

# Intro. to machine learning (CSI 5325)

## Lecture 23: Support vector learning

Greg Hamerly

Spring 2008

Some content from Andrew Moore  
<http://www.cs.cmu.edu/~awm/tutorials>.

**1** Kernelizing other algorithms

**2** Other issues

## Distance $\rightarrow$ kernel distance

Euclidean (squared) distance between  $\mathbf{a}$  and  $\mathbf{b}$ :

$$\begin{aligned}d(\mathbf{a}, \mathbf{b})^2 &= \|\mathbf{a} - \mathbf{b}\|^2 = (\mathbf{a} - \mathbf{b}) \cdot (\mathbf{a} - \mathbf{b}) \\ &= \mathbf{a} \cdot \mathbf{a} - 2\mathbf{b} \cdot \mathbf{a} + \mathbf{b} \cdot \mathbf{b}\end{aligned}$$

which is just a few dot products...

Replace dot products with a kernel:

$$\begin{aligned}d_\Phi(\mathbf{a}, \mathbf{b})^2 &= \|\Phi(\mathbf{a}) - \Phi(\mathbf{b})\|^2 \\ &= \Phi(\mathbf{a}) \cdot \Phi(\mathbf{a}) - 2\Phi(\mathbf{a}) \cdot \Phi(\mathbf{b}) + \Phi(\mathbf{b}) \cdot \Phi(\mathbf{b}) \\ &= K(\mathbf{a}, \mathbf{a}) - 2K(\mathbf{b}, \mathbf{a}) + K(\mathbf{b}, \mathbf{b})\end{aligned}$$

This gives the straight-line distance of  $\mathbf{a}$  and  $\mathbf{b}$  in the mapped space of the kernel.

# Kernelizing other algorithms

Any algorithm that can be expressed entirely with dot products can be 'kernelized.'

Just replace all dot products with a chosen kernel.

Examples: PCA,  $k$ -nearest neighbor,  $k$ -means – any distance-based learning algorithm!

## What is a kernel?

A kernel is a function that represents an inner product in some mapped space.

Mercer's condition:  $K(\mathbf{a}, \mathbf{b})$  is a kernel representing a basis expansion

$$K(\mathbf{a}, \mathbf{b}) = \sum_i \phi(\mathbf{a})_i \phi(\mathbf{b})_i$$

iff for any  $g(\mathbf{a})$  such that  $\int g(\mathbf{a})^2 d\mathbf{a}$  is finite, then

$$\int K(\mathbf{a}, \mathbf{b}) g(\mathbf{a}) g(\mathbf{b}) d\mathbf{a} d\mathbf{b} \geq 0$$

In other words,  $K$  is positive semi-definite.

## VC dimension of an SVM

The VC dimension of any SVM can be bounded in several ways, but one good way for thinking about it is:

$$VC(SVM) \leq \left\lceil \frac{\text{diameter}}{\text{margin}} \right\rceil$$

where

- diameter is the diameter of the smallest sphere that can enclose all the high-dimensional term vectors from the training set
- margin is the smallest margin we'll let the SVM use

We can use the VC dimension with SRM (structural risk minimization) to choose the kernel, the kernel parameters, etc. But it's easier just to use cross-validation.

## VC dimension of an SVM

Interestingly, the VC dimension depends on the margin, but not necessarily on the complexity of the kernel.

In fact, some kernels have infinite VC dimension themselves.

- Gaussian (aka RBF) kernels represent an infinite feature space  $\Phi(\mathbf{x})$ , and also have infinite VC dimension!

## Another SVM error bound

Another bound on SVM error bounds: the number of support vectors.

$$E[P(\text{error})] \leq \frac{E[\text{number support vectors}]}{n}$$

where the expectation on the right is over all samples of size  $n$ .

So the number of support vectors has a relationship with accuracy!

# Constructing kernels

We can learn with any data for which there is a kernel.

- vectors in  $\mathbb{R}^d$  – simple
- text documents
- strings
- trees
- graphs (think subgraph isomorphism – expensive!)
- ...

Areas of active research are how to choose appropriate kernels, and developing appropriate and efficient kernels for structured data.

# Constructing kernels from existing kernels

Suppose

- $K_1$  and  $K_2$  are both kernels over  $X \times X$ ,  $X \subseteq \mathbb{R}^d$
- $f$  is a real-valued function over  $X$
- $\Phi$  is a feature map from  $X$  to  $\mathbb{R}^m$
- $K_3$  is a kernel over  $\mathbb{R}^m \times \mathbb{R}^m$
- $B$  is a positive semi-definite  $d \times d$  matrix

Then the following are all kernels:

$$K(\mathbf{a}, \mathbf{b}) = K_1(\mathbf{a}, \mathbf{b}) + K_2(\mathbf{a}, \mathbf{b})$$

$$K(\mathbf{a}, \mathbf{b}) = \alpha K_1(\mathbf{a}, \mathbf{b})$$

$$K(\mathbf{a}, \mathbf{b}) = K_1(\mathbf{a}, \mathbf{b})K_2(\mathbf{a}, \mathbf{b})$$

$$K(\mathbf{a}, \mathbf{b}) = f(\mathbf{a})f(\mathbf{b})$$

$$K(\mathbf{a}, \mathbf{b}) = K_3(\Phi(\mathbf{a}), \Phi(\mathbf{b}))$$

$$K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T B \mathbf{b}$$

## Multi-class with SVMs

SVMs are excellent for 2 classes, but what about more than that?

Standard approach: choose a method to decompose one  $m$ -class into multiple 2-class problems

- 1-versus-all (requires  $m$  SVMs)
- pairwise classifiers (requires  $m(m - 1)/2$  SVMs)
- error-correcting output codes (multiple  $p$ -versus- $q$  SVMs)

Training is easy – break the initial multi-class dataset into multiple binary-class datasets, and train a binary SVM for each.

Combining answers of multiple classifiers for making predictions can be tricky, but voting or looking for the most confident predictions works.

## SVMs and probabilities

The SVM is a discriminative model (as opposed to a generative one).

Thus, it's good at classification, but not at providing probabilities.

Some work has been done (Vapnik, Wabha, Platt, others) to take an SVM and use it to produce  $p(y = 1|x)$ .

Platt's idea: fit a sigmoid to  $f(x)$ .

# Support vector regression

Support vectors machines are for classification; but regression is also possible with similar ideas.

A key difference from least squares is that SVR cares only about errors that are further than  $\varepsilon$  from the regression line.

- this is called  $\varepsilon$ -insensitive loss

Such regressors can be sparse, just like SVMs, and can also use kernel functions for non-linear regression.

# SVM learning in practice

Don't write the software – use someone else's:

- SVMlight (Joachims)
- Matlab toolkits
- [www.support-vector-machines.org/SVM\\_soft.html](http://www.support-vector-machines.org/SVM_soft.html)