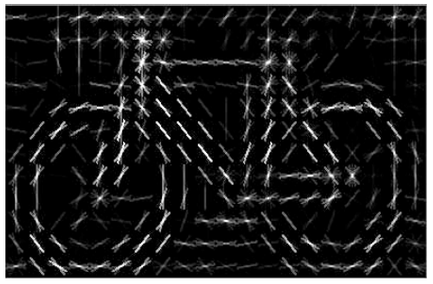
Feature maps for large-scale non-linear learning

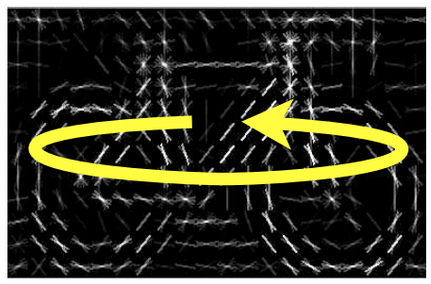
Andrea Vedaldi
University of Oxford

modelling of structure

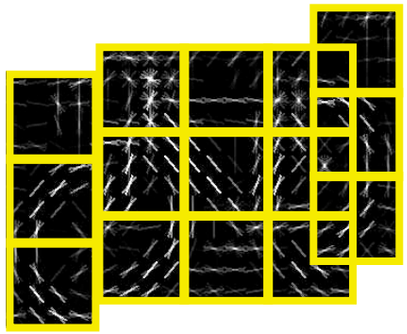
location



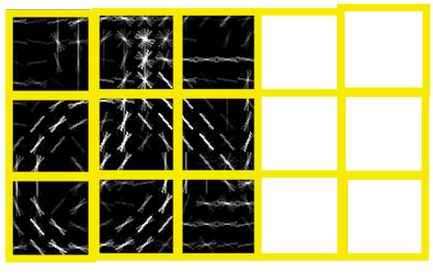
direction



deformation



truncation



[Vedaldi Zisserman 09]

massive datasets

image.net by Getty Images

Challenge

1000 classes

1.5M images

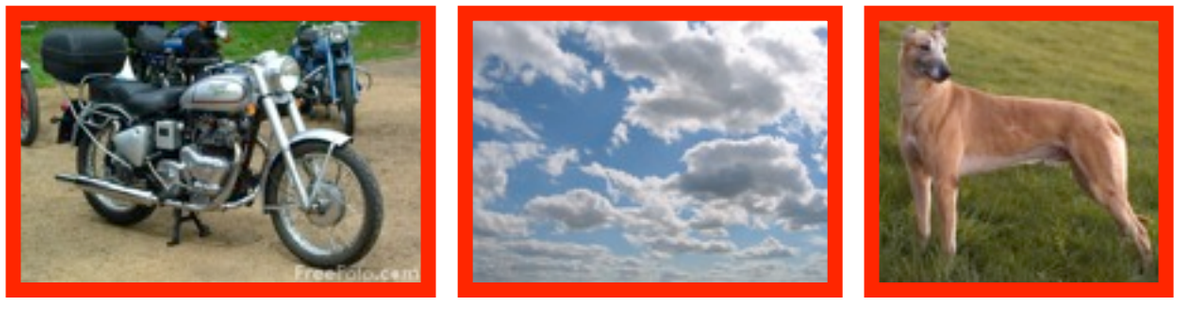
>50k-dim. descriptors

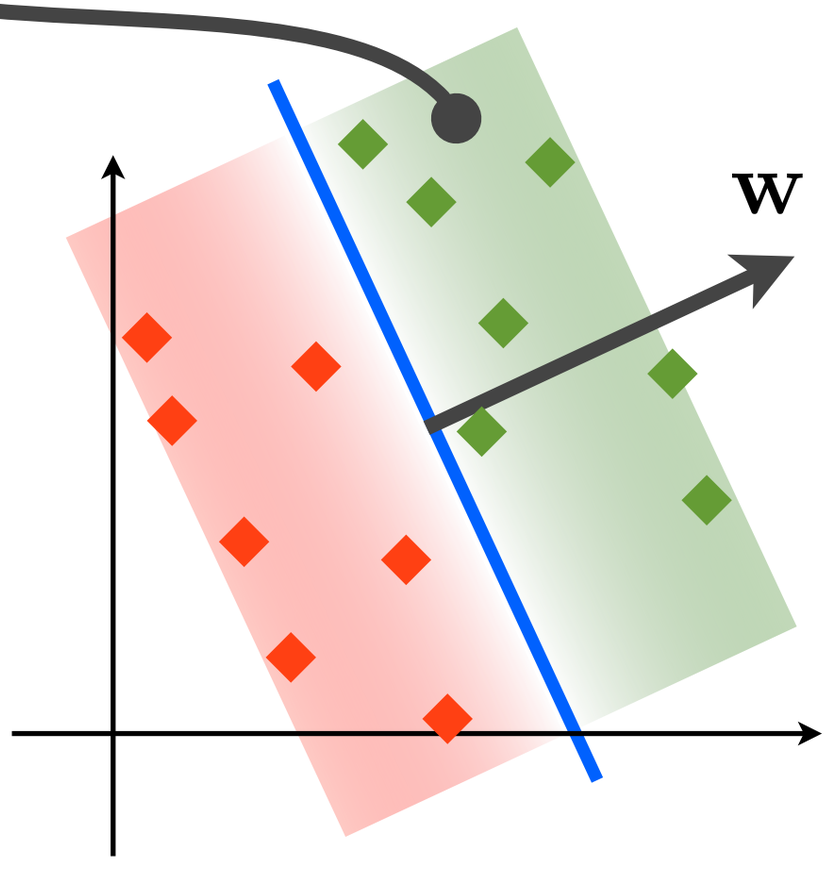
Very efficient learning

Predictive modelling

bicycle?

$\mathbf{x}$





binary classifier

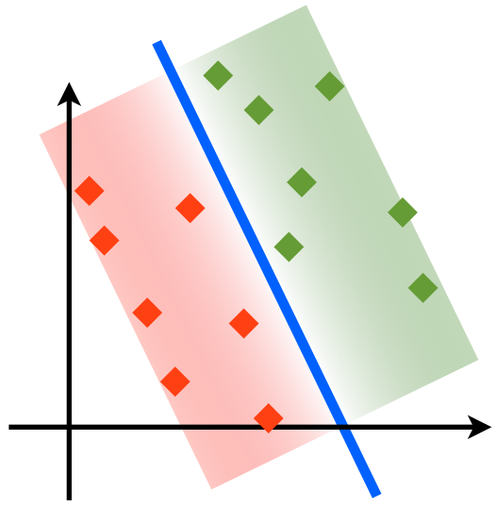
$$F(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$$

