Workshop on Web-scale Vision and Social Media
ECCV 2012, Firenze, Italy
October, 2012

Linearized Smooth Additive Classifiers

Subhransu Maji
Research Assistant Professor
Toyota Technological Institute at Chicago

Generalized Additive Models

$$f(x_1, x_2, \ldots, x_n) = f_1(x_1) + f_2(x_2) + \ldots + f_n(x_n)$$

- Why use them?
  - **Efficiency** : can be efficiently evaluated
  - **Interpretability** : Simple generalization of linear classifiers, i.e., may lead to models that are interpretable
- Well known in the statistics community
  - Generalized Additive Models (Hastie & Tibshirani '90)
- However traditional learning algorithms do not scale well (e.g. "backfitting algorithm")

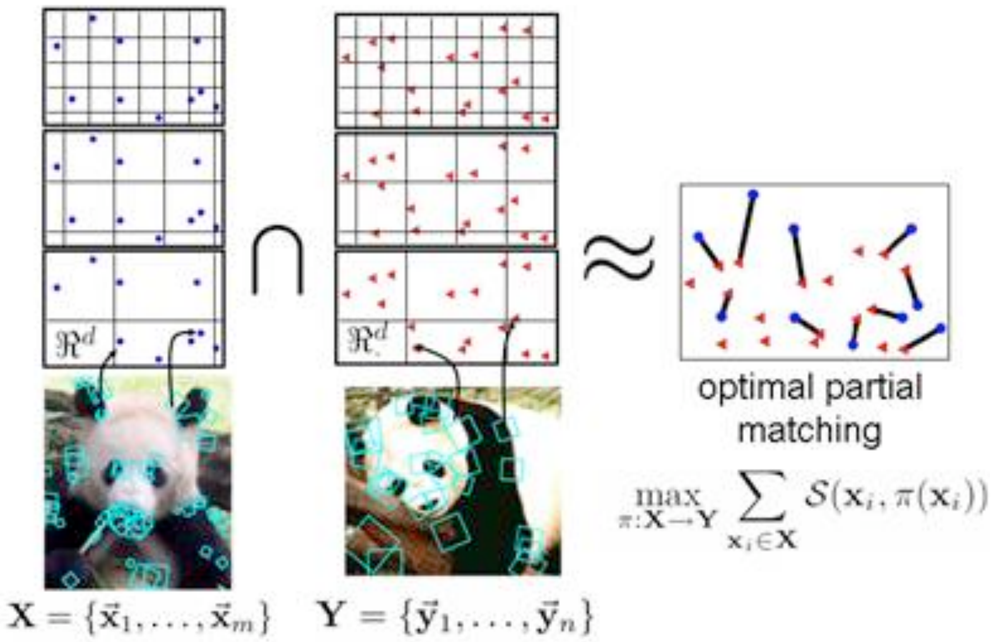
Additive kernels in computer vision

- Images are represented as histograms of low level features such as color and texture [Swain and Ballard 01, Odone et al. 05]

- Histogram based similarity measures are typically additive

$$K_{\min}(\mathbf{x}, \mathbf{y}) = \sum \min(x_i, y_i) \qquad K_{\chi^2}(\mathbf{x}, \mathbf{y}) = \sum \frac{2x_i y_i}{x_i + y_i}$$

- Other examples of additive kernels based on approximate correspondence :



Pyramid Match Kernel,
Grauman and Darrell, CVPR'05

Spatial Pyramid Match Kernel,
Lazebnik, Schmidt and Ponce, CVPR'06

Directly learning additive classifiers

$$f(x_1, x_2, \ldots, x_n) = f_1(x_1) + f_2(x_2) + \ldots + f_n(x_n)$$

A SVM like optimization framework

$$\min_{f \in F} \sum_k l\left(y^k, f(\mathbf{x}^k)\right) + \lambda R(f)$$

$$\mathbf{x}^k \in \mathbb{R}^d$$

$$y^k \in \{+1, -1\}$$

Loss function on the data
e.g. hinge loss function

$$l\left(y^k, f(\mathbf{x}^k)\right) = \max(0, 1 - y^k f(\mathbf{x}^k))$$

Regularization
e.g. derivative norm

Strategy for learning additive classifiers

$$\min_{f \in F} \sum_k l\left(y^k, f(\mathbf{x}^k)\right) + \lambda R(f)$$

$$\mathbf{x}^k \in \mathbb{R}^d$$

$$y^k \in \{+1, -1\}$$

Strategy for learning additive classifiers

$$\min_{f \in F} \sum_k l\left(y^k, f(\mathbf{x}^k)\right) + \lambda R(f)$$

$$\mathbf{x}^k \in \mathbb{R}^d$$

$$y^k \in \{+1, -1\}$$

$$f_i = \sum_j w_j^i \phi_j^i$$

Basis expansion

Strategy for learning additive classifiers

$$\min_{f \in F} \sum_k l\left(y^k, f(\mathbf{x}^k)\right) + \lambda R(f)$$

$$\mathbf{x}^k \in \mathbb{R}^d$$

$$y^k \in \{+1, -1\}$$

$$f_i = \sum_j w_j^i \phi_j^i$$

Basis expansion

Smoothness penalty on the weights

Strategy for learning additive classifiers

$$\min_{f \in F} \sum_k l\left(y^k, f(\mathbf{x}^k)\right) + \lambda R(f)$$

$$\mathbf{x}^k \in \mathbb{R}^d$$

$$y^k \in \{+1, -1\}$$

$$f_i = \sum_j w_j^i \phi_j^i$$

Basis expansion

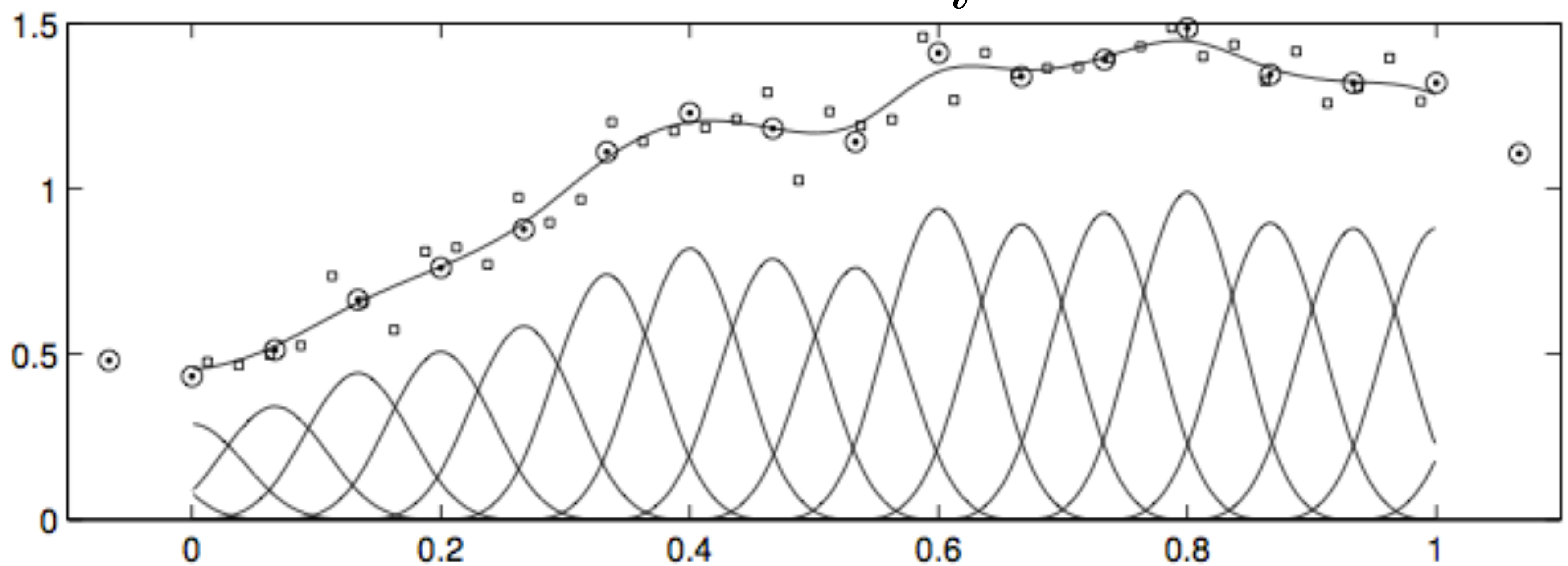
Smoothness penalty on the weights

- Search for *representations* of the function and *regularization* for which the optimization can be efficiently solved
- In particular we want to leverage the latest methods for learning linear classifiers (almost linear time algorithms)

Spline embeddings : motivation

$$f(x_1, x_2, \ldots, x_n) = f_1(x_1) + f_2(x_2) + \ldots + f_n(x_n)$$

$$\sum_i w_i \phi_i$$



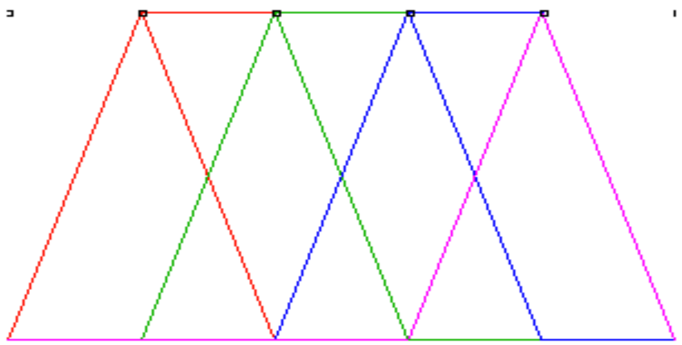
- Represent each function using a uniformly spaced spline basis
- Motivated by our earlier analysis that splines approximate additive classifiers well
- Popularized by Eilers and Marx (P-Splines '02, '04) for additive modeling
- Well known in graphics (DeBoor '01)
- Question: What is the regularization on **w**?

