CS281B/Stat241B: Advanced Topics in Learning & Decision Making

String Kernels

Lecturer: Michael I. Jordan *Scribe: Ryan White*

1 p-spectrum kernels

Here we look for substrings of length p. The kernel counts the number of substrings in common between the two strings.

We can define this kernel as:

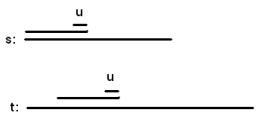
$$\begin{aligned} k_p(s,t) &= \sum_{u \in \Sigma^p} \phi_u^p(s)\phi_u^p(t) \\ \phi_u^p(s) &= |\{(v_1, v_2) : s = v_1 \cup v_2\}| \\ & \quad u \in \Sigma^p \end{aligned}$$

This kernel can be computed efficiently using a recursion built on the "k-suffix kernel":

$$k_p^s = \begin{cases} 1 & \text{if } s = s_1 u,\ t = t_1 u \text{ for } u \in \Sigma^k \\ 0 & \text{otherwise} \end{cases}$$

In other words, a score of one indicates that the suffixes agree, and a score of zero indicate that they disagree. Now, the kernel for substrings of length p is:

$$k_p(s,t) = \sum_{i=1}^{|s|-p+1} \sum_{j=1}^{|t|-p+1} k_p^s(s(i:i+p), t(j:j+p))$$



The algorithm is quadratic in lengths $|s|$ and $|t|$. However, more efficient algorithms have complexities on the order of $|s| + |t|$. The secret to these more efficient algorithms is to walk backwards when placing down strings, but shift forwards. Data structures call tries (specifically suffix trees) allow for fast comparisons.

2 All-subsequences kernels

The all subsequences kernel builds on the concept of the p-spectrum kernel, but allows gaps within the strings. To do this, we define:

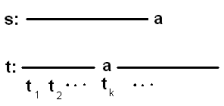
$$\begin{aligned} \phi_u(s) &= |\{\vec{i} : u = s(\vec{i})\}| \\ & \quad u \in \Sigma^* \end{aligned}$$

- where $\vec{i}$ is a sequence of integers used to index into s
- for example, $\vec{i}$ could be [2 3 7]

$$k(s,t) = \sum_{u \in \Sigma^*} \phi_u(s)\phi_u(t)$$

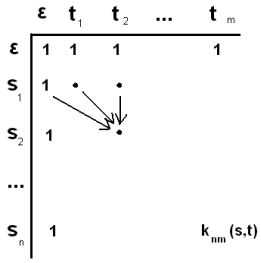
Now we can use a simple recursion that leads to dynamic programming:

$$\begin{aligned} k(sa,t) &= k(s,t) + \sum_{k:t_k=a} k(s,t(1:k-1)) \\ & \quad a \in \Sigma \\ & \quad s,t \in \Sigma^* \end{aligned}$$



$$\begin{aligned} k(s,tb) &= k(s,t) + \sum_{k:s_k=a} k(s(1:k-1),t) \\ k(s,\epsilon) &= 1 \\ k(\epsilon,t) &= 1 \end{aligned}$$

To implement this algorithm as dynamic programming, we need to build a table with columns corresponding to t and rows corresponding to s. Computing all of the entries of the table appears to have complexity $|n|^2|m|$, but can be done in complexity $|n||m|$ by pre-computing the sum of elements so far. Note that using suffix trees leads to an $O(|n| + |m|)$ algorithm.



Several modifications can be made to this algorithm. We could extend it by penalizing gaps or by running the algorithm on trees. We could compare how similar trees are by comparing sub-trees or counting the number of sub-trees in common.

3 side note

Fact:

$$\|x-y\|^2 \quad = \quad < x,x > -2 < x,y > + < y,y >$$

Since distances and inner products are similar, one could imagine interchanging them, but this really depends on the use. Note that strings are NOT a metric space.