Linear vs non-linear kernels

Linear SVM

✔ fast

✘ restrictive

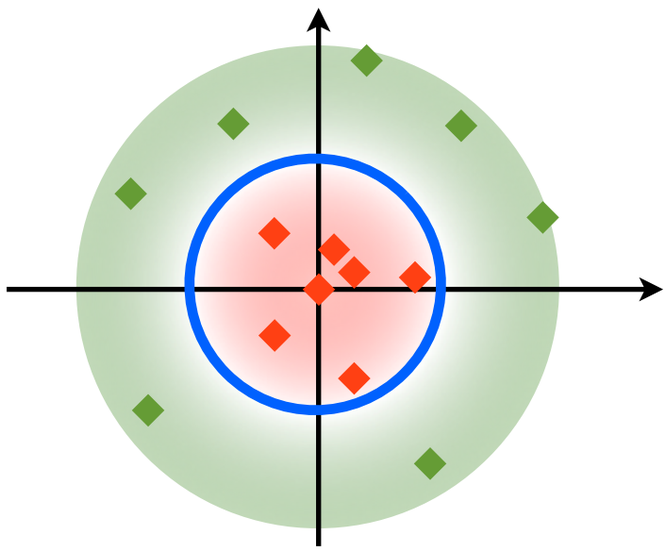


$$F(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$$

Non-linear SVM

✘ much slower

✔ powerful



$$F(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i)$$

Non-linear kernel classifiers are slow

$$F(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i)$$

thousand bicycles

many more non-bicycle

Explicit linear representations of kernels

$$F(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i)$$

feature map

$$K(\mathbf{x}, \mathbf{x}_i) = \langle \Psi(\mathbf{x}), \Psi(\mathbf{x}_i) \rangle$$

$$F(\mathbf{x}) = \langle \mathbf{w}, \Psi(\mathbf{x}) \rangle \qquad \mathbf{w} = \sum_{i=1}^{N} \alpha_i \Psi(\mathbf{x}_i)$$

Explicit linear representations of kernels

$$F(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i)$$

approximated feature map

$$K(\mathbf{x}, \mathbf{x}_i) \approx \langle \widehat{\Psi}(\mathbf{x}), \widehat{\Psi}(\mathbf{x}_i) \rangle$$

$$F(\mathbf{x}) = \langle \mathbf{w}, \widehat{\Psi}(\mathbf{x}) \rangle \qquad \mathbf{w} = \sum_{i=1}^{N} \alpha_i \widehat{\Psi}(\mathbf{x}_i)$$

Preview: *homogeneous kernel map*

X² kernel

Excellent for bag-of-words, ...

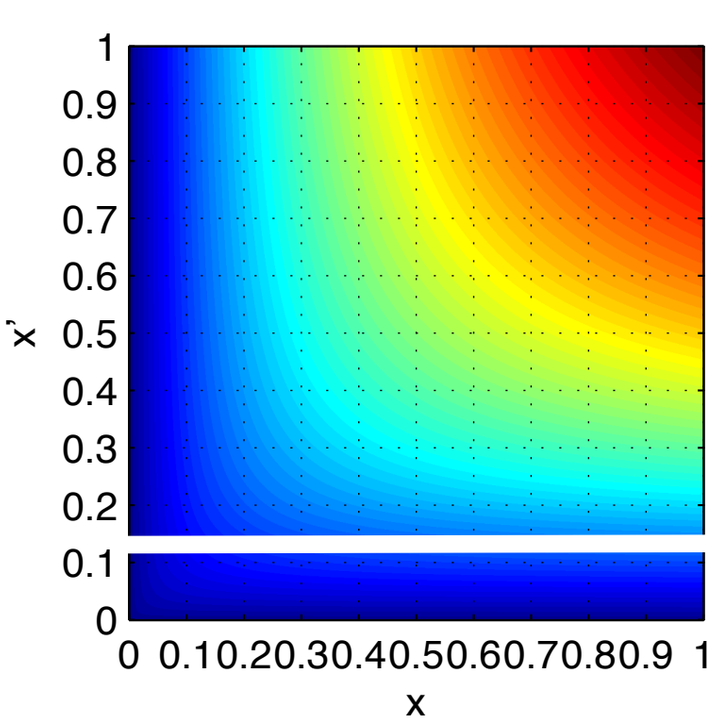
$$k(x, x') = \frac{2xx'}{x + x'}$$

Homogeneous kernel map

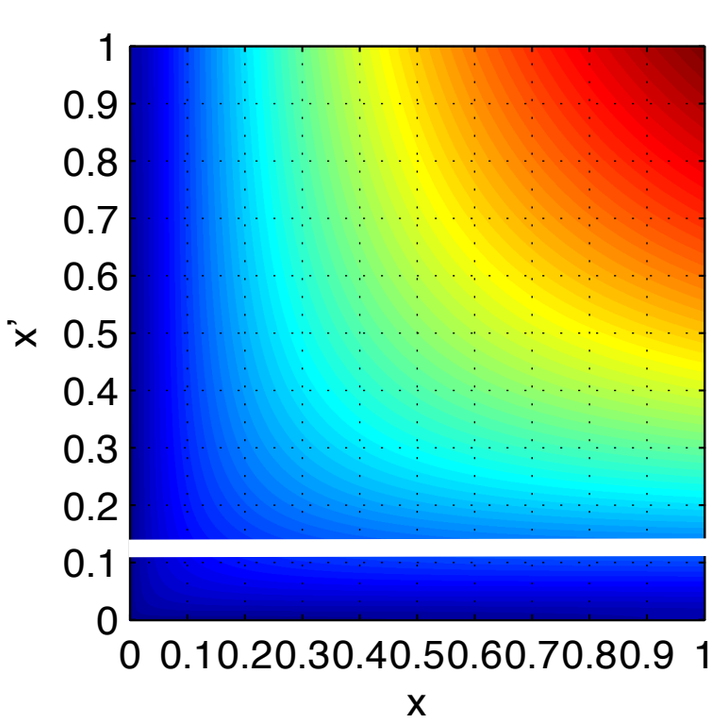
Closed form, simple, small

$$\Phi(x) = \sqrt{x} \begin{bmatrix} 0.8 \\ 0.6\cos(0.6\ \log x) \\ 0.6\sin(0.6\ \log x) \end{bmatrix}$$

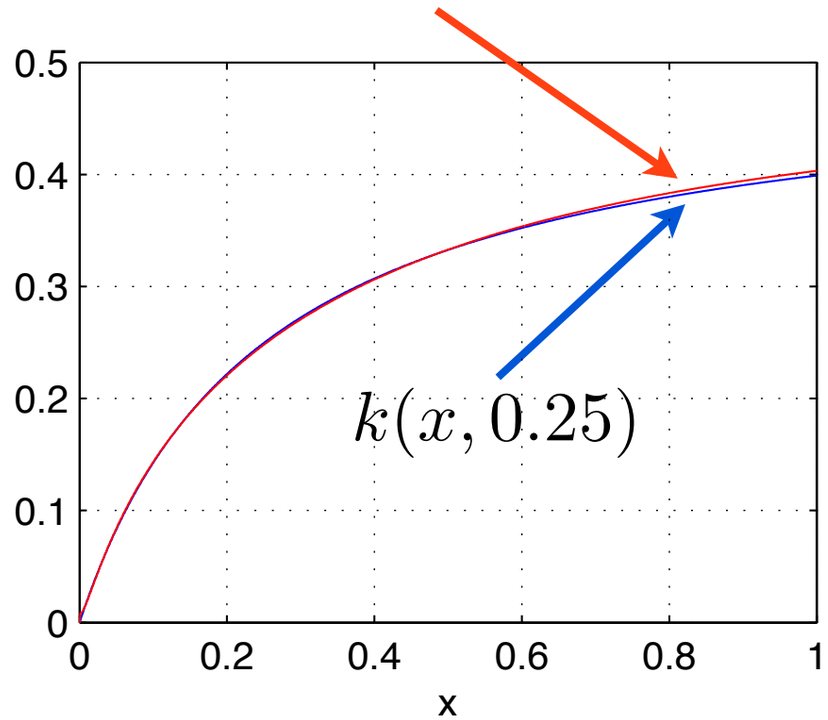
$k(x, x')$



$\langle \Phi(x), \Phi(x') \rangle$



$\langle \Phi(x), \Phi(0.25) \rangle$



[Vedaldi Zisserman 10,11]