Figure from Eilers & Marx '04

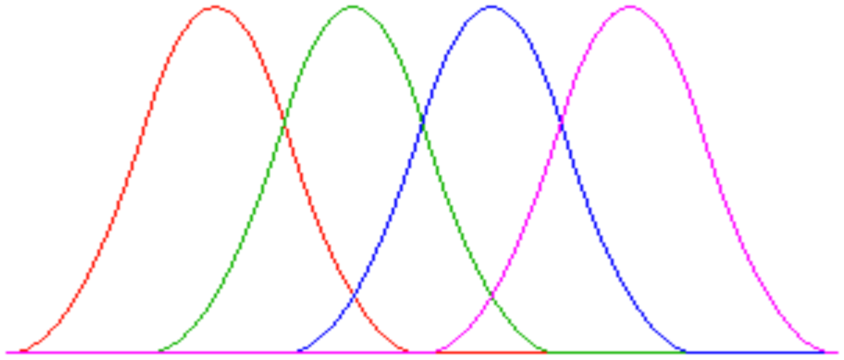
Spline embeddings : representation

Representation

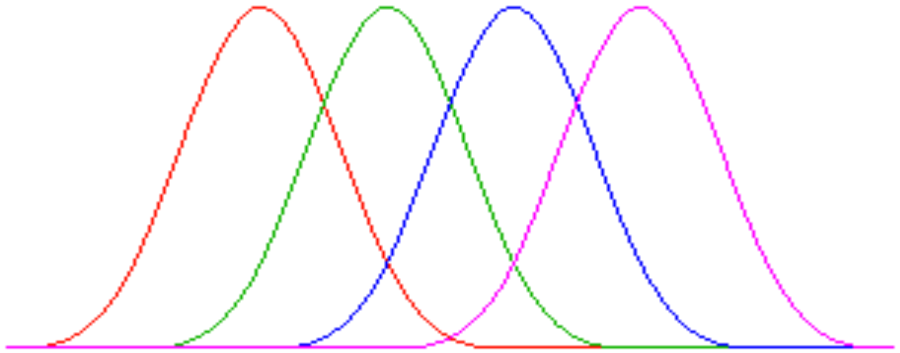
$$f(x) = \mathbf{w}^T \mathbf{\Phi}(x)$$



Linear B-Spline



Quadratic B-Spline



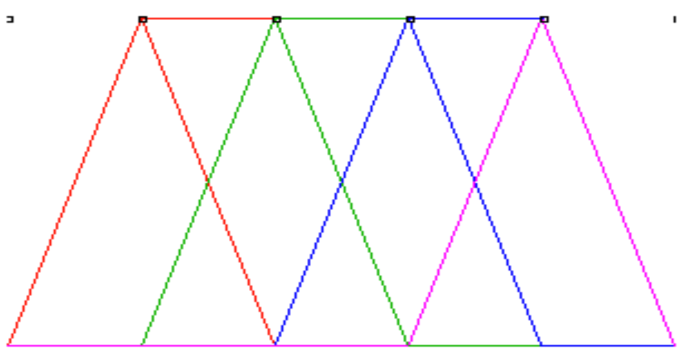
Cubic B-Spline

Project data onto a uniformly spaced spline basis

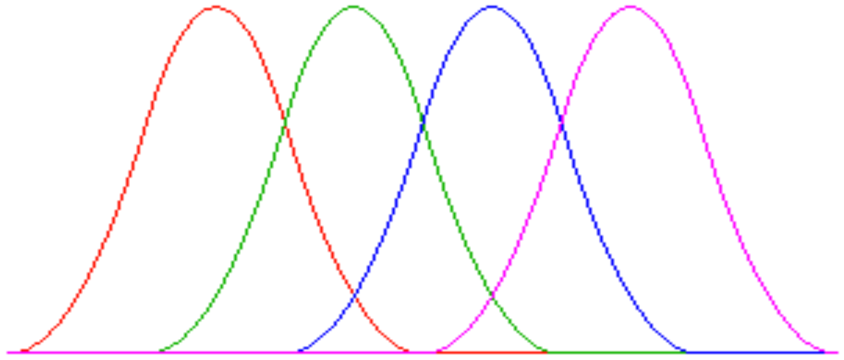
Spline embeddings : regularization

Representation

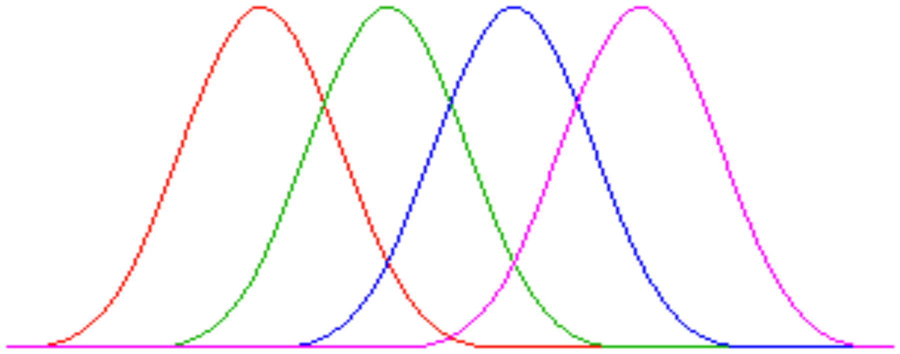
$$f(x) = \mathbf{w}^T \mathbf{\Phi}(x)$$



Linear B-Spline



Quadratic B-Spline



Cubic B-Spline

Regularization

$$R(f) = \mathbf{w}^T \mathbf{H} \mathbf{w} \qquad \mathbf{H} = \mathbf{D}^T \mathbf{D}$$

$$\mathbf{D}_0 = \mathbf{I} \qquad \mathbf{D}_1 = \begin{pmatrix} 1 & & & & \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & & \cdots & \\ & & & -1 & 1 \end{pmatrix} \qquad \mathbf{D}_2 = \begin{pmatrix} 1 & & & & \\ -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & & & \cdots & \\ & & 1 & -2 & 1 \end{pmatrix}$$

Penalize differences between adjacent weights

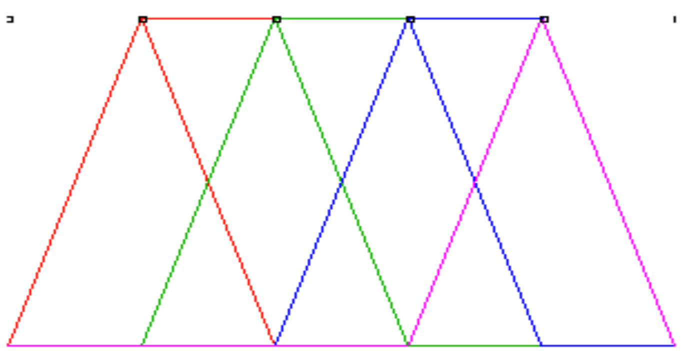
Ensures that the learned function is smooth

Similar to Penalized Splines of Eilers and Marx '02

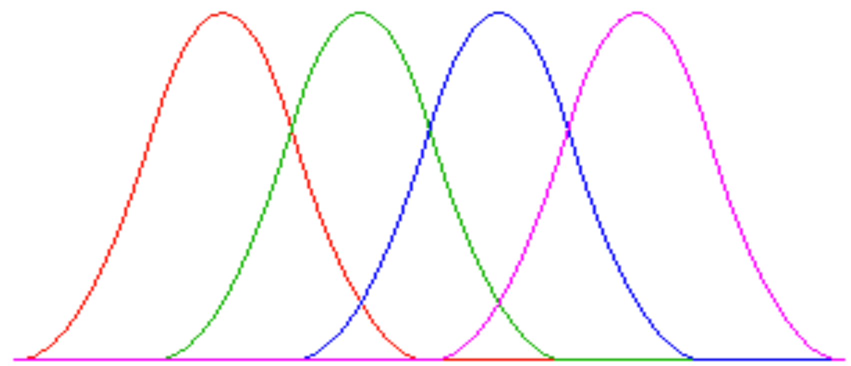
Spline embeddings : optimization

Representation

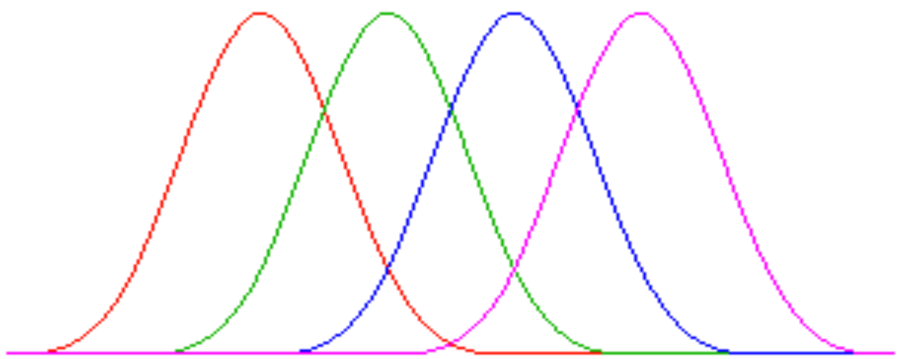
$$f(x) = \mathbf{w}^T \mathbf{\Phi}(x)$$



Linear B-Spline



Quadratic B-Spline



Cubic B-Spline

Regularization

$$R(f) = \mathbf{w}^T \mathbf{H} \mathbf{w} \qquad \mathbf{H} = \mathbf{D}^T \mathbf{D}$$

$$\mathbf{D}_0 = \mathbf{I} \qquad \mathbf{D}_1 = \begin{pmatrix} 1 & & & & \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & & \cdots & \\ & & & -1 & 1 \end{pmatrix} \qquad \mathbf{D}_2 = \begin{pmatrix} 1 & & & & \\ -2 & 1 & & & \\ & 1 & -2 & 1 & \\ & & & \cdots & \\ & & 1 & -2 & 1 \end{pmatrix}$$

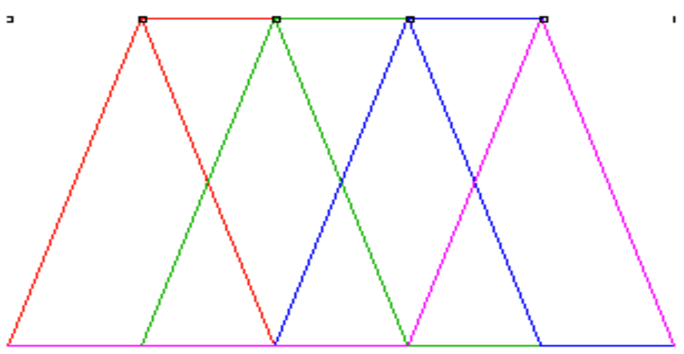
Optimization

$$c(\mathbf{w}) = \frac{\lambda}{2} \mathbf{w}^T \mathbf{D}_d^T \mathbf{D}_d \mathbf{w} + \frac{1}{n} \sum_k \max\left(0, 1 - y^k \left(\mathbf{w}^T \mathbf{\Phi}(x^k)\right)\right)$$

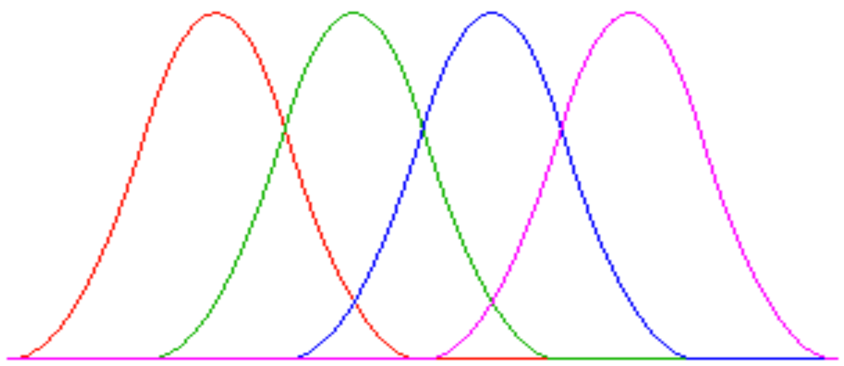
Spline embeddings : linear case

Representation

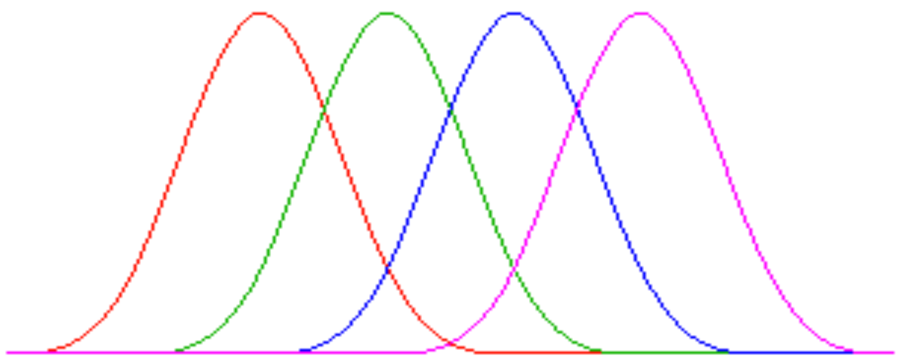
$$f(x) = \mathbf{w}^T \mathbf{\Phi}(x)$$



Linear B-Spline



Quadratic B-Spline



Cubic B-Spline

Regularization

$$\mathbf{D}_0 = \mathbf{I}$$

zero order differences [Maji and Berg, ICCV '09]

Reduces to a standard linear SVM

Projected features are sparse. At most k non-zero entries for a basis of degree k.

