Lecture 11. Kernel Methods

COMP90051 Statistical Machine Learning

Semester 2, 2017
Lecturer: Andrey Kan

THE UNIVERSITY OF MELBOURNE

This lecture

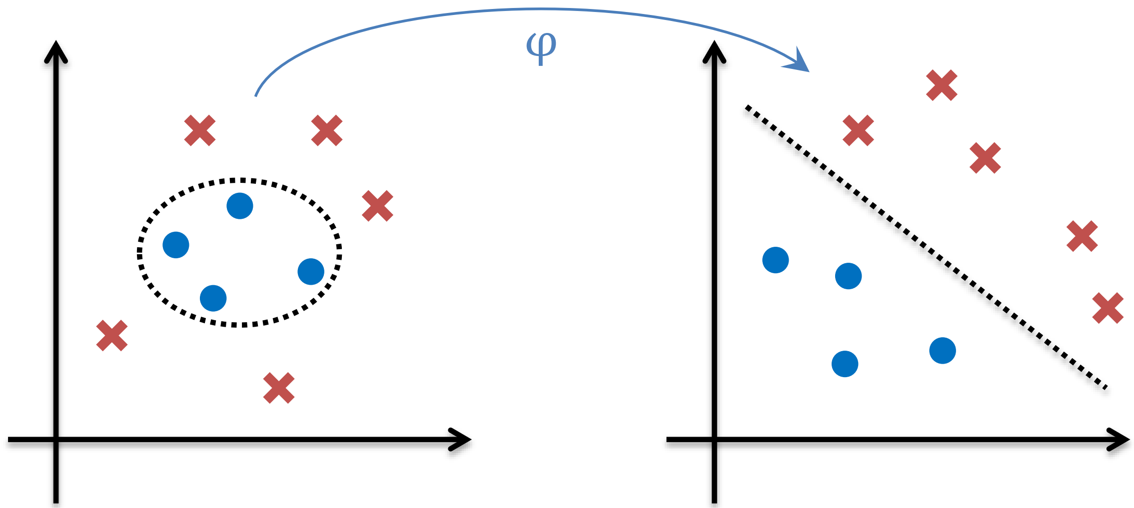
- The kernel trick
  - Efficient computation of a dot product in transformed feature space
- Modular learning
  - Separating the "learning module" from feature space transformation
- Constructing kernels
  - An overview of popular kernels and their properties
- Kernel as a similarity measure
  - Extending machine learning beyond conventional data structure

The Kernel Trick

An approach that we introduce in the context of SVMs. However, this approach is compatible with a large number of methods

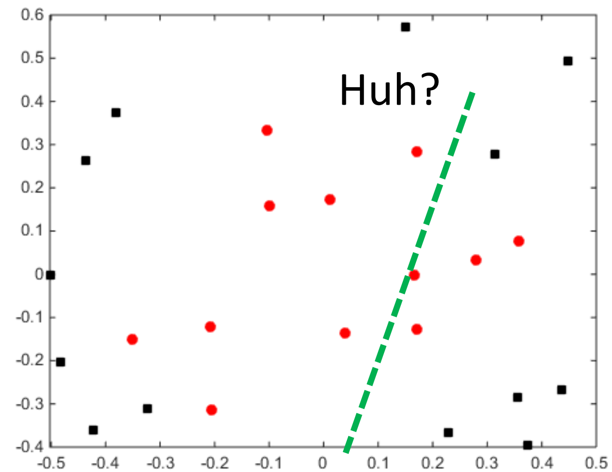
Handling non-linear data with SVM

- Method 1: Soft margin SVM
- Method 2: Feature space transformation
  - Map data into a new feature space
  - Run hard margin or soft margin SVM in new space
  - Decision boundary is non-linear in original space

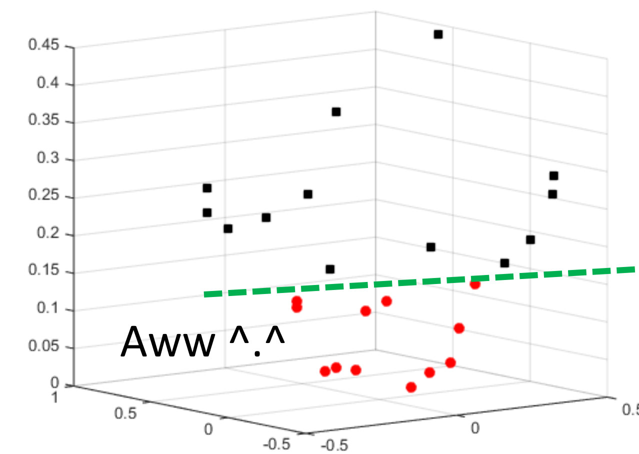


Example of feature transformation

- Consider a binary classification problem
- Each example has features $[x_1, x_2]$
- Not linearly separable



- Now 'add' a feature $x_3 = x^2 + x_2^2$
- Each point is now $[x_1, x_2, x_1^2 + x_2^2]$
- Linearly separable!