Preview: *homogeneous kernel map*

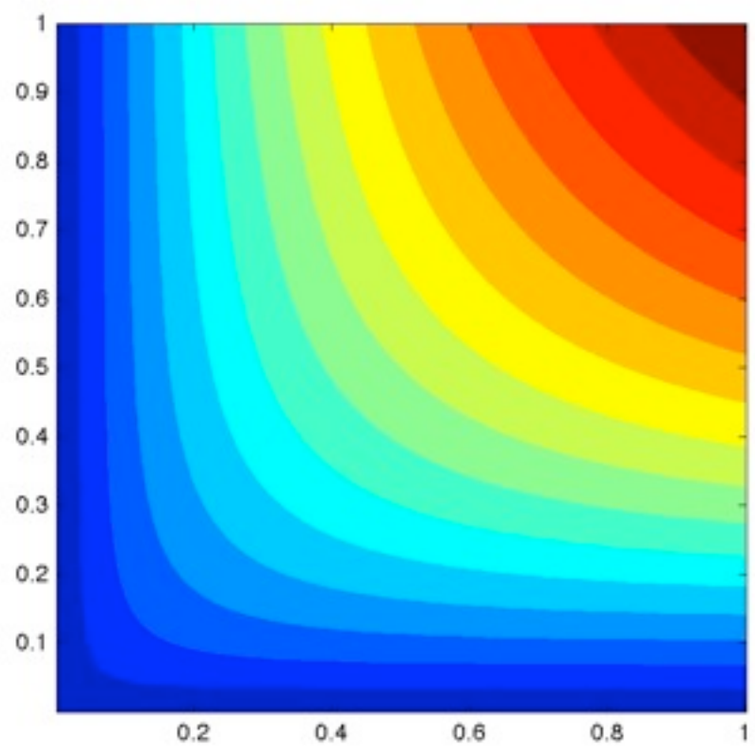
MATLAB code for Chi2 kernel

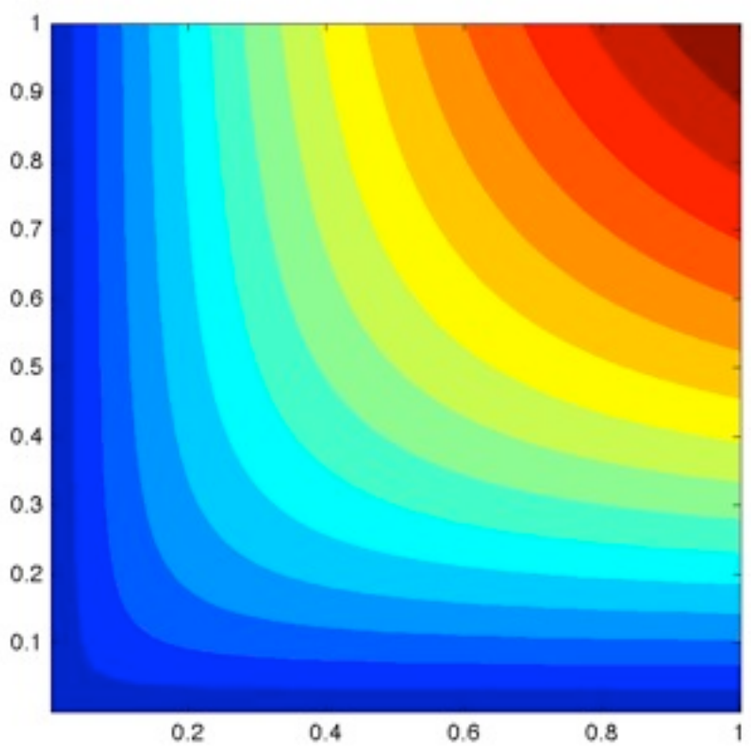
```
x = .01:.01:1 ;
for i = 1:100
  for j = 1:100
    K(i,j) = ...
      2*x(i)*x(j)/(x(i)+x(j));
  end
end
```

With the hom. kernel feature map

```
x = .01:.01:1 ;
psi = vl_homkermap(x,1) ;
K = psi'*psi ;
```

