Fast Stringkernels

Alexander J. Smola
Alex.Smola@anu.edu.au

Department of Computer Science
Machine Learning Group
RSISE, The Australian National University

Joint work with S.V.N. Vishwanathan

Overview

THE AUSTRALIAN NATIONAL UNIVERSITY

- Kernels on Strings
- Suffix Trees
  - Definition and Examples
  - Matching Statistics
  - Counting Substrings
- Weights and Kernels
  - Annotation and Weighting Function
  - Linear Time Prediction
- Extensions and Future Work

String Kernel Basics

THE AUSTRALIAN NATIONAL UNIVERSITY

Some Notation

Alphabet: what we build strings from
Sentinel Character: usually $\$$, it terminates the string
Concatenation: xy obtained by assembling strings x, y
Prefix / Sufix: If $x = yz$ then y is a prefix and z is a suffix

Exact Matching Kernels

$$k(x, x') := \sum_{s \sqsubseteq x, s' \sqsubseteq x'} w_s \delta_{s,s'} = \sum_{s \in \mathcal{A}^*} \mathrm{num}_s(x) \, \mathrm{num}_s(x') w_s.$$

Inexact Matching Kernels:

Christina's talk. Much more expensive but she and Eleazar have clever tricks.

String Kernel Examples

THE AUSTRALIAN NATIONAL UNIVERSITY

Bag of Characters

$w_s = 0$ for all $|s| > 1$ counts single characters. Can be computed in linear time and linear-time predictions (Joachims, 1999).

Bag of Words

s is bounded by whitespace. Linear time (Joachims, 1999).

Limited Range Correlations

Setting $w_s = 0$ for all $|s| > n$ yields limited range correlations of length n.

K-spectrum kernel

This takes into account substrings of length k (Leslie et al., 2002), where $w_s = 0$ for all $|s| \neq k$. Linear time kernel computation, and quadratic time prediction.

Motifs

Library of motifs for classification (Ben Hur and Brutlag, 2003).

General Case

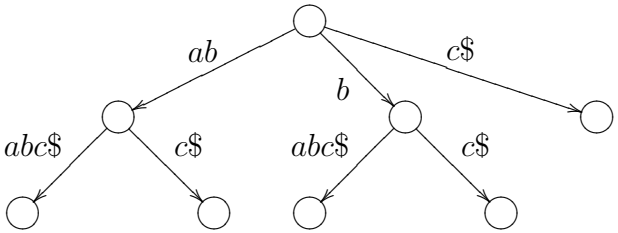
Quadratic time kernel computation (Haussler, 1998, Watkins, 1998), cubic time prediction.

Suffix Trees

THE AUSTRALIAN NATIONAL UNIVERSITY

Definition

Compact tree built from all the suffixes of a word. Suffix tree of `ababc`



Properties

- Can be built and stored in linear time (Ukkonen, 1995)
- Leaves on subtree give number of matching substrings

Suffix Links

Connections *across* the tree. Vital for parsing strings (e.g., if we parsed `abracadabra` this speeds up the parsing of `bracadabra`).

Matching Statistics

THE AUSTRALIAN NATIONAL UNIVERSITY

Definition

Given strings x, y with $|x| = n$ and $|y| = m$, the matching statistics of x with respect to y are defined by $v, c \in \mathbb{N}^n$, where

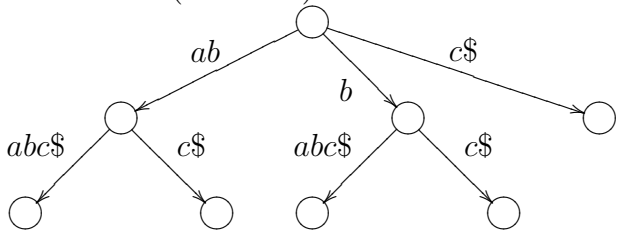
- v_i is the length of the longest substring of y matching a prefix of $x[i:n]$
- $\overline{v_i} := i + v_i - 1$
- c_i is a pointer to $\mathrm{ceil}(x[i:\overline{v_i}])$ in $S(y)$.

This can be computed in linear time (Chang and Lawler, 1994).

Example

Matching statistic of `abba` with respect to $S(\texttt{ababc})$.

String	a	b	b	a
v_i	2	1	2	1
$\mathrm{ceil}(c_i)$	ab	b	babc$	ab



Matching Substrings

THE AUSTRALIAN NATIONAL UNIVERSITY

Prefixes

w is a substring of x iff there is an i such that w is a prefix of $x[i:n]$. The number of occurrences of w in x can be calculated by finding all such i.

Substrings

The set of matching substrings of x and y is the set of all prefixes of $x[i:\overline{v_i}]$.

Next Step

If we have a substring w of x, prefixes of w may occur in x with higher frequency. We need an efficient computation scheme.

Key Trick

Theorem

Let x and y be strings and c and v be the matching statistics of x with respect to y. Assume that

$$W(y,t) = \sum_{s \in \text{prefix}(v)} w_{us} - w_u \text{ where } u = \text{ceil}(t) \text{ and } t = uv.$$

can be computed in constant time for any t. Then $k(x,y)$ can be computed in $O(|x| + |y|)$ time as

$$k(x,y) = \sum_{i=1}^{|x|} \text{val}(x[i:\overline{v_i}]) = \sum_{i=1}^{|x|} \text{val}(c_i) + \text{lvs}(\text{floor}(x[i:\overline{v_i}]))W(y, x[i:\overline{v_i}])$$

where $\text{val}(t) := \text{lvs}(\text{floor}(t)) \cdot W(y,t) + \text{val}(\text{ceil}(t))$ and $\text{val}(\text{root}) := 0$.

