

Lecture 9. Kernel Methods

COMP90051 Statistical Machine Learning

Lecturer: Feng Liu



This lecture

- Dual formulation of the SVM
- 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

Lagrangian Duality for the SVM

An equivalent formulation, with
important consequences.

Soft-margin SVM recap

- Soft-margin SVM objective:

$$\operatorname{argmin}_{\mathbf{w}, b, \xi} \left(\frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \right)$$

$$\text{s.t. } y_i(\mathbf{w}'\mathbf{x}_i + b) \geq 1 - \xi_i \text{ for } i = 1, \dots, n$$

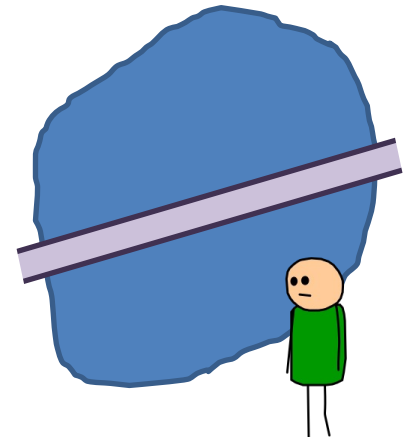
$$\xi_i \geq 0 \text{ for } i = 1, \dots, n$$

- While we can optimise the above “**primal**”, often instead work with the **dual**

Constrained optimisation

- Constrained optimisation: **canonical form**

$$\begin{aligned} & \text{minimise } f(\mathbf{x}) \\ & \text{s.t. } g_i(\mathbf{x}) \leq 0, i = 1, \dots, n \\ & \quad h_j(\mathbf{x}) = 0, j = 1, \dots, m \end{aligned}$$



- * E.g., find deepest point in the lake, *south of the bridge*
- Gradient descent doesn't immediately apply
- Hard-margin SVM: $\operatorname{argmin}_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$ s.t. $1 - y_i(\mathbf{w}'\mathbf{x}_i + b) \leq 0$ for $i = 1, \dots, n$
- Method of **Lagrange multipliers**
 - * Transform to unconstrained optimisation
 - * Transform **primal program** to a related **dual program**, alternate to primal
 - * Analyse necessary & sufficient conditions for solutions of both programs

The Lagrangian and duality

- Introduce auxiliary objective function via auxiliary variables

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f(\mathbf{x}) + \sum_{i=1}^n \lambda_i g_i(\mathbf{x}) + \sum_{j=1}^m \nu_j h_j(\mathbf{x})$$

Primal constraints became penalties

- * Called the *Lagrangian* function
- * New $\boldsymbol{\lambda}$ and $\boldsymbol{\nu}$ are called the *Lagrange multipliers* or *dual variables*
- (Old) **primal program**: $\min_{\mathbf{x}} \max_{\boldsymbol{\lambda} \geq 0, \boldsymbol{\nu}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$
- (New) **dual program**: $\max_{\boldsymbol{\lambda} \geq 0, \boldsymbol{\nu}} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$

May be easier to solve, advantageous
- Duality theory relates primal/dual:
 - * Weak duality: dual optimum \leq primal optimum
 - * For convex programs (inc. SVM!) **strong duality**: optima coincide!

Karush-Kuhn-Tucker Necessary Conditions

- Lagrangian: $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f(\mathbf{x}) + \sum_{i=1}^n \lambda_i g_i(\mathbf{x}) + \sum_{j=1}^m \nu_j h_j(\mathbf{x})$
- Necessary conditions for optimality of a primal solution
- **Primal feasibility:**
 - * $g_i(\mathbf{x}^*) \leq 0, i = 1, \dots, n$
 - * $h_j(\mathbf{x}^*) = 0, j = 1, \dots, m$
- **Dual feasibility:** $\lambda_i^* \geq 0$ for $i = 1, \dots, n$
- **Complementary slackness:** $\lambda_i^* g_i(\mathbf{x}^*) = 0, i = 1, \dots, n$
- **Stationarity:** $\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) = \mathbf{0}$

Souped-up version of necessary condition “derivative is zero” in **unconstrained** optimisation.

