Kernel Machines

Andrea Passerini
passerini@disi.unitn.it

Machine Learning

Kernel Machines

Kernel trick

- Feature mapping $\Phi(\cdot)$ can be very high dimensional (e.g. think of polynomial mapping)
- It can be highly expensive to explicitly compute it
- Feature mappings **appear only in dot products** in dual formulations
- The *kernel trick* consists in replacing these dot products with an equivalent kernel function:

$$k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^T \Phi(\mathbf{x}')$$

- The kernel function uses examples in input (not feature) space

Kernel trick

Support vector classification

- Dual optimization problem

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^m} \quad \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \underbrace{\Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j)}_{k(\mathbf{x}_i, \mathbf{x}_j)}$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C \quad i = 1, \ldots, m$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0$$

- Dual decision function

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i \underbrace{\Phi(\mathbf{x}_i)^T \Phi(\mathbf{x})}_{k(\mathbf{x}_i, \mathbf{x})}$$

Kernel trick

Polynomial kernel

- Homogeneous:

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^d$$

- E.g. $(d = 2)$

$$\begin{aligned}
k\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}\right) &= (x_1 x_1' + x_2 x_2')^2 \\
&= (x_1 x_1')^2 + (x_2 x_2')^2 + 2 x_1 x_1' x_2 x_2' \\
&= \underbrace{\begin{pmatrix} x_1^2 & \sqrt{2} x_1 x_2 & x_2^2 \end{pmatrix}^T}_{\Phi(\mathbf{x})^T} \underbrace{\begin{pmatrix} x_1'^2 \\ \sqrt{2} x_1' x_2' \\ x_2'^2 \end{pmatrix}}_{\Phi(\mathbf{x}')}
\end{aligned}$$

Kernel trick

Polynomial kernel

- Inhomogeneous:

$$k(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^d$$

- E.g. $(d = 2)$

$$k\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}\right) = (1 + x_1 x_1' + x_2 x_2')^2$$

$$= 1 + (x_1 x_1')^2 + (x_2 x_2')^2 + 2x_1 x_1' + 2x_2 x_2' + 2x_1 x_1' x_2 x_2'$$

$$= \underbrace{\begin{pmatrix} 1 & \sqrt{2}x_1 & \sqrt{2}x_2 & x_1^2 & \sqrt{2}x_1 x_2 & x_2^2 \end{pmatrix}^T}_{\Phi(\mathbf{x})^T} \underbrace{\begin{pmatrix} 1 \\ \sqrt{2}x_1' \\ \sqrt{2}x_2' \\ x_1'^2 \\ \sqrt{2}x_1' x_2' \\ x_2'^2 \end{pmatrix}}_{\Phi(\mathbf{x}')}$$

Valid Kernels

Dot product in feature space

- A valid kernel is a (similarity) function defined in cartesian product of input space:

$$k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$$

- corresponding to a dot product in a (certain) feature space:

$$k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^T \Phi(\mathbf{x}')$$

Note

- The kernel generalizes the notion of dot product to arbitrary input space (e.g. protein sequences)
- It can be seen as a measure of similarity between objects

Valid Kernels

Gram matrix

- Given examples $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ and kernel function k
- The *Gram matrix* K is the (symmetric) matrix of pairwise kernels between examples:

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) \quad \forall i, j$$

Valid Kernels

Positive definite matrix

- A symmetric $m \times m$ matrix K is *positive definite* (p.d.) if

$$\sum_{i,j=1}^{m} c_i c_j K_{ij} \geq 0, \quad \forall \mathbf{c} \in \mathbb{R}^m$$

  If equality only holds for $\boldsymbol{c} = \mathbf{0}$, the matrix is *strictly positive definite* (s.p.d)

Alternative conditions

- All eigenvalues are non-negative (positive for s.p.d.)
- There exists a matrix B such that

$$K = B^T B$$

Valid Kernels

Positive definite kernels

- A positive definite kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ giving rise to a p.d. Gram matrix for any m and $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$
- Positive definiteness is necessary and sufficient condition for a kernel to correspond to a dot product of *some* feature map Φ

How to verify kernel validity

- Prove its positive definiteness (difficult)
- Find out a corresponding feature map (see polynomial example)
- Use kernel combination properties (we'll see)

Support vector regression

- Dual problem:

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^m} \quad -\frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)\underbrace{\Phi(\mathbf{x}_i)^T\Phi(\mathbf{x}_j)}_{k(\mathbf{x}_i,\mathbf{x}_j)}$$

$$-\epsilon\sum_{i=1}^{m}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{m} y_i(\alpha_i^* - \alpha_i)$$

$$\text{subject to} \quad \sum_{i=1}^{m}(\alpha_i - \alpha_i^*) = 0 \quad \alpha_i, \alpha_i^* \in [0, C] \quad \forall i \in [1, m]$$

- Regression function:

$$f(\mathbf{x}) = \mathbf{w}^T\Phi(\mathbf{x}) + w_0 = \sum_{i=1}^{m}(\alpha_i - \alpha_i^*)\underbrace{\Phi(\mathbf{x}_i)^T\Phi(\mathbf{x})}_{k(\mathbf{x}_i,\mathbf{x})} + w_0$$

Kernel machines

(Stochastic) Perceptron: $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

1. Initialize $\mathbf{w} = \mathbf{0}$
2. Iterate until all examples correctly classified:
   1. For each incorrectly classified training example $(\boldsymbol{x}_i, y_i)$:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$$

Kernel Perceptron: $f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \mathbf{x})$

1. Initialize $\alpha_i = 0 \; \forall i$
2. Iterate until all examples correctly classified:
   1. For each incorrectly classified training example $(\boldsymbol{x}_i, y_i)$:

$$\alpha_i \leftarrow \alpha_i + \eta y_i$$

Kernels

Basic kernels

- linear kernel:

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$

- polynomial kernel:

$$k_{d,c}(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^d$$

Kernels

Gaussian kernel

$$k_\sigma(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right) = \exp\left(-\frac{\mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\mathbf{x}' + \mathbf{x}'^T\mathbf{x}'}{2\sigma^2}\right)$$

- Depends on a *width* parameter σ
- The smaller the width, the more prediction on a point only depends on its nearest neighbours
- Example of *Universal* kernel: they can uniformly approximate any arbitrary continuous target function (pb of number of training examples and choice of σ)

Kernels

Kernels on structured data

- Kernels are generalization of dot products to arbitrary domains
- It is possible to design kernels over structured objects like sequences, trees or graphs
- The idea is designing a pairwise function measuring the similarity of two objects
- This measure has to sastisfy the p.d. conditions to be a valid kernel

Match (or delta) kernel

$$k_\delta(x, x') = \delta(x, x') = \begin{cases} 1 & \text{if } x = x' \\ 0 & \text{otherwise.} \end{cases}$$

- Simplest kernel on structures
- x does not need to be a vector! (no boldface to stress it)

E.g. string kernel: 3-gram spectrum kernel

x = ABAABA x' = AAABB

$$\mathtt{x} \xrightarrow{\Phi(x)} \quad\quad \mathtt{x'} \xrightarrow{\Phi(x')}$$

$$\begin{array}{c} \mathrm{AAA} \\ \mathrm{AAB} \\ \mathrm{ABA} \\ \mathrm{ABB} \\ \mathrm{BAA} \\ \mathrm{BAB} \\ \mathrm{BBA} \\ \mathrm{BBB} \end{array} \quad \begin{pmatrix} 0 \\ 1 \\ 2 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad k(x, x') = 1$$

Kernels

Kernel combination

- Simpler kernels can combined using certain operators (e.g. sum, product)
- Kernel combination allows to design complex kernels on structures from simpler ones
- Correctly using combination operators guarantees that complex kernels are p.d.

Note

- Simplest constructive approach to build valid kernels

Kernel combination

Kernel Sum

- The sum of two kernels corresponds to the *concatenation* of their respective feature spaces:

$$\begin{aligned}(k_1 + k_2)(x, x') &= k_1(x, x') + k_2(x, x') \\ &= \Phi_1(x)^T\Phi_1(x') + \Phi_2(x)^T\Phi_2(x') \\ &= (\Phi_1(x)\ \Phi_2(x)) \begin{pmatrix} \Phi_1(x') \\ \Phi_2(x') \end{pmatrix}\end{aligned}$$

- The two kernels can be defined on **different** spaces (*direct* sum, e.g. string spectrum kernel plus string length)