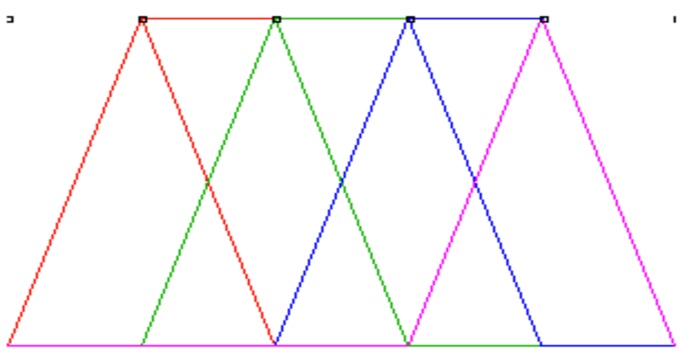
Optimization

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{\Phi}(x^k)\right)\right)$$

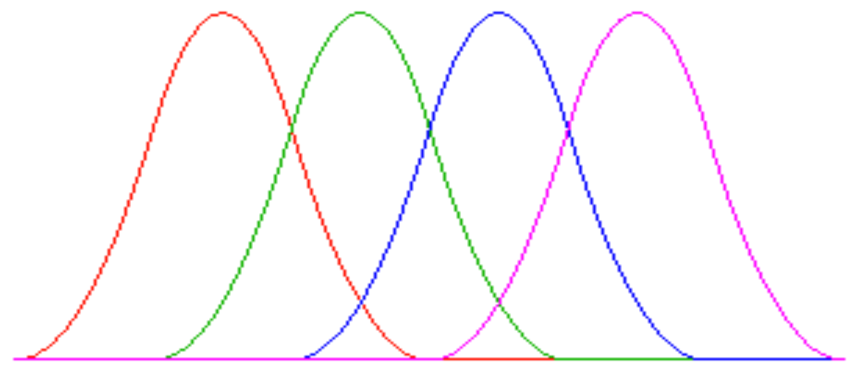
Spline embeddings : general case

Representation

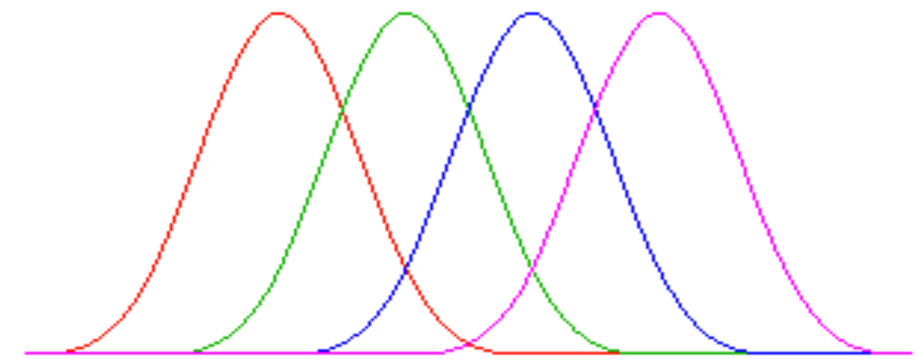
$$f(x) = \mathbf{w}^T \mathbf{\Phi}(x)$$



Linear B-Spline



Quadratic B-Spline



Cubic B-Spline

Regularization

$$\mathbf{D}_1 = \begin{pmatrix} 1 & & & & \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & \cdots & & \\ & & & -1 & 1 \end{pmatrix}$$

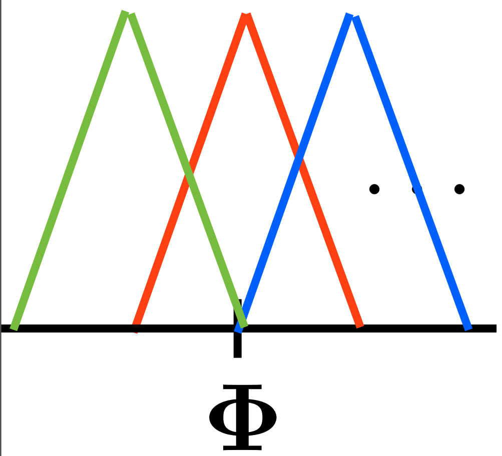
Penalize first order differences. Non-standard SVM due to regularization [Eilers & Marx '02]

Can offer better smoothness when some bins have few data points

Optimization

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{D}_1^T\mathbf{D}_1\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{\Phi}(x^k)\right)\right)$$

Spline embeddings : linearization

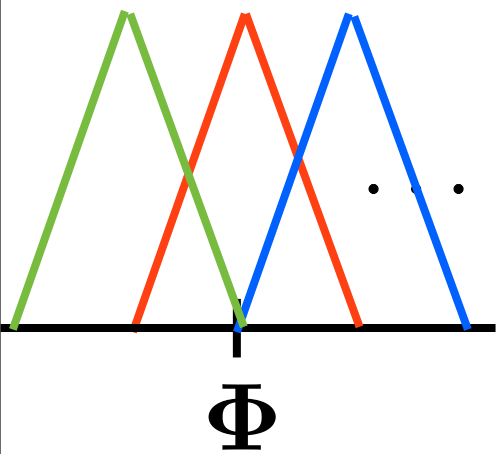


$$\mathbf{D}_1 = \begin{pmatrix} 1 & & & & \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & & \cdots & \\ & & & -1 & 1 \end{pmatrix}$$

Original Problem

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{D}_d^T\mathbf{D}_d\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{\Phi}(x^k)\right)\right)$$

Spline embeddings : linearization



$$\mathbf{D}_1 = \begin{pmatrix} 1 & & & & \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & \cdots & & \\ & & & -1 & 1 \end{pmatrix}$$

$$\mathbf{D}_1^{-T} = \begin{pmatrix} 1 & 1 & \dots & 1 & 1 \\ 0 & 1 & \dots & 1 & 1 \\ & 0 & \dots & 1 & 1 \\ & & \dots & & \\ & & & 1 & 1 \\ & & & 0 & 1 \end{pmatrix}$$

Original Problem

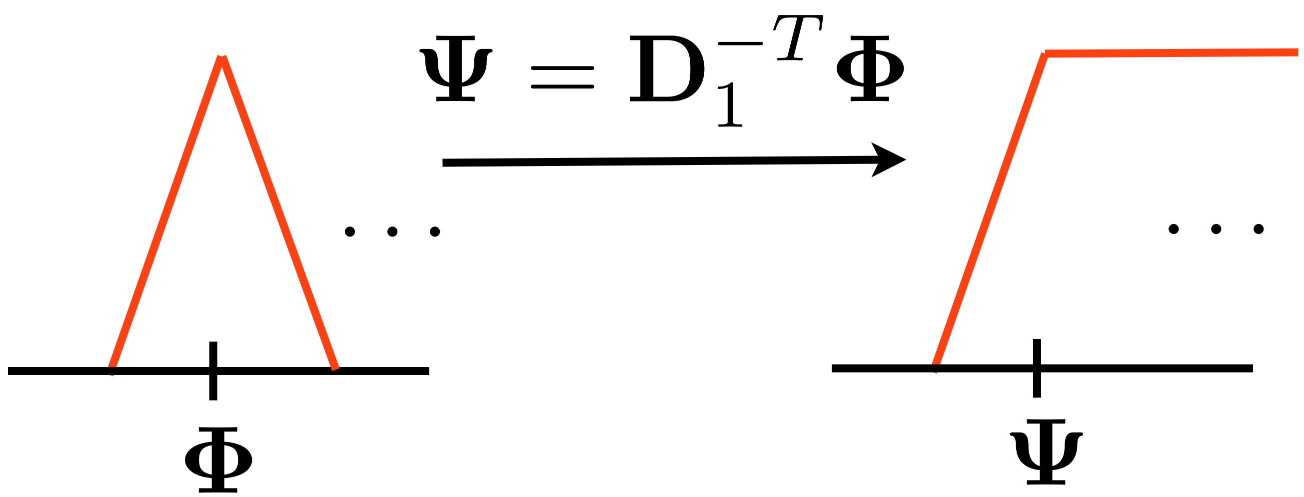
Upper Triangular

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{D}_d^T\mathbf{D}_d\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{\Phi}(x^k)\right)\right)$$

Re-Parameterization of the weight

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{D}_1^{-T}\mathbf{\Phi}(x^k)\right)\right)$$

Spline embeddings : linearization



$$\mathbf{D}_1^{-T} = \begin{pmatrix} 1 & 1 & \dots & 1 & 1 \\ 0 & 1 & \dots & 1 & 1 \\ & 0 & \dots & 1 & 1 \\ & & \dots & & \\ & & & 1 & 1 \\ & & & 0 & 1 \end{pmatrix}$$