Naïve workflow

- Choose/design a linear model
- Choose/design a high-dimensional transformation $\varphi(\boldsymbol{x})$
  - Hoping that after adding a lot of various features some of them will make the data linearly separable
- For each training example, and for each new instance compute $\varphi(\boldsymbol{x})$
- Train classifier/Do predictions

- Problem: impractical/impossible to compute $\varphi(\boldsymbol{x})$ for high/infinite-dimensional $\varphi(\boldsymbol{x})$

Hard margin SVM

- Training: finding $\boldsymbol{\lambda}$ that solve

dot-product

$$\underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j$$

s.t. $\lambda_i \geq 0$ and $\sum_{i=1}^{n} \lambda_i y_i = 0$

- Making predictions: classify new instance $\boldsymbol{x}$ based on the sign of

dot-product

$$s = b^* + \sum_{i=1}^{n} \lambda_i^* y_i \boldsymbol{x}_i' \boldsymbol{x}$$

- Here b^* can be found by noting that for arbitrary training example j we must have $y_j\left(b^* + \sum_{i=1}^{n} \lambda_i^* y_i \boldsymbol{x}_i' \boldsymbol{x}_j\right) = 1$

Hard margin SVM

- <u>Training</u>: finding $\boldsymbol{\lambda}$ that solve

$$\underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \varphi(\boldsymbol{x}_i)' \varphi(\boldsymbol{x}_j)$$

s.t. $\lambda_i \geq 0$ and $\sum_{i=1}^{n} \lambda_i y_i = 0$

- <u>Making predictions</u>: classify new instance $\boldsymbol{x}$ based on the sign of

$$s = b^* + \sum_{i=1}^{n} \lambda_i^* y_i \varphi(\boldsymbol{x}_i)' \varphi(\boldsymbol{x})$$

- Here b^* can be found by noting that for arbitrary training example j we must have $y_j\left(b^* + \sum_{i=1}^{n} \lambda_i^* y_i \varphi(\boldsymbol{x}_i)' \varphi(\boldsymbol{x}_j)\right) = 1$

Observation: Dot product representation

- Both parameter estimation and computing predictions depend on data only in a form of a dot product
  - In original space $\boldsymbol{u}'\boldsymbol{v} = \sum_{i=1}^{m} u_i v_i$
  - In transformed space $\varphi(\boldsymbol{u})'\varphi(\boldsymbol{v}) = \sum_{i=1}^{l} \varphi(\boldsymbol{u})_i \varphi(\boldsymbol{v})_i$

- Kernel is a function that can be expressed as a dot product in some feature space $K(\boldsymbol{u}, \boldsymbol{v}) = \varphi(\boldsymbol{u})'\varphi(\boldsymbol{v})$

Example of a kernel

- For some feature maps there exists a shortcut computation of the dot product via kernels
- For example, consider two vectors original space $\boldsymbol{u} = [u_1]$ and $\boldsymbol{v} = [v_1]$ and a transformation $\varphi(\boldsymbol{x}) = [x_1^2, \sqrt{2c}x_1, c]$
- So $\varphi(\boldsymbol{u}) = \left[u_1^2, \sqrt{2c}u_1, c\right]'$ and $\varphi(\boldsymbol{v}) = \left[v_1^2, \sqrt{2c}v_1, c\right]'$
- Then $\varphi(\boldsymbol{u})'\varphi(\boldsymbol{v}) = (u_1^2 v_1^2 + 2cu_1v_1 + c^2)$
- This can be alternatively computed as
$$\varphi(\boldsymbol{u})'\varphi(\boldsymbol{v}) = (u_1v_1 + c)^2$$
- Here $K(\boldsymbol{u}, \boldsymbol{v}) = (u_1v_1 + c)^2$ is a kernel