4 R kernels

Skipping over R kernels for the most part. The material is in the back of Chapter 11: Section 11.7.2.

$$K_R(x,z) \quad = \quad \sum_{\vec{x} \in R^{-1}(x)} \sum_{\vec{z} \in R^{-1}(z)} [T(x) = T(z)] \prod_{i=1}^{|T(x)|} k_i(x_i, z_i)$$

$$R(((x_1,k_1),(x_2,k_2)...(x_d,k_d)),x) \quad = \quad \begin{cases} 1 & \text{if x is decomposible in this way} \\ 0 & \text{otherwise} \end{cases}$$

- now inverse is all decompositions of a given x

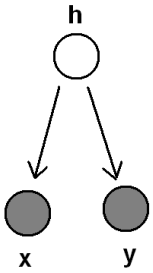
$$T(x) \quad = \quad (k_1, k_2, ...k_d)$$

- T(x) captures all of the kernels we have seen so far
- one can prove that this is always a kernel

5 P kernels

Using a probability distribution p, we can define a kernel that compares objects by assigning a high score if the two objects have high joint probability:

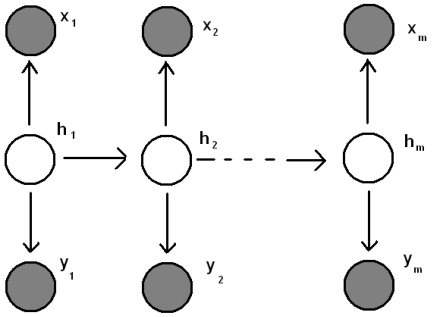
$$k(x,y) \quad = \quad \sum_h p(x|h)p(y|h)p(h)$$



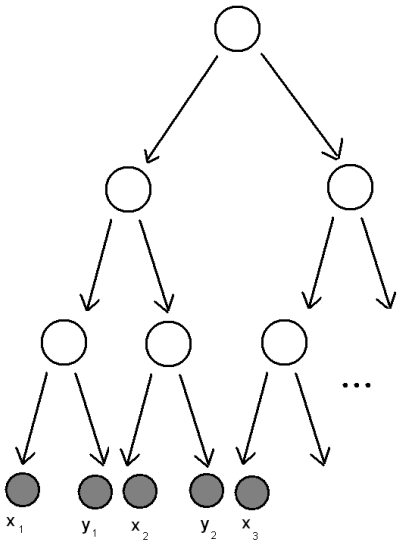
This is a kernel if the two probability distributions ($p(x|h)$ and $p(y|h)$ are the same. The features become:

$$\phi(x) \quad = \quad p(x|h)\sqrt{p(h)}$$

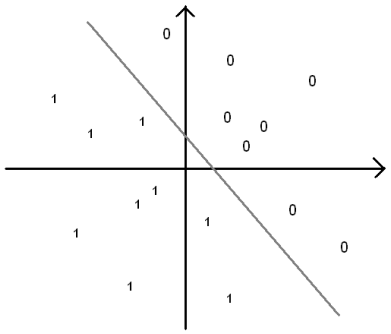
We could have a graphical model that looks like:



or a hidden markov tree:



We can use the methods from CS281A/Stat241A to efficiently compute the sums over these models.



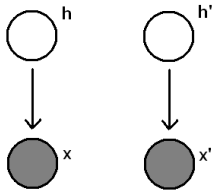
- only need to model the boudary
- use mediocre model to train SVM
- this is "discriminative modelling"

6 Marginalized kernels

- Marginalized kernels are not covered in the book

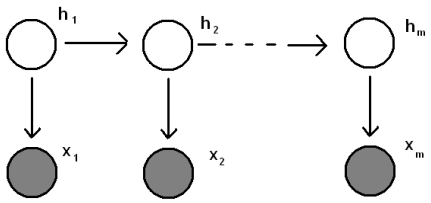
In this model we assume that we are given a graphical model involving observables x and x' with latent variables h and h' and a kernel $k(z, z')$ where $z = (x, h)$ and $z' = (x', h')$. Using these assumptions, we define the kernel as:

$$k(x, x') = \sum_{h,h'} p(h|x)p(h'|x')k(z, z')$$



Here, h could be a string parsing. The kernel would operate on the parsing and the observables. Since we don't know the actual parsing, we use $p(h|x)$ instead.

We can use many different graphical models, including an HMM:



This is a kernel because it is a non-negative linear sum of positive semidefinite functions