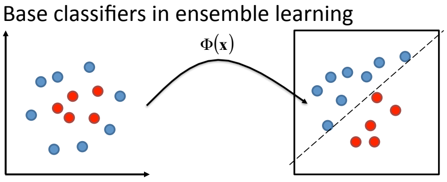
Neural Networks and Learning Systems
TBMI 26, 2017

Lecture 7
Kernel methods

Magnus Borga
magnus.borga@liu.se

Introduction

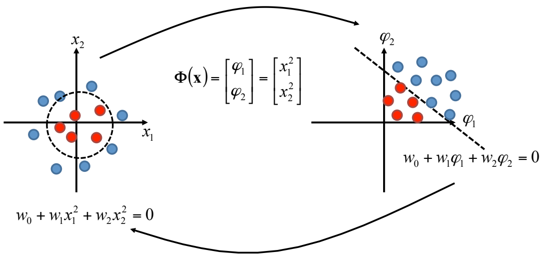
- We have seen nonlinear mappings of input features to a new feature space:
  - Hidden layer in a neural network
  - Base classifiers in ensemble learning



Cover's theorem: The probability that classes are linearly separable increases when the features are nonlinearly mapped to a higher dimensional feature space.

(*cf. the extreme case of putting each sample in a dimension of its own!*)

Nonlinear mapping example



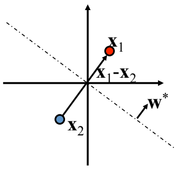
Kernel methods

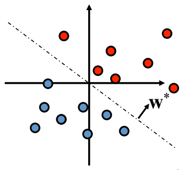
A general approach to making linear methods non-linear.

The name *kernel* refers to positive definite kernels in operator theory mathematics.

Consider a linear classifier

$$f(\mathbf{x};\mathbf{w}) = \mathbf{w}^T\mathbf{x} + w_0$$





$$\mathbf{w}^* = 1.0\mathbf{x}_1 - 1.0\mathbf{x}_2$$

Seems plausible that the optimal direction can be expressed like this!

$$\mathbf{w}^* = \sum_{n=1}^{N} \alpha_n \mathbf{x}_n$$

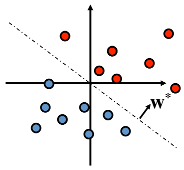
5

Linear classifier in scalar product form

$$\left.\begin{aligned} f(\mathbf{x};\mathbf{w}) &= \mathbf{w}^T\mathbf{x} + w_0 \\ \mathbf{w} &= \sum_{n=1}^{N} \alpha_n \mathbf{x}_n \end{aligned}\right\} \quad f(\mathbf{x};\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n \mathbf{x}_n^T \mathbf{x} + \alpha_0$$

$\alpha_0 = w_0$

Scalar product = distance and angle between the vectors



For the bias weight

(i) $f(\mathbf{x};\mathbf{w}) = \mathbf{w}^T\mathbf{x}$ $\quad \mathbf{x} \leftarrow \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$

(ii) $f(\mathbf{x};\boldsymbol{\alpha}) = \sum_{n=0}^{N} \alpha_n \mathbf{x}_n^T \mathbf{x}$

Add a dummy training example $\mathbf{x}_0 = \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix}$ for α_0.

NOTE: Classifier form (ii) must store all training examples for the classification, whereas form (i) must not.

Is there any advantage of form (ii)?

6

Scalar product

a.k.a. dot product or inner product

Usual Euclidian space

$$\boxed{\mathbf{x}\cdot\mathbf{z} = \mathbf{x}^T\mathbf{z} = \|\mathbf{x}\|\|\mathbf{z}\|\cos(\theta)}$$

$$\mathbf{x}^T\mathbf{x} = \|\mathbf{x}\|^2 \Rightarrow \|\mathbf{x}\| = \sqrt{\mathbf{x}^T\mathbf{x}}$$

$$\|\mathbf{x}-\mathbf{z}\|^2 = (\mathbf{x}-\mathbf{z})^T(\mathbf{x}-\mathbf{z}) = \mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\mathbf{z} + \mathbf{z}^T\mathbf{z}$$