The kernel trick

- Consider two training points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and their dot product in the transformed space. Define this quantity as $k_{ij} \equiv \varphi(\boldsymbol{x}_i)'\varphi(\boldsymbol{x}_j)$
- This can be computed as:
  1. Compute $\varphi(\boldsymbol{x}_i)'$
  2. Compute $\varphi(\boldsymbol{x}_j)$
  3. Compute $k_{ij} = \varphi(\boldsymbol{x}_i)'\varphi(\boldsymbol{x}_j)$
- However, for some transformations φ, there exists a function that gives exactly the same answer $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = k_{ij}$
  * In other words, sometimes there is a different way ("shortcut") to compute the same quantity k_{ij}
- This different way, <u>does not involve steps 1 – 3</u>. In particular, we do not need to compute $\varphi(\boldsymbol{x}_i)$ and $\varphi(\boldsymbol{x}_j)$
  * Usually kernels can be computed in $O(m)$, whereas computing $\varphi(\boldsymbol{x})$ requires $O(l)$, where $l \gg m$ or $l = \infty$

Hard margin SVM

- Training: finding $\boldsymbol{\lambda}$ that solve

$$\underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \varphi(\boldsymbol{x}_i)' \varphi(\boldsymbol{x}_j)$$

$$\text{s.t. } \lambda_i \geq 0 \text{ and } \sum_{i=1}^{n} \lambda_i y_i = 0$$

- Making predictions: classify new instance $\boldsymbol{x}$ based on the sign of

$$s = b^* + \sum_{i=1}^{n} \lambda_i^* y_i \varphi(\boldsymbol{x}_i)' \varphi(\boldsymbol{x})$$

- Here b^* can be found by noting that for arbitrary training example j we must have $y_j\left(b^* + \sum_{i=1}^{n} \lambda_i^* y_i \varphi(\boldsymbol{x}_i)' \varphi(\boldsymbol{x}_j)\right) = 1$

Hard margin SVM

- Training: finding $\boldsymbol{\lambda}$ that solve

feature mapping is implied by kernel

$$\underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

s.t. $\lambda_i \geq 0$ and $\sum_{i=1}^{n} \lambda_i y_i = 0$

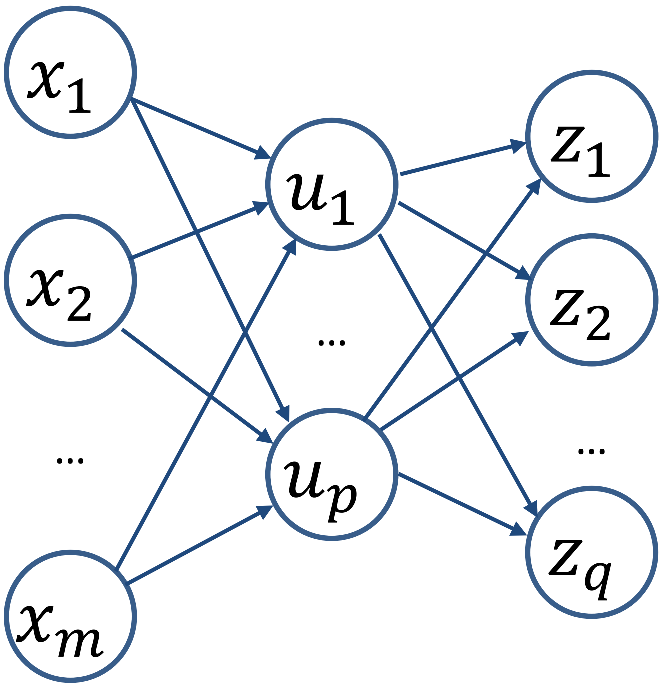
- Making predictions: classify new instance $\boldsymbol{x}$ based on the sign of

$$s = b^* + \sum_{i=1}^{n} \lambda_i^* y_i K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

feature mapping is implied by kernel

- Here b^* can be found by noting that for arbitrary training example j we must have $y_j \left(b^* + \sum_{i=1}^{n} \lambda_i^* y_i K\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)\right) = 1$

ANN approach to non-linearity



In this ANN, elements of $\boldsymbol{u}$ can be thought as the transformed input $\boldsymbol{u} = \varphi(\boldsymbol{x})$

