String and Tree Kernels
Algorithms and Applications

S.V.N. “Vishy” Vishwanathan
vishy@axiom.anu.edu.au

National ICT of Australia
and
Indian Insitutute of Science
Bangalore, India

Joint work with Alex Smola

Overview - I

- Motivation
- Exact Kernels on Strings
  - Definition and Examples
- Suffix Trees
  - Definition
  - Matching Statistics
  - Counting Substrings
- Weights and Kernels
  - Annotation
  - Weighting Functions
  - Linear Time Prediction

Overview - II

NATIONAL ICT AUSTRALIA

- Sliding Windows
- Position Dependent Weights
- Kernels on Trees
  - Definition
  - Sorting Trees
  - Tree to String Conversion
  - Coarsening Levels
- Inexact Kernels on Strings
  - Definition
  - Mismatch Kernel
  - Space Time Tradeoffs
- Extensions and Future Work

Motivation

NATIONAL ICT AUSTRALIA

Kernel Methods

- Exciting theoretical bounds
- Numerous applications
- Limited to vectorial data

Strings

- Bio-informatics
- Spam filtering
- Internet search engines

String Kernels

- Must respect *structure*
- Fast and easy to compute
- Semantically meaningful

Notation

NATIONAL ICT AUSTRALIA

Alphabet

- Set of characters denoted by $\mathcal{A}$

String

- Any $x \in \mathcal{A}^k$ for some k

Sentinel

- Some $\$ \notin \mathcal{A}$ used to terminate a string

Prefix/Suffix/Substring

- Let $x = uvw$ for some possibly empty u, v and w
- u is called the prefix and w the suffix of x
- v is called a substring of x
- We write $v \sqsubseteq x$

String Kernels - I

NATIONAL ICT AUSTRALIA

Exact Matching Kernels

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

- Count all matching substrings
- Flexible weighting scheme
- Different applications $\implies$ different weights
- Noise in training data $\implies$ does not work :-(
- Successful applications in bio-informatics (Vishwanathan and Smola, 2002) (Leslie et. al., 2002)
- Linear time algorithms using suffix trees

Exact String Kernels

NATIONAL ICT AUSTRALIA

Bag of Characters

Counts single characters (Joachims, 1999). Set $w_s = 0$ for all $|s| > 1$

Bag of Words

s is bounded by whitespace (Joachims, 1999)

Limited Range Correlations

Set $w_s = 0$ for all $|s| > n$ given a fixed n

K-spectrum kernel

Account for matching substrings of length k (Leslie et al., 2002). Set $w_s = 0$ for all $|s| \neq k$

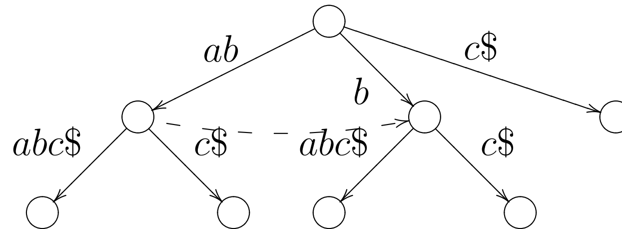
General Case

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

Suffix Trees

Definition

Compact tree built from all the suffixes of a word. Suffix tree of `ababc` denoted by $S(ababc)$.



- Node label $:=$ unique path from the root
- Suffix links are used to speed up parsing of strings
- Suppose we are at a node ax then suffix links help us to jump to node x
- Each internal node has a unique suffix link (McCreight, 76)

Suffix Trees Contd. . .

Properties

- Represents all the substrings of the given string
- Can be constructed in linear time
- Offline algorithms - (Weiner 73) and (McCreight 76)
- Online algorithm - (Ukkonen 93)
- Can be stored using linear space
- Edges are encoded by indices of the substring
- Each leaf corresponds to a unique suffix
- Leaves on subtree give number of occurrences
- Each internal node has at least 2 distinct children
- Annotation by performing DFS is easy
- LCA queries in constant time (linear pre-processing)

Matching Statistics

NATIONAL ICT AUSTRALIA

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

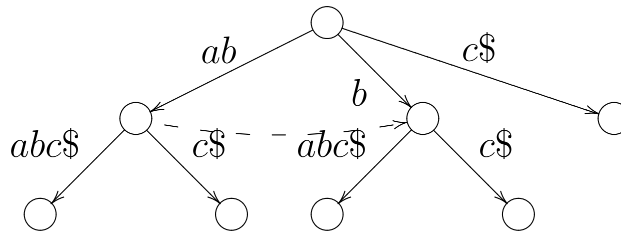
- v_i is the length of the longest substring of y matching a prefix of $x[i:n]$
- $\overline{v_i} := i + v_i - 1$
- c_i is a pointer to $\mathrm{ceil}(x[i:\overline{v_i}])$ in $S(y)$.
- $\mathrm{ceil}(s)$ is the last node on the path from the root to s

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

Example

Matching statistic of abba with respect to $S($ababc$)$.