$W(y,t)$ in Constant Time

THE AUSTRALIAN NATIONAL UNIVERSITY

Length-Dependent Weights

Assume that $w_s = w_{|s|}$, then

$$W(y,t) = \sum_{j=|\operatorname{ceil}(t)|}^{|t|} w_j - w_{|\operatorname{ceil}(t)|} = \omega_{|t|} - \omega_{|\operatorname{ceil}(t)|} \text{ where } \omega_j := \sum_{i=1}^{j} w_j$$

Generic Weights

- Simple option: pre-compute the annotation of all suffix trees beforehand.
- Better: build suffix tree on all strings (linear time) and annotate this tree.
- Simplifying assumption for TFIDF weights, $w_s = \phi(|s|)\psi(\#s)$

$$W(y,t) = \sum_{s \in \operatorname{prefix}(t)} w_s - \sum_{s \in \operatorname{prefix}(\operatorname{ceil}(t))} w_s = \phi(\operatorname{freq}(t)) \sum_{i=|\operatorname{ceil}(t)|+1}^{|t|} \phi(i)$$

$W(y,t)$ for Feature Lists

THE AUSTRALIAN NATIONAL UNIVERSITY

Weights

$$w_s = \begin{cases} 1 & \text{if } s \text{ occurs in list} \\ 0 & \text{otherwise} \end{cases}$$

Suffix Trie of Motifs

- Compact all motifs into suffix trie Ω (can be done in linear time, all we need to do is chop off excess strings due to concatenation). Need to include **suffix links**.
- Annotate Ω to contain motif weights (cheap: only one number per motif) and store top down sums ω on vertices (linear via BFS).

Lookup Procedure for $W(y,t)$

- Compute matching statistics of x, y with respect to Ω (linear time) and of x with respect to $S(y)$
- Identify positions of vertices of $S(y)$ on Ω.
- $W(y,t)$ is difference between values of ω on the branches of Ω.

Thanks for comments from Eleazar . . .

Linear Time Prediction

THE AUSTRALIAN NATIONAL UNIVERSITY

Problem

For prediction we need to compute $f(x) = \sum_i \alpha_i k(x_i, x)$.

- This depends on the number of SVs.
- Bad for large databases (e.g., spam filtering). The classifier degrades in runtime, the more data we have.
- We are repeatedly parsing s

Idea

We can merge matching weights from all the SVs. All we need is a compressed lookup function.

Linear Time Prediction

- Merge all SVs into one suffix tree Σ.
- Compute matching statistics of x wrt. Sigma.
- Update weights on every node of Σ as

$$\mathrm{weight}(\overline{w}) = \sum_{i=1}^{m} \alpha_i \, \mathrm{lvs}_{x_i}(\overline{w})$$

- Extend the definition of $\mathrm{val}(x)$ to Σ via

  $\mathrm{val}_\Sigma(t) := \mathrm{weight}(\mathrm{floor}(t)) \cdot W(\Sigma, t) + \mathrm{weight}(\mathrm{ceil}(t))$ and $\mathrm{val}_\Sigma(\mathrm{root}) := 0.$

- Here $W(\Sigma, t)$ denotes the sum of weights between $\mathrm{ceil}(t)$ and t, with respect to Σ rather than $S(y)$. We only need to sum over $\mathrm{val}_\Sigma(x[i : \overline{v_i}])$ to compute f.

We can classify texts in linear time regardless of the size of the SV set!

Position Dependent Weights

Position Kernel

$$k(x, x') := \sum_{s \in \mathcal{A}^*} \mathrm{num}_s(x)\, \mathrm{num}_s(x') w_s.$$

where we redefine position dependent count

$$\mathrm{num}_s(x) := \sum_{i=1}^{n} w_i \{x[i:n] = ss'\}$$

So there is **no need for a hard cutoff at boundary**.

Computing it

Use $k(x, y) = \sum_{i=1}^{|x|} \mathrm{val}(x[i : \overline{v_i}])$ and replace it by

$$k(x, y) = \sum_{i=1}^{|x|} w_i \,\mathrm{val}(x[i : \overline{v_i}])$$

More replace leaf weights by w_i (depending on starting position).

Summary and Extensions

THE AUSTRALIAN NATIONAL UNIVERSITY

- Similar results hold for kernels on trees: redux of tree to string kernels (heaps, stacks, bags, etc. trivial)
- Linear prediction and kernel computation time (previously quadratic or cubic). Makes things practical.
- Can be used in SMO-type algorithms which need to compute many $f(x_i)$: updates $f \Leftarrow f + \alpha_i k(x_i, \cdot)$ can be done in $O(|x| \cdot \min(|x|, m))$ worst case time (typical is $O(|x|)$).
- Storage of SVs needed. Can be greatly reduced if redundancies abound in SV set. E.g. for `anagram` and `analphabet` we need only `analphabet` and `gram`.
- Coarsening for trees (can be done in linear time, too)
- Approximate matching and wildcards (see Christina's talk)
- Automata and dynamical systems
- Do "expensive" things with string kernel classifiers.