# Lecture 11. Kernel Methods

COMP90051 Statistical Machine Learning

Semester 2, 2018
Lecturer: Ben Rubinstein

# This lecture

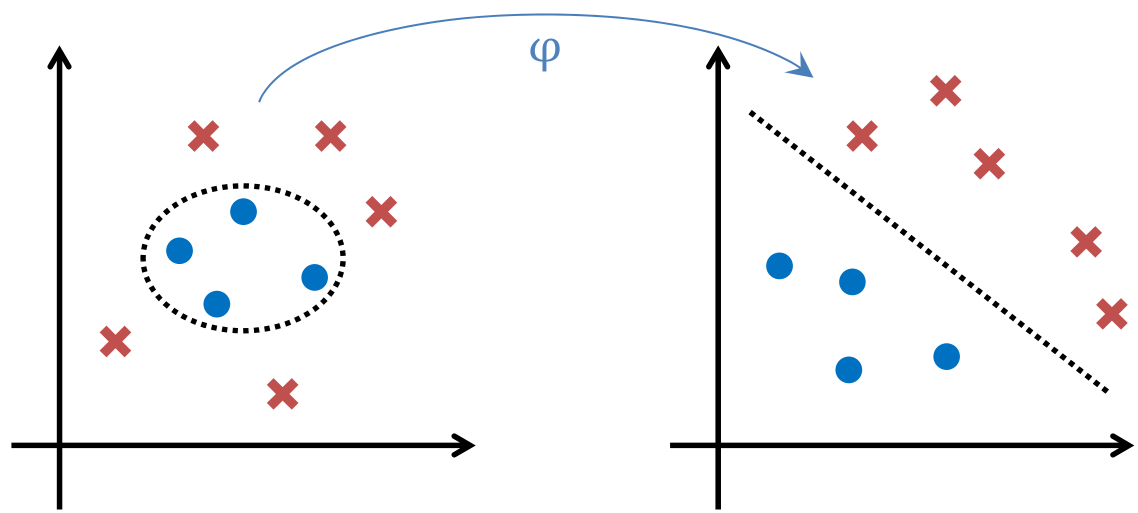
- Kernelisation
  - Basis expansion on dual formulation of SVMs
  - "Kernel trick"; Fast computation of feature space dot product
- Modular learning
  - Separating "learning module" from feature transformation
  - Representer theorem
- Constructing kernels
  - Overview of popular kernels and their properties
  - Mercer's theorem
  - Learning on unconventional data types

# Kernelising the SVM

Feature transformation by basis expansion; sped up by direct evaluation of kernels – the ‘kernel trick’

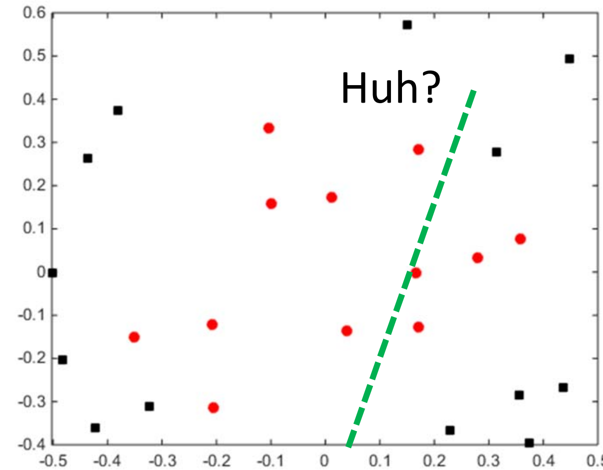
# Handling non-linear data with the SVM

- Method 1: Soft-margin SVM (deck #10)
- Method 2: Feature space transformation (deck #4)
  - 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

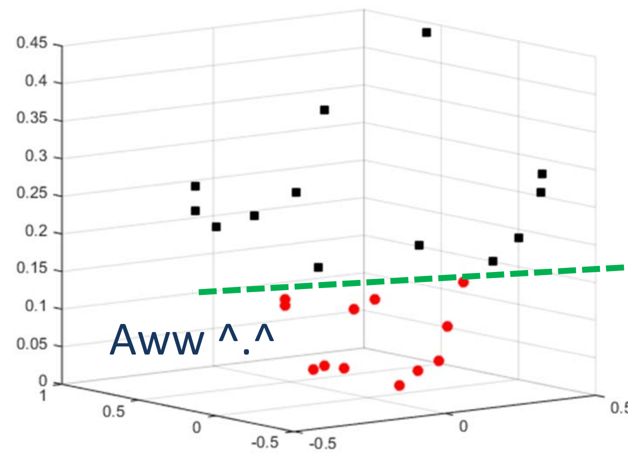


# Feature transformation (Basis expansion)

- 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's dual formulation

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

dot-product

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

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

- Making predictions: classify instance $\boldsymbol{x}$ as sign of

dot-product

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

Note: $b^*$ found by solving for it in $y_j\left(b^* + \sum_{i=1}^{n} \lambda_i^* y_i \boxed{\boldsymbol{x}_i' \boldsymbol{x}_j}\right) = 1$ for any support vector $j$