Don't penalise if
constraint satisfied

KKT conditions for hard-margin SVM

The Lagrangian

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\lambda}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \lambda_i (y_i (\mathbf{w}' \mathbf{x}_i + b) - 1)$$

KKT conditions:

- * Feasibility: $y_i ((\mathbf{w}^*)' \mathbf{x}_i + b^*) - 1 \geq 0$ for $i = 1, \dots, n$
- * Feasibility: $\lambda_i^* \geq 0$ for $i = 1, \dots, n$
- * Complementary slackness: $\lambda_i^* (y_i ((\mathbf{w}^*)' \mathbf{x}_i + b^*) - 1) = 0$
- * Stationarity: $\nabla_{\mathbf{w}, b} \mathcal{L}(\mathbf{w}^*, b^*, \boldsymbol{\lambda}^*) = \mathbf{0}$

Let's minimise Lagrangian w.r.t primal variables

- Lagrangian:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\lambda}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \lambda_i (y_i (\mathbf{w}' \mathbf{x}_i + b) - 1)$$

- Stationarity conditions give us more information:

$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^n \lambda_i y_i = 0 \quad \longrightarrow \quad \text{New constraint}$$

$$\frac{\partial \mathcal{L}}{\partial w_j} = w_j^* - \sum_{i=1}^n \lambda_i y_i (\mathbf{x}_i)_j = 0 \quad \longrightarrow \quad \text{Eliminates primal variables}$$

- The Lagrangian becomes (with additional constraint, above)

$$\mathcal{L}(\boldsymbol{\lambda}) = \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y_i y_j \mathbf{x}_i' \mathbf{x}_j$$

Dual program for hard-margin SVM

- Having minimised the Lagrangian with respect to primal variables, now maximising w.r.t dual variables yields the **dual program**

$$\begin{aligned} \operatorname{argmax}_{\lambda} \quad & \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y_i y_j \mathbf{x}_i' \mathbf{x}_j \\ \text{s.t. } & \lambda_i \geq 0 \text{ and } \sum_{i=1}^n \lambda_i y_i = 0 \end{aligned}$$

- Strong duality:** Solving dual, solves the primal!!
- Like primal: A so-called *quadratic program* - off-the-shelf software can solve – more later
- Unlike primal:
 - * Complexity of solution is $O(n^3)$ instead of $O(d^3)$ – more later
 - * Program depends on dot products of data only – more later on kernels!

Making predictions with dual solution

Recovering primal variables

- Recall from stationarity: $\mathbf{w}_j^* - \sum_{i=1}^n \lambda_i y_i (\mathbf{x}_i)_j = 0$
- Complementary slackness: b^* can be recovered from dual solution, noting for any example j with $\lambda_i^* > 0$, we have $y_j(b^* + \sum_{i=1}^n \lambda_i^* y_i \mathbf{x}_i' \mathbf{x}_j) = 1$ (these are the **support vectors**)

Testing: classify new instance \mathbf{x} based on sign of

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

Soft-margin SVM's dual

- Training: find λ that solves

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

box constraints

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

- Making predictions: same pattern as in as in hard-margin case

Finally... Training the SVM

- The SVM dual problems are quadratic programs, solved in $O(n^3)$, or $O(d^3)$ for the primal.
- This can be inefficient; specialised solutions exist
 - * chunking: original SVM training algorithm exploits fact that many λ s will be zero (sparsity)
 - * sequential minimal optimisation (SMO), an extreme case of chunking. An iterative procedure that analytically optimises randomly chosen pairs of λ s per iteration

Mini summary

- Dual vs primal formulation of SVM
- Method of Lagrange Multipliers
- Approaches to make predictions and train

