Support Vector Machines

Kernel Methods

Why SVMs?

- Question: at what serial number did the new $5 bill enter circulation?

Old Old Old Old New New New

Serial No.

Why SVMs?

- Question: at what serial number did the new $5 bill enter circulation?

- If we assume approximately uniformly-distributed observations, then the likelihood will be approximately uniform over [max(Old),min(New)]
- Min expected squared-error is max margin

Why SVMs?

- Sometimes the boundaries of the classes are more "informative" than the overall distribution of the classes.

Why SVMs?

- Sometimes the boundaries of the classes are more “informative” than the overall distribution of the classes.
- SVMs are faster.

Hard-margin SVMs

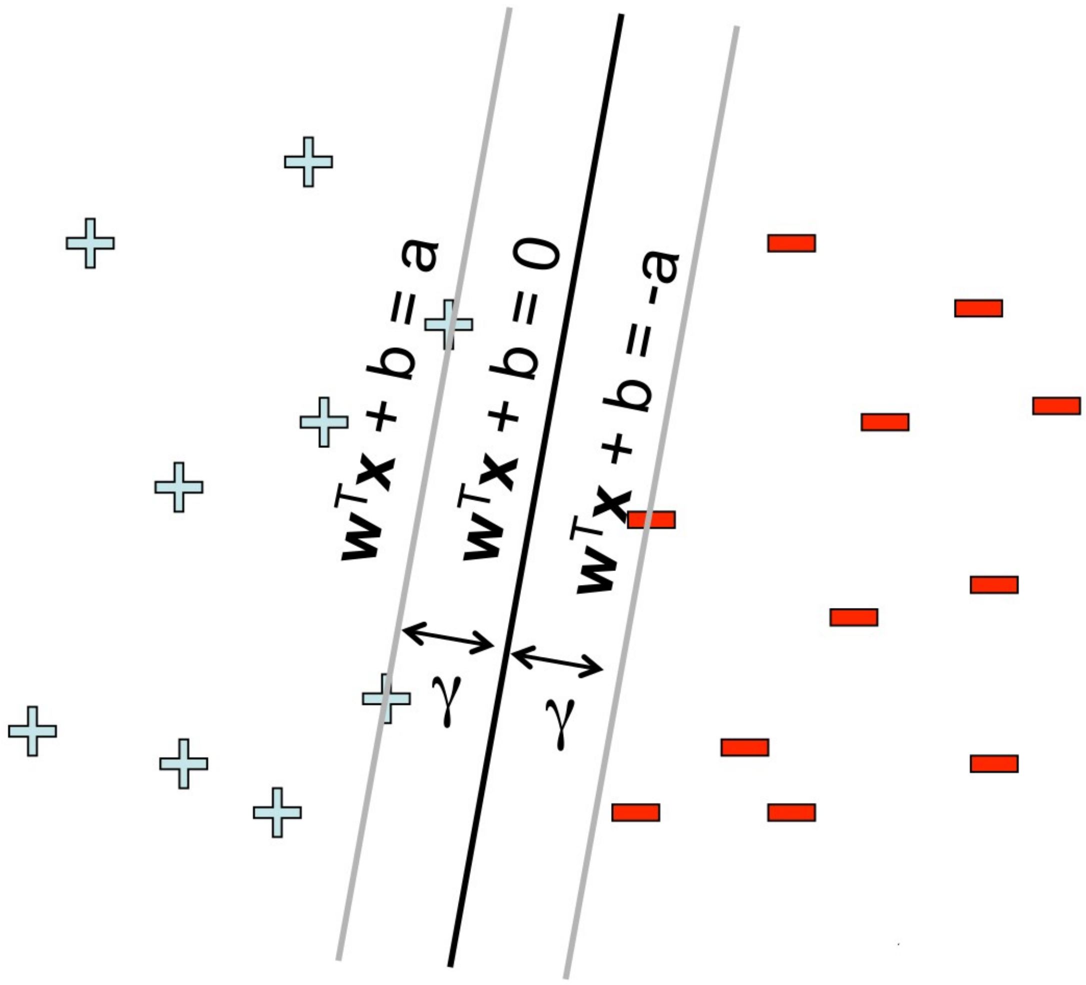
- Enforce that all points are out of the margin:

$$(\boldsymbol{w}^T \boldsymbol{x}_j + b)\, y_j \geq a$$

- Then maximize margin:

$$\max_w \gamma = \frac{a}{\|\boldsymbol{w}\|}$$

- Here, *a* is the margin after points are projected onto *w*
- solution is the same for any *a*