# Hard-margin SVM in *feature space*

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

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

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

- Making predictions: classify new instance $\boldsymbol{x}$ as sign of

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

Note: $b^*$ found by solving for it in $y_j\left(b^* + \sum_{i=1}^{n} \lambda_i^* y_i \boxed{\varphi(\boldsymbol{x}_i)' \varphi(\boldsymbol{x}_j)}\right) = 1$ for support vector $j$

# Observation: Kernel 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 $K(\boldsymbol{u}, \boldsymbol{v}) = \varphi(\boldsymbol{u})'\varphi(\boldsymbol{v})$

# Kernel as shortcut: Example

- For *some* $\varphi(\boldsymbol{x})$'s, kernel is faster to compute directly than first mapping to feature space then taking dot product.
- For example, consider two vectors $\boldsymbol{u} = [u_1]$ and $\boldsymbol{v} = [v_1]$ and transformation $\varphi(\boldsymbol{x}) = [x_1^2, \sqrt{2c}x_1, c]$, some $c$
  - So $\varphi(\boldsymbol{u}) = \underbrace{\left[u_1^2, \sqrt{2c}u_1, c\right]'}_{\text{2 operations}}$ and $\varphi(\boldsymbol{v}) = \underbrace{\left[v_1^2, \sqrt{2c}v_1, c\right]'}_{\text{+2 operations}}$
  - Then $\varphi(\boldsymbol{u})'\varphi(\boldsymbol{v}) = \left(u_1^2 v_1^2 + 2cu_1v_1 + c^2\right)$ +5 operations = 9 ops.
- This can be alternatively computed directly as

$$\varphi(\boldsymbol{u})'\varphi(\boldsymbol{v}) = (u_1v_1 + c)^2 \quad \text{3 operations}$$

  - Here $K(\boldsymbol{u}, \boldsymbol{v}) = (u_1v_1 + c)^2$ is the corresponding kernel

# More generally: The "kernel trick"

- Consider two training points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and their dot product in the transformed space.
- $k_{ij} \equiv \varphi(\boldsymbol{x}_i)'\varphi(\boldsymbol{x}_j)$ 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 $\varphi$, there's a "shortcut" function that gives exactly the same answer $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = k_{ij}$
  * Doesn't involve steps 1 – 3 and no computation of $\varphi(\boldsymbol{x}_i)$ and $\varphi(\boldsymbol{x}_j)$
  * Usually $k_{ij}$ computable in $O(m)$, but computing $\varphi(\boldsymbol{x})$ requires $O(l)$, where $l \gg m$ (impractical) and even $l = \infty$ (infeasible)

# Kernel hard-margin SVM

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

feature mapping is implied by kernel

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

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

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

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

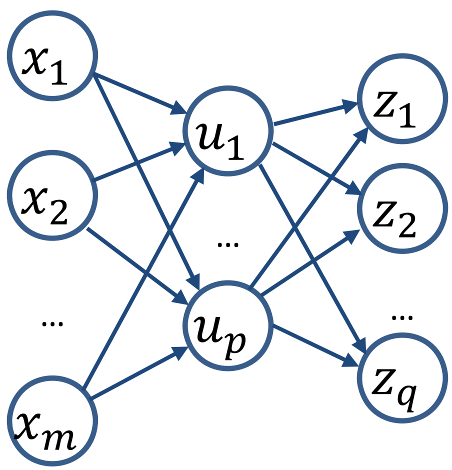
feature mapping is implied by kernel

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

# Approaches to non-linearity

## ANNs

- Elements of $\boldsymbol{u} = \varphi(\boldsymbol{x})$ are transformed input $\boldsymbol{x}$
- This $\varphi$ has weights learned from data




## SVMs

- Choice of kernel $K$ determines features $\varphi$
- Don't learn $\varphi$ weights
- But, don't even need to compute $\varphi$ so can support v high dim. $\varphi$
- Also support arbitrary data types