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



Matching Substrings

NATIONAL ICT AUSTRALIA

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 - I

NATIONAL ICT AUSTRALIA

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

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

where $\text{val}(s)$ indicates the contribution to the kernel due to string s and all its prefixes.

Key Trick - II

NATIONAL ICT AUSTRALIA

Observation

All substrings ending on the same edge of $s(y)$ occur the same number of times in string y

- For any matching substring t we can write

$$\mathrm{val}(t) := \mathrm{lvs}(\mathrm{floor}(t)) \cdot W(y,t) + \mathrm{val}(\mathrm{ceil}(t))$$

- For each node v we can pre-compute $\mathrm{val}(v)$ by a simple DFS on the suffix tree
- We can compute in constant time

$$\mathrm{val}(x[i:\overline{v_i}]) = \mathrm{lvs}(\mathrm{floor}(x[i:\overline{v_i}])) \cdot W(y, x[i:\overline{v_i}]) + \mathrm{val}(c_i)$$

Computing $W(y,t)$

NATIONAL ICT AUSTRALIA

Length-Dependent Weights

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

$$W(y,t) = \sum_{j=|\operatorname{ceil}(t)|}^{|t|} w_j - w_{|\operatorname{ceil}(t)|} = \omega_{|t|} - \omega_{|\operatorname{ceil}(t)|}$$

where $\omega_j := \sum_{i=1}^{j} w_j$, which can be pre-computed and stored for $j = 1, 2, \ldots \max(|x|, |y|)$.

K-spectrum Kernel

It is easy to see that

$$W(y,t) = \begin{cases} 1 & \text{if } |\operatorname{ceil}(t)| < k \text{ and } |t| \geq k \\ 0 & \text{if otherwise} \end{cases}$$

Generic Weights

NATIONAL ICT AUSTRALIA

TFIDF Weights

- Assume $w_s = \psi(\mathrm{freq}(s))\phi(|s|)$
- Strings on the same edge $\implies$ same frequency

$$W(y,t) = \psi(\mathrm{freq}(t)) \sum_{i=|\,\mathrm{ceil}(t)|+1}^{|t|} \phi(i)$$

- To take into account frequency in entire training set build a master suffix tree of all strings in the training set
- In case weights are completely arbitrary it suffices to annotate all the nodes of this master suffix tree (Vishwanathan and Smola, 2002).

The Estimation Problem

NATIONAL ICT AUSTRALIA

Problem

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

- This depends on the number of SVs
- Web search engines and spam filtering
  - Large number of SVs
  - Real time prediction is critical

Key Observation

- We are repeatedly parsing the SV strings
- Pre-processing the SV set can speed up prediction

Idea

We can merge matching weights from all the SVs. All we need is a master suffix tree!

Master Suffix Tree

NATIONAL ICT AUSTRALIA

- A collection of strings $\mathcal{X} = \{x_1, x_2 \ldots x_m\}$
- We define $S(\mathcal{X})$ as a *natural* union of $S(x_i)$
- Define $X := x_1\$_1 x_2\$_2 \ldots x_m\$_m$
- Construct $S(X)$
- Prune away the extra labels on the leaves
- The $\$_i$'s induce a extra $\log$ penalty in construction time
- Aamir et. al. algorithm avoids this penalty
- It uses a clever modification of the McCreight algorithm
- We can construct the master suffix tree $S(\mathcal{X})$ in $O(\sum_i |x_i|)$ time

Estimation Algorithm

NATIONAL ICT AUSTRALIA

- A substring s in a Support Vector x_i is *counted* (i.e. contributes a weight value) α_i times to the kernel
- We merge the suffix trees of all the Support Vectors into a master suffix tree
- In the master suffix tree we associate weight α_i with each leaf derived from Support Vector x_i
- For a node v, we define $\mathrm{lvs}(v)$ as the sum of weights associated with the subtree rooted at v
- The key theorem can now be applied unchanged to compute $f(x)$
- Our algorithm runs in time linear in the size of x and is independent of the size of the Support Vector set!