VLFeat Toolbox
http://www.vlfeat.org





Preview: *homogeneous kernel map*

Caltech-101 category recognition

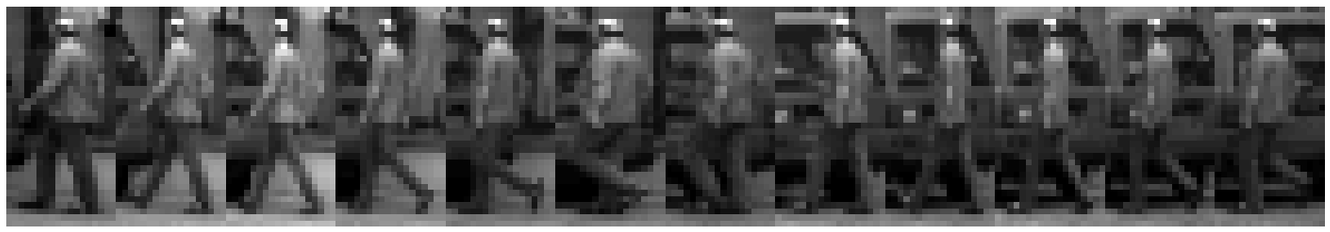
#1,500

training time

1 h → 5 m

4× speedup

DaimlerChrylser pedestrian recognition



#20,000

1/2 h → 14 s

100× speedup

Trecvid 2009 video indexing

#70,000

> 1 h → 22.6 s

160× speedup

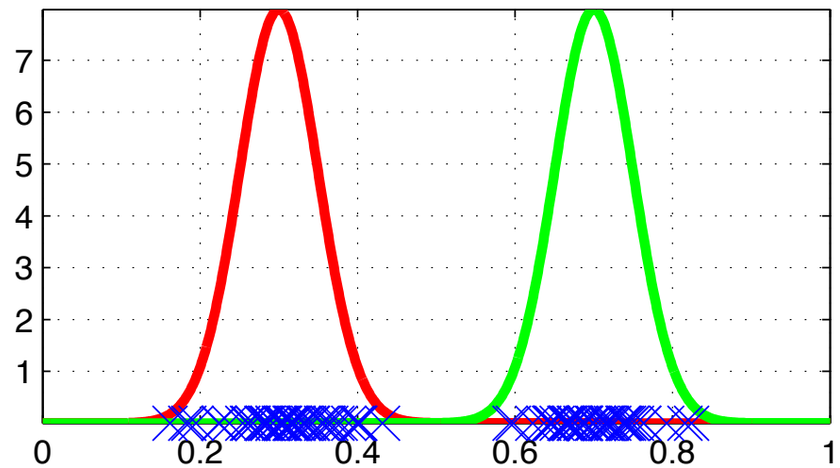
Information captured by a kernel

Kernel function

Example: Gaussian kernel

$$k(x, x') = \exp\left(-\frac{(x - x')^2}{2\sigma^2}\right)$$

Data distribution

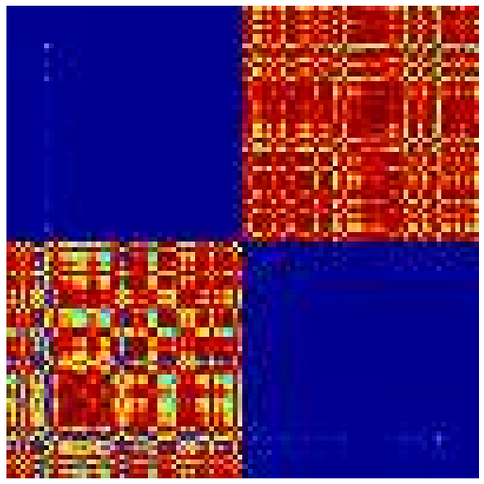


samples $X = (x_1, x_2, \ldots, x_n)$

Empirical kernel matrix

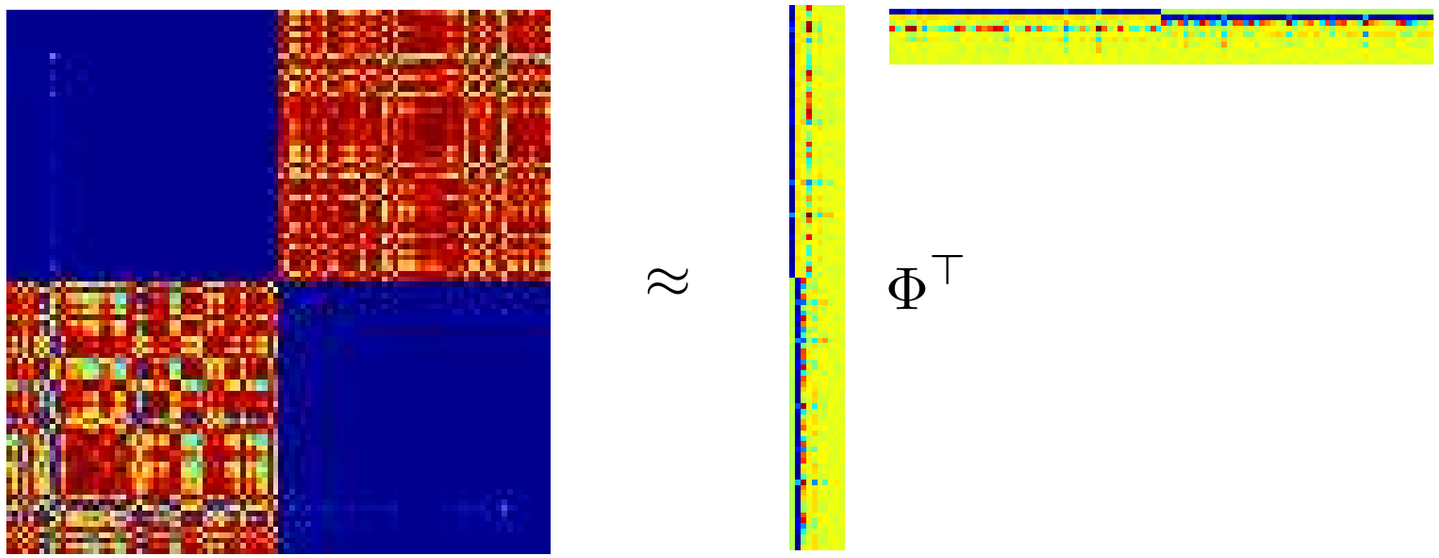
(data sorted by class for visualization clarity)

$$K = K_{XX} = \begin{bmatrix} \ddots & & \\ & k(x_i, x_j) & \\ & & \ddots \end{bmatrix} =$$



Low-rank kernel approximations

$$K \approx \Phi^\top \Phi, \qquad \Phi \in \mathbb{R}^{D \times n}, \quad D \ll n$$



Optimal decomposition: eigen-vectors (spectrum)

$$KU = U \begin{bmatrix} \kappa_1^2 & & \\ & \kappa_2^2 & \\ & & \ddots \end{bmatrix} \qquad U = \begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_n \\ | & | & & | \end{bmatrix} \qquad \Phi = \begin{bmatrix} \kappa_1 \mathbf{u}_1^\top \\ \kappa_2 \mathbf{u}_2^\top \\ \vdots \\ \kappa_D \mathbf{u}_D^\top \end{bmatrix}$$

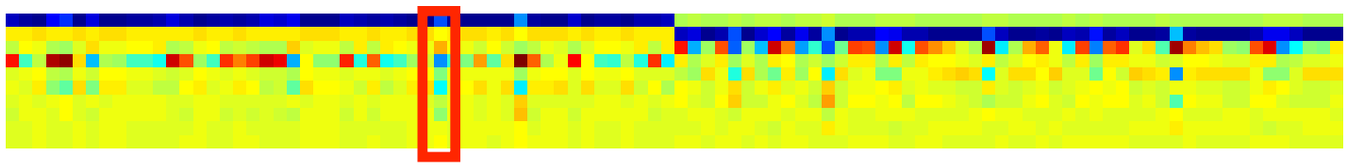
sorted eigenvalues

orthonormal eigenvectors

[Williams Seeger 01]

Eigen-features

$$K \approx \Phi^\top \Phi, \qquad \Phi \in \mathbb{R}^{D \times n}, \quad D \ll n$$

