CS485/685
Lecture 9: Jan 31, 2012

Kernel methods

[B] Sections 6.1, 6.2

Non-linear Models Recap

- Generalized linear models:

- Neural networks:

Kernel Methods

- Idea: use large (possibly infinite) set of fixed non-linear basis functions
- Normally, complexity depends on number of basis functions, but by a “dual trick”, **complexity depends on the amount of data**
- Examples:
  - **Gaussian Processes** (next class)
  - **Support Vector Machines** (next week)
  - Kernel Perceptron
  - Kernel Principal Component Analysis

Kernel Function

- Let $\phi(\boldsymbol{x})$ be a set of basis functions that map inputs x to a feature space.
- In many algorithms, this feature space only appears in the dot product $\phi(\boldsymbol{x})^T\phi(\boldsymbol{x}')$ of pairs inputs $\boldsymbol{x}, \boldsymbol{x}'$.
- Define the kernel function $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T\phi(\boldsymbol{x}')$ to be the dot product of any pair x, x' in feature space.
  - **We only need to know $\boldsymbol{k}(\boldsymbol{x}, \boldsymbol{x}')$**, not $\phi(\boldsymbol{x})$

Dual Representations

- Recall linear regression objective

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}[\boldsymbol{w}^T\phi(\boldsymbol{x}_n) - y_n]^2 + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$$

- Solution: set gradient to 0

$$\nabla E(\boldsymbol{w}) = \sum_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n) - y_n)\phi(\boldsymbol{x}_n) + \lambda\boldsymbol{w} = 0$$

$$\boldsymbol{w} = -\frac{1}{\lambda}\sum_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n) - y_n)\phi(\boldsymbol{x}_n)$$

$\therefore$ $\boldsymbol{w}$ is a linear combination of inputs in feature space $\{\phi(\boldsymbol{x}_n)|1 \le n \le N\}$

Dual Representations

- Substitute $\mathbf{w} = \boldsymbol{\Phi a}$
- Where $\boldsymbol{\Phi} = [\phi(\boldsymbol{x}_1)\ \phi(\boldsymbol{x}_2)\ \ldots\ \phi(\boldsymbol{x}_N)]$

$$\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix} \quad \text{and } a_n = -\frac{1}{\lambda}(\boldsymbol{w}^T\phi(\boldsymbol{x}_n) - y_n)$$

- Dual objective: minimize E with respect to $\boldsymbol{a}$

$$E(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{a} - \boldsymbol{a}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{y} + \frac{\boldsymbol{y}^T\boldsymbol{y}}{2} + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{a}$$

Gram Matrix

- Let $\boldsymbol{K} = \boldsymbol{\Phi}^T\boldsymbol{\Phi}$ be the Gram matrix
- Substitute in objective:

$$E(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^{\boldsymbol{T}}\boldsymbol{K}\boldsymbol{K}\boldsymbol{a} - \boldsymbol{a}^{\boldsymbol{T}}\boldsymbol{K}\boldsymbol{y} + \frac{\boldsymbol{y}^{\boldsymbol{T}}\boldsymbol{y}}{2} + \frac{\lambda}{2}\boldsymbol{a}^{\boldsymbol{T}}\boldsymbol{K}\boldsymbol{a}$$

- Solution: set gradient to 0

$$\nabla E(\boldsymbol{a}) = \boldsymbol{K}\boldsymbol{K}\boldsymbol{a} - \boldsymbol{K}\boldsymbol{y} + \lambda\boldsymbol{K}\boldsymbol{a} = 0$$
$$\boldsymbol{K}(\boldsymbol{K} + \lambda\boldsymbol{I})\boldsymbol{a} = \boldsymbol{K}\boldsymbol{y}$$
$$\boldsymbol{a} = (\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y}$$

- Prediction:
  $y_n = \boldsymbol{w}^T\phi(\boldsymbol{x}_n) = \boldsymbol{a}^{\boldsymbol{T}}\boldsymbol{\Phi}\phi(\boldsymbol{x}_{\boldsymbol{n}}) = k(\cdot, \boldsymbol{x}_{\boldsymbol{n}})^T(\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y}$
  where $k(\cdot, \boldsymbol{x}_n) = K(\boldsymbol{x}, \boldsymbol{x}_{\boldsymbol{n}})\ \forall\boldsymbol{x}$

