Text Classification using String Kernels

Prashant Ullegaddi
200807013
prashant.ullegaddi@research.iiit.ac.in

November 1, 2009

Outline.

- The Problem.
- What is a String Kernel?
- Salient Features of String Kernel.
- Previous Work.
- Basic Notations and Definitions.
- Feature Space of String Kernel.
- Basic Algorithm – Kernel as an Inner Product.
- An Example.
- Normalizing the Kernel Value.
- Key to Recursive Computation.
- Recursive Computation of String Kernel - Kernel Trick.
- SSK (String Subsequence Kernel) Algorithm.
- SSK with an Example.
- Discussion on SSK.
- References.

The Problem.

- Given two text document, how to compare them?
- Are two text documents similar to each other?
- If yes, what is the similarity score?
- This helps in classifying/categorizing the large text collections.

What is a String Kernel?

- A string kernel between two strings is an inner product in the feature space generated by all subsequences of length n.
- A subsequence is any ordered sequence of n characters occurring in the text though need not occur contiguously.
- E.g., subsequence 'car' is occurring in '**car**d' contiguously, but non-contiguously in '**c**ust**ar**d'.
- Main idea: Compare two text documents by means of the substrings they contain in common: the more substrings in common, the more similar they are.

Salient Features of String Kernel.

- String Kernel considers non-contiguous occurrences of subsequences as well but they are weighted less compared to continguous occurrences.
- It considers the ordering of the word information in text.
  E.g. the two strings:
  "The interest rate goes up, US dollar goes down." and
  "The interest rate goes down, US dollar goes up."
  These two will be identified as different strings by String kernel where as traditional methods like Word Kernels miss such ordering information.
- Doesn't take any semantic information about text, only works at low-level (character level matching). Boon or bane ☺.
- E.g. computer and desktop are treated differently.

Previous Work.

- Bag-of-Words (BOW) approach (also known as Word Kernel approach)
  - Breaks the documents into words
  - Words occurring more than a threshold are considered features
- N-grams approach (N-gram kernel)
  - Same as BOW, but instead of words here n-grams are considered.
  - E.g., 2-grams of CUSTARD are CU, US, ST, TA, AR, and RD.
  - Note: It's a language-independent model.
- String Kernel approach is compared to these two approaches.

Basic Notations and Definitions.

- Let $\sum$ be the alphabet.
- A string s has length $|s|$ and denoted by $s = s_1...s_{|s|}$.
- st denotes concatenation of strings s and t.
- The string $s[i:j]$ is the substring $s_i...s_j$ of s.
- A string u is a subsequence of s if u occurs in s though need not occur contiguously! That is there exist indices $\mathbf{i} = (i_1, ..., i_{|u|})$ with $1 \leq i_1 < ... < i_{|u|} \leq |s|$ such that $u = s[\mathbf{i}]$.
- Length $l(\mathbf{i})$ of subsequence u in string s is $i_{|u|} - i_1 + 1$, i.e., the window the subsequence spans in s.
- $\sum^n$ denotes all strings of length n from alphabet $\sum$.

Feature Space of String Kernel.

- String Kernels work in $|\sum|^n$ dimensional space.
- Feature space is $F_n = \mathbb{R}^{|\sum|^n}$.
- Feature value corresponding to some feature u (for any $u \in \sum^n$) for string s is given by

$$\phi_u(s) = \sum_{u=s[\mathbf{i}]} \lambda^{l(\mathbf{i})} \qquad \text{with} \quad \lambda \in (0,1).$$

- E.g. consider the string, $s = custard$ and feature $u = car$. The subsequence 'car' has a value λ^{6-1+1} $(= \lambda^6)$ in the string 'custard' since $u = s[\mathbf{i}]$ for $\mathbf{i} = (1, 5, 6)$. Similarly its value in 'card' is $\lambda^{3-1+1} = \lambda^3$, so that $\phi_{car}(card) > \phi_{car}(custard)$.
- This shows that contiguous occurrences weighted high compared to non-contiguous occurrences of the same subsequence.

String Kernel as an Inner Product.

For given two string s and t and length n, we can compute string kernel as inner product of feature vectors Φ of strings s and t. Thus we have,

$$
\begin{aligned}
K_n(s,t) &= \langle \Phi(s), \Phi(t) \rangle \\
&= \sum_{u \in \sum^n} \phi_u(s).\phi_u(t) \\
&= \sum_{u \in \sum^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \lambda^{l(\mathbf{i})} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{l(\mathbf{j})} \\
&= \sum_{u \in \sum^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{l(\mathbf{i})+l(\mathbf{j})}
\end{aligned}
$$

Basic Algorithm – Kernel as an Inner Product

Step 1: Enumerate all substrings of length n.