$\Phi =$ 

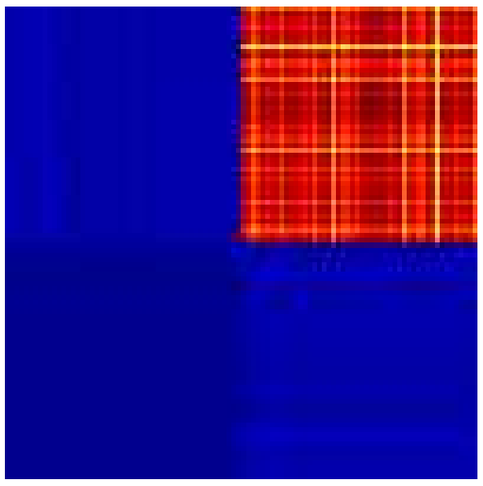
$\Phi(x_i) \in \mathbb{R}^d$

$k(x_i, x_j) \approx \langle \Phi(x_i), \Phi(x_j) \rangle$

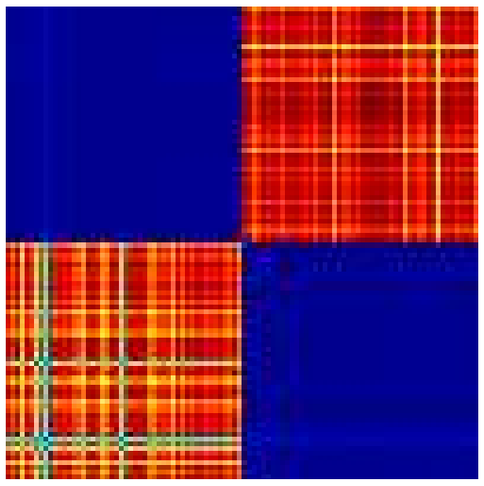
Each column is a **code** for a point

The code is an **approximated feature**

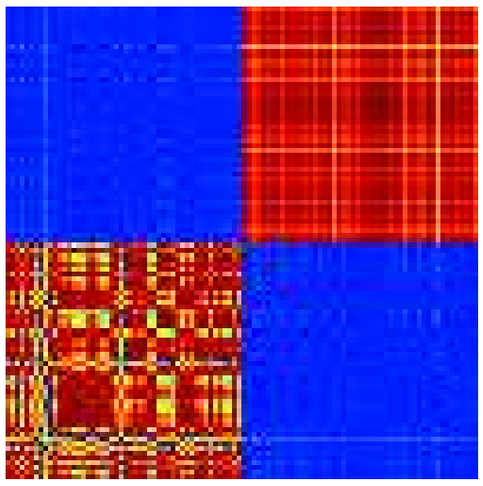
D = 1



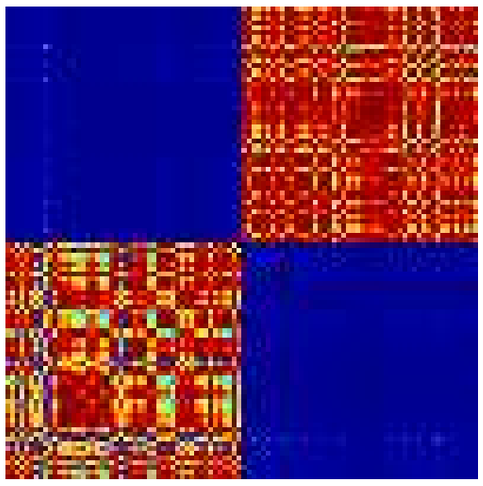
D = 2



D = 3

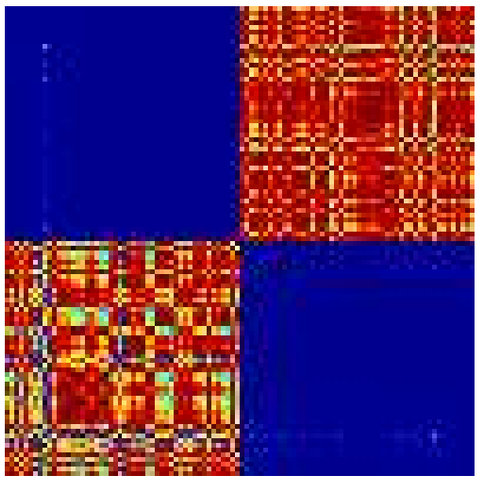


D = 5



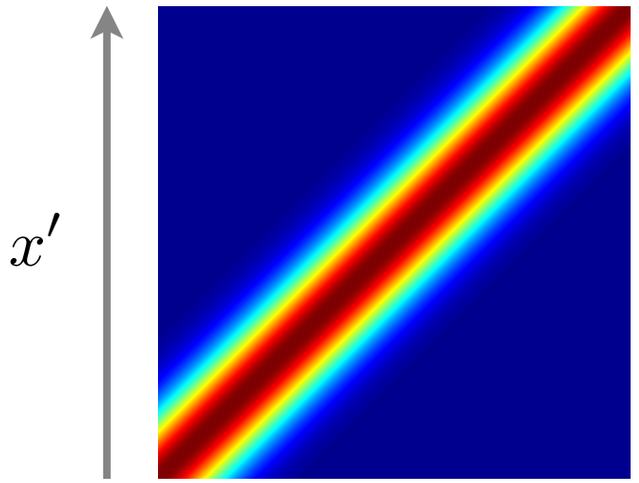
Extending eigen-features to new samples

$k(x_i, x_j)$

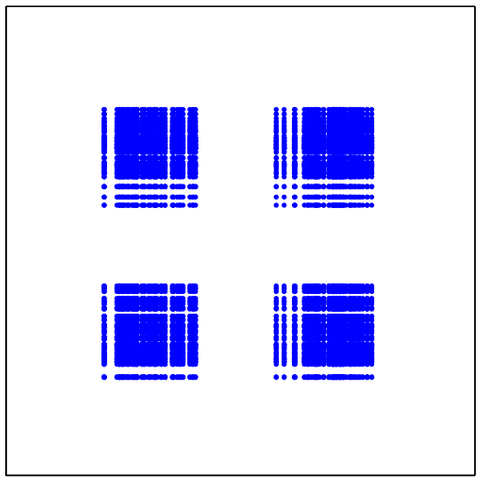


data samples

$k(x, x')$



eigen-features support



$$KU = U\,\mathrm{diag}(\boldsymbol{\kappa})^2$$

$$\Phi = \begin{bmatrix} \kappa_1 \mathbf{u}_1^\top \\ \kappa_2 \mathbf{u}_2^\top \\ \vdots \\ \kappa_D \mathbf{u}_D^\top \end{bmatrix} = \mathrm{diag}(\boldsymbol{\kappa}_D) U_D^\top = \mathrm{diag}(\boldsymbol{\kappa}_D)^{-1} U_D^\top K$$

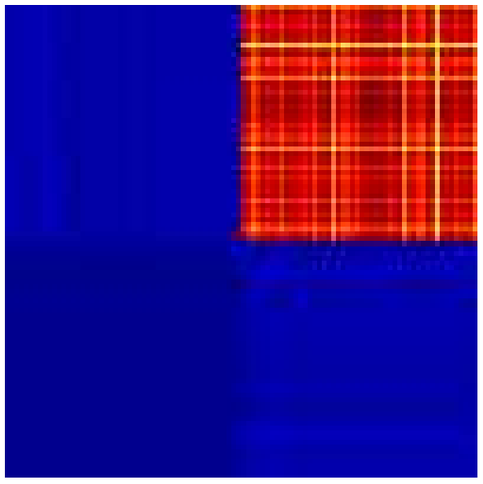
one column is one eigen-feature

extension to any point *x*

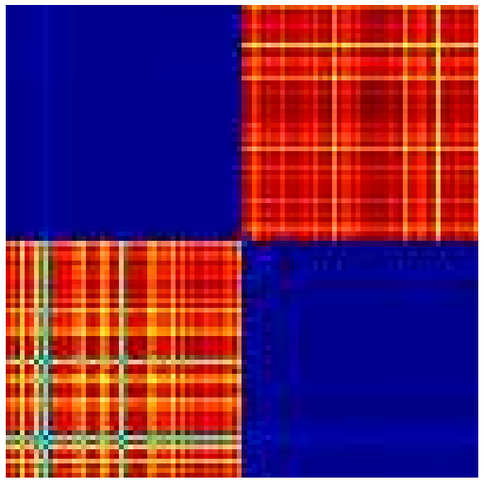
$$\Phi(x) = \mathrm{diag}(\boldsymbol{\kappa}_D)^{-1} U_D^\top \begin{bmatrix} k(x_1, x) \\ k(x_2, x) \\ \vdots \\ k(x_n, x) \end{bmatrix}$$

