CSCE 990 Lecture 9:
Designing Kernels*

Stephen D. Scott

March 23, 2006

Introduction

- We are now very aware of the importance and power of kernels in SVMs
- We also know from Chapter 2 about some basic kernels and simple ways to build new kernels out of old ones
  - Linear scaling, addition, multiplication, etc. of existing kernels
- We'll look at other ways to construct new kernels from existing ones, plus other completely different types of kernels
- Some of them might look familiar ...

Outline

- Tricks for constructing kernels
- String kernels
- Spectrum kernels
- Locality-improved kernels
- Kernels defined on graphs
- Sections 13.1–13.3, 13.5, assorted papers

Tricks for Constructing Kernels

- If k_1 and k_2 are kernels, then so are

  $\alpha_1 k_1 + \alpha_2 k_2$ for $\alpha_1, \alpha_2 \geq 0$

  $\Rightarrow$ If input vectors can be partitioned into subvectors of different types (e.g. strings and real values), can apply direct sum:

  $$(k_1 \oplus k_2)(x_1, x_2, x_1', x_2') = k_1(x_1, x_1') + k_2(x_2, x_2')$$

  where $x_1, x_1' \in \mathcal{X}_1$ (e.g. $\mathbb{R}^n$) and $x_2, x_2' \in \mathcal{X}_2$ (e.g. strings)

  $k_1 k_2$

  $\Rightarrow$ Similar to application of direct sum, use tensor product:

  $$(k_1 \otimes k_2)(x_1, x_2, x_1', x_2') = k_1(x_1, x_1')\, k_2(x_2, x_2')$$

Tricks for Constructing Kernels
Conformal Transformations

- For a real-valued function f, $k'(x, x') = f(x)f(x')$ is a kernel

- This leads to conformal transformations:

$$k_f(x, x') = f(x)k(x, x')f(x')$$

  - If k is a kernel, then so is k_f
  - Recall that if $\|x\| = \|x'\| = 1$, then $\langle x, x' \rangle = \cos(\angle(x, x'))$; thus

$$\begin{aligned}\cos(\angle(\Phi_f(x), \Phi_f(x'))) &= \frac{f(x)k(x,x')f(x')}{\sqrt{f(x)k(x,x)f(x)}\sqrt{f(x')k(x',x')f(x')}} \\ &= \frac{k(x,x')}{\sqrt{k(x,x)}\sqrt{k(x',x')}} \\ &= \cos(\angle(\Phi(x), \Phi(x')))\end{aligned}$$

I.e. angles in feature space are preserved in a conformal transformation

Tricks for Constructing Kernels
Convolution Kernels

- Notions of tensor products and direct sums lead to R-convolution kernels

- E.g. consider partitioning the string $x = ATG$ into two distinct, contiguous, nonemtpy substrings:

$$R_1: \quad x_{1,R_1} = A \quad \text{AND} \quad x_{2,R_1} = TG$$

$$\text{OR}$$

$$R_2: \quad x_{1,R_2} = AT \quad \text{AND} \quad x_{2,R_2} = G$$

(similarly, decompose x')

- Now can compute a kernel for each substring of each partitioning and combine:

$$\begin{aligned}k(x, x') &= k_1(x_{1,R_1}, x'_{1,R_1})k_2(x_{2,R_1}, x'_{2,R_1}) \\ &+ k_1(x_{1,R_2}, x'_{1,R_2})k_2(x_{2,R_2}, x'_{2,R_2})\end{aligned}$$

Tricks for Constructing Kernels
Convolution Kernels (cont'd)

- Generally, define the set of allowed decompositions as a relation $R(x_1, \ldots, x_D, x)$ and define the R-convolution

$$(k_1 \star \cdots \star k_D)(x, x') := \sum_R \prod_{d=1}^{D} k_d(x_d, x'_d)$$

(i.e. sum over all allowable decompositions of x into $x_1, \ldots, x_D$, etc.)

