We will discuss string kernels today:

Text Classification using String Kernels

Huma Lodhi HUMA@CS.RHUL
Craig Saunders CRAIG@CS.RHUL
John Shawe-Taylor JOHN@CS.RHUL
Nello Cristianini NELLO@CS.RHUL
Chris Watkins CHRISW@CS.RHUL
Department of Computer Science,
Royal Holloway, University of London,
Egham, Surrey TW20 0EX, UK

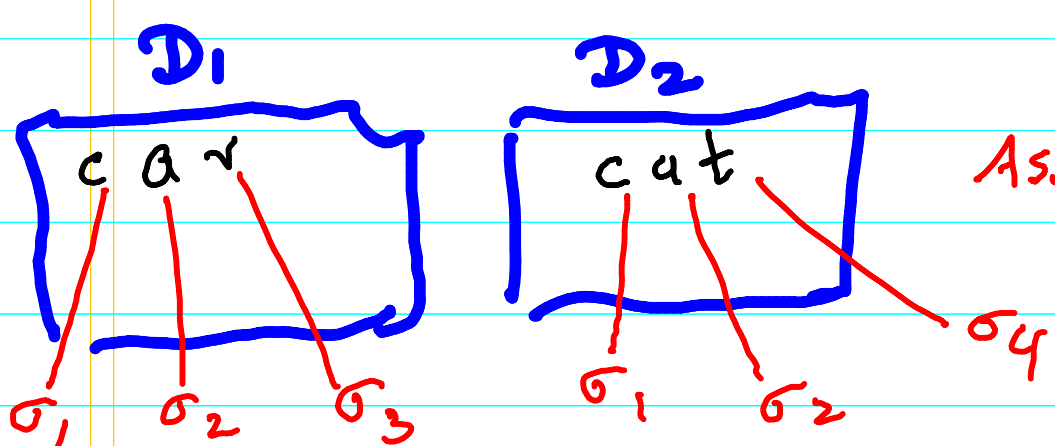
Editor: Bernhard Schölkopf

Definition 1 (String subsequence kernel- SSK) *Let Σ be a finite alphabet. A string is a finite sequence of characters from Σ, including the empty sequence. For strings s, t, we denote by $|s|$ the length of the string $s = s_1 \ldots s_{|s|}$, and by st the string obtained by concatenating the strings s and t. The string $s[i:j]$ is the substring $s_i \ldots s_j$ of s. We say that u is a subsequence of s, if there exist indices $\mathbf{i} = (i_1, \ldots, i_{|u|})$, with $1 \leq i_1 < \cdots < i_{|u|} \leq |s|$, such that $u_j = s_{i_j}$, for $j = 1, \ldots, |u|$, or $u = s[\mathbf{i}]$ for short. The length $l(\mathbf{i})$ of the subsequence in s is $i_{|u|} - i_1 + 1$. We denote by Σ^n the set of all finite strings of length n, and by Σ^* the set of all strings*

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n. \tag{1}$$

We now define feature spaces $F_n = \mathbb{R}^{\Sigma^n}$. The feature mapping ϕ for a string s is given by defining the u coordinate $\phi_u(s)$ for each $u \in \Sigma^n$. We define

$$\phi_u(s) = \sum_{\mathbf{i}: u = s[\mathbf{i}]} \lambda^{l(\mathbf{i})}, \tag{2}$$



Assume $\sigma_1, \sigma_2 .. \sigma_4$ are words (i.e. indivisible units or characters)

$\Sigma = \{\sigma_1 .. \sigma_4\}$ is the vocabulary

You could generate sequences of characters from Σ

Eg: $S_1 = c - a$ $\qquad S_5 = a - a$

$S_2 = c \_ t$

$S_3 = r \_ t$ $\qquad \vdots$

$S_4 = c \_ c$

Goal: 1 Define feature ϕ so as to measure similarity between D_1 & D_2 in the space of sequences. $S \in \Sigma^*$

2 Want to compute $\phi^T \phi$ efficiently

3 Go about 1 & 2 so that "gaps" in the matchings are allowed but with penalty. (special case of no gaps is with infinite penalty)