This transformation is explicitly constructed by varying the ANN topology

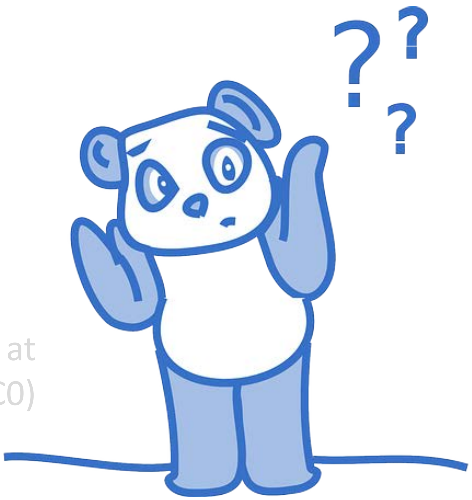
Moreover, the weights are learned from data

SVM approach to non-linearity

- Choosing a kernel implies some transformation $\varphi(\boldsymbol{x})$. Unlike ANN case, we don't have control over relative weights of components of $\varphi(\boldsymbol{x})$
- However, the advantage of using kernels is that we don't need to actually compute components of $\varphi(\boldsymbol{x})$. This is beneficial when the transformed space is multidimensional. In addition, it makes it possible to transform the data into an infinite-dimensional space
- Kernels also offer an additional advantage discussed in the last part of this lecture

Checkpoint

- Which of the following statements is always true?
  - Any method that uses a feature space transformation $\varphi(\boldsymbol{x})$ uses kernels
  - Support vectors are points from the training set
  - Feature mapping $\varphi(\boldsymbol{x})$ makes data linearly separable



Modular Learning

Separating the “learning module”
from feature space transformation

Representer theorem

- Theorem: a large class of linear methods can be formulated (represented) such that both training and making predictions require data only in a form of a dot product
- Hard margin SVM is one example of such a method
- The theorem predicts that there are many more. For example:
  - Ridge regression
  - Logistic regression
  - Perceptron
  - Principal component analysis
  - and so on …

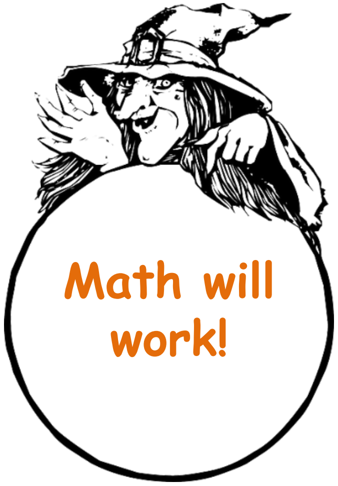


Image: ClkerFreeVectorImages@ pixabay.com (CC0)

Kernelised perceptron (1/3)

When classified correctly, weights are unchanged

When misclassified: $\boldsymbol{w}^{(k+1)} = -\eta(\pm\boldsymbol{x})$
($\eta > 0$ is called *learning rate)*

<u>If $y = 1$, but $s < 0$</u>
$w_i \leftarrow w_i + \eta x_i$
$w_0 \leftarrow w_0 + \eta$

<u>If $y = -1$, but $s \geq 0$</u>
$w_i \leftarrow w_i - \eta x_i$
$w_0 \leftarrow w_0 - \eta$

Suppose weights are initially set to 0

First update: $\boldsymbol{w} = \eta y_{i_1}\boldsymbol{x}_{i_1}$
Second update: $\boldsymbol{w} = \eta y_{i_1}\boldsymbol{x}_{i_1} + \eta y_{i_2}\boldsymbol{x}_{i_2}$
Third update $\boldsymbol{w} = \eta y_{i_1}\boldsymbol{x}_{i_1} + \eta y_{i_2}\boldsymbol{x}_{i_2} + \eta y_{i_3}\boldsymbol{x}_{i_3}$
etc.

Kernelised perceptron (2/3)

- Weights always take the form $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i$, where $\boldsymbol{\alpha}$ some coefficients
- Perceptron weights are always a linear combination of data!
- Recall that prediction for a new point $\boldsymbol{x}$ is based on sign of $w_0 + \boldsymbol{w}'\boldsymbol{x}$
- Substituting $\boldsymbol{w}$ we get $w_0 + \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i'\boldsymbol{x}$
- The dot product $\boldsymbol{x}_i'\boldsymbol{x}$ can be replaced with a kernel

Kernelised perceptron (3/3)

Choose initial guess $\boldsymbol{w}^{(0)}$, $k = 0$

Set $\boldsymbol{\alpha} = \mathbf{0}$

For t from 1 to T (epochs)

For each training example $\{\boldsymbol{x}_i, y_i\}$

Predict based on $w_0 + \sum_{j=1}^{n} \alpha_j y_j \boldsymbol{x}_i' \boldsymbol{x}_j$