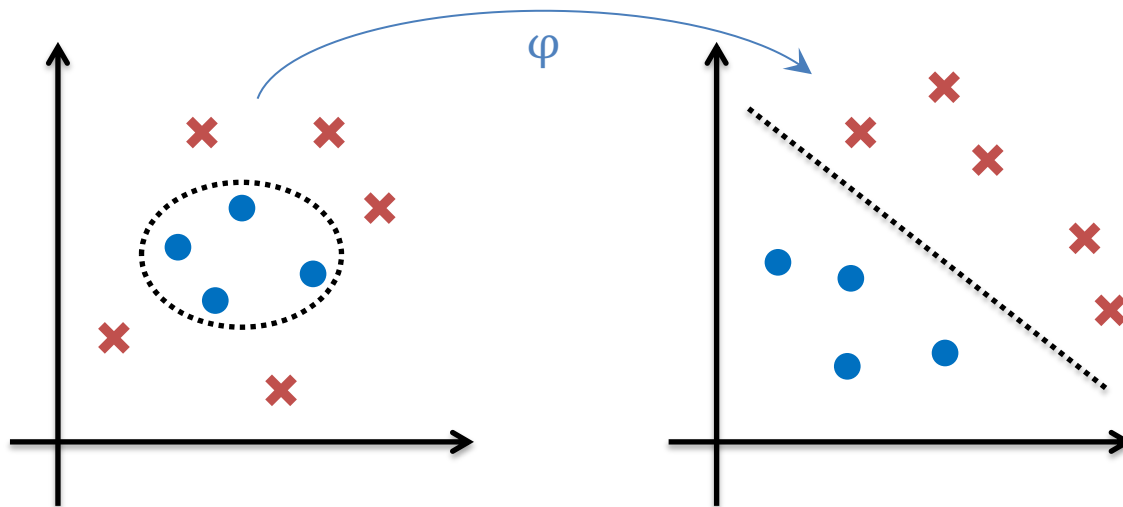
Next: Kernelising the SVM

Kernelising the SVM

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

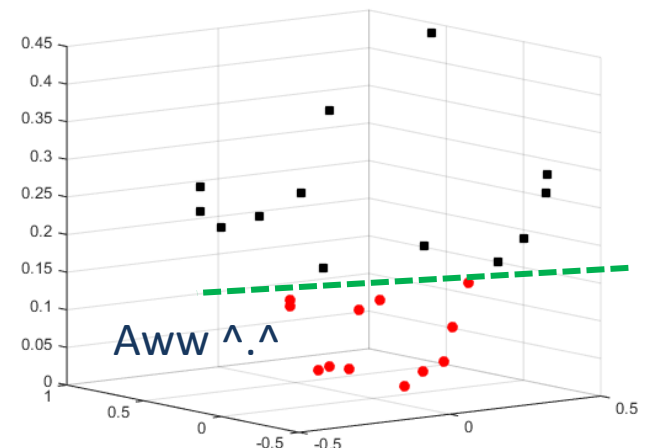
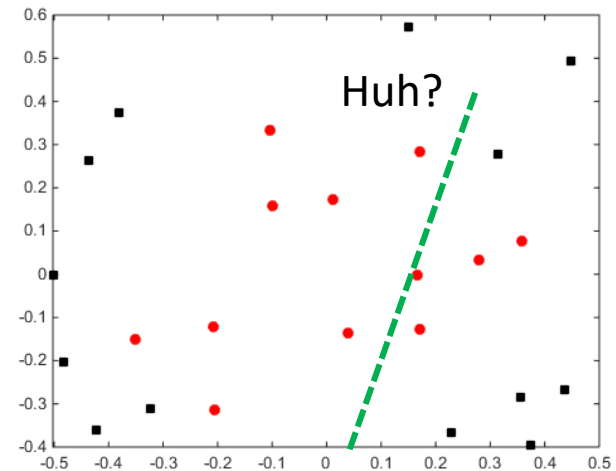
Handling non-linear data with the SVM

- Method 1: Soft-margin SVM
- 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



Feature transformation (Basis expansion)

- Consider a binary classification problem
- Each example has features $[x_1, x_2]$
- Not linearly separable
- Now 'add' a feature $x_3 = x_1^2 + x_2^2$
- 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 $\varphi(\mathbf{x})$
 - * 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 $\varphi(\mathbf{x})$
- Train classifier/Do predictions
- Problem: impractical/impossible to compute $\varphi(\mathbf{x})$ for high/infinite-dimensional $\varphi(\mathbf{x})$