- Based on earlier results, we know this to be a valid kernel

- A special case: ANOVA kernel of order D

$$k_D(x, x') := \sum_{1 \le i_1 < \cdots < i_D \le N} \prod_{d=1}^{D} k^{(i_d)}(x_{i_d}, x'_{i_d})$$

($D = N \Rightarrow$ tensor prod, $D = 1 \Rightarrow$ direct sum)

String Kernels

- To apply SVMs to text classification, can map documents to bag-of-words representation and use kernels defined on $\mathbb{R}^n$
  - Each dimension is one word, value in that dimension is word frequency
  - Ignores word ordering

- Alternatively, can use a string kernel, which computes similarities between two strings based on their common substrings

- Related to R-convolution kernel

String Kernels
(cont'd)

- Let Σ be a finite alphabet, Σ^n be set of all length-n strings over Σ, and $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$
- Given $s \in \Sigma^*$, let $\mathbf{i} := (i_1, \ldots, i_{|u|})$ be an index sequence with $1 \leq i_1 < \cdots < i_{|u|} \leq |s|$ and $u := s(\mathbf{i}) := s(i_1) \cdots s(i_{|u|})$ be a (possibly non-contiguous) subsequence of s
- $l(\mathbf{i}) := i_{|u|} - i_1 + 1$ is the length of u in s
  - E.g. if $s = ABBA$, then $l(1,2,3) = 3$ (for ABB), $l(1,4) = 4$ (for AA)
  - $\Phi_n(s)$ defines one dimension per substring $u \in \Sigma^n$, and the uth component of $\Phi_n(s)$ is

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

for $0 < \lambda \leq 1$

String Kernels
(cont'd)

- E.g. if $s = ABBA$, then $[\Phi_2(s)]_{AB} = \lambda^2 + \lambda^3$
- $[\Phi_n(s)]_u$ larger if u (nearly) contiguous and common in s
- The string kernel is then

$$\begin{aligned} k_n(s,t) &= \sum_{u \in \Sigma^n} [\Phi_n(s)]_u [\Phi_n(t)]_u \\ &= \sum_{u \in \Sigma^n} \sum_{(\mathbf{i},\mathbf{j}):s(\mathbf{i})=t(\mathbf{j})=u} \lambda^{l(\mathbf{i})} \lambda^{l(\mathbf{j})} \end{aligned}$$

- If want to vary n, use $k := \sum_n c_n k_n$
- Since value of k_n (and therefore k) depend on lengths of s and t, normalize k in feature space

String Kernels
(cont'd)

- To efficiently compute the kernel, define for $i = 1, \ldots, n-1$

$$k'_i(s,t) = \sum_{u \in \Sigma^i} \sum_{(\mathbf{i},\mathbf{j}):s(\mathbf{i})=t(\mathbf{j})=u} \lambda^{|s|+|t|-i_1-j_1+2}$$

- Then if $x \in \Sigma^1$, can recursively compute $k_n(s,t)$:

$$\begin{aligned} &k'_0(s,t) = 1 \quad \text{for all } s,t \\ &k'_i(s,t) = 0 \quad \text{if } \min(|s|,|t|) < i \\ &k_i(s,t) = 0 \quad \text{if } \min(|s|,|t|) < i \end{aligned}$$

$$\begin{aligned} k'_i(sx,t) &= \lambda k'_i(s,t) + \sum_{j:t_j=x} k'_{i-1}(s,t[1,\ldots,j-1])\lambda^{|t|-j+2} \\ k_n(sx,t) &= k_n(s,t) + \sum_{j:t_j=x} k'_{n-1}(s,t[1,\ldots,j-1])\lambda^2 \end{aligned}$$