If misclassified, update $\alpha_i \leftarrow \alpha_i + 1$

Modular learning

- All information about feature mapping is concentrated within the kernel
- In order to use a different feature mapping, simply change the kernel function
- Algorithm design decouples into choosing a “learning method” (e.g., SVM vs logistic regression) and choosing feature space mapping, i.e., kernel

Constructing Kernels

An overview of popular kernels
and kernel properties

A large variety of kernels

In this section, we review polynomial and Gaussian kernels

www.kernel-methods.net

Polynomial kernel

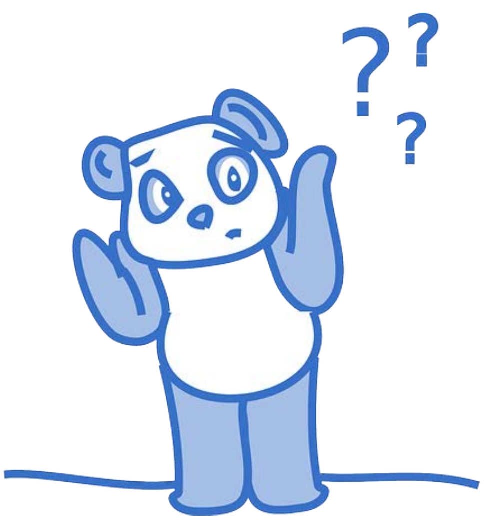
- Function $K(\boldsymbol{u},\boldsymbol{v}) = (\boldsymbol{u}'\boldsymbol{v}+c)^d$ is called ***polynomial kernel***
  * Here $\boldsymbol{u}$ and $\boldsymbol{v}$ are vectors with m components
  * $d \geq 0$ is an integer and $c \geq 0$ is a constant
- Without the loss of generality, assume $c = 0$
  * If it's not, add $\sqrt{c}$ as a dummy feature to $\boldsymbol{u}$ and $\boldsymbol{v}$
- $(\boldsymbol{u}'\boldsymbol{v})^d = (u_1v_1 + \cdots + u_mv_m)(u_1v_1 + \cdots + u_mv_m) \dots (u_1v_1 + \cdots + u_mv_m)$
- $= \sum_{i=1}^{l}(u_1v_1)^{a_{i1}} \dots (u_mv_m)^{a_{im}}$
  * Here $0 \leq a_{ij} \leq d$ and l are integers
- $= \sum_{i=1}^{l}\left(u_1^{a_{i1}} \dots u_m^{a_{im}}\right)\left(v_1^{a_{i1}} \dots v_m^{a_{im}}\right)$
- $= \sum_{i=1}^{l} \varphi(\boldsymbol{u})_i\varphi(\boldsymbol{v})_i$
- Feature map $\varphi: \mathbb{R}^m \rightarrow \mathbb{R}^l$, where $\varphi_i(\boldsymbol{x}) = \left(x_1^{a_{i1}} \dots x_m^{a_{im}}\right)$

Identifying new kernels

- Method 1: Using identities, such as below. Let $K_1(\boldsymbol{u},\boldsymbol{v})$, $K_2(\boldsymbol{u},\boldsymbol{v})$ be kernels, $c > 0$ be a constant, and $f(\boldsymbol{x})$ be a real-valued function. Then each of the following is also a kernel:
  - $K(\boldsymbol{u},\boldsymbol{v}) = K_1(\boldsymbol{u},\boldsymbol{v}) + K_2(\boldsymbol{u},\boldsymbol{v})$
  - $K(\boldsymbol{u},\boldsymbol{v}) = cK_1(\boldsymbol{u},\boldsymbol{v})$
  - $K(\boldsymbol{u},\boldsymbol{v}) = f(\boldsymbol{u})K_1(\boldsymbol{u},\boldsymbol{v})f(\boldsymbol{v})$
  - *See Bishop's book for more identities*

Prove these!

- Method 2: Using Mercer's theorem



art: OpenClipartVectors at pixabay.com (CC0)