Hard-margin SVM's dual formulation

- Training: finding λ that solve

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

dot-product
↓

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

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

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

dot-product
↓

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

Hard-margin SVM in feature space

- Training: finding λ that solve

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

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

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

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

Note: b^* found by solving for it in $y_j(b^* + \sum_{i=1}^n \lambda_i^* y_i \boxed{\varphi(\mathbf{x}_i)' \varphi(\mathbf{x}_j)}) = 1$ 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 $\mathbf{u}'\mathbf{v} = \sum_{i=1}^m u_i v_i$
 - * In transformed space $\varphi(\mathbf{u})'\varphi(\mathbf{v}) = \sum_{i=1}^l \varphi(\mathbf{u})_i \varphi(\mathbf{v})_i$
- **Kernel** is a function that can be expressed as a dot product in some feature space $K(\mathbf{u}, \mathbf{v}) = \varphi(\mathbf{u})'\varphi(\mathbf{v})$

Kernel as shortcut: Example

- For *some* $\varphi(\mathbf{x})$'s, **kernel is faster to compute** directly than first mapping to feature space then taking dot product.
- For example, consider two vectors $\mathbf{u} = [u_1]$ and $\mathbf{v} = [v_1]$ and transformation $\varphi(\mathbf{x}) = [x_1^2, \sqrt{2c}x_1, c]$, some c
 - * So $\varphi(\mathbf{u}) = [u_1^2, \sqrt{2c}u_1, c]'$ and $\varphi(\mathbf{v}) = [v_1^2, \sqrt{2c}v_1, c]'$

2 operations +2 operations
 - * Then $\varphi(\mathbf{u})'\varphi(\mathbf{v}) = (u_1^2v_1^2 + 2cu_1v_1 + c^2)$ +4 operations = 8 ops.
- This can be alternatively **computed directly** as

$$\varphi(\mathbf{u})'\varphi(\mathbf{v}) = (u_1v_1 + c)^2 \quad \text{3 operations}$$
 - * Here $K(\mathbf{u}, \mathbf{v}) = (u_1v_1 + c)^2$ is the corresponding kernel

More generally: The “kernel trick”

- Consider two training points \mathbf{x}_i and \mathbf{x}_j and their dot product in the transformed space.
- $k_{ij} \equiv \varphi(\mathbf{x}_i)' \varphi(\mathbf{x}_j)$ **kernel matrix** can be computed as:
 1. Compute $\varphi(\mathbf{x}_i)'$
 2. Compute $\varphi(\mathbf{x}_j)$
 3. Compute $k_{ij} = \varphi(\mathbf{x}_i)' \varphi(\mathbf{x}_j)$
- However, for some transformations φ , there's a “shortcut” function that gives exactly the same answer $K(\mathbf{x}_i, \mathbf{x}_j) = k_{ij}$
 - * Doesn't involve steps 1 – 3 and no computation of $\varphi(\mathbf{x}_i)$ and $\varphi(\mathbf{x}_j)$
 - * Usually k_{ij} computable in $O(m)$, but computing $\varphi(\mathbf{x})$ requires $O(l)$, where $l \gg m$ (**impractical**) and even $l = \infty$ (**infeasible**)

Kernel hard-margin SVM

- Training: finding λ that solve

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

feature mapping is implied by kernel

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

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

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

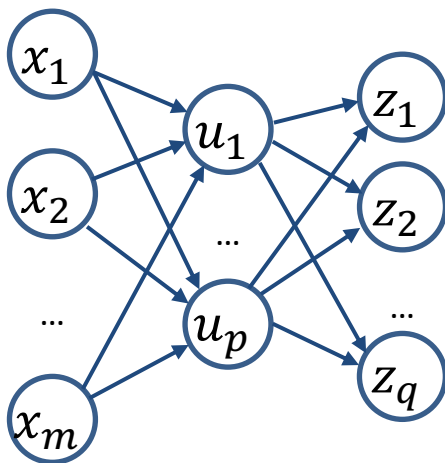
feature mapping is implied by kernel

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

Approaches to non-linearity

NNets

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



SVMs

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

Mini summary

- Kernelisation
 - * Basis expansion on dual formulation of SVMs
 - * “Kernel trick”; Fast computation of feature space dot product

Next: Kernel methods as modular machine learning

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
- But how to know if an algorithm is a kernel method?