Sliding Windows

NATIONAL ICT AUSTRALIA

Motivation

- Bioinformatics: Estimate on windows of a long string

Problem

- Long sequence x and a window width N are given
- Estimate the function value for windows of length N

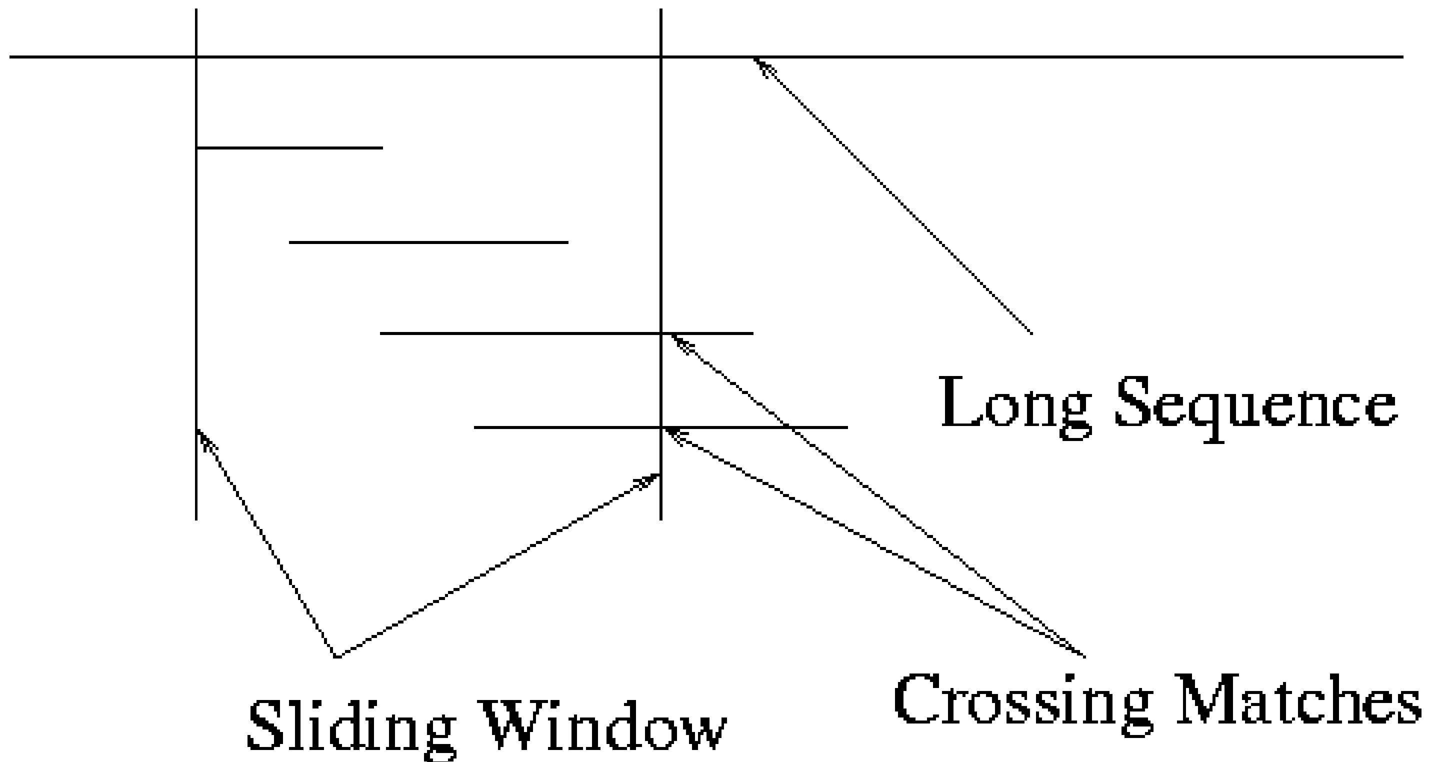
Observation

- *Interesting* matches $\Longrightarrow$ cross a window boundary

Algorithm

- Compute the matching statistics for x
- Account for cross boundary matches (at most N)
- In practice very few matches cross the boundary
- Expected sub-linear time for sliding a window!