Vector lengths and distances between points are determined by the scalar product.

$$\cos(\theta) = \frac{\mathbf{x}^T\mathbf{z}}{\sqrt{\mathbf{x}^T\mathbf{x}}\sqrt{\mathbf{z}^T\mathbf{z}}}$$

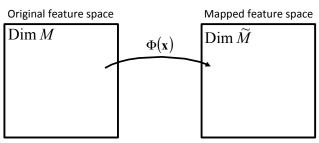
$\mathbf{x}^T\mathbf{z} = 0 \Leftrightarrow \mathbf{x}$ and $\mathbf{z}$ orthogonal

Angles are determined by the scalar product.

7

Non-linear mappings

$$\Phi(\mathbf{x}): R^M \rightarrow R^{\tilde{M}}, \text{ with } \tilde{M} > M$$



(i) $f(\mathbf{x};\mathbf{w}) = \mathbf{w}^T\mathbf{x}$

(ii) $f(\mathbf{x};\boldsymbol{\alpha}) = \sum_{n=0}^{N} \alpha_n \mathbf{x}_n^T \mathbf{x}$

(i) $f(\mathbf{x};\mathbf{w}) = \mathbf{w}^T\Phi(\mathbf{x})$

(ii) $f(\mathbf{x};\boldsymbol{\alpha}) = \sum_{n=0}^{N} \alpha_n \Phi(\mathbf{x}_n)^T \Phi(\mathbf{x})$

Examples: $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

$$\Phi(\mathbf{x}) = \begin{bmatrix} x_1^2 \\ x_1x_2 \\ x_2^2 \end{bmatrix}$$

$$\Phi(\mathbf{x}) = \begin{bmatrix} \cos(x_1) \\ \sin(x_2) \\ x_2e^{-x_1} \\ x_2^{1000} \end{bmatrix}$$

8

Explicit and implicit mapping

Classifier form (ii) offers two different ways of defining $\Phi(\mathbf{x})$!

$$f(\mathbf{x};\boldsymbol{\alpha}) = \sum_{n=0}^{N} \alpha_n \Phi(\mathbf{x}_n)^T \Phi(\mathbf{x})$$

Reminder: We only need the scalar product!

Explicit: Do the actual mapping, for example $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ $\quad \Phi(\mathbf{x}) = \begin{bmatrix} x_1^2 \\ x_1 x_2 \\ x_2^2 \end{bmatrix}$

Definition

But this is only an intermediate vector that we do not really need.

Implicit: Define the new feature space by defining the scalar product in that space, i.e., how distances and angles are measured. For example:

$$\kappa(\mathbf{x},\mathbf{z}) = \Phi(\mathbf{x})^T \Phi(\mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2$$

Kernel function Definition

9

Explicit and implicit mappings are equivalent

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad \mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$$

$$\kappa(\mathbf{x},\mathbf{z}) = (\mathbf{x}^T\mathbf{z})^2 = (x_1 z_1 + x_2 z_2)^2 = (x_1^2 z_1^2 + 2x_1x_2z_1z_2 + x_2^2 z_2^2) = \underbrace{\begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}^T \begin{pmatrix} z_1^2 \\ \sqrt{2}z_1z_2 \\ z_2^2 \end{pmatrix}}_{\Phi(\mathbf{x})^T \Phi(\mathbf{z})}$$

Define!

The kernel function $\kappa(\mathbf{x},\mathbf{z}) = (\mathbf{x}^T\mathbf{z})^2$ defines the same space as the explicit mapping $\mathbf{x} \rightarrow \Phi(\mathbf{x})$.

Only in some special cases can we find the explicit mapping function from the implicit kernel function!

10

Why not always use explicit mappings?

- Assume we have 20 input features….
- Create all polynomial combinations up to degree 5 (e.g., x_1, x_1^5, $x_2^2 x_9^3$,….)
- Generates a new feature space with dimension > 50,000!
- For example, PCA in new space: Eigendecomposition of a 50,000 x 50,000 matrix.