Representer theorem

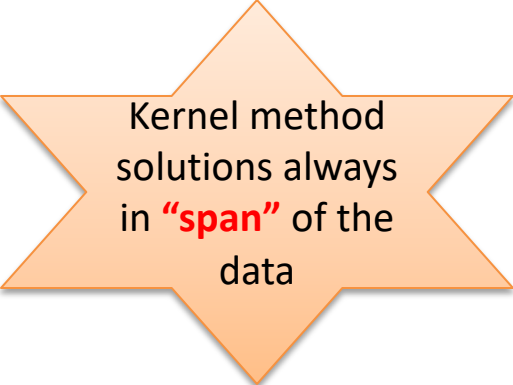
Theorem: For any training set $\{\mathbf{x}_i, y_i\}_{i=1}^n$, any empirical risk function E , monotonic increasing function g , then any solution

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

has representation for some coefficients

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

- Tells us when a (decision-theoretic) learner is kernelizable
- The dual tells us the form this linear kernel representation takes
- SVM not the only case:
 - * Ridge regression
 - * Logistic regression
 - * Principal component analysis (PCA)
 - * Canonical correlation analysis (CCA)
 - * Linear discriminant analysis (LDA)
 - * and many more ...



Kernel method
solutions always
in “**span**” of the
data

Mini summary

- Kernel methods are modular
 - * Choose learning algorithm
 - * Choose kernel
- Representer thm: recognises kernelisable learners

Next: Constructing and recognising kernels

Constructing Kernels

An overview of popular kernels,
kernel properties for building and
recognising new kernels

Polynomial kernel

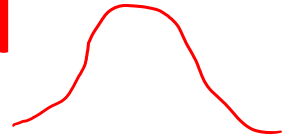
- Function $K(\mathbf{u}, \mathbf{v}) = (\mathbf{u}'\mathbf{v} + c)^d$ is called polynomial kernel
 - * Here \mathbf{u} and \mathbf{v} are vectors with m components
 - * $d \geq 0$ is an integer and $c \geq 0$ is a constant
- Without loss of generality, assume $c = 0$
 - * If it's not, add \sqrt{c} as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

Identifying new kernels

- Method 1: Let $K_1(\mathbf{u}, \mathbf{v})$, $K_2(\mathbf{u}, \mathbf{v})$ be kernels, $c > 0$ be a constant, and $f(\mathbf{x})$ be a real-valued function. Then each of the following is also a kernel:
 - * $K(\mathbf{u}, \mathbf{v}) = K_1(\mathbf{u}, \mathbf{v}) + K_2(\mathbf{u}, \mathbf{v})$
 - * $K(\mathbf{u}, \mathbf{v}) = cK_1(\mathbf{u}, \mathbf{v})$
 - * $K(\mathbf{u}, \mathbf{v}) = f(\mathbf{u})K_1(\mathbf{u}, \mathbf{v})f(\mathbf{v})$
 - * *See Bishop for more identities*
- Method 2: Using Mercer's theorem (coming up!)

Prove
these!

Radial basis function kernel



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

Power series
expansion
- Here, each $(\mathbf{u}'\mathbf{v})^d$ 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 $\varphi(\mathbf{u})$, is there a good kernel to use?
- Inverse question: given some function $K(\mathbf{u}, \mathbf{v})$, **is this a valid kernel?** In other words, is there a mapping $\varphi(\mathbf{u})$ implied by the kernel?

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

Handling arbitrary data structures

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

$K(\text{"science is organized knowledge"}, \text{"wisdom is organized life"})$

- However, not every function on two objects is a valid kernel
- Remember that we need that function $K(\mathbf{u}, \mathbf{v})$ 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 diffusion kernel 307
 Definition 9.34 von Neumann diffusion kernel 307
 Computation 9.35 Evaluating diffusion 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

Mini Summary

- Constructing kernels
 - * An overview of popular kernels and their properties
 - * Mercer's theorem
 - * Extending machine learning beyond conventional data structure

Next lecture: Perceptron