Defn:

Substring: $S = c\_a\_t$

Substrings (contiguous of S)
$$\left\{\begin{aligned} S[1:3] &= c\_a\_t \\ S[2:3] &= a\_t \\ S[2:2] &= a \end{aligned}\right.$$

Subsequence: $S = r\_o\_l\_l\_e\_r$

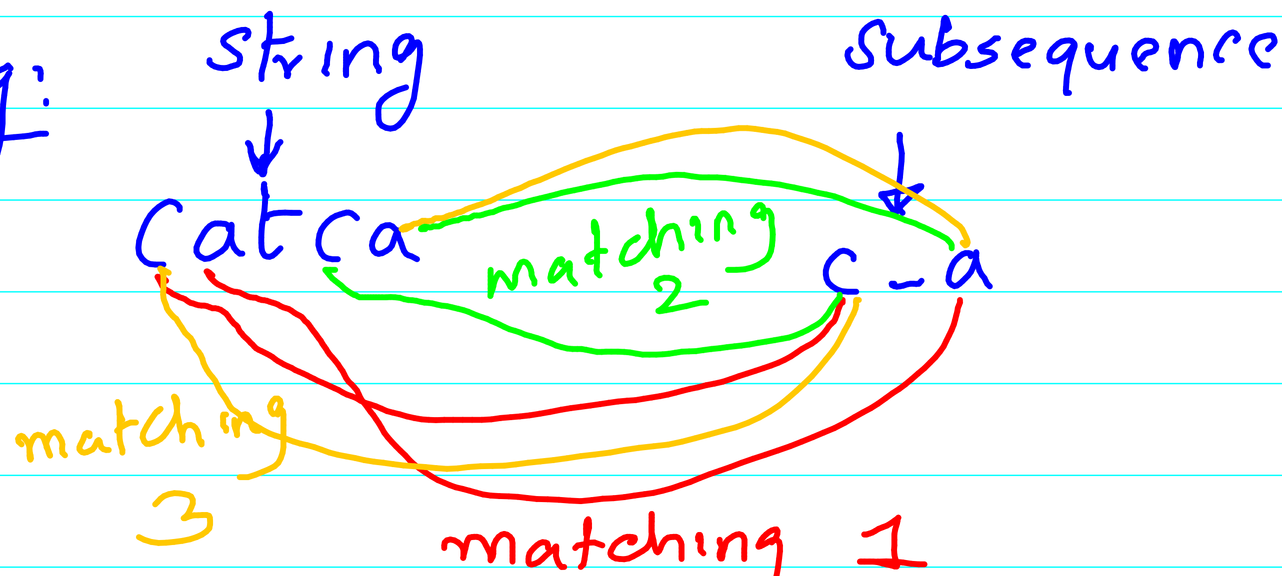
subsequences (non contiguous of S)
$$\left\{\begin{aligned} S[1,3,6] &= r\_l\_r \\ S[2,4] &= o\_l \end{aligned}\right.$$

subsequence (any non-zero entry indicates subsequence)

$\lambda \in (0,1]$

	c-a	c-t	a-t	b-a	b-t	c-r	a-r	b-r
ϕ(cat)	λ^2	λ^3	λ^2	0	0	0	0	0
ϕ(car)	λ^2	0	0	0	0	λ^3	λ^2	0
ϕ(bat)	0	0	λ^2	λ^2	λ^3	0	0	0
ϕ(bar)	0	0	0	λ^2	0	0	λ^2	λ^3

Eg:



$$\phi(\text{catca}) \Big|^{\,c\_a}\ \lambda^2 + \lambda^2 + \lambda^2 \times \lambda^3$$

$$= 2\lambda^2 + \lambda^5$$

In this setting, each match carries a penalty of λ & each gap carries a penalty of λ.

14/03

⟹ To handle "no gap" situation you might need to separate the two penalties & set the gap penalty to ∞

→ why penalize "matches"?

- of course, if you separate the two penalties, match penalty could be set to 1
- Nevertheless, you often want "short" feature/string matches to be rewarded more than "long" feature/string matches

→ Reason multiplicative scheme for penalty is used is because $\phi^T \phi$ becomes efficiently computable.