11

The kernel function

$$\boxed{\mathbf{x} \cdot \mathbf{z} = \mathbf{x}^T \mathbf{z}}$$

Needs to define a valid scalar product in some space

$\mathbf{x} \cdot \mathbf{z} = \mathbf{z} \cdot \mathbf{x}$

$a\mathbf{x} \cdot b\mathbf{z} = ab(\mathbf{x} \cdot \mathbf{z})$

$\mathbf{x} \cdot (\mathbf{z}_1 + \mathbf{z}_2) = \mathbf{x} \cdot \mathbf{z}_1 + \mathbf{x} \cdot \mathbf{z}_2$

⋮

Properties of a scalar product

Polynomial kernels	Gaussian kernel	Sigmoid kernel
$\kappa(\mathbf{x}_i,\mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^d$	$\kappa(\mathbf{x}_i,\mathbf{x}_j) = \exp\left(-\frac{\lVert \mathbf{x}_i - \mathbf{x}_j \rVert^2}{2\sigma^2}\right)$	$\kappa(\mathbf{x}_i,\mathbf{x}_j) = \tanh(\mathbf{x}_i^T \mathbf{x}_j)$

Includes all monomials up to degree d, e.g., for d=2: 1, x_1, x_2, x_1^2, x_1x_2, x_2^2

Many other kernels, see for example: http://crsouza.blogspot.com/2010/03/kernel-functions-for-machine-learning.html

12

Summary so far and open questions

- Assume that the optimal solution for a linear classifier can be expressed as: $\mathbf{w} = \sum_{n=1}^{N} \alpha_n \mathbf{x}_n$ This must be verified!
- The linear classifier can then be expressed as:
$$f(\mathbf{x};\boldsymbol{\alpha}) = \sum_{n=0}^{N} \alpha_n \mathbf{x}_n^T \mathbf{x}$$ How do we find the α's?
- Apply the linear classifier in a higher-dimensional space by defining its scalar product via the kernel function
$$\kappa(\mathbf{x},\mathbf{z}) = \Phi(\mathbf{x})^T \Phi(\mathbf{z})$$
$$f(\mathbf{x};\boldsymbol{\alpha}) = \sum_{n=0}^{N} \alpha_n \kappa(\mathbf{x}_n, \mathbf{x})$$ How do we select the kernel function?

13

Example: Linear perceptron with square error cost

From lecture 2!

Minimize the following cost function

$$\varepsilon(\mathbf{w}) = \sum_{i=1}^{N} \left(\mathbf{w}^T \mathbf{x}_i - y_i\right)^2$$

N = # training samples
$y_i \in \{-1,1\}$ depending on the class of training sample *i*

14

Example: Linear perceptron algorithm

From lecture 2!

$$\varepsilon(\mathbf{w}) = \sum_{i=1}^{N} \left(\mathbf{w}^T \mathbf{x}_i - y_i\right)^2$$

$$\frac{\partial \varepsilon}{\partial \mathbf{w}} = 2\sum_{i=1}^{N} \left(\mathbf{w}^T \mathbf{x}_i - y_i\right)\mathbf{x}_i$$

Gradient descent:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{\partial \varepsilon}{\partial \mathbf{w}} = \mathbf{w}_t - \eta \sum_{i=1}^{N} \left(\mathbf{w}_t^T \mathbf{x}_i - y_i\right)\mathbf{x}_i \quad (\text{Eq.}1)$$

Algorithm:
1. Start with a random **w**
2. Iterate Eq. 1 until convergence

$$\mathbf{w}^* = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i \text{ as } t \to \infty$$

15

Example: Kernel perceptron algorithm

Gradient descent:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \sum_{i=1}^{N} \left(\mathbf{w}_t^T \mathbf{x}_i - y_i\right)\mathbf{x}_i$$ Original space

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \sum_{i=1}^{N} \underbrace{\left(\mathbf{w}_t^T \Phi(\mathbf{x}_i) - y_i\right)}_{\beta_{t,i}} \Phi(\mathbf{x}_i)$$ Mapped space