Equivalently a change of basis

Upper Triangular

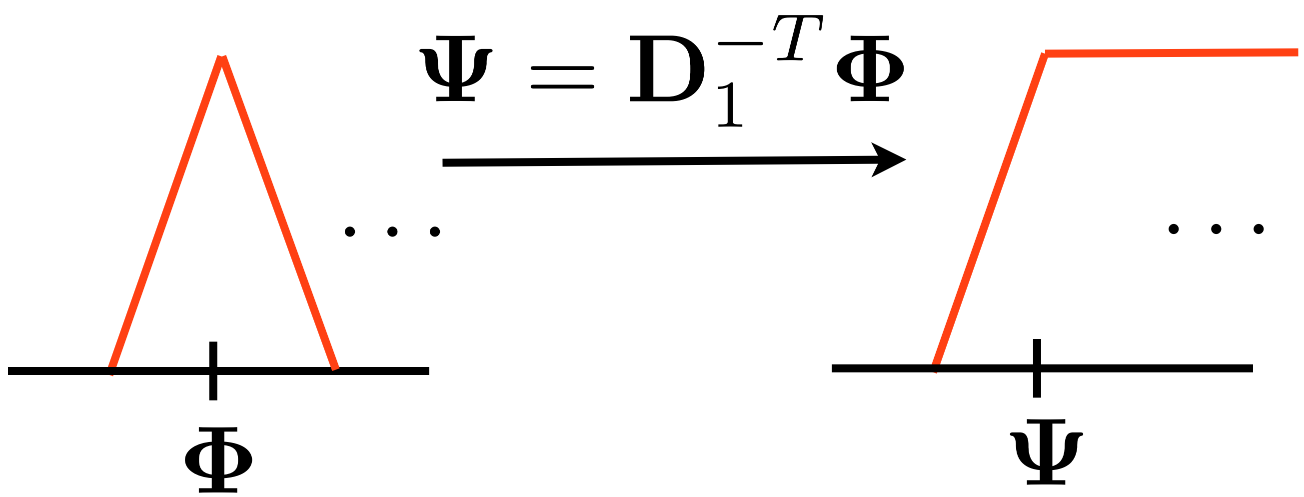
$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{D}_d^T\mathbf{D}_d\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{\Phi}(x^k)\right)\right)$$

Re-Parameterization of the features

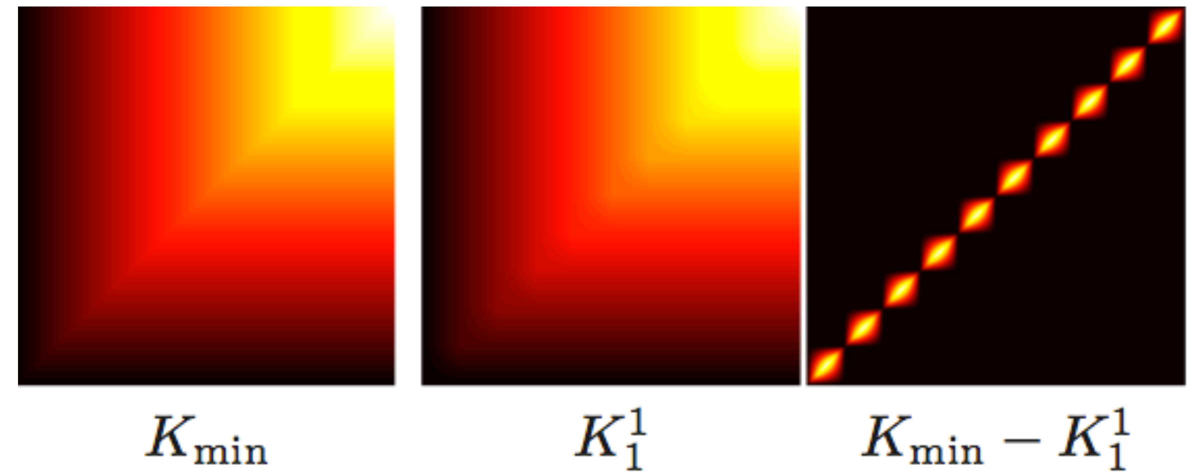
$\mathbf{\Psi}$

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\boxed{\mathbf{D}_1^{-T}\mathbf{\Phi}(x^k)}\right)\right)$$

Spline embeddings : linearization



Implicit kernel



$$K_1^1 = \mathbf{\Psi}^T\mathbf{\Psi}$$

Equivalently a change of basis

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{D}_d^T\mathbf{D}_d\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{\Phi}(x^k)\right)\right)$$

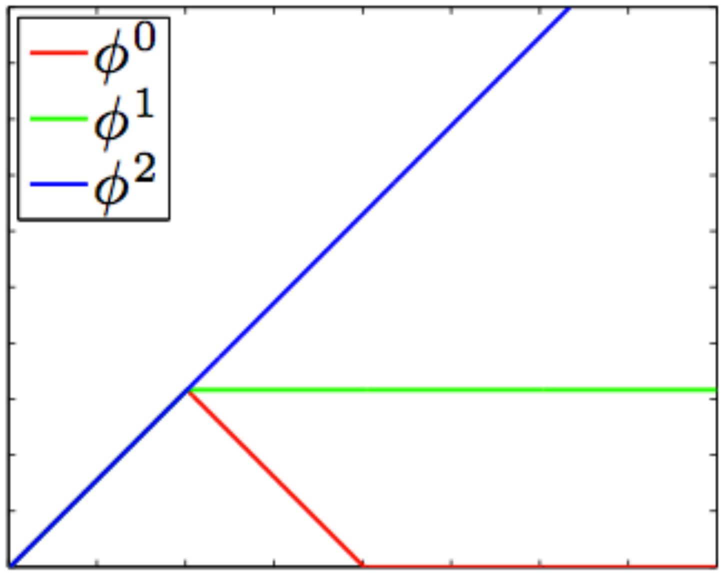
Re-Parameterization of the features

$\mathbf{\Psi}$

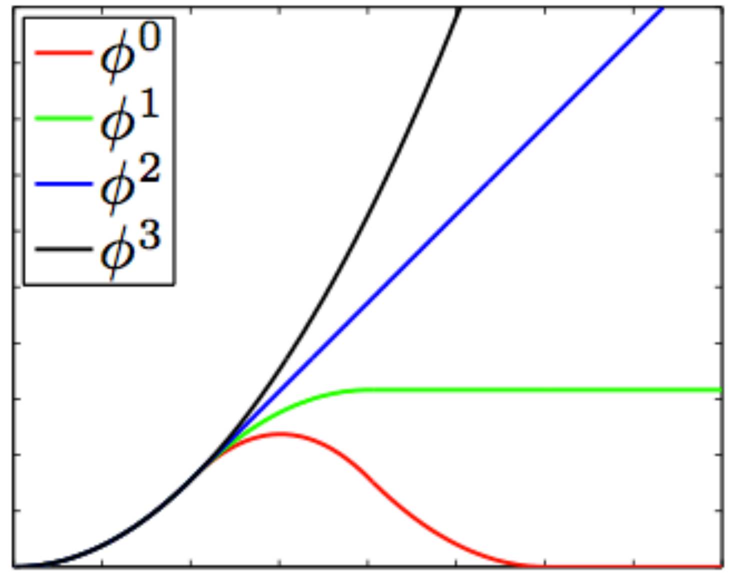
$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{D}_1^{-T}\mathbf{\Phi}(x^k)\right)\right)$$

Spline embeddings : visualizing the kernel

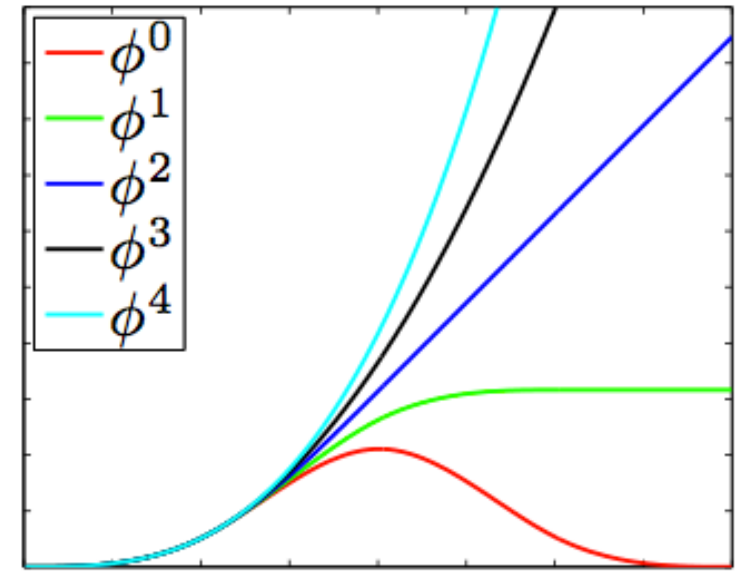
Various basis and regularization



Linear B-Spline

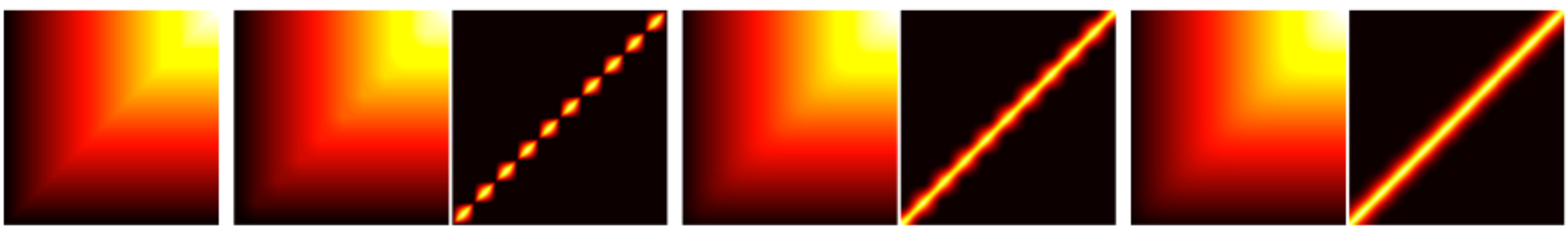


Quadratic B-Spline



Cubic B-Spline

Fixing order(regularization)=1, and varying degree(basis)

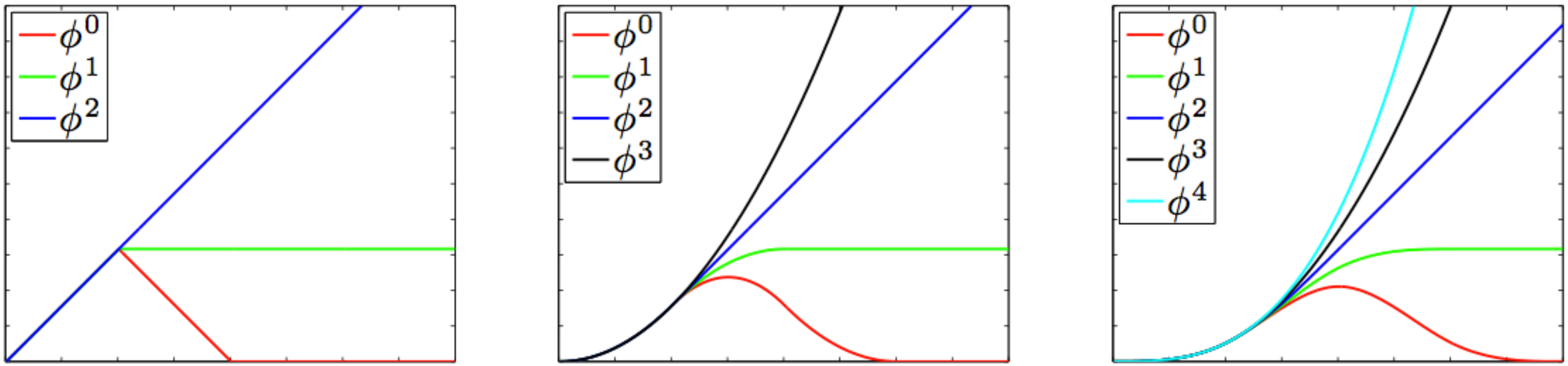


$K_{\min}$ $\quad K_1^1$ $\quad K_{\min} - K_1^1$ $\quad K_2^1$ $\quad K_{\min} - K_2^1$ $\quad K_3^1$ $\quad K_{\min} - K_3^1$