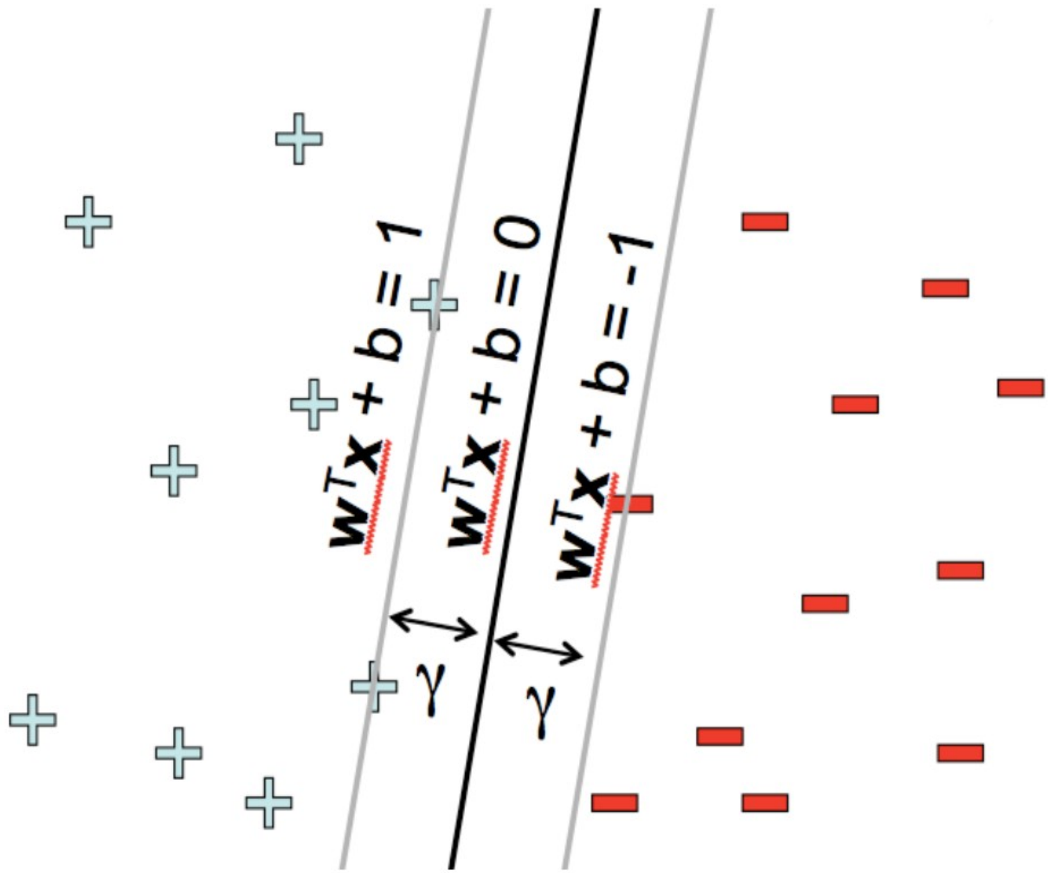


Hard-margin SVMs

- set *a*=1, rewrite:

$$\min_{w} \|\boldsymbol{w}\|$$

$$(\boldsymbol{w}^T \boldsymbol{x}_j + b)\, y_j \geq 1$$

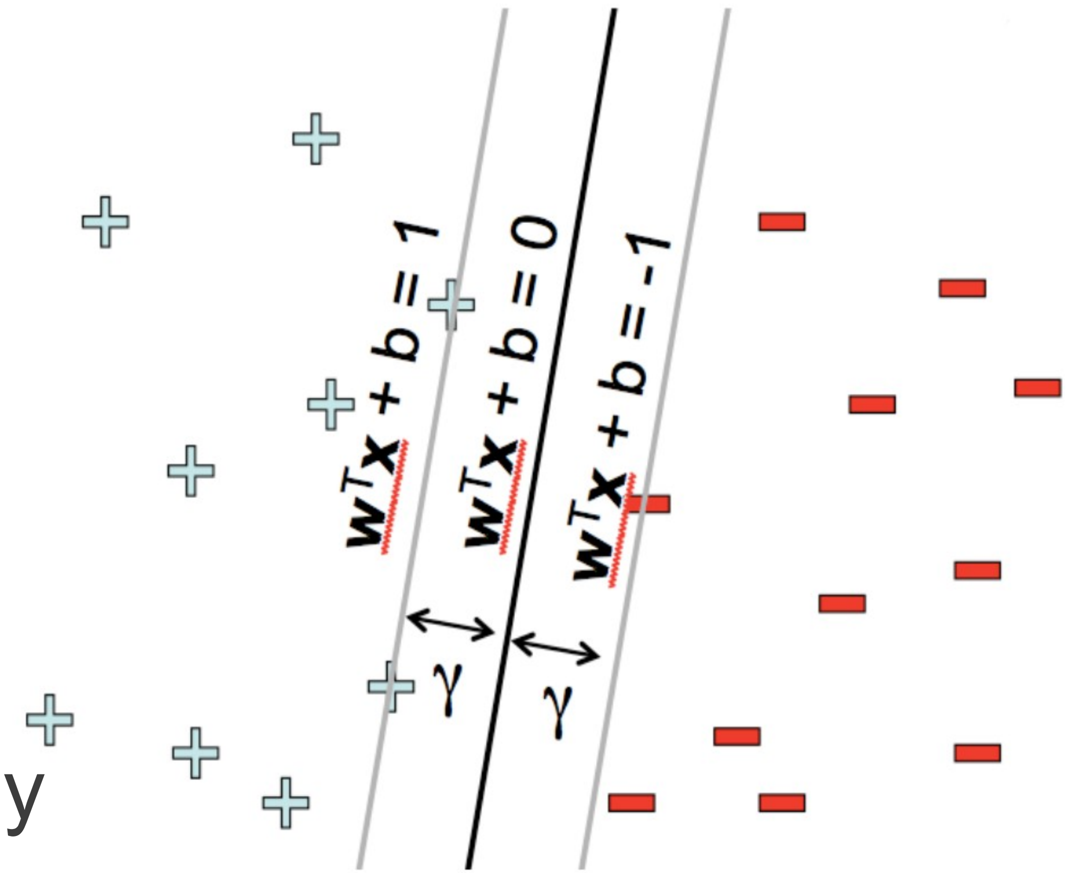


Hard-margin SVMs

- Dual form: W is a linear combination of training examples.

$$\boldsymbol{w} = \sum_{l=1}^{M} \alpha_l y_l \boldsymbol{x}_l$$

- Can optimize α's directly
- α's will be 0 except for support vectors.