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \sum_{i=1}^{N} \beta_{t,i} \Phi(\mathbf{x}_i)$$

$$\mathbf{w}^* = \sum_{i=1}^{N} \alpha_i \Phi(\mathbf{x}_i) \text{ as } t \to \infty$$

16

Example: Kernel perceptron algorithm

$$\left.\begin{array}{l}\varepsilon(\mathbf{w}) = \sum_{i=1}^{N}\left(y_i - \mathbf{w}^T\Phi(\mathbf{x}_i)\right)^2 \\ \mathbf{w} = \sum_{i=1}^{N}\alpha_i\Phi(\mathbf{x}_i)\end{array}\right\} \varepsilon(\boldsymbol{\alpha}) = \sum_{i=1}^{N}\left(y_i - \sum_{j=1}^{N}\alpha_j \underline{\Phi(\mathbf{x}_j)^T\Phi(\mathbf{x}_i)}\right)^2 = \sum_{i=1}^{N}\left(y_i - \sum_{j=1}^{N}\alpha_j \underline{\kappa(\mathbf{x}_j,\mathbf{x}_i)}\right)^2$$

Kernel trick!

Gradient:

$$\frac{\partial\varepsilon}{\partial\alpha_k} = -2\sum_{i=1}^{N}\left(y_i - \sum_{j=1}^{N}\alpha_j\kappa(\mathbf{x}_j,\mathbf{x}_i)\right)\kappa(\mathbf{x}_k,\mathbf{x}_i)$$

Gradient descent in α!

$$\boxed{\alpha_{k,t+1} = \alpha_{k,t} - \eta\frac{\partial\varepsilon}{\partial\alpha_k}}$$

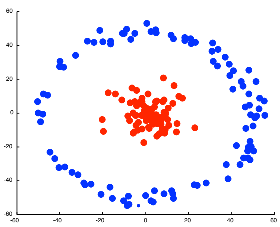
17

Example: Kernel perceptron summary

1. Showed that $\mathbf{w}^* = \sum_{i=1}^{N}\alpha_i\Phi(\mathbf{x}_i)$
2. Cost function in α: $\varepsilon(\boldsymbol{\alpha}) = \sum_{i=1}^{N}\left(y_i - \sum_{j=1}^{N}\alpha_j\Phi(\mathbf{x}_j)^T\Phi(\mathbf{x}_i)\right)^2$
3. Choose kernel function: $\kappa(\mathbf{x}_j,\mathbf{x}_i) = \Phi(\mathbf{x}_j)^T\Phi(\mathbf{x}_i)$
4. Gradient descent in α: $\alpha_{k,t+1} = \alpha_{k,t} - \eta\frac{\partial\varepsilon}{\partial\alpha_k}$
5. Apply classifier: $f(\mathbf{x};\boldsymbol{\alpha}) = \sum_{i=0}^{N}\alpha_i\kappa(\mathbf{x}_i,\mathbf{x})$

18

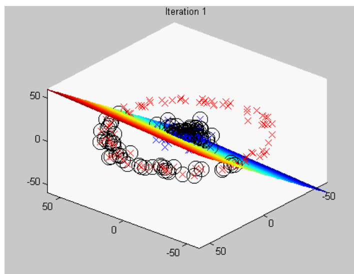
Kernel Perceptron example



19

Kernel Perceptron example, cont

Movie!

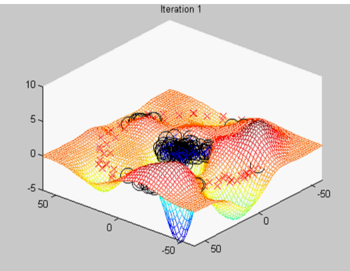


The original linear perceptron algorithm will not work because the classes are not linearly separable

20

Kernel Perceptron example, cont

Movie!