Spectrum Kernel

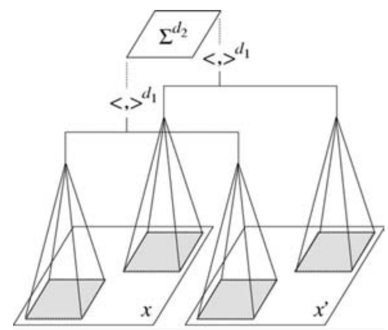
- Another type of string kernel
- For a fixed integer $\gamma \geq 1$, define the γ-spectrum of a sequence to be the set of all length-γ contiguous sequences it contains
- Feature map for spectrum kernel is indexed by all possible length-γ subsequences from alphabet Σ (similar to bag of words)
- For each $a \in \Sigma^\gamma$, let $\phi_a(x)$ = number of times a occurs in x contiguously
- Now define $\Phi_\gamma(x) = (\phi_a(x))_{a \in \Sigma^\gamma}$
  - This is a weighted representation of x's γ-spectrum
  - A sparse vector

Spectrum Kernels
(cont'd)

- Can compute $k_\gamma(x, x') = \langle \Phi_\gamma(x), \Phi_\gamma(x') \rangle$ in time $O(|x| + |x'|)$
  1. Collect set of length-γ subsequences of x into array A_x and sort it (same with x')
     - A_x contains non-zero entries of $\Phi_\gamma(x)$
  2. Scan A_x and $A_{x'}$, multiplying entries that match, and sum the products

Locality-Improved Kernels

- A variation on existing kernels to emphasize local correlations over long-range (global) ones
- E.g. in image processing, replace polynomial kernel $\langle x, x' \rangle^d$ with a variant that focuses on subimages first
- Generally, take the dot product over all corresponding subimages of the two images, raise to the d_1 power, sum these values, then raise to the d_2 power



Locality-Improved Kernels
Image Processing (cont'd)

- Specifically:
  1. Compute $(x. * x')$, the pixel-wise product of x and x'
  2. Sample $(x.*x')$ with pyramidal receptive fields:
  $$z_{ij} := \sum_{i',j'} w(\max(|i - i'|, |j - j'|))(x. * x')_{i'j'}$$
  where e.g. weighting function $w(n) = \max(q - n, 0)$; i.e. only include pixels in a width-p window ($p = 2q + 1$) centered at (i, j)
  3. Raise each z_{ij} to the d_1 power (this gives local correlations)
  4. Sum $z_{ij}^{d_1}$ over entire image and raise this sum to the d_2 power (long-range correlations)
- If $d_1 = 1$, get standard polynomial kernel $\langle x, x' \rangle^{d_2}$

Locality-Improved Kernels
Image Processing (cont'd)

Classifier	Error on MNIST (%)
$k^{1,4}$	4.0
$k_9^{2,2}$	3.1
$k_9^{4,1}$	3.4
Virt SV	2.8
VSV $k_9^{2,2}$	2.0

Locality-Improved Kernels
DNA Start Codon Recognition

- Problem: in a DNA sequence (from alphabet $\{A, C, T, G\}$), identify subsequences that encode genes
  - Typically such a coding region begins with ATG
  - But not all ATG occurrences imply a coding region
  - Thus the learning problem is to take a length-200 window centered at an ATG and predict if it's a coding region
- For this problem, long-range dependencies aren't very important, so use a kernel to emphasize local correlations

Locality-Improved Kernels
DNA Start Codon Recognition (cont'd)

- We'll consider correlations inside small windows of length $2\ell + 1$:

$$\text{win}_p(x, x') = \left(\sum_{j=-\ell}^{+\ell} v_j \text{match}_{p+j}(x, x') \right)^{d_1}$$

  where $\text{match}_{p+j}(x, x') = 1$ if x and x' match at position $p + j$ and 0 otherwise, and v_j is a weight for window position j (larger near 0)

- Now we sum the values of win_p:

$$k(x, x') = \left(\sum_{p=1}^{\ell} \text{win}_p(x, x') \right)^{d_2}$$

  (Should summation really be only to ℓ?)