in-sample reconstructions

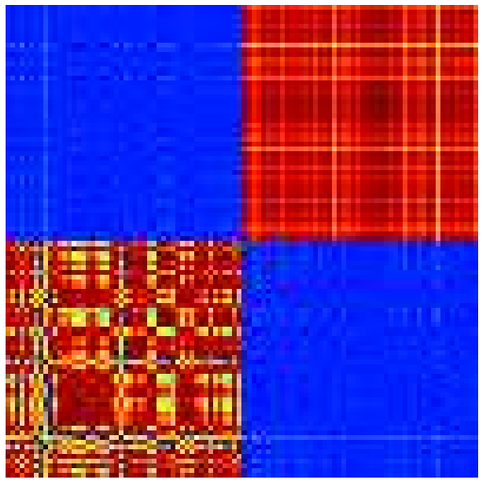
D = 1



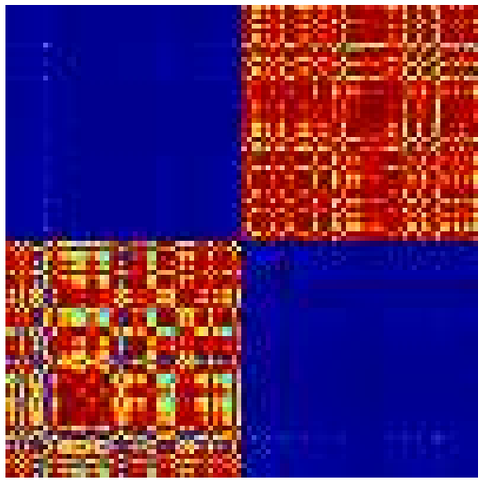
D = 2



D = 3

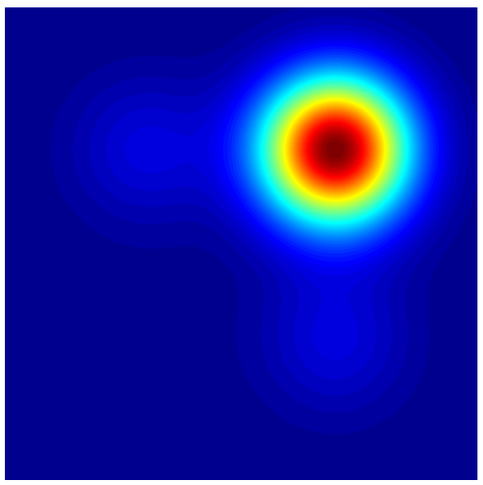


D = 5

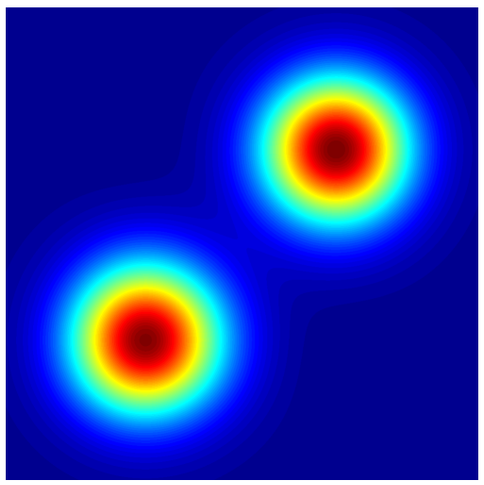


out-of-sample reconstructions

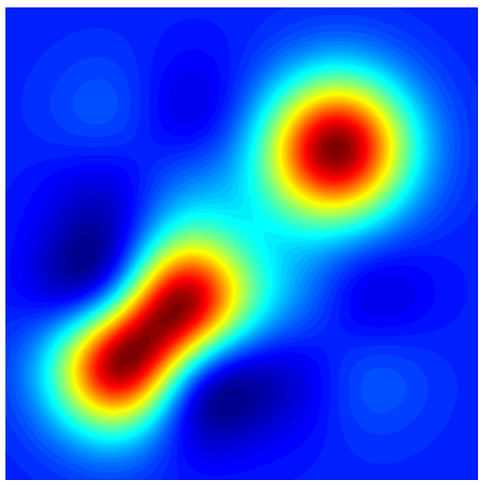
D = 1



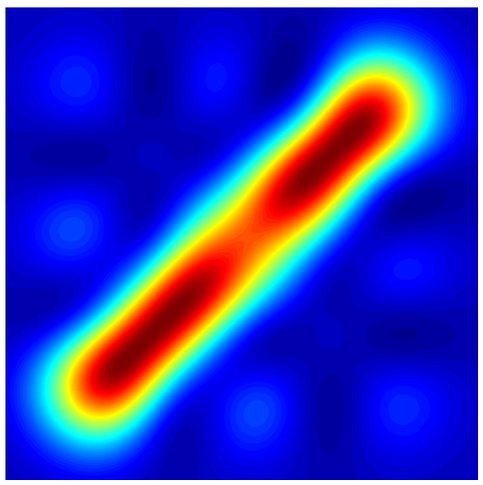
D = 2



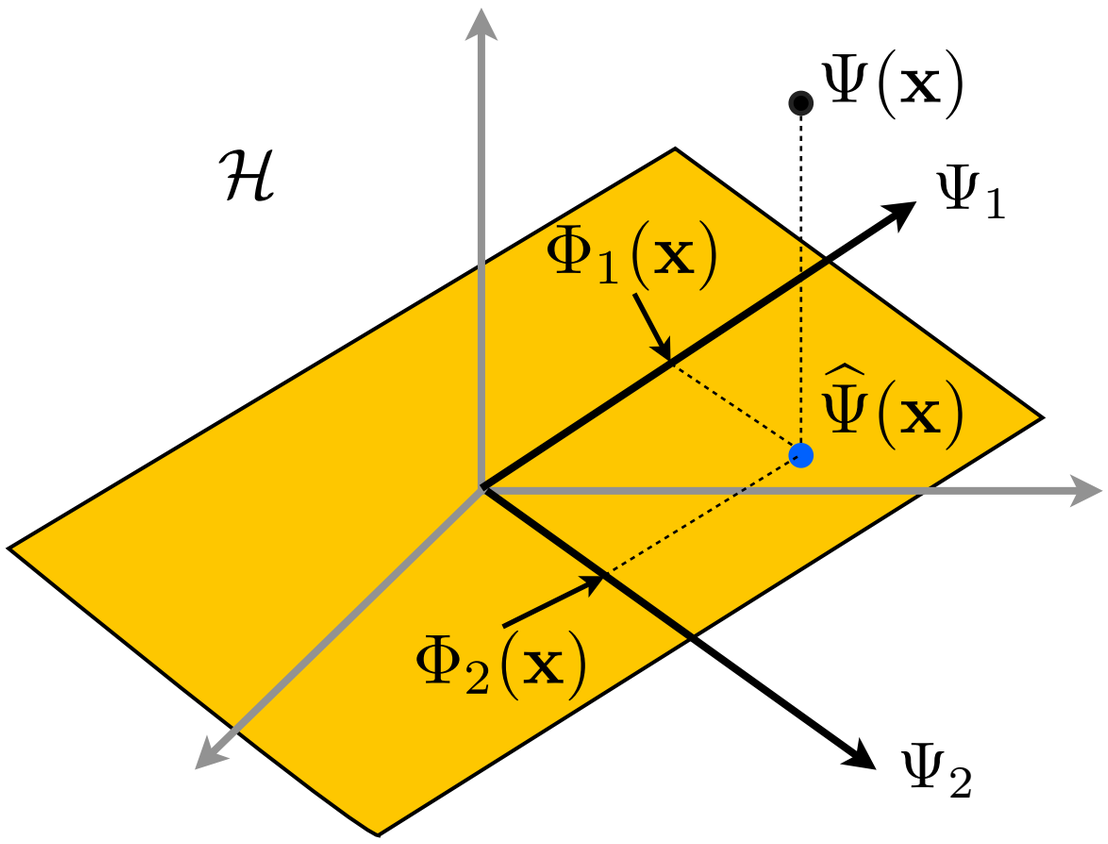
D = 3



D = 5



Approximated features as projections



- **Exact features**
  - always exist
  - abstract
  - infinite dimensional
- **Approximated features**
  - project on a *D*-dimensional subspace
  - find an orthonormal basis
  - express projections relative to basis to obtain feature in R^D