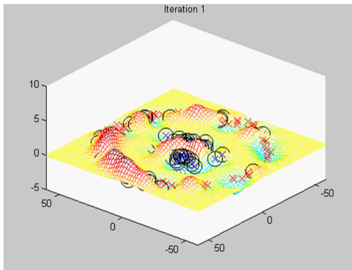
The surface shows the value of $\sum_{k=1}^{N} \alpha_k \kappa(\mathbf{x}_k, \mathbf{x})$ for different **x**.

Gaussian kernel with σ=10

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

Kernel Perceptron example, cont

Movie!



The surface shows the value of $\sum_{k=1}^{N} \alpha_k \kappa(\mathbf{x}_k, \mathbf{x})$ for different **x**.

Gaussian kernel with σ=10

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

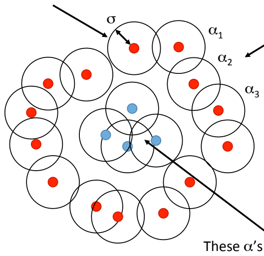
Structure of the classification function

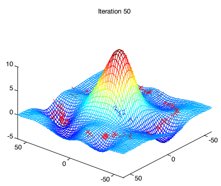
$$f(\mathbf{x}) = \sum_{k=1}^{N} \alpha_k \kappa(\mathbf{x}_k, \mathbf{x})$$

Weighted distance to all training samples

Gaussian kernel

These α's will probably be negative





These α's will probably be positive

Kernelization of linear methods

- Perceptron
- LDA
- SVM
- PCA
- k-means

Linear classifiers

Linear structure discovery methods

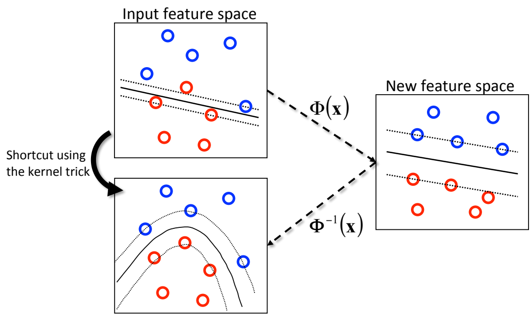
Clustering

Number of parameters equals the number of training samples

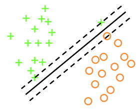
$$f(\mathbf{x}) = \sum_{k=1}^{N} \alpha_k \kappa(\mathbf{x}_k, \mathbf{x})$$

Have to store all training samples

Nonlinear SVM

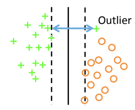


Kernelizing the linear SVM



$$\min_{\mathbf{w}} \mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i$$

subject to $y_i(\mathbf{w}^T\mathbf{x}_i + w_0) \geq 1 - \xi_i$



Assume that we can show that $\mathbf{w} = \sum_{i=1}^{N}\alpha_i\mathbf{x}_i$

$$\min_{\boldsymbol{\alpha}} \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j\underline{\mathbf{x}_i^T\mathbf{x}_j} + C\xi_i$$

subject to $y_i(\sum_{j=1}^{N}\alpha_n\underline{\mathbf{x}_i^T\mathbf{x}_j} + \alpha_0) \geq 1 - \xi_i$

Nonlinear SVM

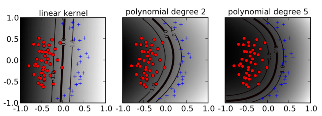
$$\min_{\boldsymbol{\alpha}} \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j\kappa(\mathbf{x}_i, \mathbf{x}_j) + C\xi_i$$

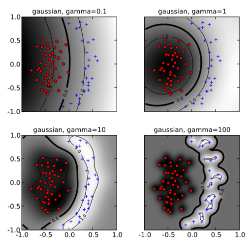
subject to $y_i(\sum_{j=1}^{N}\alpha_n\kappa(\mathbf{x}_i, \mathbf{x}_j) + \alpha_0) \geq 1 - \xi_i$

C: Trade-off parameter between the importance of a low error on the training data vs. finding wide margins that may give better generalization on test data.