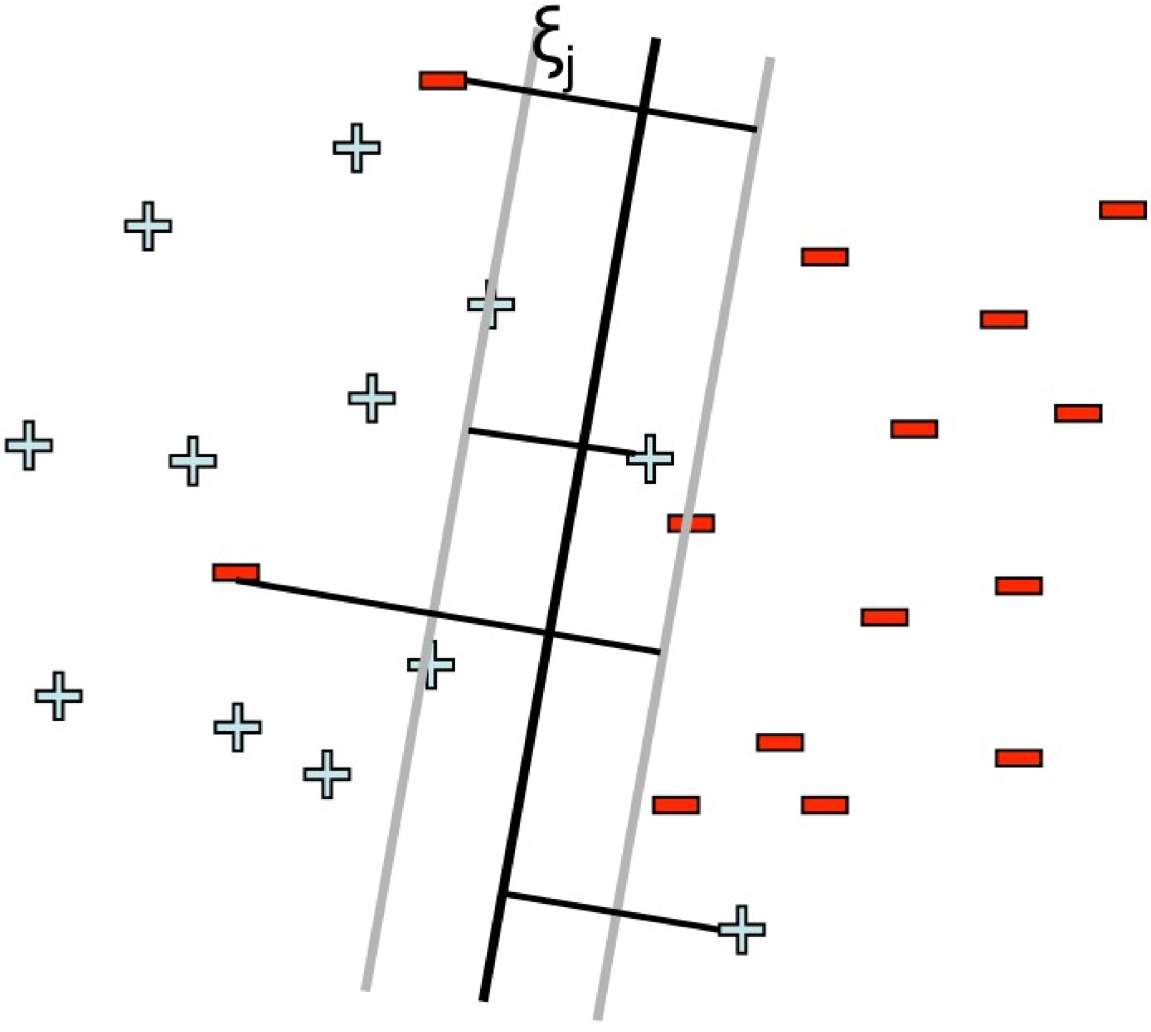


Soft-margin SVMs

- Slack variables ξ which represent how 'wrong' our prediction is.

$$\min_{w} \|\boldsymbol{w}\| + C \sum_{j} \xi_j$$

$$(\boldsymbol{w}^T \boldsymbol{x}_j + b)\, y_j \geq 1 - \xi_j$$



Support Vector Machines

Kernel Methods

Why Kernels?

- The HOG features of a patch:

(Dalal & Triggs 2005)

Why Kernels?

- Given this dog as input:

- This window is very close:

- And both of these windows are *somewhat* close:

Why Kernels?

- This window is very far:

- Distances mean nothing past a certain point
- We want a classifier that gives more weight to 'nearby' examples

Why Kernels?

