FASTER CALCULATION OF STRING SUBSEQUENCE KERNEL

DAESEOK LEE

Abstract. We present a simple recursive formula for calculating the String Subsequence Kernel(SSK) introduced by Lodhi et al.. We reduced the time complexity from $O(n^2m)$ to $O(nm)$.

1. introduction

String Subsequence Kernel(SSK) was introduced by Lodhi et al.([LSST$^+$02]), as a measure that quantifies similarity between a pair of strings by looking for common subsequences. It has been applied in a wide variety of contexts including biological contexts and Natural Language Processing(NLP), in which it served as a basis of classification or clustering algorithms.

Let A be a finite set of characters and let $0 < \lambda \leq 1$. Then the string subsequence kernel between strings $s_1, s_2 \in A^*$ is defined as

$$K(s_1, s_2) = \sum_{t \in A^*} \sum_{\substack{s_1[\mathbf{i}_1]=t \\ s_2[\mathbf{i}_2]=t}} \lambda^{|\mathbf{i}_1|+|\mathbf{i}_2|}, \tag{1.1}$$

where $\mathbf{i}$ represents a multi-index and $|\mathbf{i}|$ represents the length of it (defined as (last index)-(first index)+1). For example, $K(aacb, adbb) = 4\lambda^2 + \lambda^6 + 2\lambda^7 + \lambda^8$, where the first term came from the common substrings $(a), (b)$ and the remaining terms all came from common substring (ab)(in different pairs of multi-indices).

One drawback of this particular kernel was that it required too much computation time. In fact, it required $O(|s| \cdot |t| \cdot \min(|s|, |t|))$ time, when computed as in [LSST$^+$02]. Consequently, many alternatives were propsed such as modification of SSK([SK07]) and cheaper kernels([SV03],[LK04]). Instead in this paper, we propose a simpler and faster dynamic programming method for computing SSK, which takes only $O(|s| \cdot |t|)$ time.

2. algorithm

Let A, λ as before and let $'*'$ be a character that does not appear in A. For our recursive algortihm, we introduce a function $I(s, t)$ that indicates the "marginal increment" of the kernel, which can be written as follows

$$I(s, t) = K(s +' *', t +' *') - K(s, t) - \lambda^2 \tag{2.1}$$

Without this marginal increment function, an effective recursive calculation is hard to be done since each common subsequence has different effect on the kernel when a new common character appears, because of the difference in the distance between the last index of it and the new position. The marginal increment function keeps track of the sum of these different effects, allowing a convenient increment of the

Date: March 20, 2018.

kernel. Our main observation is that the marginal increment function itself can be calculated recursively as well.

Theorem 2.1. *Let* $a_{i,j} = K(s[:\ i], t[:\ j])$, $b_{i,j} = I(s[:\ i], t[:\ j])$, *and* $\delta_{i,j} = \begin{cases} 1 & s[i] = t[j] \\ 0 & s[i] \neq t[j] \end{cases}$. *Then we have*

$$a_{i,j} = a_{i,j-1} + a_{i-1,j} - a_{i-1,j-1} + (b_{i-1,j-1} + \lambda^2)\delta_{i,j} \tag{2.2}$$

$$b_{i,j} = \lambda b_{i-1,j} + \lambda b_{i,j-1} - \lambda^2 b_{i-1,j-1} + (b_{i-1,j-1} + \lambda^2)\lambda^2\delta_{i,j} \tag{2.3}$$

In particular, the kernel $K(s,t)$ *can be obtained as* $a_{|s|,|t|}$ *after the iterative calculations.*

Proof. easy to check, from definition of K and I. □

3. DISCUSSION

Although our algorithm is an improvement over the previous one, it still requires quardratic time, which can be too long compared to linear time kernels such as that of [SV03]. However, there might be some cases where we would better consider the effect of common substrings of arbitrary length properly. In those cases, SSK might be a decent kernel, and our algorithm would help.

REFERENCES

[LK04] Christina Leslie and Rui Kuang. Fast string kernels using inexact matching for protein sequences. *Journal of Machine Learning Research*, 5(Nov):1435–1455, 2004.

[LSST+02] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2(Feb):419–444, 2002.

[SK07] Alexander K Seewald and Florian Kleedorfer. Lambda pruning: an approximation of the string subsequence kernel for practical svm classification and redundancy clustering. *Advances in Data Analysis and Classification*, 1(3):221–239, 2007.

[SV03] Alex J Smola and SVN Vishwanathan. Fast kernels for string and tree matching. In *Advances in neural information processing systems*, pages 585–592, 2003.