathbf{W}'$ during the first iteration of SVMRationalKernels on the data in Table 1.

Table 2. First iteration of SVMRationalKernels 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. 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)			

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

$$\begin{aligned} \mathbf{Q}_i^\top \boldsymbol{\alpha} &= \mathrm{D}\left(y_i \mathbf{X}_i \circ \mathbf{T} \circ \mathbf{T}^{-1} \circ \mathbf{W}\right) = \mathrm{D}((y_i \mathbf{X}_i \circ \mathbf{T}) \circ (\mathbf{W} \circ \mathbf{T})^{-1}) \\ &= \mathrm{D}\left(\Pi_2(y_i \mathbf{X}_i \circ \mathbf{T}) \circ \Pi_2(\mathbf{W} \circ \mathbf{T})\right) = \mathrm{D}(\mathbf{\Phi}'_i \circ \mathbf{W}'), \end{aligned} \tag{43}$$

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:

$$\mathbf{W}' \leftarrow \mathbf{W}' + \Delta(\alpha_i)\mathbf{\Phi}'_i. \tag{44}$$

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 (43) to compute the gradient and (44) to update $\mathbf{W}'$, we can generalize Algorithm SVMCoordinateDescent of Figure 2 and obtain Algorithm SVMRationalKernels of Figure 3. It follows from Theorem 1 and [22] that this algorithm converges at least linearly towards a global optimal solution. Moreover, the heuristics used by [15] 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 SVMRATIONALKERNELS on the dataset given by Table 1 when using a bigram kernel.

6. 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 $\mathbf{A}$.

6.1. *Naive representation of* $\mathbf{W}'$

A straightforward choice follows directly from the definition of $\mathbf{W}'$. $\mathbf{W}'$ is represented as a non-deterministic weighted automaton, $\mathbf{W}' = \sum_{j=1}^{m} \alpha_j \mathbf{\Phi}'_j$, with a single initial state and m outgoing ϵ-transitions, where the weight of the jth transition is α_j and its destination state is the initial state of $\mathbf{\Phi}'_j$. The size of this choice of $\mathbf{W}'$ is $|\mathbf{W}'| = m + \sum_{j=1}^{m} |\mathbf{\Phi}'_j|$. The benefit of this representation is that the update of $\boldsymbol{\alpha}$ using (44) 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 (43) is in $O(|\mathbf{\Phi}'_i||\mathbf{W}'|) = O(|\mathbf{\Phi}'_i| \sum_{j=1}^{m} |\mathbf{\Phi}'_j|)$. From an algorithmic point of view, using this naive representation of $\mathbf{W}'$ is equivalent to using (41) 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.

6.2. *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 (44). 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 (43). 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 $O\left(n(\mathbf{\Phi}'_i)\right)$.

6.3. *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 (43) 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. Perform-

Table 3. 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 ($\mathbf{W}'_n$)	$O(\lvert\mathbf{\Phi}'_i\rvert \sum_{j=1}^{m} \lvert\mathbf{\Phi}'_j\rvert)$	$O(1)$	$O(m)$
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	$O(\lvert\mathbf{W}'_m\rvert)$

ing the update of $\mathbf{W}'$ using (44) can be more costly though. With the straightforward approach of using the general union, weighted determinization and minimization algorithms [7, 23], 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 [10, 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 $|\mathbf{W}'_t|$ the number of distinct n-grams of order less than or equal to k in the dataset.

7. 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 [11] implemented using LIBSVM [5] and modified to handle the on-demand computation of rational kernels; and (b) SVM-RATIONALKERNELS implemented using a trie representation for $\mathbf{W}'$. Table 4 reports the training time observed using a dual-core 2.2 GHz AMD Opteron workstation with 16GB

Table 4. 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	Training Time		Size of W'	
		SMO-like	New Algo.	trie	min. aut.
Reuters (subset)	4-gram	2m 18s	25s	66,331	34,785
	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 (full)	4-gram	618m 43s	16m 30s	242,570	106,640
	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

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 as described in Section 6.3, 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 4 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 $\mathbf{W}'$.

8. 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] Cyril Allauzen, Mehryar Mohri, and Ameet Talwalkar. Sequence kernels for predicting protein essentiality. In *ICML 2008*, pages 9–16. ACM, 2008.

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

[3] Christian Berg, Jens Peter Reus Christensen, and Paul Ressel. *Harmonic Analysis on Semigroups*. Springer-Verlag: Berlin-New York, 1984.

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

[5] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

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

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

[8] Corinna Cortes and Mehryar Mohri. Moment kernels for regular distributions. *Machine Learning*, 60(1-3):117–134, 2005.

[9] Corinna Cortes and Vladimir N. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[10] Jan Daciuk, Stoyan Mihov, Bruce W. Watson, and Richard Watson. Incremental construction of minimal acyclic finite state automata. *Computational Linguistics*, 26(1):3–16, 2000.

[11] Rong-En Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training SVM. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

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

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

[14] Alan J. Hoffman. On approximate solutions of systems of linear inequalities. *Journal of Research of the National Bureau of Standards*, 49:263–265, 1952.

[15] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *ICML 2008*, pages 408–415. ACM, 2008.

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

[17] Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*. Number 5 in EATCS Monographs on Theoretical Computer Science. Springer, New York, 1986.

[18] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. On sampling-based approximate spectral decomposition. In *ICML 2009*. ACM, 2009.

[19] Christina Leslie and Rui Kuang. Fast String Kernels using Inexact Matching for Protein Sequences. *Journal of Machine Learning Research*, 5:1435–1455, 2004.

[20] Christina S. Leslie, Eleazar Eskin, and William Stafford Noble. The Spectrum Kernel: A String Kernel for SVM Protein Classification. In *Pacific Symposium on Biocomputing*, pages 566–575, 2002.

[21] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–44, 2002.

[22] Zhi-Quan Luo and Paul Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35, 1992.

[23] Mehryar Mohri. Weighted automata algorithms. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, chapter 6, pages 213–254. Springer, 2009.

[24] Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer, 1978.

[25] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge Univ. Press, 2004.

[26] Ivor W. Tsang, James T. Kwok, and Pak-Ming Cheung. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6:363–392, 2005.

[27] Christopher K. I. Williams and Matthias Seeger. Using the Nyström method to speed up kernel

machines. In *NIPS*, pages 682–688, 2000.

[28] 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):799–807, 2000.