Radial basis function kernel

- Function $K(\boldsymbol{u}, \boldsymbol{v}) = \exp(-\gamma \|\boldsymbol{u} - \boldsymbol{v}\|^2)$ is called ***radial basis function kernel*** (aka Gaussian kernel)
  - Here $\gamma > 0$ is the spread parameter
- $\exp(-\gamma \|\boldsymbol{u} - \boldsymbol{v}\|^2) = \exp\left(-\gamma (\boldsymbol{u} - \boldsymbol{v})'(\boldsymbol{u} - \boldsymbol{v})\right)$
- $= \exp\left(-\gamma (\boldsymbol{u}'\boldsymbol{u} - 2\boldsymbol{u}'\boldsymbol{v} + \boldsymbol{v}'\boldsymbol{v})\right)$
- $= \exp(-\gamma \boldsymbol{u}'\boldsymbol{u}) \exp(2\gamma \boldsymbol{u}'\boldsymbol{v}) \exp(-\gamma \boldsymbol{v}'\boldsymbol{v})$
- $= f(\boldsymbol{u}) \exp(2\gamma \boldsymbol{u}'\boldsymbol{v}) f(\boldsymbol{v})$
- $= f(\boldsymbol{u}) \left(\sum_{d=0}^{\infty} r_d (\boldsymbol{u}'\boldsymbol{v})^d\right) f(\boldsymbol{v})$

Power series expansion

- Here, each $(\boldsymbol{u}'\boldsymbol{v})^d$ is a polynomial kernel. Using kernel identities, we conclude that the middle term is a kernel, and hence the whole expression is a kernel

Mercer's Theorem

- Question: given $\varphi(\boldsymbol{u})$, is there a good kernel to use?
- Inverse question: given some function $K(\boldsymbol{u}, \boldsymbol{v})$, is this a valid kernel? In other words, is there a mapping $\varphi(\boldsymbol{u})$ implied by the kernel?

- Mercer's theorem:
  * Consider a finite sequences of objects $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$
  * Construct $n \times n$ matrix of pairwise values $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$
  * $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is a kernel if this matrix is positive-semidefinite, and this holds for all possible sequences $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$

Kernel as a Similarity Measure

Extending machine learning beyond conventional data structure

Yet another use of kernels

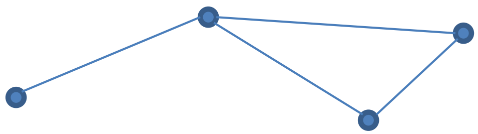
- Remember how (re-parameterised) SVM makes predictions. The prediction depends on the sign of $\left(b + \sum_{i=1}^{n} \lambda_i y_i \boldsymbol{x}_i' \boldsymbol{x}\right)$
  - So point $\boldsymbol{x}$ is “dot-producted” with each training support vector
- This term can be re-written using a kernel $\left(b + \sum_{i=1}^{n} \lambda_i y_i K(\boldsymbol{x}_i, \boldsymbol{x})\right)$
  - This can be seen as comparing $\boldsymbol{x}$ to each of the support vectors
- E.g., consider Gaussian kernel $K(\boldsymbol{x}_i, \boldsymbol{x}) = \exp(-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}\|^2)$
- Here $\|\boldsymbol{x}_i - \boldsymbol{x}\|$ is the distance between the points and $K(\boldsymbol{x}_i, \boldsymbol{x})$ is monotonically decreasing with the distance
- $K(\boldsymbol{x}_i, \boldsymbol{x})$ can be interpreted as a *similarity measure*

Kernel as a similarity measure

- More generally, any kernel $K(\boldsymbol{u}, \boldsymbol{v})$ can be viewed as a similarity measure: it maps two objects to a real number
- In other words, choosing/designing a kernel can be viewed as defining how to compare the objects
- This is a very powerful idea, because we can extend kernel methods to objects that are not vectors
- This is the first time in this course, when we are going to encounter a notion of a different data type
- So far, we've been concerned with vectors of fixed dimensionality, e.g., $\boldsymbol{x} = [x_1, \ldots, x_m]'$

Data comes in a variety of shapes

- But what if we wanted to do machine learning on …
- Graphs
  - Facebook, Twitter, …



- Sequences of variable lengths
  - “science is organized knowledge”, “wisdom is organized life”*, …
  - “CATTC”, “AAAGAGA”
- Songs, movies, etc.

* Both quotations are from Immanuel Kant

Handling arbitrary data structures

- Kernels offer a way to deal with the variety of data types
- For example, we could define a function that somehow measures similarity of variable length strings

K(“science is organized knowledge”, “wisdom is organized life”)

- However, not every function on two objects is a valid kernel
- Remember that we need that function $K(\boldsymbol{u}, \boldsymbol{v})$ to imply a dot product in some feature space

This lecture

- The kernel trick
  - Efficient computation of a dot product in transformed feature space
- Modular learning
  - Separating the “learning module” from feature space transformation
- Constructing kernels
  - An overview of popular kernels and their properties
- Kernel as a similarity measure
  - Extending machine learning beyond conventional data structure