Step 2: Compute the feature vectors (where each feature is string enumerated above) for given two documents.

Step 3: Compute the similarity between the documents as inner product between the feature vectors ($K_n(s,t) = \langle \Phi(s), \Phi(t) \rangle$).

NOTE:

1. Efficiency is $O(|\sum|^n)$, just to compute the features!
2. With increase in n, computing all features becomes computationally impractical.

An Example.

- Let *cat*, *car*, *bat*, and *bar* be four documents.
- The features (for n = 2) for these documents are *c-a*, *c-t*, *b-a*, *b-t*, *a-t*, *a-r*, *c-r*, and *b-r*.
- The feature vector for each of the documents is given below:

Table: Feature vectors for the four documents.

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

- Applying equation (2), we have $K_2(cat, bat) = \lambda^2.\lambda^2 = \lambda^4$

Normalizing the Kernel Value.

Nôrmalized kernel value is given by

$$\hat{K}_n(s,t) = \frac{K_n(s,t)}{\sqrt{K_n(s,s)K_n(t,t)}} \tag{1}$$

For the example above the normalized kernel value is

$$\begin{aligned}
\hat{K}_2(cat,bat) &= \frac{K_2(cat,bat)}{\sqrt{K_2(cat,cat)K_2(bat,bat)}} \\
&= \frac{\lambda^4}{\sqrt{(2\lambda^4+\lambda^6)(2\lambda^4+\lambda^6)}} \\
\hat{K}_2(cat,bat) &= \frac{1}{2+\lambda^2}
\end{aligned}$$

Key to Recursive Computation.

Suppose we know the value of the kernel for string s and t, then how can we use this value to compute kernel for sx and t where x is any symbol such that $x \in \sum$?

- Subsequences common to strings s and t are also common to string sx and t.
- Also consider new matching subsequences ending in x in t and whose $(n-1)$ suffix is in s.

Recursive Computation of String Kernel – Kernel Trick.

The recursion for computing the Subsequence Kernel:

$$
\begin{aligned}
K'_0(s,t) &= 1 \qquad \textit{for all } s,\ t && (2)\\
K'_i(s,t) &= 0 \qquad \textit{if } min(|s|,|t|) < i && (3)\\
K'_i(sx,t) &= \lambda K'_i(s,t) + \sum_{j:t_j=x} K'_{i-1}(s,t[1:(j-1)])\lambda^{|t|-j+2} && (4)\\
K_i(s,t) &= 0 \qquad \textit{if } min(|s|,|t|) < i, \textit{ for } i = 1,...,n-1 && (5)\\
K_n(sx,t) &= K_n(s,t) + \lambda^2 \sum_{j:t_j=x} K'_{n-1}(s,t[1:j-1]) && (6)
\end{aligned}
$$

- Uses recursion to solve the problem without explicitly computing features (the subsequences of length n).
- Uses Dynamic Programming to store intermediate results to speed up the computation.

SSK($s[1:p]$, $t[1:q]$, n, λ)

/* s and t are strings of lengths p, and q, n is subsequences length, λ is the decay factor.*/

$K'_{prev}(0:q, 0:p) \leftarrow 1$ /* eq.(2) */
for $l \leftarrow 1$ *to* $n-1$ **do**
  for $i \leftarrow 0$ *to* q **do**
    for $j \leftarrow 0$ *to* p **do**
      if $min(i,j) < l$ **then**
        $K'_{next}(i,j) \leftarrow 0$ /* eq.(3) */
      else
        $K'_{next}(i,j) \leftarrow \lambda\, K'_{next}(i,j-1)$ /* 1st term, eq.(4)*/
        for $k \leftarrow 1$ *to* i **do** /* 2nd term of eq.(4) */
          if $s[j] = t[k]$ **then**
            $K'_{next}(i,j) \leftarrow K'_{next}(i,j) + \lambda^{i-k+2}\, K'_{prev}(k-1,j-1)$
          endif
        endfor
      endif
    endfor
  endfor
  $K'_{prev} \leftarrow K'_{next}$
endfor

SSK – String Subsequence Kernel (...continued)

```
for i ← 0 to q do
    for j ← 0 to p do
        if min(i, j) < n then
            K_n(i, j) ← 0    /* eq.(5)*/
        else
            K_n(i, j) ← K_n(i,j − 1)    /* 1st term, eq.(6)*/
            for k ← 1 to i do                /* 2nd term of eq.(6) */
                if s[j] = t[k] then
                    K_n(i, j) ← K_n(i, j) + λ^2 K'_next(k − 1,j − 1)
                endif
            endfor
        endif
    endfor
endfor
return K_n(q, p)
```

SSK with an Example.