H/W: How about additive or o/r forms of penalty or reward?

Let us now formally define ϕ:

$\phi_u(s)$ *for each* $u \in \Sigma^n$. *We define*

$$\phi_u(s) = \sum_{\mathbf{i}:u=s[\mathbf{i}]} \lambda^{l(\mathbf{i})},$$

match index array.

eg: $u = c\_a$

$s = a\ b\ r\ c\ t\ q\ a\ c\ a$

$\Rightarrow i = [4, 7], [4, 9], [8, 9]$

& $\lambda^3 + \lambda^5 + \lambda$

Definition of String kernel

$$K_n(s,t) = \sum_{u\in\Sigma^n} \langle \phi_u(s)\cdot\phi_u(t)\rangle = \sum_{u\in\Sigma^n}\sum_{\mathbf{i}:u=s[\mathbf{i}]}\lambda^{l(\mathbf{i})}\sum_{\mathbf{j}:u=t[\mathbf{j}]}\lambda^{l(\mathbf{j})}$$
$$= \sum_{u\in\Sigma^n}\sum_{\mathbf{i}:u=s[\mathbf{i}]}\sum_{\mathbf{j}:u=t[\mathbf{j}]}\lambda^{l(\mathbf{i})+l(\mathbf{j})}.$$

The two i/p documents

Enumerating the u's is a formidable task !!!

Can we obtain $K_n(s,t)$ from $K_r(p,q)$ for p/q substring of s/t, $r < n$ using recursion?

Obviously:

$$K_n(sx,t) = K_n(s,t) + \ldots\ldots$$

Vinod's recursion

$$K_n(sx, t) = K_n(s, t) + \sum_{i=1}^{|s|} K'_{n-1}(s, i, t)\, x^{|s|-i}$$

where

$$K'_n(s, i, t) = \sum_{j=1}^{i-1} K'_{n-1}(s, j, t) * x^{i-j+1}$$

We will come back to Vinod's recursion after looking at that of the authors:

Base cases:

$$K_n(s, t) = 0 \quad \text{if } n = 0$$
$$= 0 \quad \text{if } |s| \text{ or } |t| < n$$

$$K_i'(s,t) = \sum_{u \in \Sigma^i} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{|s|+|t|-i_1-j_1+2},$$
$$i = 1, \ldots, n-1,$$

Like us, the authors also realize the need for an auxilliary $K_i'(s,t)$

$K_i'(s,t)$ is almost similar to $K_n(s,t)$ except that penalty is not based on $l(\mathbf{i})$ or $l(\mathbf{j})$ but is based on the distances from ends of s & t to the first matches i_1 in $\mathbf{i}$ & j_1 in $\mathbf{j}$

Let us write recursion for

$K_i'(sx, t)$:

Base case:

$K_i'(s,t) = 0$ — if $i = 0$; if $|s|$ or $|t| < i$

$$K_i'(sx, t) = \lambda K_i'(s,t) + \sum_{j : t_j = x} K_{i-1}'(s, t_{1..j-1}) * \lambda^{|t| - j + 2}$$

factoring in case where x has no match in any $u \in \Sigma$

to account for 'x' that matched t_j (neither x nor t_j are in argument for $K_{i-1}(s, t_{1..j-1})$)

H/W: Write $K_n(sx, t)$ in terms of $K_n(s, t)$ & $K_i'(s, t)$

$$
\begin{aligned}
K_0'(s,t) &= 1, \textit{ for all } s, t, \\
K_i'(s,t) &= 0, \textit{ if } \min(|s|, |t|) < i, \\
K_i(s,t) &= 0, \textit{ if } \min(|s|, |t|) < i, \\
K_i'(sx,t) &= \lambda K_i'(s,t) + \sum_{j:t_j=x} K_{i-1}'(s, t[1:j-1])\lambda^{|t|-j+2}, \\
&\qquad i = 1, \ldots, n-1, \\
K_n(sx,t) &= K_n(s,t) + \sum_{j:t_j=x} K_{n-1}'(s, t[1:j-1])\lambda^2.
\end{aligned}
$$