Classifier	Error (%)
ANN	15.4
Poly kernel, $d = 1$	13.8
L-I kernel, $d_1 = 4, \ell = 4$	11.9
Codon-improved kernel, $d_1 = 2, \ell = 3$	12.2

Kernels on Graphs

- Very general form of structured data
- Can represent many data types, including chemical structures
- Will consider directed graphs with labels on edges and nodes
- Let $\mathcal{G}$ be the space of all graphs, modulo isomorphism

Complete Graph Kernels

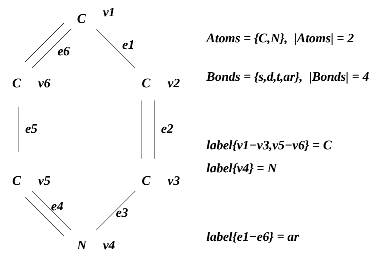
- A complete graph kernel k is one whose implicit remapping $\Phi : \mathcal{G} \to \mathcal{H}$ distinguishes all pairs of graphs $(G, G') \in \mathcal{G} \times \mathcal{G}$, i.e. Φ is injective
- Example (Subgraph feature space): Let each dimension in $\Phi(G)$ correspond to a distinct connected graph $H \in \mathcal{G}$. Then $[\Phi(G)]_H =$ number of times an isomorphism of H appears in G.
- Gärtner et al. [2003] showed that for injective Φ, $k(G, G) + k(G', G') - 2k(G, G') = \langle \Phi(G) - \Phi(G'), \Phi(G) - \Phi(G') \rangle = 0$ iff $G \simeq G'$

  ⇒ Computing k is as hard as graph isomorphism, for which no efficient algorithm is currently known
- Further, the kernel for the subgraph mapping is in fact NP-hard to compute (reduce from Hamiltonian path), even to approximate and/or if H comes from a restricted class of graphs

Kernels Based on Label Pairs

- Now consider more restrictive kernels that can be efficiently considered
- Focus on graphs with labels on nodes but not edges; labels come from $\mathcal{L} = \{\ell_1, \ldots, \ell_m\}$
- Let label matrix L be such that $[L]_{ri} = 1$ if node v_i's label is ℓ_r and $[L]_{ri} = 0$ otherwise
- Let adjacency matrix E be such that $[E]_{ij} = 1$ if directed edge (v_i, v_j) exists in graph G and $[E]_{ij} = 0$ otherwise; $[E^n]_{ij}$ is number of length-n walks from v_i to v_j
- $\left[LL^\top\right]_{rr}$ = number of times label ℓ_r is assigned to a vertex in G
- $\left[LE^nL^\top\right]_{ij}$ = number of walks of length n between vertices labeled ℓ_i and vertices labeled ℓ_j

Matrix Example



$$L = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad E = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$LEL^\top = \begin{bmatrix} 8 & 2 \\ 2 & 0 \end{bmatrix} \quad LE^2L^\top = \begin{bmatrix} 18 & 2 \\ 2 & 2 \end{bmatrix}$$

Kernels Based on Label Pairs (cont'd)

- $\mathcal{W}_n(G)$ = set of all n-edge walks in G
- For walk $w \in \mathcal{W}_n(G)$, $l_1(w)$ = label of first vertex of w and $l_{n+1}(w)$ = label of last vertex
- λ = sequence of nonnegative weights $\lambda_0, \lambda_1, \ldots$
- Define mapping $\Phi(G)$ to have one feature per pair of labels (ℓ_i, ℓ_j): $[\Phi(G)]_{\ell_i,\ell_j} =$

$$\sum_{n=0}^{\infty} \lambda_n \left| \left\{ w \in \mathcal{W}_n(G) : l_1(w) = \ell_i \wedge l_{n+1}(w) = \ell_j \right\} \right|$$

i.e. the weighted sum of the number of length-n walks from an ℓ_i-labeled vertex to an ℓ_j-labeled vertex, weighted by λ_n, summed over all $n \to \infty$