Let $s = cat$, and $t = bat$ be the two strings. $K_2(s, t)$ can be computed as follows starting with K'_0.

Table: For i = 0

K'_0	ϵ	c	a	t
ϵ	1	1	1	1
b	1	1	1	1
a	1	1	1	1
t	1	1	1	1

SSK with an Example (...continued)

Table: For i = 1.

K'_1	ϵ	c	a	t
ϵ	0	0	0	0
b	0	0	0	0
a	0	0	λ^2	λ^3
t	0	0	λ^3	$\lambda^2 + \lambda^4$

E.g. let us compute the value for cell K'_1(ca, ba).

$$\begin{aligned} K'_1(ca, ba) &= \lambda K'_1(c, ba) + \lambda^{|ba|-2+2} K'_0(c, b) \\ &= \lambda.0 + \lambda^{2-2+2}.1 \\ &= \lambda^2 \end{aligned}$$

SSK with an Example (...continued)

Table: For last recursion to compute Kernel.

K_2	ϵ	c	a	t
ϵ	0	0	0	0
b	0	0	0	0
a	0	0	0	0
t	0	0	0	λ^4

$$\begin{aligned} K_2(ca, ba) &= K_2(c, ba) + \lambda^2 K_1'(c, b) \\ &= 0 + \lambda^2.0 = 0, \text{ and} \end{aligned}$$

$$\begin{aligned} K_2(cat, bat) &= K_2(ca, bat) + \lambda^2 K_1'(ca, bb) \\ &= 0 + \lambda^2.\lambda^2 = \lambda^4 \end{aligned}$$

The value of K_2(s,t) is given by the value in the cell ($|t|$, $|s|$) in the table K_2

Discussion on SSK.

- The efficiency of SSK to compute $K_n(s,t)$ is $O(n|s||t|^2)$.
- Improvment can be made by looking at the repeated k-loop in the algorithm in computation. This can be eliminated by computing it and storing it for later use to compute K'_i.
- Define

$$K''_i(sx,t) = \sum_{j:t_j=x} K'_{i-1}(s,t[1:j-1])\lambda^{|t|-j+2} \tag{7}$$

- $K'_i(sx,t)$ can be computed as follows:

$$K'_i(sx,t) = \lambda K'_i(s,t) + K''_i(sx,t) \tag{8}$$

- With this change, a loop on $|t|$ is eliminated, making the efficiency $O(n|s||t|)$.

Approximation of SSK on Large Datasets

- Since cost of computation of SSK is linear in document lengths, for large datasets computing SSK can still be very expensive. Hence an approximation of SSK is desired.
- An approximation method is specified here:
  1. Pick k most occurring n-grams from the first N documents in the collection. Call this set T_n.
  2. For every document d, compute SSK between d and t for all $t \in T_n$. This computation must be less expensive as one of the strings is shorter.
  3. Find the inner product between documents with SSK computed above. This forms the new approximation to SSK.

New Approximated SSK.

$$\tilde{K}_n(s,t) = \sum_{z \in T_n} \hat{K}_n(s,z)\hat{K}_n(z,t) \tag{9}$$

where T_n is the set of all most occurring n-grams in the data set D with $s, t \in D$.

References.

1. Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, Chris Watkins, *Text Classification using String Kernels*, JMLR 2001.
2. Huma Lodhi, John Shawe-Taylor, Nello Cristianini, Chris Watkins, *Text Classification using String Kernels*, NIPS 2000.
3. Nicola Cancedda, Eric Gaussier, Cyril Goutte, Jean-Michel Renders, *Word Sequence Kernels*, Appendix A. Recursive Formulation of Sequence Kernels, JMLR 2003.
4. John Shawye-Taylor and Nello Christianini, *Kernel Methods for Pattern Analysis*, Chapter 11 on *Kernels for Structured Data: Strings, trees etc.*

Thank You!