Kernel combination

Kernel Product

- The product of two kernels corresponds to the Cartesian products of their features:

$$
\begin{aligned}
(k_1 \times k_2)(x, x') &= k_1(x, x') k_2(x, x') \\
&= \sum_{i=1}^{n} \Phi_{1i}(x)\Phi_{1i}(x') \sum_{j=1}^{m} \Phi_{2j}(x)\Phi_{2j}(x') \\
&= \sum_{i=1}^{n} \sum_{j=1}^{m} (\Phi_{1i}(x)\Phi_{2j}(x))(\Phi_{1i}(x')\Phi_{2j}(x')) \\
&= \sum_{k=1}^{nm} \Phi_{12k}(x)\Phi_{12k}(x') = \Phi_{12}(x)^T \Phi_{12}(x')
\end{aligned}
$$

- where $\Phi_{12}(x) = \Phi_1(x) \times \Phi_2(x)$ is the Cartesian product
- the product can be between kernels in different spaces (*tensor* product)

Kernel combination

Linear combination

- A kernel can be rescaled by an arbitrary positive constant: $k_\beta(x,x') = \beta k(x,x')$
- We can e.g. define linear combinations of kernels (each rescaled by the desired weight):

$$k_{sum}(x,x') = \sum_{k=1}^{K} \beta_k k_k(x,x')$$

Note

- The weights of the linear combination can be learned simultaneously to the predictor weights (the alphas)
- This amounts at performing *kernel learning*

Kernel combination

Decomposition kernels

- Use the combination operators (sum and product) to define kernels on structures.
- Rely on a decomposition relationship $R(x) = (x_1, \dots, x_D)$ breaking a structure into its *parts*

E.g. for strings

- $R(x) = (x_1, \dots, x_D)$ could be break string x into substrings such that $x_1 \circ \dots x_D = x$ (where $\circ$ is string concatenation)
- E.g. ($D = 3$, empty string not allowed):

$$\mathtt{x} = \mathtt{AAABB} \qquad \mathtt{R(x)} = \left\{ \begin{array}{ll} \mathtt{A\ A\ ABB} & \mathtt{AA\ A\ BB} \\ \mathtt{A\ AA\ BB} & \mathtt{AA\ AB\ B} \\ \mathtt{A\ AAB\ B} & \mathtt{AAA\ B\ B} \end{array} \right\}$$

Kernel combination

Convolution kernels

- decomposition kernels defining a kernel as the convolution of its parts:

$$(k_1 \star \cdots \star k_D)(x, x') = \sum_{(x_1,\dots,x_D) \in R(x)} \sum_{(x'_1,\dots,x'_D) \in R(x')} \prod_{d=1}^{D} k_d(x_d, x'_d)$$

- where the sums run over all possible decompositions of x and x'.

Convolution kernels

Set kernel

- Let $R(x)$ be the set membership relationship (written as $\in$)
- Let $k_{member}(\xi, \xi')$ be a kernel defined over set elements
- The set kernel is defined as:

$$k_{set}(X, X') = \sum_{\xi \in X} \sum_{\xi' \in X'} k_{member}(\xi, \xi')$$

Set intersection kernel

- For delta membership kernel we obtain:

$$k_{\cap}(X, X') = |X \cap X'|$$

Kernel combination

Kernel normalization

- Kernel values can often be influenced by the dimension of objects
- E.g. a longer string has more substrings $\rightarrow$ higher kernel value
- This effect can be reduced *normalizing* the kernel

Cosine normalization

- Cosine normalization computes the cosine of the dot product in feature space:

$$\hat{k}(x, x') = \frac{k(x, x')}{\sqrt{k(x, x)k(x', x')}}$$

Kernel combination

Kernel composition

- Given a kernel over structured data $k(x, x')$
- it is always possible to use a basic kernel on top of it, e.g.:

$$
\begin{aligned}
(k_{d,c} \circ k))(x, x') &= (k(x, x') + c)^d \\
(k_\sigma \circ k)(x, x') &= \exp\left(-\frac{k(x, x) - 2k(x, x') + k(x', x')}{2\sigma^2}\right)
\end{aligned}
$$

- it corresponds to the **composition** of the mappings associated with the two kernels
- E.g. all possible conjunctions of up to d k-grams for string kernels

References

kernel trick: C. Burges, *A tutorial on support vector machines for pattern recognition*, Data Mining and Knowledge Discovery, 2(2), 121-167, 1998.

kernel properties: J.Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, 2004 (Section 3)

kernels: J.Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, 2004 (Section 9)