The kernels for life, universe and everything

Tomas Singliar

CS3750 Advanced Machine Learning

Overview

- SVM
- Design requirements and considerations
- Design approaches
- Examples
  - String kernels
  - Tree kernels
  - Graph kernels
- Conclusion and questions

SVM

- n datapoints $\boldsymbol{x}_i$
- Two classes: $y_i = +1$ and $y_i = -1$
- We search for hyperplane separating the classes
- Hyperplane not unique – want max-margin hyperplane
- Learning is quadratic optimization of Lagrange parameters α_i
- $\alpha_i = 0$ for all points except those on boundary – the *support vectors*
- Classification of new datapoint (bias weight in)

$$y = \mathrm{sgn}(\mathbf{w}^T \mathbf{x}) = \mathrm{sgn}\left(\sum_{i \in SV} \alpha_i y_i (\mathbf{x}_i^T \mathbf{x}) \right)$$

Kernels

- The dot product $\mathbf{x}^T \mathbf{x}$ is a distance measure
  - precisely cosine of angle if normalized
- Kernels can be seen as distance measures
  - Or conversely express degree of similarity
- Design criteria - we want kernels to be
  - **valid** – Satisfy Mercer condition of positive semidefiniteness
  - **good** – embody the “true similarity” between objects
  - **appropriate** – generalize well
  - **efficient** – the computation of k(x,x’) is feasible
    - NP-hard problems abound with graphs

Concept classes and good kernels

- Valid - Mercer positive semidefiniteness condition
- Concept – mapping $c : X \to \{0,1\}$
- Concept class – set of concepts
- Kernel is **complete** iff it is “fine-grained” enough

$$\forall c : k(x,\cdot) = k(x',\cdot) \Rightarrow c(x) = c(x')$$

- Kernel is **correct** (wrt a concept class C) iff

$$\forall c \in C \exists \alpha_i : \sum_i \alpha_i k(x_i, x) \geq 0 \Leftrightarrow c(x)$$

i.e. if an SVM (with perfect separation) can be learned with it

Appropriate & computable kernels

- We want kernels that generalize well
- Matching kernel $k(x,x') = \delta(x,x')$
  - always correct, always complete, mostly useless
- Correctness & completeness ~ training performance
- Appropriateness ~ testing (generalization) perf.
- We want realistically computable kernels:
  - $k(x,x') = (c(x) == c(x'))$ is great
  - but solves the **whole** problem
  - can be NP-hard or non-computable

Design of kernels

- Two approaches to kernel design
  - Model driven
    - encodes knowledge about domain
    - From generative models: Fisher kernel
    - Diffusion kernel – local relationships
    - Ex. : Hidden Markov models DNA sequences, speech
  - Syntax driven
    - exploits structure of problem – special case or parameter
    - Ex.: strings, trees, terms

Model based kernels – Fisher kernel

- Knowledge about the objects to classify in form of a generative probability model
- Fisher information matrix
  - sensitivity of probability to parameters at x ~ variance
  - Cramer-Rao bound: $\mathrm{var}(x_i) \geq I_{ii}^{-1}$

$$U_x = \nabla_\theta \log P(x \mid \theta) \qquad I = \left\langle U_x U_x^T \right\rangle_{P(x|\theta)}$$

- Fisher kernel

$$k_F(x, x') = U_x^T I^{-1} U_{x'}$$

- performs well if class is latent variable in the model
- used widely for sequence data (HMM)
- I^{-1} is sometimes dropped (also drops requirement on the matrix)

Matrix exponents and diffusion kernels

- Instance space has local relations
- Generator matrix H, kernel matrix $K = e^{\beta H}$
- Key identity is Taylor expansion $e^{x} = \lim_{n \to \infty} \sum_{i=0}^{n} \frac{x^i}{i!}$
- So $e^{\beta H} = \lim_{n \to \infty} \sum_{i=0}^{n} \frac{\beta^i H^i}{i!}$
- H is symmetric $\Rightarrow e^{\beta H}$ is positive semidefinite
- β - bandwidth parameter
  - as β grows, local structure encoded by H propagates
  - results in global structure
- Diffusion comes from MRF dynamics
  - covariance of the field at time t is

$$Cov(t) = \sigma^2 e^{2\alpha t H}$$

The Convolution kernel

- Syntax-driven kernel – defined (recursively) on structure
- Idea is compositional semantics – define semantics of object as function of their parts' semantics
- Let $x, x' \in X$ be the objects of X and let $\vec{x}, \vec{\underline{x}}' \in X_1, \ldots, X_D$ be tuples of parts of *x, x'* , let R be 'is composed of'
- Then convolution kernel is given by

$$k_{conv}(x, x') = \sum_{\vec{x} \in R^{-1}(x), \vec{x}' \in R^{-1}(x')} \prod_{d} k_d(x_d, x_d')$$