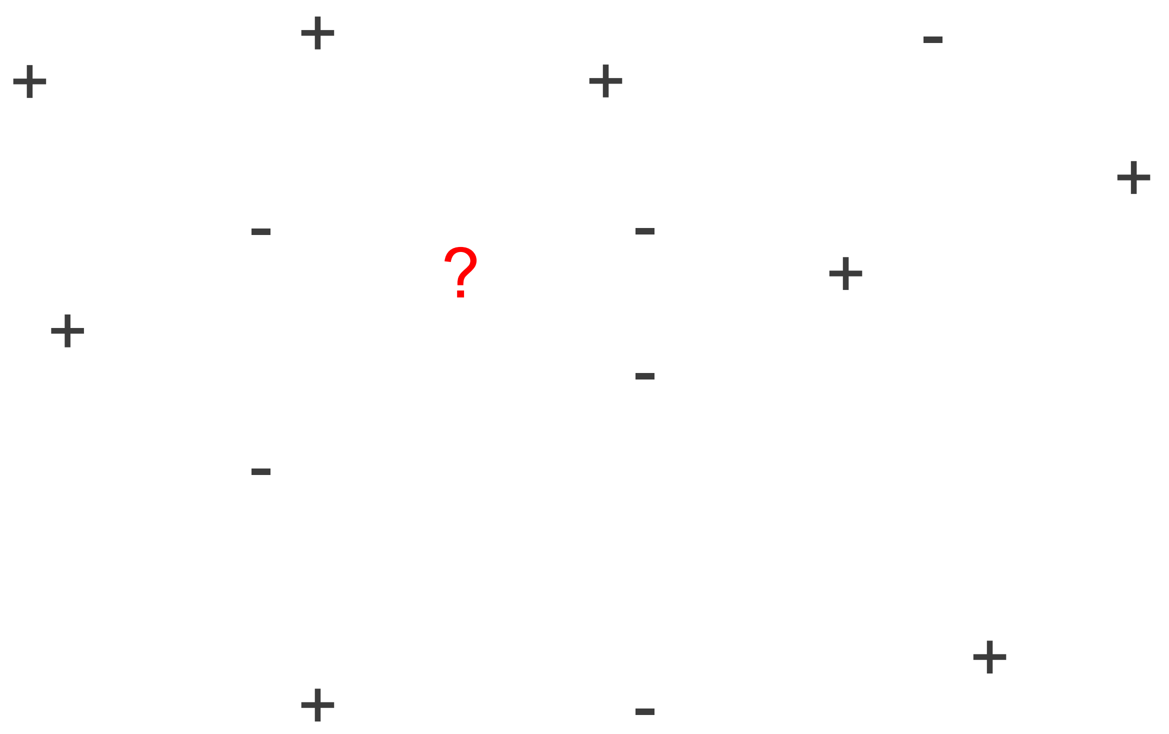
- Sometimes it is easier to define similarity between examples than it is to embed them in a feature space!
- Similarity of two patches *a* and *b*, for example:

$$\exp\left(-\frac{\|HOG(a)-HOG(b)\|_2^2}{2\sigma^2}\right)$$

- The kernel lets us not worry about the underlying HOG space.