A Picture Helps



Positional Weights

Motivation

- Bioinformatics: weigh exons and introns differently

Definition

$$k(x, x') := \sum_{s \sqsubseteq x, s' \sqsubseteq x'} w_{(s,x)} \rho_{(s',x')} \delta_{s,s'}$$

where $w_{(s,x)}$ and $\rho_{(s',x')}$ are position dependent.

Observation

- Each leaf of $S(x)$ corresponds to a unique suffix of x

Algorithm

- Assign a different weight to each leaf
- As before sum the weights on each subtree
- The original algorithm runs unchanged!

Tree Kernels

NATIONAL ICT AUSTRALIA

Subset Trees

Set of connected nodes of a tree T

Definition (Colins and Duffy, 2001)

Denote by T, T' trees and by $t \models T$ a subset tree of T, then

$$k(T, T') = \sum_{t \models T, t' \models T'} w_t \delta_{t,t'}.$$

Our Definition (Vishwanathan and Smola, 2002)

In case we count matching subtrees then $t \models T$ denotes that t is a subtree of T and we get

$$k(T, T') = \sum_{t \models T, t' \models T'} w_t \delta_{t,t'}.$$

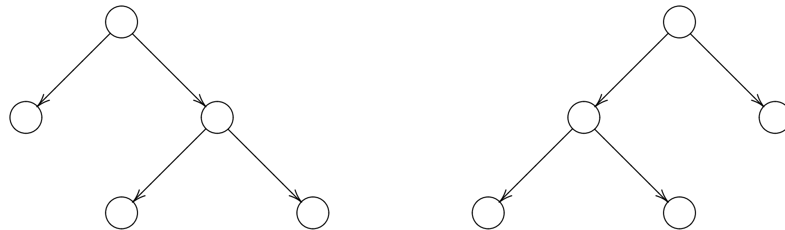
Permutation Invariance

Problem

We want permutation invariance of unordered trees.

Example:

The following two unordered trees are mirror images



Solution

- Sort trees before computing kernel
- Maps equivalent trees to a single representative

Sorting Trees - I

Sorting Rules

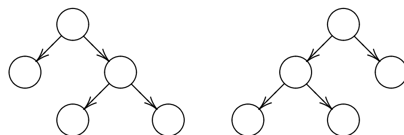
- Assume existence of lexicographic order on labels
- Introduce symbols '[', ']' satisfy '[' < ']', and that ']', '[' < $\mathrm{label}(n)$ for all labels.

Algorithm

- For node n with children $n_1, \ldots, n_c$ sort the tags of the children in lexicographical order such that $\mathrm{tag}(n_i) \leq \mathrm{tag}(n_j)$ if $i < j$ and define

$$\mathrm{tag}(n) = [\,\mathrm{label}(n)\,\mathrm{tag}(n_1)\,\mathrm{tag}(n_2)\ldots\mathrm{tag}(n_c)].$$

Example

The trees  have label `[[][[][]]]`.

Sorting Trees - II

NATIONAL ICT AUSTRALIA

Theorem

Let l be the number of nodes and λ the length of a label

1. $\mathrm{tag}(\mathrm{root})$ can be computed in $(\lambda + 2)(l \log_2 l)$ time and linear storage in l.
2. Substrings s of $\mathrm{tag}(\mathrm{root})$ starting with '$[$' and ending with a balanced '$]$' correspond to subtrees t of T where s is the tag on t.
3. $\mathrm{tag}(\mathrm{root})$ is invariant under permutations of the leaves and allows the reconstruction of an unique element of the equivalence class (under permutation).

Proof

- Proof of $1.$ by induction. Rest follows from definition.
- Extension to k-ary trees straightforward

Tree to String Conversion

NATIONAL ICT AUSTRALIA

Consequence

We can compute tree kernel by

1. Converting trees to strings
2. Computing string kernels

Advantages

- More general subtree operations possible: we may include non-balanced subtrees (cutting a slice from a tree).
- Simple storage and simple implementation (dynamic array suffices)
- All speedups for strings work for tree kernels, too (XML documents, etc.)