# Any method that uses a feature space transformation $\varphi(x)$ uses kernels

When poll is active, respond at **PollEv.com/bipr**

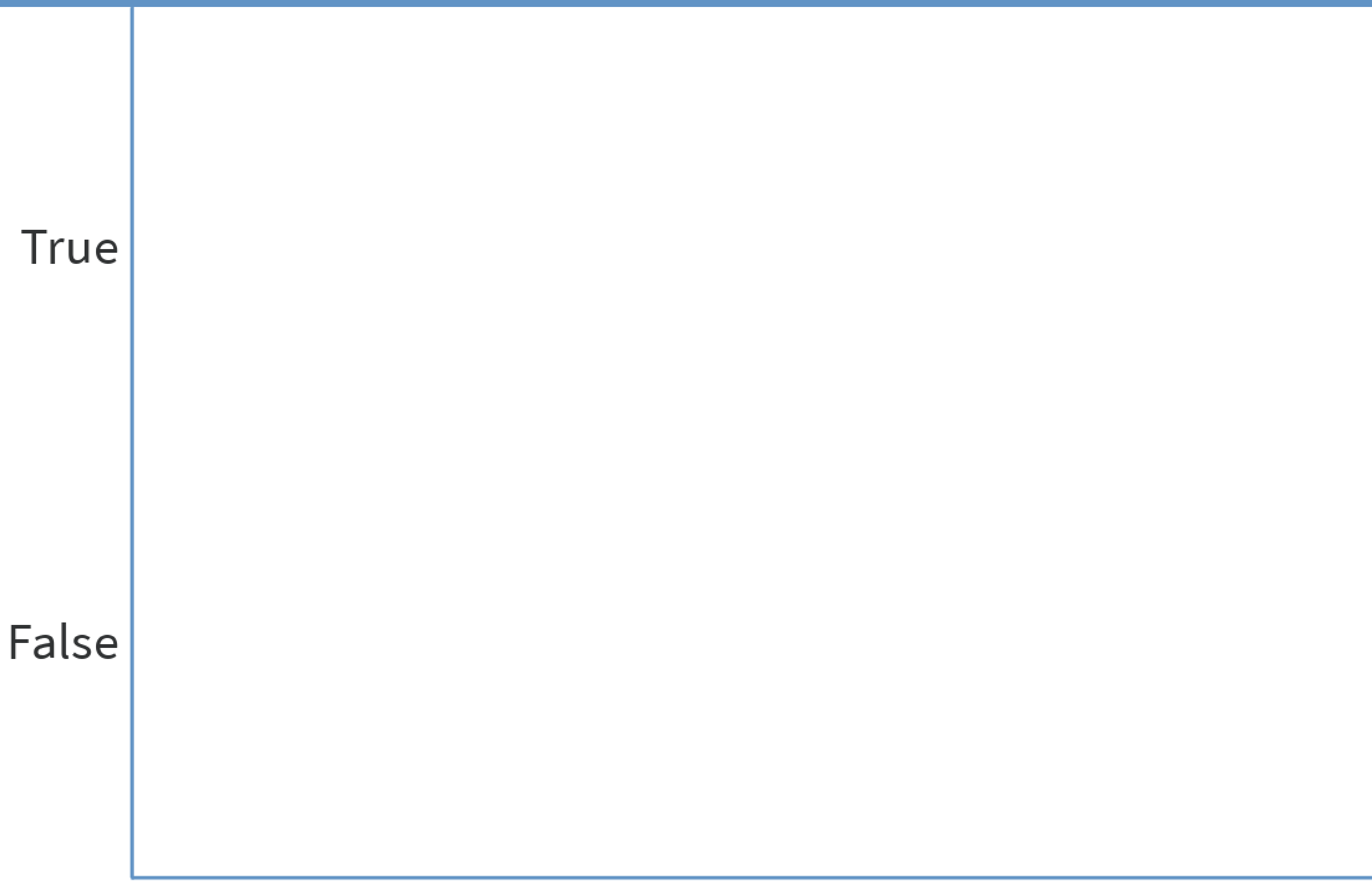
Text **BIPR** to **+61 427 541 357** once to join

Answers to this poll are anonymous

True

False

# Support vectors are points from the training set



# Feature mapping $\varphi(x)$ makes data linearly separable

True

False

# Modular Learning

Kernelisation beyond SVMs;
Separating the “learning module”
from feature space transformation

# 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

# 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 $\alpha_i \leftarrow \alpha_i + 1$

# Representer theorem

- Theorem: For any training set $\{\boldsymbol{x}_i, y_i\}_{i=1}^{n}$, any empirical risk function $E$, monotonic increasing function $g$, then any solution

$$f^* \in \arg\min_f E(\boldsymbol{x}_1, y_1, f(\boldsymbol{x}_1), \dots, \boldsymbol{x}_n, y_n, f(\boldsymbol{x}_n)) + g(\|f\|)$$

has representation for some coefficients

$$f^*(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}_i)$$

Aside: f sits in a reproducing kernel Hilbert space (RKHS)

- Tells us when a (decision-theoretic) learner is kernelizable
- The dual tells us the form this linear kernel representation takes
- SVM is but one example!
  * Ridge regression
  * Logistic regression
  * Principal component analysis (PCA)
  * Canonical correlation analysis (CCA)
  * Linear discriminant analysis (LDA)
  * and many more …