Smooth variants of the min kernel

Spline embeddings : visualizing the basis

Various basis and regularization



Linear B-Spline   Quadratic B-Spline   Cubic B-Spline

$$\phi_i(x) = (x - \tau_i)_+^r$$

The basis are equivalent to truncated polynomial basis if:
degree(basis) - order(regularization) = 1 [DeBoor '01]

Solving the optimization efficiently

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{D}_d^T\mathbf{D}_d\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{\Phi}(x^k)\right)\right)$$

Original problem : non-standard regularization

Solving the optimization efficiently

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{D}_d^T\mathbf{D}_d\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{\Phi}(x^k)\right)\right)$$

Original problem : non-standard regularization

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{D}_d^{-T}\mathbf{\Phi}(x^k)\right)\right)$$

Modified problem : dense features (memory bottleneck)

Solving the optimization efficiently

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{D}_d^T\mathbf{D}_d\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{\Phi}(x^k)\right)\right)$$

Original problem : non-standard regularization

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{D}_d^{-T}\mathbf{\Phi}(x^k)\right)\right)$$

Modified problem : dense features (memory bottleneck)

Solution : implicit representation

maintain: $\mathbf{w}_d = \mathbf{D}_d^{-1}\mathbf{w}$

classification: $f(x) = \mathbf{w}_d^T\mathbf{\Phi}(x)$ ⟵ O(d) vs O(n)

Solving the optimization efficiently

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{D}_d^T\mathbf{D}_d\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{\Phi}(x^k)\right)\right)$$

Original problem : non-standard regularization

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{D}_d^{-T}\mathbf{\Phi}(x^k)\right)\right)$$

Modified problem : dense features (memory bottleneck)

Solution : implicit representation

maintain: $\mathbf{w}_d = \mathbf{D}_d^{-1}\mathbf{w}$

classification: $f(x) = \mathbf{w}_d^T\mathbf{\Phi}(x)$ ⟵ O(d) vs O(n)

update: $\mathbf{w} \leftarrow \mathbf{w} - \eta\mathbf{D}_d^{-T}\boldsymbol{\Phi}(x^k)$

$\mathbf{w}_d \leftarrow \mathbf{w}_d - \eta\mathbf{L}_d\boldsymbol{\Phi}(x^k)$

$\mathbf{L}_d = \mathbf{D}_d^{-1}\mathbf{D}_d^{-T}$

Solving the optimization efficiently

Computing the updates

maintain: $\mathbf{w}_d = \mathbf{D}_d^{-1}\mathbf{w}$

classification: $f(x) = \mathbf{w}_d^T \mathbf{\Phi}(x)$

update: $\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{D}_d^{-T} \boldsymbol{\Phi}(x^k)$

$$\mathbf{w}_d \leftarrow \mathbf{w}_d - \eta \mathbf{L}_d \boldsymbol{\Phi}(x^k)$$

$$\mathbf{L}_d = \mathbf{D}_d^{-1}\mathbf{D}_d^{-T}$$

Computing the updates

maintain: $\mathbf{w}_d = \mathbf{D}_d^{-1}\mathbf{w}$

classification: $f(x) = \mathbf{w}_d^T \mathbf{\Phi}(x)$

update: $\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{D}_d^{-T} \boldsymbol{\Phi}(x^k)$

$$\mathbf{w}_d \leftarrow \mathbf{w}_d - \eta \mathbf{L}_d \boldsymbol{\Phi}(x^k)$$

$$\mathbf{L}_d = \mathbf{D}_d^{-1}\mathbf{D}_d^{-T}$$

Given n linearly spaced basis of degree d, computing updates to w_d takes $O(n^2)$ time.

Solving the optimization efficiently

Computing the updates

maintain: $\mathbf{w}_d = \mathbf{D}_d^{-1}\mathbf{w}$

classification: $f(x) = \mathbf{w}_d^T \mathbf{\Phi}(x)$

update: $\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{D}_d^{-T} \boldsymbol{\Phi}(x^k)$

$$\mathbf{w}_d \leftarrow \mathbf{w}_d - \eta \mathbf{L}_d \boldsymbol{\Phi}(x^k) \qquad \mathbf{L}_d = \mathbf{D}_d^{-1}\mathbf{D}_d^{-T}$$

Given n linearly spaced basis of degree d, computing updates to w_d takes $O(n^2)$ time.

However, one can compute it on O(nd) by exploiting the structure of D (see paper below)

S. Maji, Linearized Smooth Additive Classifiers, ECCV WS 2012

Solving the optimization efficiently

Computing the updates

maintain: $\mathbf{w}_d = \mathbf{D}_d^{-1}\mathbf{w}$

classification: $f(x) = \mathbf{w}_d^T \mathbf{\Phi}(x)$

update: $\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{D}_d^{-T} \boldsymbol{\Phi}(x^k)$

$$\mathbf{w}_d \leftarrow \mathbf{w}_d - \eta \mathbf{L}_d \boldsymbol{\Phi}(x^k) \qquad \mathbf{L}_d = \mathbf{D}_d^{-1}\mathbf{D}_d^{-T}$$

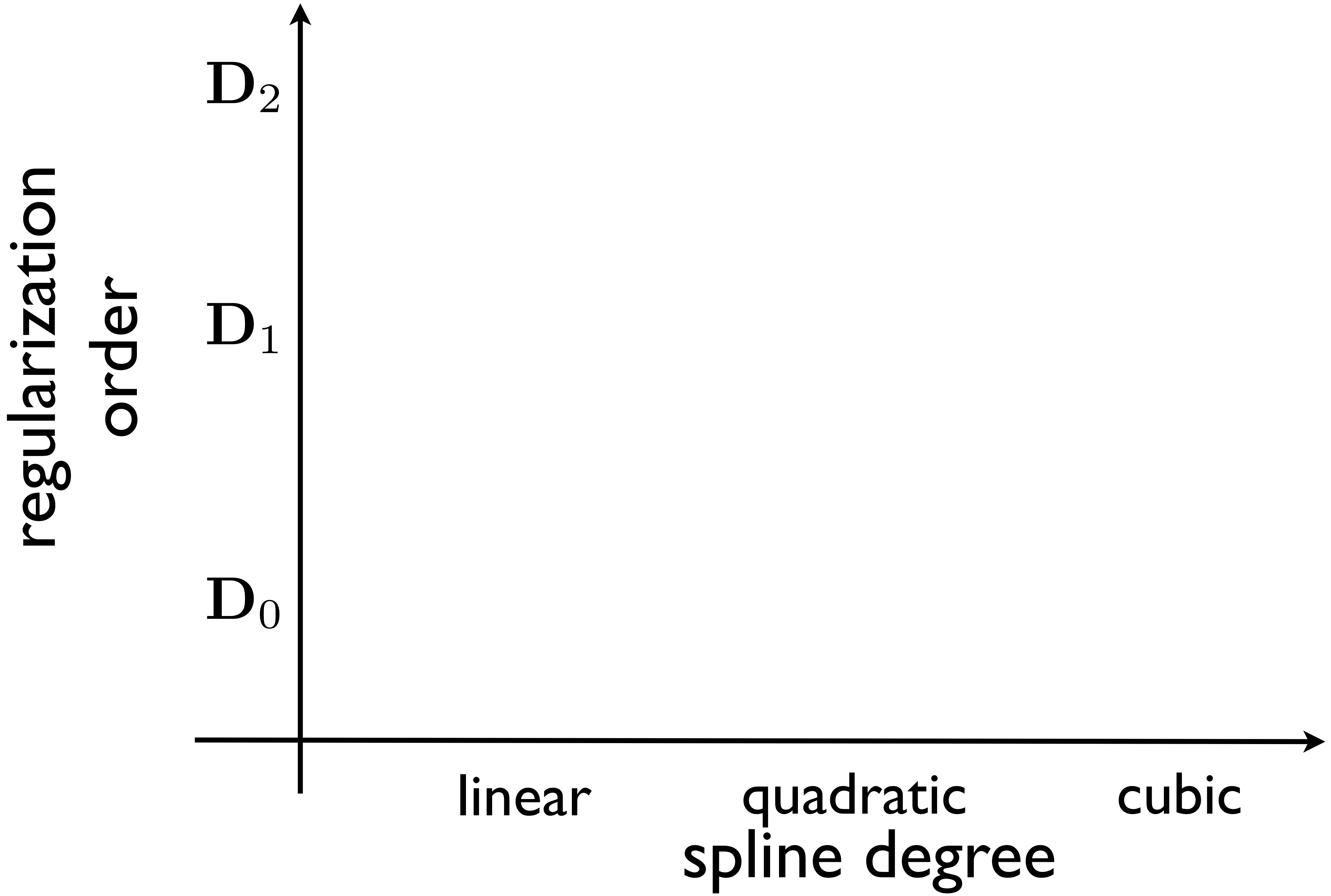
Given n linearly spaced basis of degree d, computing updates to w_d takes $O(n^2)$ time.