Coarsening Levels

NATIONAL ICT AUSTRALIA

Motivation

- Two trees are very similar if we ignore a few nodes
- Applications: image processing, document analysis

Definition

- $T_d \implies$ chop off nodes at height d in T
- If tree is labeled, need to propagate labels also
- We can then define

$$k_{\text{coarse}}(T, T') = \sum_i W_i k(T_i, T'_i)$$

- W_i is a down-weighting factor (typically $0 < \lambda^i < 1$)

Algorithm

- Coarsen string representations to compute k_{coarse}

String Kernel Basics - II

NATIONAL ICT AUSTRALIA

Inexact Matching Kernels

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

- Count all approximately matching substrings
- Search space is large
- Not all weighting schemes yield a proper kernel
- More expensive to compute
- Can compute special cases efficiently
- Space and time trade-offs
- Open Question: Can we do better?

Mismatch Kernels

NATIONAL ICT AUSTRALIA

Motivation

- Sequencing methods are error prone
- Sequences differ slightly due to biological reasons

Definition

$$K(x, x') = \sum_{s, s' \in \mathcal{A}^*} \mathrm{num}_s(x) \, \mathrm{num}_{s'}(x') \delta^m_{s,s'} w_{s,s'}$$

- $\delta^m_{s,s'}$ is non-zero only
  - if $|s| = |s'|$
  - No. of mismatches between s and s' is less than m

Special Case

- The (k, m)-mismatch kernel is obtained if we set $w_{s,s'} = 1$ for all $|s| = |s'| = k$ and 0 otherwise

Algorithm - I

Key Idea

- Given two strings x and x'
- Align x' with all possible positions on x
- For each alignment locate the mismatch locations
- This is known as convolution of strings
- Use a suffix tree to jump over matching pieces

Example

GATTACATA	x
TAGATACAGTAC	x'
111000011	Mismatches between x and x'
123333345	Mismatch count
0123**321012**	Mismatch count in windows of size 4

Algorithm - I Contd. . .

Analysis

- Computing mismatches per alignment - $O(|x|)$ time
- Finding all *good* windows takes $O(|x|)$ time
- There are $O(|x'|)$ possible alignments
- Total algorithm takes $O(|x||x'|)$ time

Advantages

- Faster than the previously known methods
- Can be sped up by using a suffix tree
- Can jump over matching regions in constant time

Disadvantages

- The algorithm is $\Theta(|x||x'|)$
- Scales badly when the strings are long

Algorithm - II

Key Idea

- We can rewrite the (k, m) mismatch kernel as

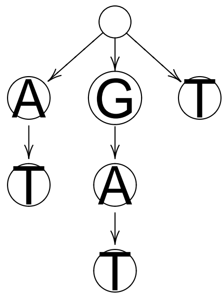
$$K(x, x') = \sum_{i=1}^{|x'|-k} K(x, x'[i : i + k - 1])$$

- If we pre-compute the value of $K(x, \cdot)$ for all possible k-mers the kernel evaluation is easy
- For each k-mer we need to generate its m neighborhood
- A suffix tree encodes all exact matching k-mers
- To generate the m neighborhood we make multiple copies of the suffix tree allowing for at most m mismatches

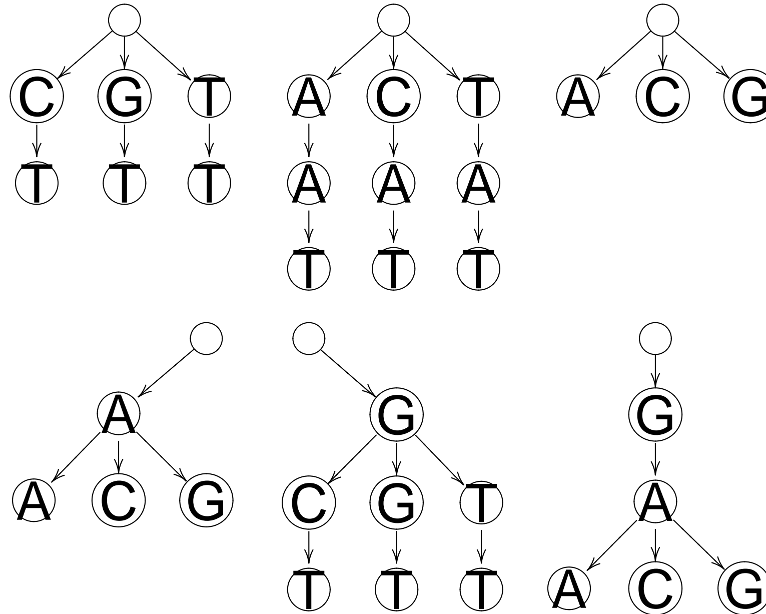
Algorithm - II Contd...

