Kernel Methods

Greg Mori - CMPT 419/726

Bishop PRML Ch. 6

Non-linear Mappings

- In the lectures on linear models for regression and classification, we looked at models with $\boldsymbol{w}^T\phi(\boldsymbol{x})$
- The feature space $\phi(\boldsymbol{x})$ could be high-dimensional
- This was good because if data aren't separable in original input space $(\boldsymbol{x})$, they may be in feature space $\phi(\boldsymbol{x})$
- We'd like to avoid computing high-dimensional $\phi(\boldsymbol{x})$
- We'd like to work with $\boldsymbol{x}$ which doesn't have a natural vector-space representation
  - e.g. graphs, sets, strings

Kernel Trick

- In previous lectures on linear models, we would explicitly compute $\phi(\boldsymbol{x}_i)$ for each datapoint
  - Run algorithm in feature space
- For some feature spaces, can compute dot product $\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$ efficiently
- Efficient method is computation of a kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$
- The kernel trick is to rewrite an algorithm to only have $\boldsymbol{x}$ enter in the form of dot products
- The menu:
  - Kernel trick examples
  - Kernel functions

A Kernel Trick

- Let's look at the nearest-neighbour classification algorithm
- For input point x_i, find point x_j with smallest distance:

$$\begin{aligned} ||\boldsymbol{x}_i - \boldsymbol{x}_j||^2 &= (\boldsymbol{x}_i - \boldsymbol{x}_j)^T(\boldsymbol{x}_i - \boldsymbol{x}_j) \\ &= \boldsymbol{x}_i{}^T\boldsymbol{x}_i - 2\boldsymbol{x}_i{}^T\boldsymbol{x}_j + \boldsymbol{x}_j{}^T\boldsymbol{x}_j \end{aligned}$$

- If we used a non-linear feature space $\phi(\cdot)$:

$$\begin{aligned} ||\phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_j)||^2 &= \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_i) - 2\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j) + \phi(\boldsymbol{x}_j)^T\phi(\boldsymbol{x}_j) \\ &= k(\boldsymbol{x}_i, \boldsymbol{x}_i) - 2k(\boldsymbol{x}_i, \boldsymbol{x}_j) + k(\boldsymbol{x}_j, \boldsymbol{x}_j) \end{aligned}$$

- So nearest-neighbour can be done in a high-dimensional feature space without actually moving to it

A Kernel Function

- Consider the kernel function $k(\boldsymbol{x},\boldsymbol{z}) = (1+\boldsymbol{x}^T\boldsymbol{z})^2$

- With $\boldsymbol{x},\boldsymbol{z} \in \mathbb{R}^2$,

$$\begin{aligned} k(\boldsymbol{x},\boldsymbol{z}) &= (1+x_1z_1+x_2z_2)^2 \\ &= 1+2x_1z_1+2x_2z_2+x_1^2z_1^2+2x_1z_1x_2z_2+x_2^2z_2^2 \\ &= (1,\sqrt{2}x_1,\sqrt{2}x_2,x_1^2,\sqrt{2}x_1x_2,x_2^2)(1,\sqrt{2}z_1,\sqrt{2}z_2,z_1^2,\sqrt{2}z_1z_2,z_2^2)^T \\ &= \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{z}) \end{aligned}$$

- So this particular kernel function does correspond to a dot product in a feature space (is valid)
- Computing $k(\boldsymbol{x},\boldsymbol{z})$ is faster than explicitly computing $\boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{z})$
  - In higher dimensions, larger exponent, much faster

Why Kernels?

- Why bother with kernels?
  - Often easier to specify how similar two things are (dot product) than to construct explicit feature space ϕ.
  - There are high-dimensional (even infinite) spaces that have efficient-to-compute kernels
    - Separability
- So you want to use kernels
  - Need to know when kernel function is valid, so we can apply the kernel trick

Valid Kernels

- Given some arbitrary function $k(\boldsymbol{x}_i,\boldsymbol{x}_j)$, how do we know if it corresponds to a dot product in some space?
- Valid kernels: if $k(\cdot,\cdot)$ satisfies:
  - Symmetric; $k(\boldsymbol{x}_i,\boldsymbol{x}_j) = k(\boldsymbol{x}_j,\boldsymbol{x}_i)$
  - Positive definite; for any $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N$, the Gram matrix $\boldsymbol{K}$ must be positive semi-definite:

$$\boldsymbol{K} = \begin{pmatrix} k(\boldsymbol{x}_1,\boldsymbol{x}_1) & k(\boldsymbol{x}_1,\boldsymbol{x}_2) & \ldots & k(\boldsymbol{x}_1,\boldsymbol{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_N,\boldsymbol{x}_1) & k(\boldsymbol{x}_N,\boldsymbol{x}_2) & \ldots & k(\boldsymbol{x}_N,\boldsymbol{x}_N) \end{pmatrix}$$