However, one can compute it on O(nd) by exploiting the structure of D (see paper below)

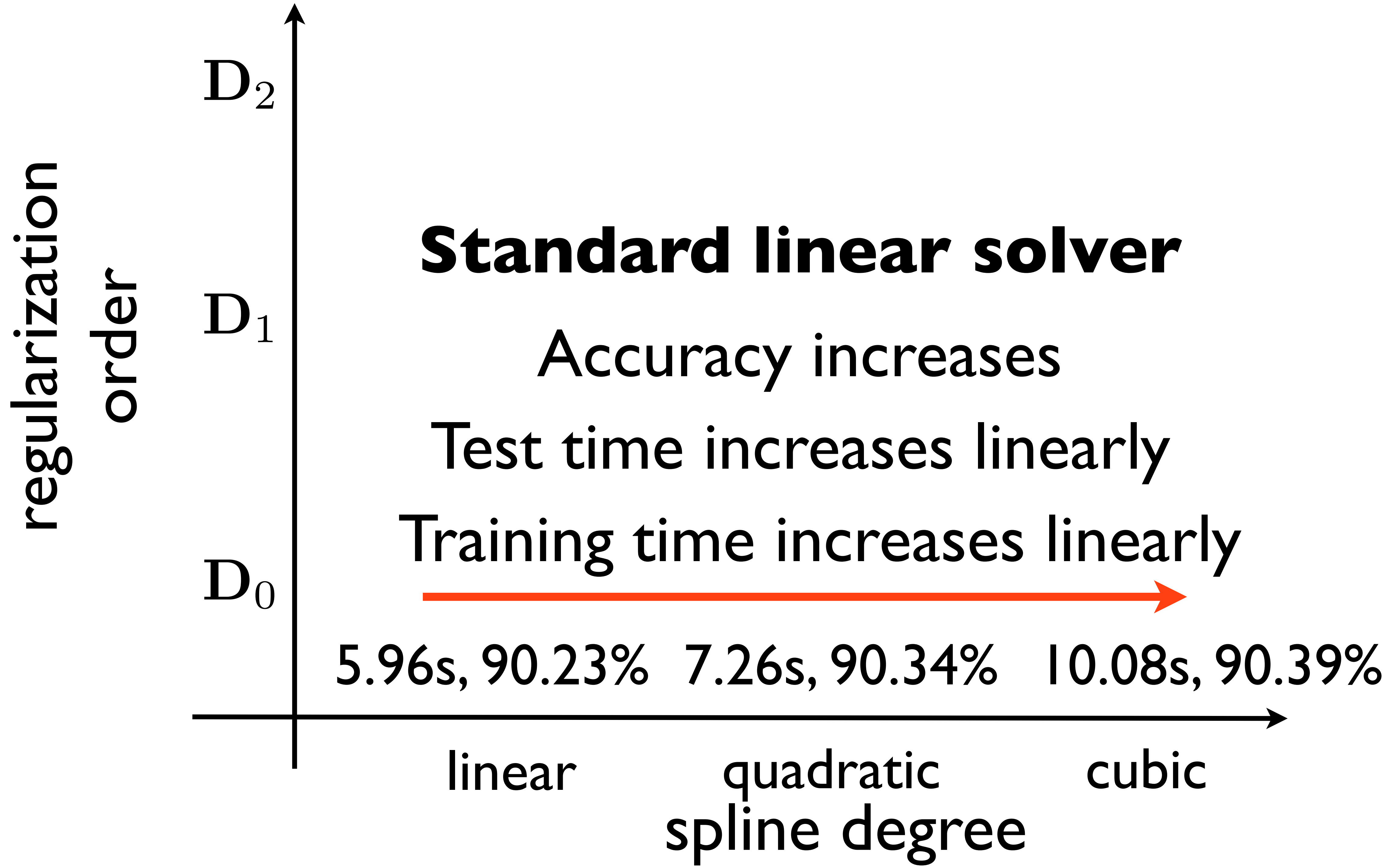
Hence these classifiers can be trained with very low memory overhead, without compromising training time

S. Maji, Linearized Smooth Additive Classifiers, ECCV WS 2012

Spline embeddings : computational tradeoffs



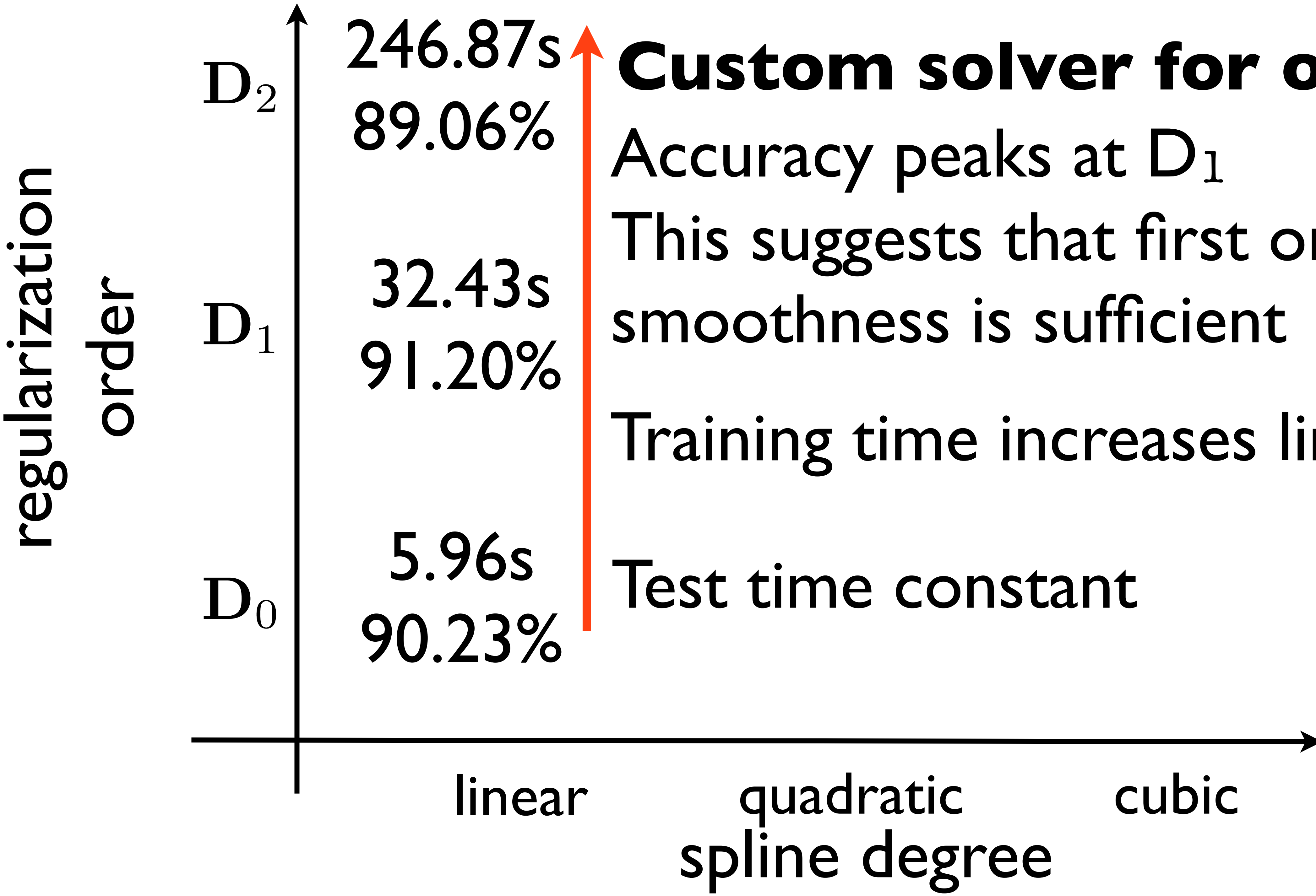
Spline embeddings : computational tradeoffs



Experiments on DC pedestrian dataset (n = 20)