# Constructing Kernels

An overview of popular kernels
and kernel properties

# 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 $\boldsymbol{v}$
- $(\boldsymbol{u}'\boldsymbol{v})^d = (u_1 v_1 + \cdots + u_m v_m)(u_1 v_1 + \cdots + u_m v_m) \ldots (u_1 v_1 + \cdots + u_m v_m)$
- $= \sum_{i=1}^{l} (u_1 v_1)^{a_{i1}} \ldots (u_m v_m)^{a_{im}}$
  - Here $0 \leq a_{ij} \leq d$ and $l$ are integers
- $= \sum_{i=1}^{l} \left(u_1^{a_{i1}} \ldots u_m^{a_{im}}\right)\left(v_1^{a_{i1}} \ldots v_m^{a_{im}}\right)$
- $= \sum_{i=1}^{l} \varphi(\boldsymbol{u})_i \varphi(\boldsymbol{v})_i$
- Feature map $\varphi: \mathbb{R}^m \rightarrow \mathbb{R}^l$, where $\varphi_i(\boldsymbol{x}) = \left(x_1^{a_{i1}} \ldots x_m^{a_{im}}\right)$

# Identifying new kernels

- Method 1: 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 for more identities*

Prove these!

- Method 2: Using Mercer's theorem (coming up!)

# Radial basis function kernel

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

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

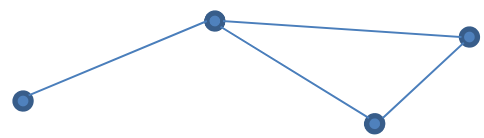
# Mercer's Theorem

- 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 $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$
  * Construct $n \times n$ matrix of pairwise values $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$
  * $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is a valid kernel if this matrix is positive-semidefinite, and this holds for all possible sequences $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$

# Data comes in a variety of shapes

- So far in COMP90051 data has been vectors of numbers
- 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.

* Immanuel Kant

# Handling arbitrary data structures

- Kernels are powerful approach to deal with many data types
- Could define similarity function on 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

# A large variety of kernels

Definition 9.1 Polynomial kernel 286
Computation 9.6 All-subsets kernel 289
Computation 9.8 Gaussian kernel 290
Computation 9.12 ANOVA kernel 293
Computation 9.18 Alternative recursion for ANOVA kernel 296
Computation 9.24 General graph kernels 301
Definition 9.33 Exponential difusion kernel 307
Definition 9.34 von Neumann difusion kernel 307
Computation 9.35 Evaluating difusion kernels 308
Computation 9.46 Evaluating randomised kernels 315
Definition 9.37 Intersection kernel 309
Definition 9.38 Union-complement kernel 310
Remark 9.40 Agreement kernel 310
Section 9.6 Kernels on real numbers 311
Remark 9.42 Spline kernels 313
Definition 9.43 Derived subsets kernel 313
Definition 10.5 Vector space kernel 325
Computation 10.8 Latent semantic kernels 332
Definition 11.7 The p-spectrum kernel 342
Computation 11.10 The p-spectrum recursion 343
Remark 11.13 Blended spectrum kernel 344
Computation 11.17 All-subsequences kernel 347
Computation 11.24 Fixed length subsequences kernel 352
Computation 11.33 Naive recursion for gap-weighted subsequences kernel 358
Computation 11.36 Gap-weighted subsequences kernel 360
Computation 11.45 Trie-based string kernels 367
Algorithm 9.14 ANOVA kernel 294
Algorithm 9.25 Simple graph kernels 302
Algorithm 11.20 All–non-contiguous subsequences kernel 350
Algorithm 11.25 Fixed length subsequences kernel 352
Algorithm 11.38 Gap-weighted subsequences kernel 361
Algorithm 11.40 Character weighting string kernel 364
Algorithm 11.41 Soft matching string kernel 365
Algorithm 11.42 Gap number weighting string kernel 366
Algorithm 11.46 Trie-based p-spectrum kernel 368
Algorithm 11.51 Trie-based mismatch kernel 371
Algorithm 11.54 Trie-based restricted gap-weighted kernel 374
Algorithm 11.62 Co-rooted subtree kernel 380
Algorithm 11.65 All-subtree kernel 383
Algorithm 12.8 Fixed length HMM kernel 401
Algorithm 12.14 Pair HMM kernel 407
Algorithm 12.17 Hidden tree model kernel 411
Algorithm 12.34 Fixed length Markov model Fisher kernel 427

www.kernel-methods.net

# This lecture

- Kernels
  - Nonlinearity by basis expansion
  - Kernel trick to speed up computation
- Modular learning
  - Separating "learning module" from feature transformation
  - Representer theorem
- Constructing kernels
  - An overview of popular kernels and their properties
  - Mercer's theorem
  - Extending machine learning beyond conventional data structure

Next lecture: Ensemble methods