NATIONAL ICT AUSTRALIA

Original Tree



One Neighborhood



Algorithm - II Contd. . .

Algorithm

- For each pattern x build suffix tree $S(x)$
- Truncate the suffix tree to depth k (i.e. k-mers)
- Expand the m neighborhood and build suffix links
- As before use matching statistics to compute $K(x, x')$

Advantages

- Kernel computation linear after pre-processing
- Linear time estimation ideas can be applied
- Most of the ideas from exact matching can be used

Disadvantages

- Large amounts of pre-processing
- Not memory efficient

Applications

NATIONAL ICT AUSTRALIA

Problem

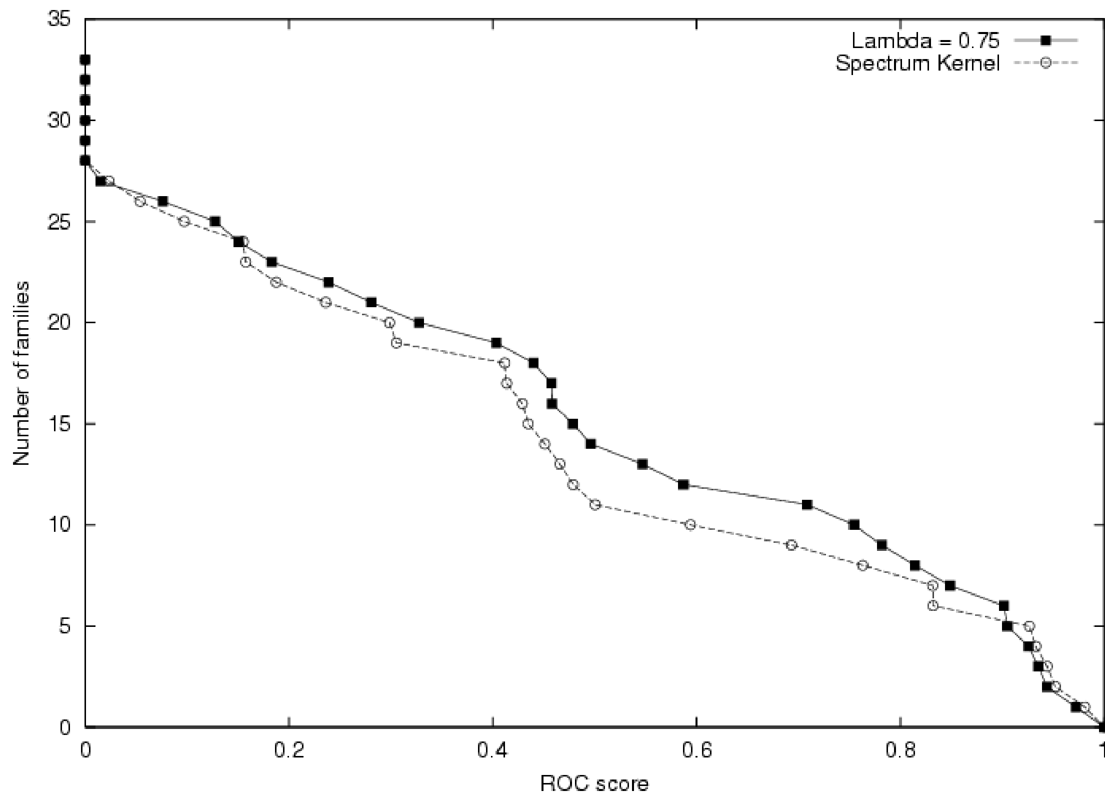
- Detect protein homologies (i.e. similarities)
- Used to predict
  - Functional properties
  - Structural properties

  of new protiens from known protiens

Kernel Used

- Exact Kernel
- Length weighted with $w_s = \lambda^{|s|}$
- Matching substrings of minimum length 3

Results



Summary and Extensions

- Reduction from quadratic or cubic to linear prediction and kernel computation time
- Kernels on heaps, stacks, bags, etc. trivial
- Compact storage of SVs if redundancies abound in SV set. E.g. for `anagram` and `analphabet` we need only `analphabet` and `gram`
- Ideas can also be extended to suffix arrays
- Approximate matching and wildcards
- Can we do better in case of approximate matches?
- Automata and dynamical systems
- Do "expensive" things with string kernel classifiers
- More information at http://www.kernel-machines.org