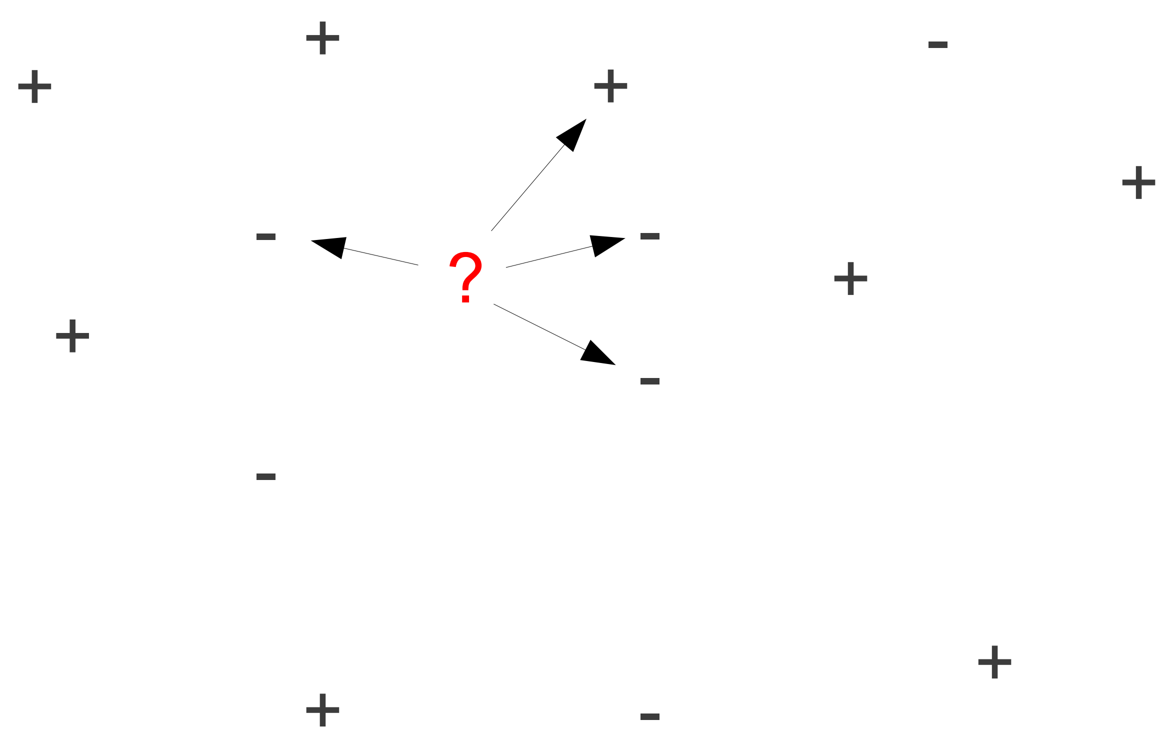
Classification & learning with Kernels

- Simplest idea: k-nearest neighbors



Classification & learning with Kernels

- Simplest idea: k-nearest neighbors
- Find nearest points using the kernel

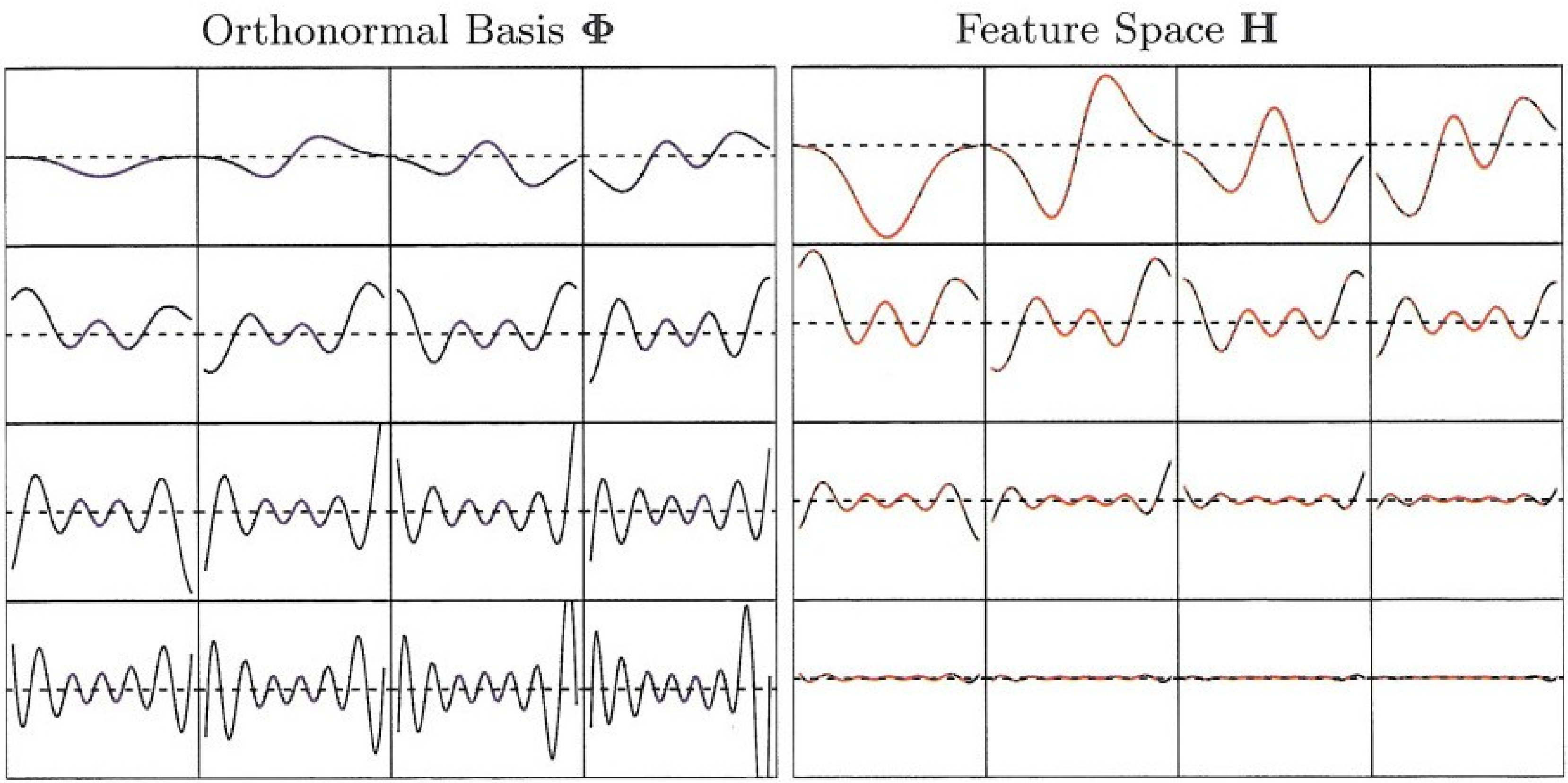


Linear methods with Kernels

- We want to maintain the properties of linear methods such as linear regression and, especially, support vector machines
- One approach: find a (possibly infinite-dimensional) space where dot product between two points in the space equals the kernel evaluated on the two points

Linear methods with Kernels

- Largest 16 bases corresponding to the Gaussian kernel in one dimension, over a bounded interval:



(Hastie, Tibshirani & Friedman 2009)

How do you compute these bases?

- You don't. You get lucky with math instead.
- This is the kernel trick: for many important problems, the final regression function has the form:

$$f(\boldsymbol{x}) = \sum_{x^l \in trainingSet} \alpha_l * \kappa(\boldsymbol{x}, \boldsymbol{x}^l)$$

- Where κ is the kernel and the α's are a function of only the training data.
- Plug in testing examples as x and get a prediction in time linear in the size of the training set.

How to compute the α's?