- Can be adapted to virtually everything
- But it's a long way to go

A String kernel

- Similarity of strings: common subsequences
- Example: *cat* and *cart*
  - Common: *'c', 'a', 't', 'ca', 'at', 'ct', 'cat'*
  - Exponential penalty for longer gaps: λ
  - Result: k('cat','cart') = $2\lambda^7 + \lambda^5 + \lambda^4 + 3\lambda^2$
- Feature transformation $\varphi(s)$:
  - *s[i]* -- subsequence of s induced by index set *i*
  - *l(i) = max(i) – min(i)* – length of *i* in *s*
  - $$\varphi_u(s) = \sum_{i:u=s[i]} \lambda^{l(i)}$$
- The kernel is given by

$$k_n(s,t) = \sum_{u\in\Sigma^n} \varphi_u(s)\varphi_u(t) = \sum_{u\in\Sigma^n}\sum_{i:u=s[i]}\sum_{j:u=s[j]} \lambda^{l(i)+l(j)}$$

Another string kernel

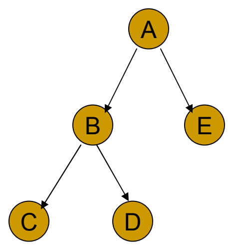
- A sliding window kernel for DNA sequences
- Classification: inition site or not
  - inition site – codon where translation begins
- Locality-improved kernel

$$k_i(x,x') = \left(\sum_{j=-l}^{l} w_j k_\delta(x_{i+j}, x'_{i+j})\right)^{d_1} \qquad k(x,x') = \left(\sum_{j=l}^{n-l} k_i(x,x')\right)^{d_2}$$

- results competitive with previous approaches
- probabilistic: replace x_i with log $p(x_i = \text{init} \mid x_{i-1})$ ("bigram")
- parameter d_1 – weight on local match

kernels

- We can encode a tree as a string by traversing in preorder and parenthesizing
- Then we can use a string kernel



tag(T)= (A(B(C)(D))(E))

- Tag can be computed in loglinear time
- Uniquely identifies the tree
- Substrings correspond to sub**set** trees
- Balanced substrings correspond to subtrees

Tree kernels

- Syntax driven kernel
- V_1, V_2 are sets of vertices of T_1, T_2
- $\delta^+(v)$ is the set of children of v, $\delta^+(v,j)$ is the j-th child
- $S(v_1,v_2)$ is the number of isomorphic subtrees of v_1,v_2
    - $S(v_1,v_2) = 1$ if labels match and no children
    - $S(v_1,v_2) = 0$ if labels don't match
    - otherwise

$$k(T_1,T_2) = \sum_{v_1 \in V_1, v_2 \in V_2} S(v_1,v_2) \qquad S(v_1,v_2) = \prod_{k=1}^{|\delta^+(v_1)|} (1 + S(\delta(v_1,j),\delta(v_2,j)))$$

- This has $O(|V_1||V_2|)$ complexity

Graphs

- Complexity a more important issue – things get NP-hard
- If you can do many walks through nodes labeled by the same names in two graphs, they are similar
- This process can be modeled as diffusion: Model driven kernel
  - Take negative Laplacian of adjacency matrix for the generator
    - $H_{ij} = 1$ if (v_i, v_j) is an edge
    - $H_{ij} = |N(v_i)|$ if $v_i = v_j$
    - $H_{ij} = 0$ otherwise
  - $K = e^{\beta H}$
- Or directlySyntactic kernel based on walks
  - Construct product graph
  - Count the 1-step walks that you do in both graphs: E_x^1
  - 2-step walks: E_x^2, 3-step walks E_x^3 , ….
    - Discounting for convergence

$$k_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} \left[\sum_{n=0}^{\infty} \lambda_i E_\times^n \right]$$

Applications and conclusions

- Kernel methods are popular and useful
  - Computational biology: gene identification, phylogenetic profiles clustering, genus prediction,
  - Computational (bio)chemistry: molecule shape prediction from NMR spectrum, drug activity prediction
  - Natural language processing: parse tree similarity, n-gram kernels,
- Syntactic and information-theoretic approach
- Design your own kernels for any type of object you deal with
  - Intuition: measure similarity between objects
  - Verify that your kernel is good and appropriate
  - Some (graph) problems are hard
    - tradeoff between fast and appropriate kernels
- SVM implementations exist that allow user-definable kernels
  - www.kernel-machines.org

- Thank you!
- Questions welcome!