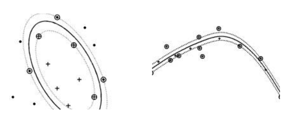
κ(.,.): Kernel function that determines the non-linear mapping. May contain additional parameters such as the width of a Gaussian kernel.

Nonlinear SVM - Examples





Source: A. Ben-Hur & J. Weston
A User's Guide to Support Vector Machines



Source: http://www.support-vector-machines.org/

SVM recipe

- Find a good software library, e.g., LIBSVM or SVM-Light.
- Normalize features, e.g., to the interval [-1,1] or [0,1]
- Choose a Gaussian kernel function
- Choose the width of the Gaussian kernel (σ) and the trade-off parameter *C* using cross-validation.

$$C = 2^{-5}, 2^{-3}, \ldots, 2^{15}$$
$$\sigma = 2^{-4}, 2^{-3}, \ldots, 2^{2} \longleftarrow \text{Depends on the scaling of the data}$$

Training data	Training data	Test data
Training data	Test data	Training data
Test data	Training data	Training data

Hsu et al.
A Practical Guide to Support Vector Classification

Nonlinear SVM - Summary

- Brings two clever and independent concepts together:
  - Large margin principle for good generalization
  - Kernel trick for making linear methods nonlinear
- Cost function ”landscape” less complex than in, e.g., neural network training.
- By many considered to be the state-of-the-art classifier around.
- Must store the support vectors, which can be many.
- Classification slower than, for example, boosting.

$$f(\mathbf{x}) = \sum_{k=1}^{N} \alpha_k \kappa(\mathbf{x}_k, \mathbf{x})$$

Kernel PCA

- Non-linear version of PCA.
- PCA can be written in terms of scalar products.
- Use the ”kernel trick”.

Kernel-PCA

$\mathbf{XX}^T\mathbf{e} = \lambda\mathbf{e}$ Ordinary PCA

Multiply from left with $\mathbf{X}^T$:

$$\mathbf{X}^T\mathbf{X}\underbrace{\mathbf{X}^T\mathbf{e}}_{\mathbf{f}} = \lambda\underbrace{\mathbf{X}^T\mathbf{e}}_{\mathbf{f}} \quad \rightarrow \mathbf{X}^T\mathbf{X}\mathbf{f} = \lambda\mathbf{f}$$

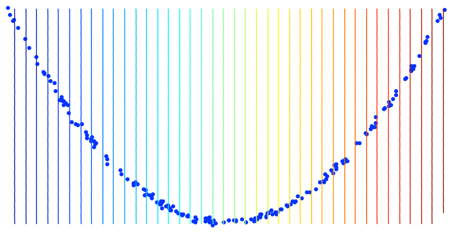
Eigen value problem on an inner product matrix
i.e. with coeficients defined by scalar products!

Kernel-PCA

- Similarly, PCA can be performed on any kernel matrix $\mathbf{K}$ whose components k_{ij} are defined by a kernel function $k_{ij} = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$
- The principal components are linear in the feature space but non-linear in the input space.

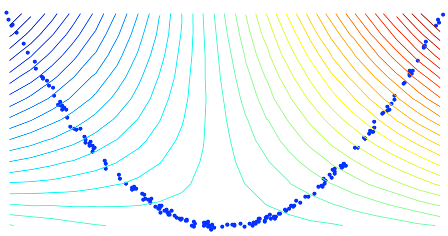
33

Linear PCA



34

KPCA with quadratic kernel



35

Kernels – Pros and cons

- Well understood linear methods carried out in a high-dimensional space where linear separability is more likely.
- Can achieve good performance

- How to choose the kernel and the kernel parameters?
- Have to store the training data.
- Need all combinations of training samples: (# samples)^2
- Training and classification can be computationally intensive

36

Some math concepts you'll see when reading about kernel methods

- **Mercer's theorem (1909)**
  Tells us when a kernel function represents a valid scalar product (in some space).
- **Reproducing Kernel Hilbert Spaces (RKHS)**
  Theory about the space for which our kernel is actually the scalar product.
- **Representer theorem**
  Tells us for which optimization problems the solution is a linear combination of the input vectors.

37