- For linear regression in the a known space, use this formula to compute the α's:

$$\alpha = (XX^T + \lambda I_m)^{-1} \boldsymbol{y}$$

$$f(x) = \sum_{x^l \in trainingSet} \alpha_l * \langle \boldsymbol{x}, \boldsymbol{x}^{\boldsymbol{l}} \rangle$$

- See Tom's slides for derivation

How to compute the α's?

- For linear regression in the expanded space, use this formula to compute the α's:

$$\boldsymbol{\alpha} = (K + \lambda I_m)^{-1} \boldsymbol{y}$$

$$f(\boldsymbol{x}) = \sum_{x^l \in trainingSet} \alpha_l * \kappa(\boldsymbol{x}, \boldsymbol{x}^l)$$

- Where $K_{ij} = \kappa(\boldsymbol{x}^i, \boldsymbol{x}^j)$
- See Tom's slides for derivation

This works for SVM's too

- For any valid kernel, the final SVM classifier will have the form:

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{x}^l \in trainingSet} \alpha_l * \kappa(\boldsymbol{x}, \boldsymbol{x}^l)$$

- Compute *α*'s via:

$$\begin{array}{lll} \max\limits_{\alpha_1 \ldots \alpha_M} & \sum_{l=1}^{M} \alpha_l - \frac{1}{2} \sum_{j=1}^{M} \sum_{k=1}^{M} \alpha_j \alpha_k y_j y_k \ \kappa(\mathbf{x_j}, \mathbf{x_k}) & \\ \text{s.t.} & \alpha_l \geq 0 & \forall l \in \text{ training examples} \\ & \sum_{l=1}^{M} \alpha_l y_l = 0 & \end{array}$$