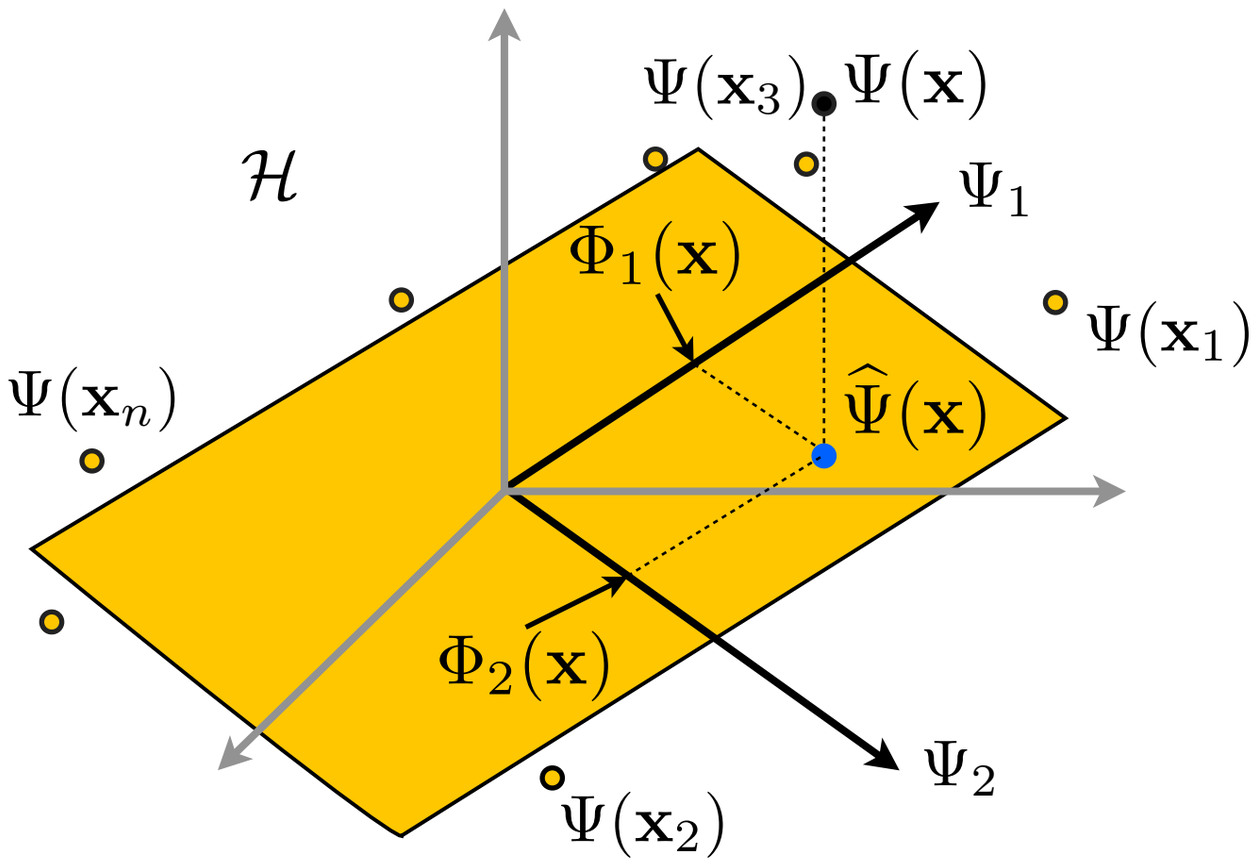
$$K(\mathbf{x}, \mathbf{x}') = \langle \Psi(\mathbf{x}), \Psi(\mathbf{x}') \rangle$$

$\Psi_1, \Psi_2, \ldots, \Psi_D$ orthonormal basis

$$\Phi(\mathbf{x}) = \begin{bmatrix} \langle \Psi_1, \Psi(\mathbf{x}) \rangle \\ \langle \Psi_2, \Psi(\mathbf{x}) \rangle \\ \vdots \\ \langle \Psi_D, \Psi(\mathbf{x}) \rangle \end{bmatrix}$$

coordinate functions

Empirical Nyström's approximation



- **The subspace should "fit" the data**
  - kernel function (= exact feature)

$$K(\mathbf{x}, \mathbf{x}') = \langle \Psi(\mathbf{x}), \Psi(\mathbf{x}') \rangle_{\mathcal{H}}$$

  - data empirical distribution

$$\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$$

- **Best fit: principal components**
  - find orthonormal eigenvectors

$$KU = U \operatorname{diag}(\boldsymbol{\kappa})^2$$

Projection

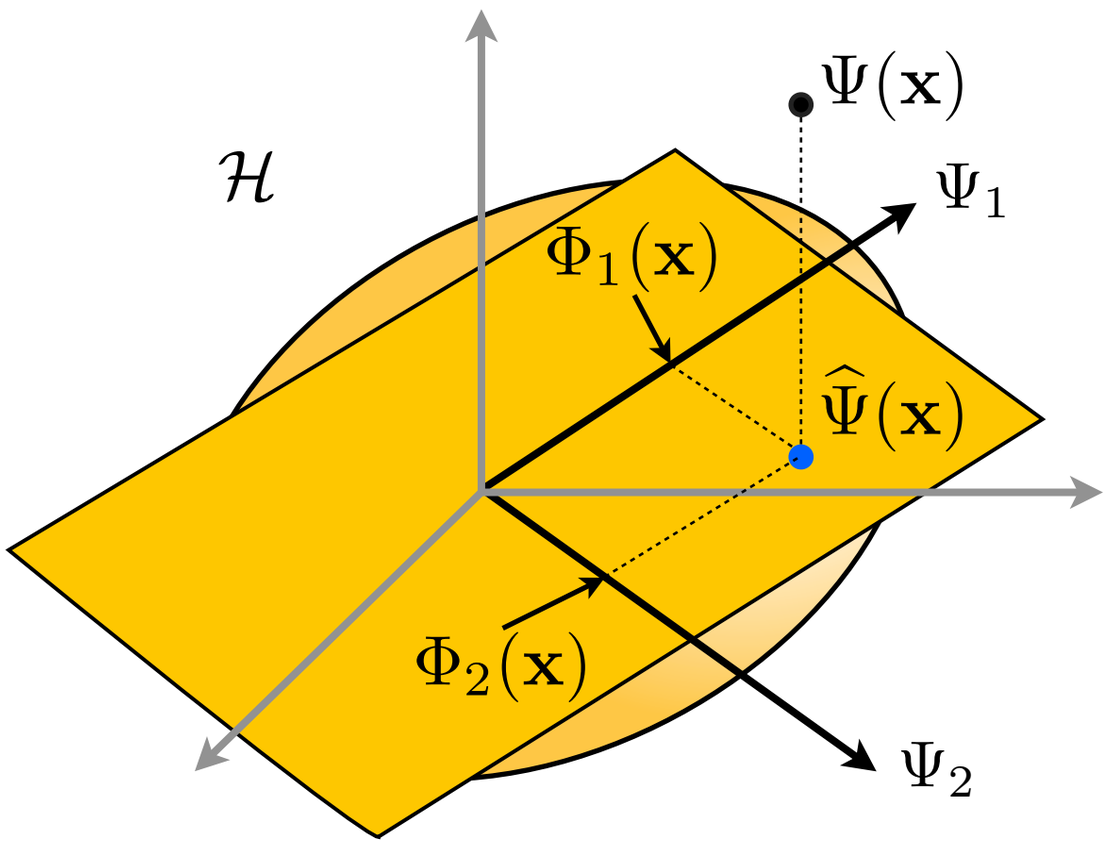
$$\Phi_i(\mathbf{x}) = \langle \Psi_i, \Psi(\mathbf{x}) \rangle = \kappa_i^{-1} \mathbf{u}_i^\top \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}) \\ k(\mathbf{x}_2, \mathbf{x}) \\ \vdots \\ k(\mathbf{x}_n, \mathbf{x}) \end{bmatrix}$$