  - Positive semi-definite means $\boldsymbol{x}^T\boldsymbol{K}\boldsymbol{x} \geq 0$ for all $\boldsymbol{x}$

  then $k(\cdot,\cdot)$ corresponds to a dot product in some space ϕ
  - a.k.a. Mercer kernel, admissible kernel, reproducing kernel

Examples of Kernels

- Some kernels:
  - Linear kernel $k(\boldsymbol{x}_1,\boldsymbol{x}_2) = \boldsymbol{x}_1^T\boldsymbol{x}_2$
    - $\phi(\boldsymbol{x}) = \boldsymbol{x}$
  - Polynomial kernel $k(\boldsymbol{x}_1,\boldsymbol{x}_2) = (1+\boldsymbol{x}_1^T\boldsymbol{x}_2)^d$
    - Contains all polynomial terms up to degree d
  - Gaussian kernel $k(\boldsymbol{x}_1,\boldsymbol{x}_2) = \exp(-||\boldsymbol{x}_1-\boldsymbol{x}_2||^2/2\sigma^2)$
    - Infinite dimension feature space

Constructing Kernels

- Can build new valid kernels from existing valid ones:
  - $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = ck_1(\boldsymbol{x}_1, \boldsymbol{x}_2)$, $c > 0$
  - $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k_1(\boldsymbol{x}_1, \boldsymbol{x}_2) + k_2(\boldsymbol{x}_1, \boldsymbol{x}_2)$
  - $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k_1(\boldsymbol{x}_1, \boldsymbol{x}_2)k_2(\boldsymbol{x}_1, \boldsymbol{x}_2)$
  - $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp(k_1(\boldsymbol{x}_1, \boldsymbol{x}_2))$
- Table on p. 296 gives many such rules

More Kernels

- Stationary kernels are only a function of the difference between arguments: $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k(\boldsymbol{x}_1 - \boldsymbol{x}_2)$
  - Translation invariant in input space: $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k(\boldsymbol{x}_1 + \boldsymbol{c}, \boldsymbol{x}_2 + \boldsymbol{c})$
- Homogeneous kernels, a. k. a. radial basis functions only a function of magnitude of difference: $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k(||\boldsymbol{x}_1 - \boldsymbol{x}_2||)$
- Set subsets $k(A_1, A_2) = 2^{|A_1 \cap A_2|}$, where $|A|$ denotes number of elements in A
- Domain-specific: think hard about your problem, figure out what it means to be similar, define as $k(\cdot, \cdot)$, prove positive definite (Feynman algorithm)

Perceptron Classifier - Kernelized

- Recall the perceptron $y(\boldsymbol{x}) = f(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}))$
- The update rule for the perceptron is

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} + \underbrace{\eta\boldsymbol{\phi}(\boldsymbol{x}_n)t_n}_{if\ \ incorrect}$$

- Hence,

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(0)} + \alpha_1\boldsymbol{\phi}(\boldsymbol{x}_1) + \alpha_2\boldsymbol{\phi}(\boldsymbol{x}_2) + \ldots \alpha_N\boldsymbol{\phi}(\boldsymbol{x}_N)$$

- The classifier is then

$$f(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x})) = f(\boldsymbol{w}^{(0),T}\boldsymbol{\phi}(\boldsymbol{x}) + \alpha_1\boldsymbol{\phi}(\boldsymbol{x}_1)^T\boldsymbol{\phi}(\boldsymbol{x}) + \alpha_2\boldsymbol{\phi}(\boldsymbol{x}_2)^T\boldsymbol{\phi}(\boldsymbol{x}) + \ldots)$$

- Kernelized! (init $\boldsymbol{w}^{(0)} = \boldsymbol{0}$)
- Similar trick can be done for the update rule

Regression - Kernelized

- Regularized least squares regression can also be kernelized
- Kernelized solution is

$$y(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^T(\boldsymbol{K} + \lambda\boldsymbol{I}_N)^{-1}\boldsymbol{t} \quad \text{vs.} \quad \phi(\boldsymbol{x})(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\boldsymbol{I}_M)^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}$$

  for original version
- N is number of datapoints (size of Gram matrix $\boldsymbol{K}$)
- M is number of basis functions (size of matrix $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$)
- Bad if $N > M$, but good otherwise

Conclusion

- Readings: Ch. 6.1-6.2 (pp. 291-297)
- Many algorithms can be re-written with only dot products of features
  - We've seen NN, perceptron, regression; also PCA, SVMs (later)
- Non-linear features, or domain-specific similarity measurements are useful
- Dot products of non-linear features, or similarity measurements, can be written as kernel functions
  - Validity by positive semi-definiteness of kernel function
- Can have algorithm work in non-linear feature space without actually mapping inputs to feature space
  - Advantageous when feature space is high-dimensional