Dual Linear Regression

- Prediction: $y_n = \boldsymbol{a}^{\boldsymbol{T}}\boldsymbol{\Phi}\phi(\boldsymbol{x}_n)$
  $= k(\cdot, \boldsymbol{x}_n)^T(\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y}$
- Linear regression where we find dual solution $\boldsymbol{a}$ instead of primal solution **w**.
- Complexity:
  - Primal solution: # of basis functions
  - Dual solution: amount of data
    - Advantage: can use very large # of basis functions
    - Just need to know kernel k

Constructing Kernels

- Two possibilities:
  - Find mapping $\boldsymbol{\phi}$ to feature space and let $\boldsymbol{K} = \boldsymbol{\phi}^{\boldsymbol{T}}\boldsymbol{\phi}$
  - Directly specify $\boldsymbol{K}$
- Can any function that takes two argument serve as a kernel?
- No, a valid kernel must be positive semi-definite
  - In other words, k must factor into the product of a transposed matrix by itself (e.g., $\boldsymbol{K} = \boldsymbol{\phi}^{\boldsymbol{T}}\boldsymbol{\phi}$)
  - Or, all eigenvalues must be greater than or equal to 0.

Example

- Let $k(\boldsymbol{x}, \boldsymbol{z}) = \left(\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{z}\right)^2$

Constructing Kernels

- Can we construct k directly without knowing ϕ?
- Yes, any positive semi-definite k is fine since there is a corresponding implicit feature space. But positive semi-definiteness is not always easy to verify.
- Alternative, construct kernels from other kernels using rules that preserve positive semi-definiteness

Rules to construct Kernels

- Let $k_1(\boldsymbol{x}, \boldsymbol{x}')$ and $k_2(\boldsymbol{x}, \boldsymbol{x}')$ be valid kernels
- The following kernels are also valid:
  1. $k(\boldsymbol{x}, \boldsymbol{x}') = c k_1(\boldsymbol{x}, \boldsymbol{x}') \quad \forall c > 0$
  2. $k(\boldsymbol{x}, \boldsymbol{x}') = f(\boldsymbol{x}) k_1(\boldsymbol{x}, \boldsymbol{x}') f(\boldsymbol{x}') \quad \forall f$
  3. $k(\boldsymbol{x}, \boldsymbol{x}') = q(k_1(\boldsymbol{x}, \boldsymbol{x}'))$ q is polynomial with coeffs ≥ 0
  4. $k(\boldsymbol{x}, \boldsymbol{x}') = \exp(k_1(\boldsymbol{x}, \boldsymbol{x}'))$
  5. $k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}')$
  6. $k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}') k_2(\boldsymbol{x}, \boldsymbol{x}')$
  7. $k(\boldsymbol{x}, \boldsymbol{x}') = k_3(\phi(\boldsymbol{x}), \phi(\boldsymbol{x}'))$
  8. $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^{\boldsymbol{T}} \boldsymbol{A} \boldsymbol{x}'$ $\boldsymbol{A}$ is symmetric positive semi-definite
  9. $k(\boldsymbol{x}, \boldsymbol{x}') = k_a(\boldsymbol{x}_{\boldsymbol{a}}, \boldsymbol{x}'_a) + k_b(\boldsymbol{x}_{\boldsymbol{b}}, \boldsymbol{x}'_b)$
  10. $k(\boldsymbol{x}, \boldsymbol{x}') = k_a(\boldsymbol{x}_a, \boldsymbol{x}'_a) k_b(\boldsymbol{x}_b, \boldsymbol{x}'_b)$

Common Kernels

- Polynomial kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \boldsymbol{x}')^M$
  - M is the degree
  - Feature space: all degree M products of entries in $\boldsymbol{x}$
  - Example: Let $\boldsymbol{x}$ and $\boldsymbol{x}'$ be two images, then feature space could be all products of M pixel intensities
- More general polynomial kernel:
  $$k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \boldsymbol{x}' + c)^M \text{ with } c > 0$$
  - Feature space: all products of up to M entries in $\boldsymbol{x}$

Common Kernels

- Gaussian Kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\left\|\boldsymbol{x}-\boldsymbol{x}'\right\|^2}{2\sigma^2}\right)$
- Valid Kernel because:
- Implicit feature space is infinite!

Non-vectorial Kernels

- Kernels can be defined with respect to other things than vectors such as sets, strings or graphs
- Example for sets: $k(A_1, A_2) = 2^{|A_1 \cap A_2|}$