IKSVM training time : 360s

Spline embeddings : computational tradeoffs



Custom solver for order > 1

Accuracy peaks at D_1

This suggests that first order smoothness is sufficient

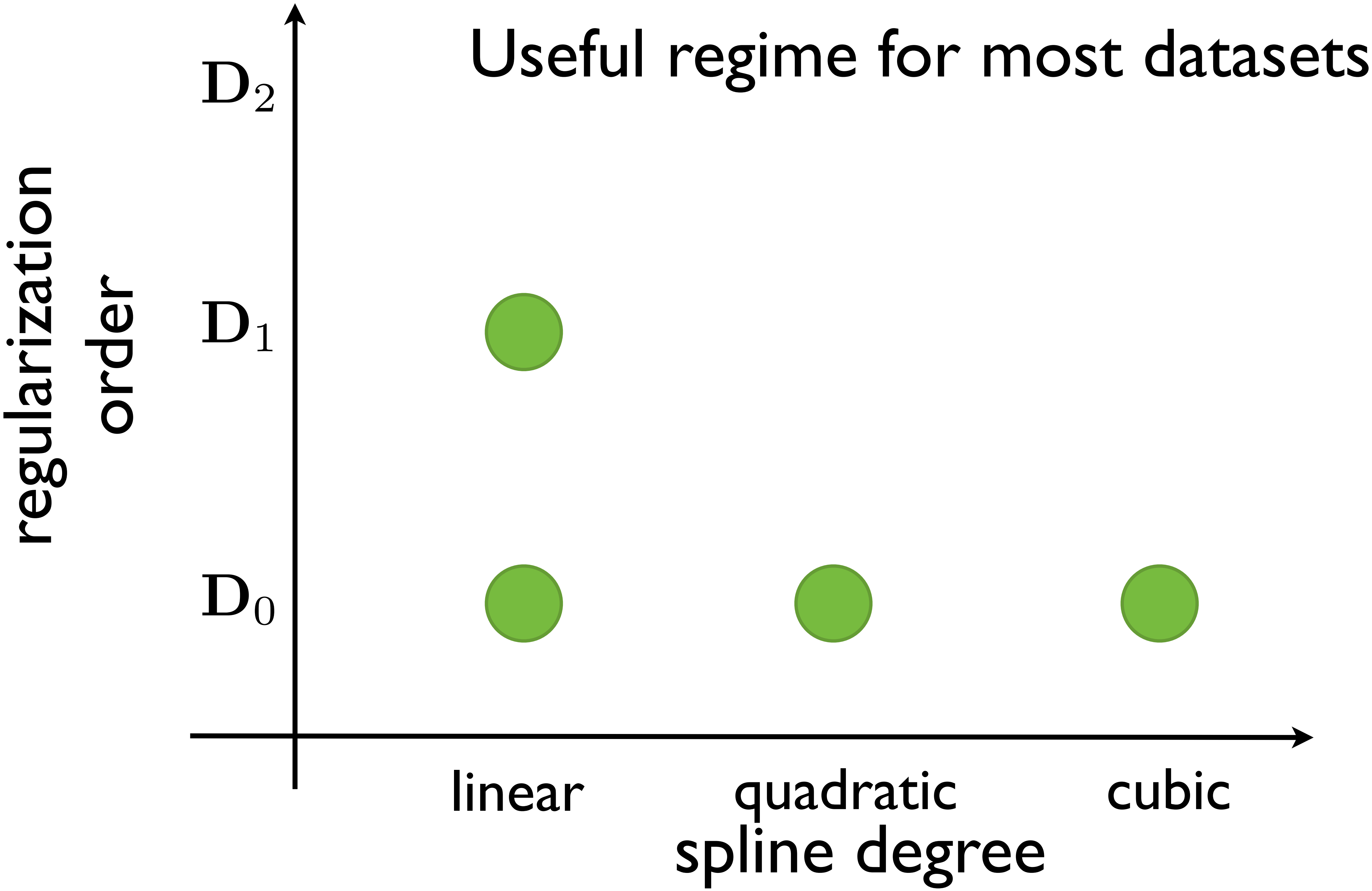
Training time increases linearly

Test time constant

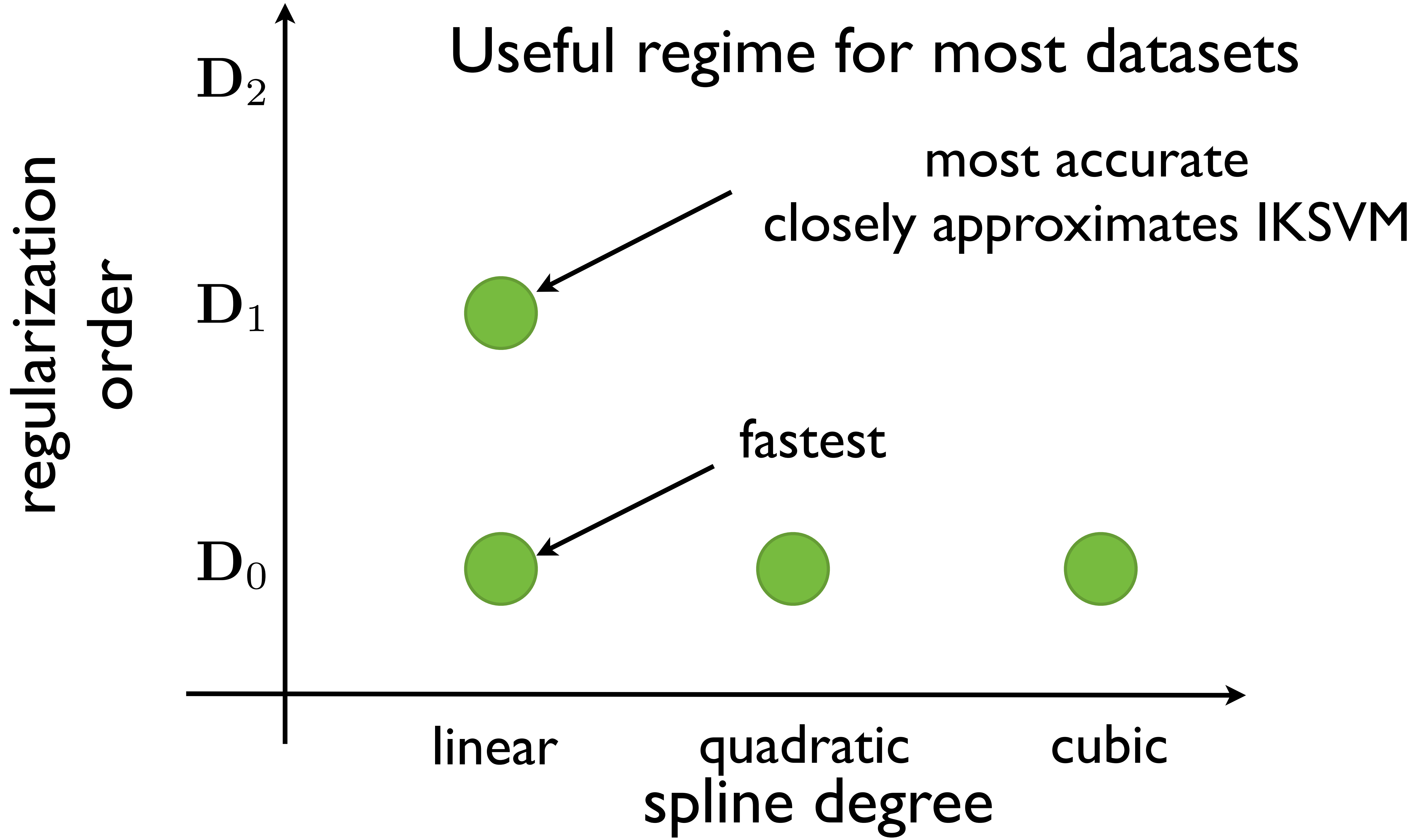
Experiments on DC pedestrian dataset (n = 20)

IKSVM training time : 360s

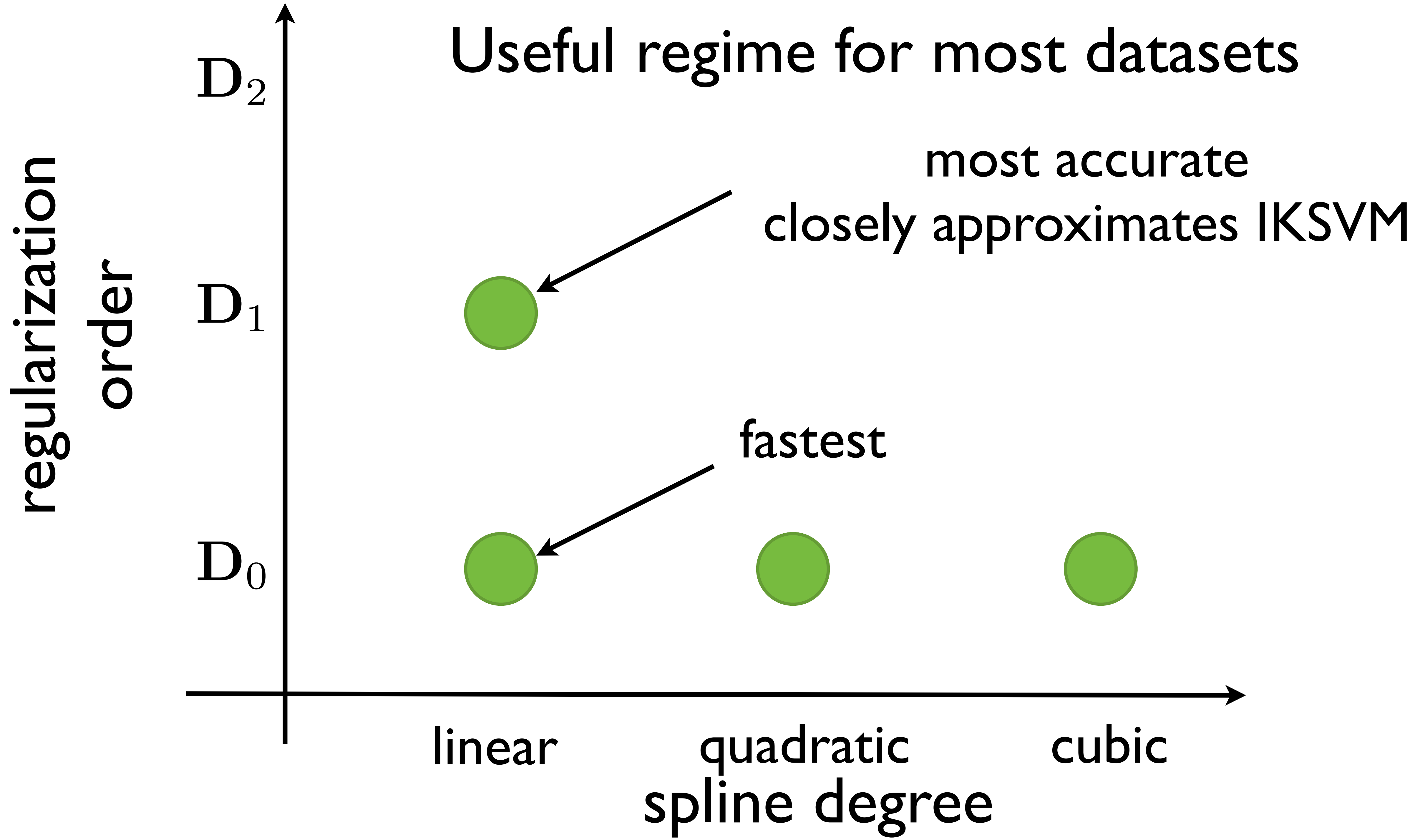
Spline embeddings : computational tradeoffs



Spline embeddings : computational tradeoffs



Spline embeddings : computational tradeoffs

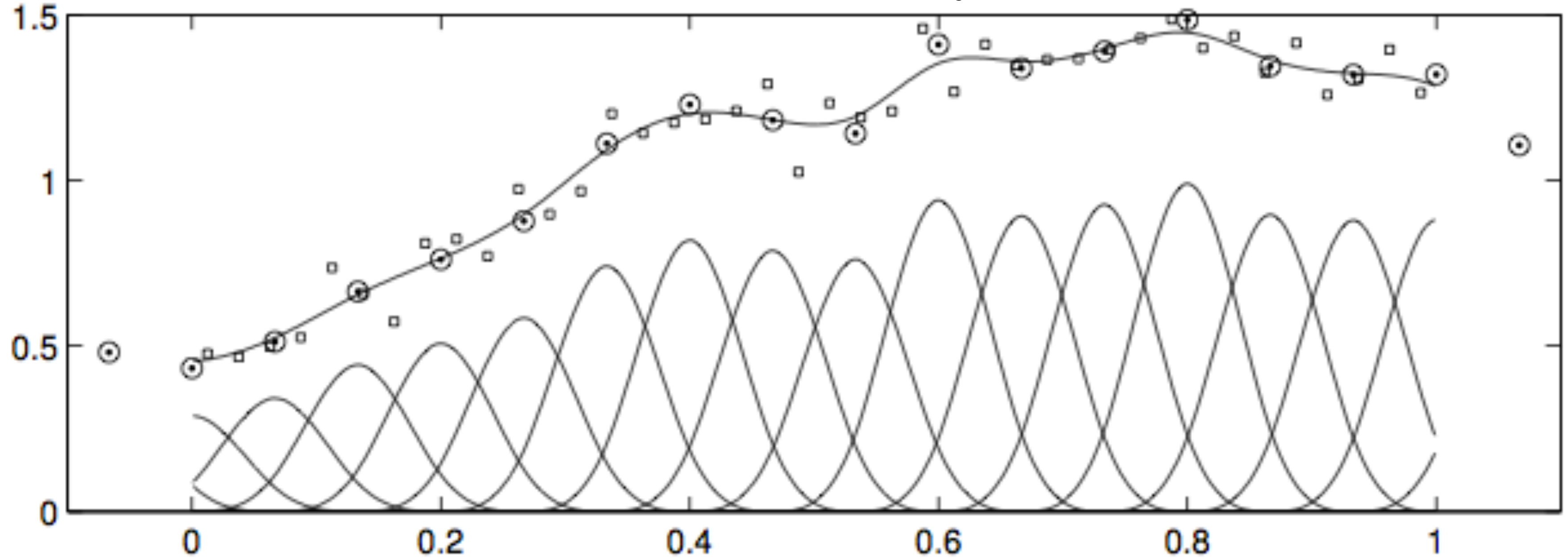


Experiments on MNIST, INRIA pedestrians, Caltech 101, DC pedestrians, etc
All these are still an order of magnitude faster than traditional SVM solver.
These have the same memory overhead as linear SVMs since the features are computed online.

General basis expansion

$$f(x_1, x_2, \ldots, x_n) = f_1(x_1) + f_2(x_2) + \ldots + f_n(x_n)$$

$$f_1(x_1) \rightarrow \sum_i w_i \phi_i$$



local splines are the basis

In general can choose any orthonormal basis

Fourier embeddings : representation

Representation

$$\psi_1(x), \psi_1(x), \ldots, \psi_n(x)$$

$$f(x) = \sum w_i \psi_i(x)$$

$$\int_a^b \psi_i(x)\psi_j(x)\gamma(x)dx = \delta_{i,j}$$

An orthonormal basis

Examples: Trigonometric functions, Wavelets, etc.