Kernels Based on Label Pairs (cont'd)

- Thus kernel is $\langle \Phi(G), \Phi(G') \rangle =$

$$\left\langle L \left(\sum_{i=0}^{\infty} \lambda_i E^i \right) L^\top, L' \left(\sum_{i=0}^{\infty} \lambda_i E'^i \right) L'^\top \right\rangle$$

- Under certain conditions, can efficiently compute the matrix power series
- E.g. if $\lambda_i = \beta^i / i!$ for some $\beta > 0$ and if E can be diagonalized such that $E = T^{-1}DT$, then $E^n = T^{-1}D^nT$ and $[D^n]_{ii} = [D_{ii}]^n$ since D is diagonal
- Now we can compute

$$\lim_{n\to\infty} \sum_{i=0}^{n} \frac{(\beta E)^i}{i!}$$

as

$$T^{-1} \left(\lim_{n\to\infty} \sum_{i=0}^{n} \frac{\beta^i D^i}{i!} \right) T \quad ,$$

where limits are taken component-wise

Kernels Based on Contiguous Label Sequences

- Previous kernel's mapping Φ has a low-dimensional feature space: $|\mathcal{L}|^2$
  - $\Rightarrow$ E.g. if all node labels are C or N, then feature space has dimension 4
- For a more expressive feature mapping, will use mapping with one dimension per label sequence rather than label pair
- Assume we have labels for both nodes and edges; if nodes or edges are not labeled, use generic symbol '#'

Kernels Based on Contiguous Label Sequences (cont'd)

- Let $\mathcal{S}_n$ be set of all possible label sequences of walks with n edges and let λ, $\mathcal{W}_n(G)$, and $l_i(w)$ be as before
- Define mapping $\Phi(G)$ to have one feature per possible label sequence $s \in \bigcup_n \mathcal{S}_n$:

$$[\Phi(G)]_s = \sqrt{\lambda_n}\,|\{w \in \mathcal{W}_n(G) : \forall i \;\; s_i = l_i(w)\}|$$

i.e. the number of walks in G with n edges whose (vertex and edge) label sequences match $s = s_1, s_2, \ldots, s_{2n+1} \in \mathcal{S}_n$, weighted by $\sqrt{\lambda_n}$

Kernels Based on Contiguous Label Sequences (cont'd)

- To compute the kernel, use the notion of a product graph: given $G_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $G_2 = (\mathcal{V}_2, \mathcal{E}_2)$, $G_\times = G_1 \times G_2$ is defined as

$$\mathcal{V}_\times = \{(v_1, v_2) \in \mathcal{V}_1 \times \mathcal{V}_2 : label(v_1) = label(v_2)\}$$

$$\mathcal{E}_\times = \{((u_1, u_2), (v_1, v_2)) \in \mathcal{V}_\times^2 : (u_1, v_1) \in \mathcal{E}_1$$

$$\wedge (u_2, v_2) \in \mathcal{E}_2 \wedge label(u_1, v_1) = label(u_2, v_2)\}$$

- One can show that

$$|\{w \in \mathcal{W}_n(G_1 \times G_2) : \forall i \;\; s_i = l_i(w)\}|$$

$$= |\{w \in \mathcal{W}_n(G_1) : \forall i \;\; s_i = l_i(w)\}|$$

$$\cdot\, |\{w \in \mathcal{W}_n(G_2) : \forall i \;\; s_i = l_i(w)\}|$$

- Since an n-edge walk in $G_1 \times G_2$ corresponds to a walk in each of G_1 and G_2, each with same label sequence, the dot product $\langle \Phi(G_1), \Phi(G_2) \rangle$ can be computed as

$$k_\times(G_1, G_2) = \sum_{i,j=1}^{\mathcal{V}_\times} \left[\sum_{n=0}^{\infty} \lambda_n E_\times^n \right]_{ij}$$

Topic summary due in 1 week!