Fourier embeddings : regularization

Representation

$$\psi_1(x), \psi_1(x), \ldots, \psi_n(x)$$

$$f(x) = \sum w_i \psi_i(x)$$

$$\int_a^b \psi_i(x)\psi_j(x)\gamma(x)dx = \delta_{i,j}$$

An orthonormal basis

Regularization : penalize d'th order derivative

$$\int_a^b f^d(x)^2 \gamma(x)dx = \int_a^b w_i w_j \psi_i^d(x)\psi_j^d(x)\gamma(x)dx$$

$$R(f) = \mathbf{w}^T\mathbf{H}\mathbf{w} \quad \mathbf{H}_{i,j} = \int_a^b \psi_i^d(x)\psi_j^d(x)w(x)dx$$

Similar to the spline case (different basis)

Requires the basis to be differentiable

Fourier embeddings : optimization

Representation

$$\psi_1(x), \psi_1(x), \dots, \psi_n(x)$$

$$f(x) = \sum w_i \psi_i(x)$$

$$\int_a^b \psi_i(x)\psi_j(x)\gamma(x)dx = \delta_{i,j}$$

An orthonormal basis

Regularization : penalize d'th order derivative

$$\int_a^b f^d(x)^2 \gamma(x)dx = \int_a^b w_i w_j \psi_i^d(x)\psi_j^d(x)\gamma(x)dx$$

$$R(f) = \mathbf{w}^T \mathbf{H}\mathbf{w} \qquad \mathbf{H}_{i,j} = \int_a^b \psi_i^d(x)\psi_j^d(x)w(x)dx$$

Optimization

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{H}\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{\Psi}(x^k)\right)\right)$$

Fourier embeddings : optimization

Optimization

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{H}\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{\Psi}(x^k)\right)\right)$$

H is not diagonal - cannot directly use fast linear solvers

In general it is not structured either (unlike splines)

Fourier embeddings : optimization

Optimization

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{H}\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{\Psi}(x^k)\right)\right)$$

H is not diagonal - cannot directly use fast linear solvers

In general it is not structured either (unlike splines)

Practical solution

Pick orthogonal basis with orthogonal derivatives

Cross terms disappear, i.e., H is diagonal again

$$\int_a^b f^d(x)^2\gamma(x)dx = \int_a^b w_i w_j \psi_i^d(x)\psi_j^d(x)\gamma(x)dx \propto w_i^2$$

Fourier embeddings : optimization

Optimization

$$c(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{H}\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}^T\mathbf{\Psi}(x^k)\right)\right)$$

H is not diagonal - cannot directly use fast linear solvers

In general it is not structured either (unlike splines)

Practical solution

Pick orthogonal basis with orthogonal derivatives

Cross terms disappear, i.e., H is diagonal again

$$\int_a^b f^d(x)^2\gamma(x)dx = \int_a^b w_i w_j \psi_i^d(x)\psi_j^d(x)\gamma(x)dx \propto w_i^2$$

Examples: Trigonometric functions, One of Jacobi, Laguerre or Hermite polynomials

M. Webster, Orthogonal polynomials with orthogonal derivatives. ***Mathematische Zeitschrift, 39:634–638, 1935***

Fourier Embeddings : Two practical ones

Two families of orthogonal basis with orthogonal derivatives

Trigonometric	Hermite
$x \in [-1, 1]$, $w(x) = 1$	$x \in N(0,1)$, $w(x) = e^{-x^2/2}$
$\Psi_n^1(x) = \{\frac{\cos(n\pi x)}{n}, \frac{\sin(n\pi x)}{n}\}$	$\Psi_n^1(x) = \frac{H_n(x)}{\sqrt{nn!}}$
$\Psi_n^2(x) = \{\frac{\cos(n\pi x)}{n^2}, \frac{\sin(n\pi x)}{n^2}\}$	$\Psi_1^2(x) = \Psi_1^1(x)$, $\Psi_n^2(x) = \frac{H_n(x)}{\sqrt{n(n-1)n!}}$, $n > 1$

Embeddings that penalize the first and second order derivatives

Learning : project data onto the first few basis and use a linear solver such as LIBLINEAR

Fourier features are low dimensional and dense, as opposed to spline features which are high dimensional but sparse.

Comparison of various additive classifiers

Method	Test Accuracy	Training Time	
SVM (linear) + LIBLINEAR	81.49 (1.29)	3.8s	
SVM (min) + LIBSVM	89.05 (1.42)	363.1s	
		online	batch
B-Spline ($\mathbf{D}_0$, Linear, $n = 05$)	88.51 (1.35)	**5.9**s	-
B-Spline ($\mathbf{D}_0$, Cubic, $n = 05$)	89.00 (1.44)	10.8s	-
B-Spline ($\mathbf{D}_1$, Linear, $n = 10$)	**89.56** (1.35)	17.2s	-
B-Spline ($\mathbf{D}_1$, Cubic, $n = 10$)	89.25 (1.39)	19.2s	-
Fourier ($d = 1$, $n = 4$)	88.44 (1.43)	159.9s	12.7s (4× memory)
Hermite ($d = 1$, $n = 4$)	87.67 (1.26)	35.5s	12.6s (4× memory)

DC pedestrian dataset

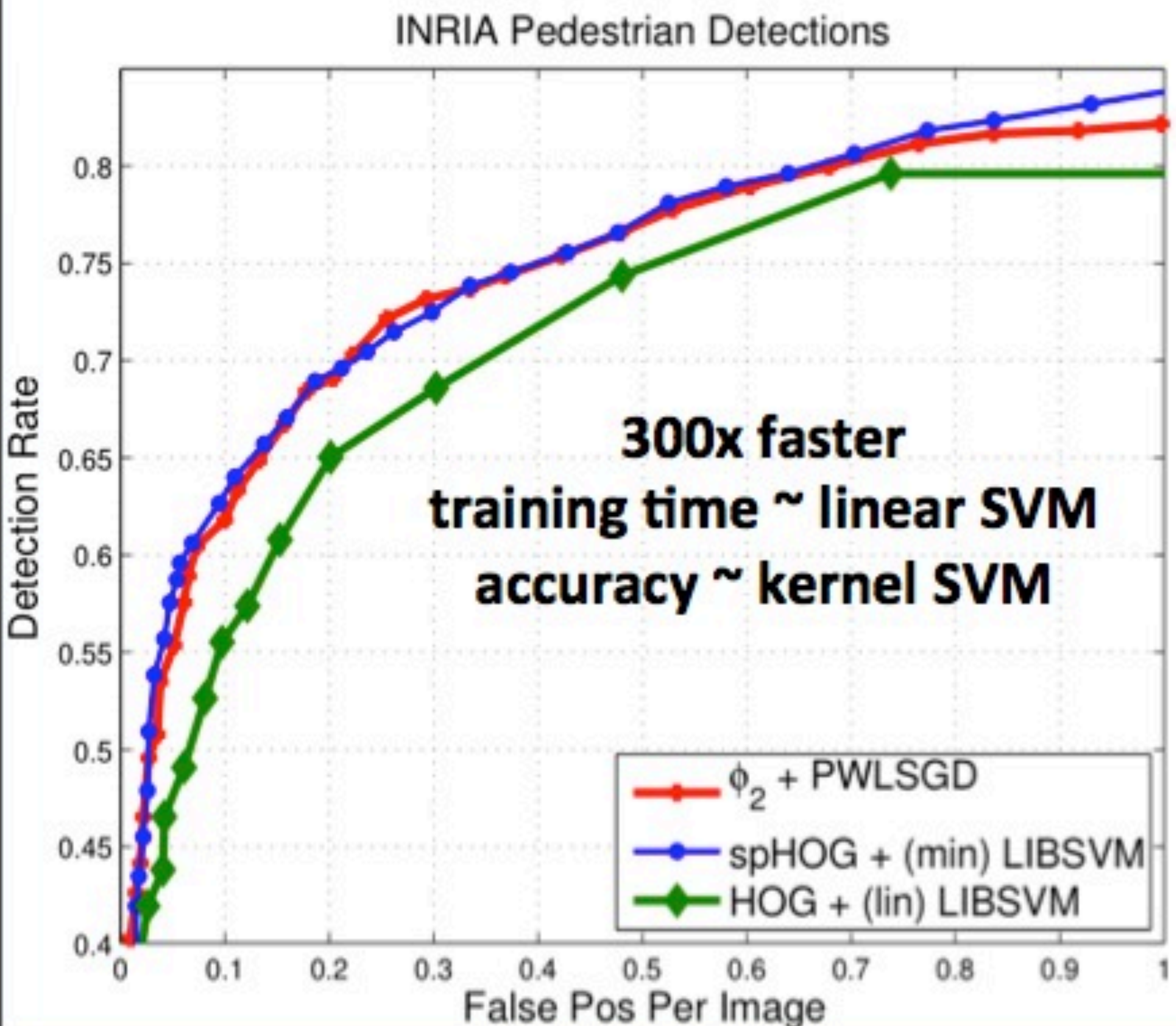
Comparison of various additive classifiers

Method	Test Error	Training Time
SVM (linear) + `LIBLINEAR`	1.44%	6.2s
SVM (min) + `LIBSVM`	0.79%	$\sim$ 2.5 hours
B-Spline ($\mathbf{D}_0$, Linear, $n = 20$)	0.88%	31.6s
B-Spline ($\mathbf{D}_0$, Cubic, $n = 20$)	0.86%	51.6s
B-Spline ($\mathbf{D}_1$, Linear, $n = 40$)	**0.81%**	157.7s
B-Spline ($\mathbf{D}_1$, Cubic, $n = 40$)	0.82%	244.9s
Hermite ($d = 1, n = 4$)	1.06%	358.6s

MNIST dataset

Comparison of various additive classifiers

Dataset	Linear		Piecewise Linear		IK SVM	
	Time	Accuracy	Time	Accuracy	Time	Accuracy
INRIA pedestrians	20s	see curve	**76s**	**see curve**	~ 3 hr	see curve
Caltech101, 15 examples	18.6s	41.2%	**238s**	**49.9%**	844s	50.1%
Caltech101, 30 examples	40.5s	46.2%	**291s**	**55.4%**	2686s	56.5%



Software

- Code to train large scale additive classifiers. Provides functions:
  - `train` : input (y,**x**) and outputs additive classifiers
    - choice of various encodings, regularizations.
    - encodings are computed online
    - implements efficient weight updates for splines features
  - `classify` : takes a learned classifier and features, outputs decision values
  - `encode` : returns encoded features which can be directly used with any linear solver
- Download at:
  - http://ttic.uchicago.edu/~smaji/libspline-release1.0.tar.gz

Conclusions

- We discussed methods to directly learn additive classifiers based on a regularized loss minimization
- We proposed two kinds of basis for which the learning problem can be efficiently solved, **Spline** and **Fourier** embeddings
  - Spline embeddings are sparse, easy to compute, and can be used to learn classifiers with *almost no memory overhead,* compared to learning linear classifiers.
  - Fourier embeddings (Trigonometric and Hermite) are low dimensional, but are relatively expensive to compute, hence are useful in setting where features can be stored in memory
- More experimental details and code can be found on the author's website (ttic.uchicago.edu/~smaji)