  - obtain orthonormal basis

$$\Psi_i = \sum_{k=1}^{n} \Psi(\mathbf{x}_k) u_{ki} \kappa_i^{-1}$$

[Williams Seeger 01]

Formal Nyström's approximation



- **The plane should "fit" the data**
  - kernel function (= exact feature)

$$K(\mathbf{x}, \mathbf{x}') = \langle \Psi(\mathbf{x}), \Psi(\mathbf{x}') \rangle_{\mathcal{H}}$$

  - data distribution

$$p(\mathbf{x}), \quad \mathbf{x} \in \mathcal{X}$$

- **Best fit: principal components**
  - find orthonormal eigenfunctions

$$\int_{\mathcal{X}} K(\mathbf{x}, \mathbf{z}) u_i(\mathbf{z}) p(\mathbf{z}) \, d\mathbf{z} = u_i(\mathbf{x}) \kappa_i^2$$

  - obtain orthonormal basis

$$\Psi_i = \int_{\mathcal{X}} \Psi(\mathbf{z}) u_i(\mathbf{z}) p(\mathbf{z}) \, d\mathbf{z}$$

Projection

$$\Phi_i(\mathbf{x}) = u_i(\mathbf{x}) \kappa_i$$

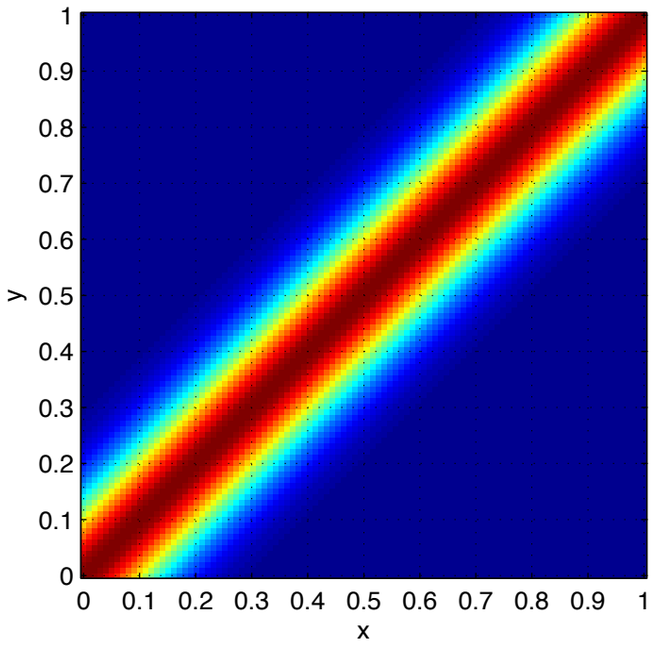
$$\left(= \kappa_i^{-1} \int_{\mathcal{X}} u_i(\mathbf{z}) K(\mathbf{z}, \mathbf{x}) p(\mathbf{z}) \, d\mathbf{z} \right)$$

[Williams Seeger 01]

Nyström's approximation of stationary kernels

Stationary kernel
(i.e. translation invariant)

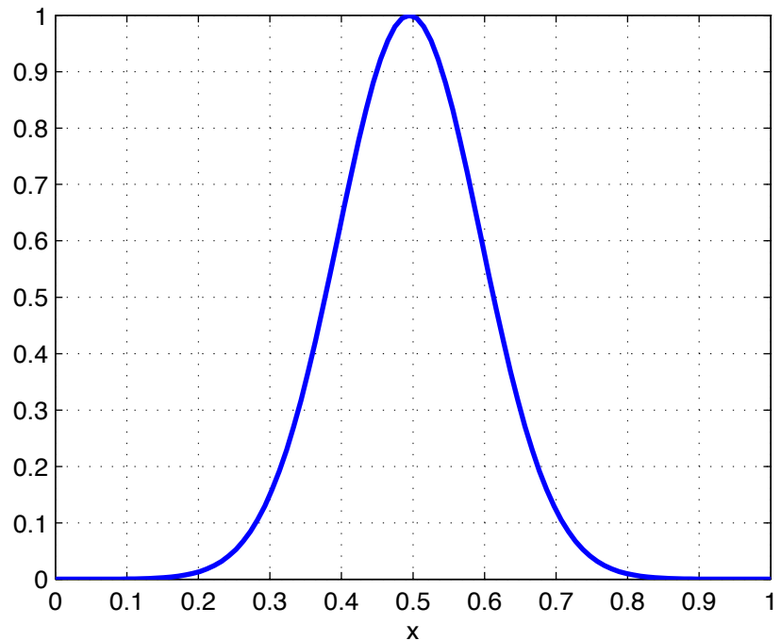
$$\forall \mathbf{t} \in \mathbb{R}^d : K(\mathbf{x}+\mathbf{t}, \mathbf{x}'+\mathbf{t}) = K(\mathbf{x}, \mathbf{x}')$$



$K(x, x')$

Signature / profile
A "cut" through the kernel

$$K(\mathbf{x}, \mathbf{x}') = \mathcal{K}(\mathbf{x}' - \mathbf{x})$$



$\mathcal{K}$

eigenfunctions = sinusoids

$$\frac{1}{(2\pi)^d} \int_{\mathbb{R}^d} \mathcal{K}(\mathbf{x}-\mathbf{z}) e^{-\mathbf{i}\langle \boldsymbol{\omega}, \mathbf{z} \rangle} \, d\mathbf{z} = \kappa_{\boldsymbol{\omega}}^2 e^{-\mathbf{i}\langle \boldsymbol{\omega}, \mathbf{x} \rangle}$$

inverse
Fourier transform

$$\Phi_{\boldsymbol{\omega}}(\mathbf{x}) = \kappa_{\boldsymbol{\omega}} e^{-\mathbf{i}\langle \boldsymbol{\omega}, \mathbf{x} \rangle}$$

Random Fourier features for stationary kernels

feature map

$$\Phi_{\boldsymbol{\omega}}(\mathbf{x}) = \kappa_{\boldsymbol{\omega}} e^{-\mathbf{i}\langle \boldsymbol{\omega}, \mathbf{x} \rangle}$$

kernel reconstruction

$$K(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle = \int_{\mathbb{R}^d} e^{-\mathbf{i}\langle \mathbf{x}-\mathbf{x}', \boldsymbol{\omega} \rangle} \kappa_{\boldsymbol{\omega}}^2 \, d\boldsymbol{\omega}$$

approximate the integral by **sampling**

$$K(\mathbf{x}, \mathbf{x}') \approx \langle \widehat{\Phi}(\mathbf{x}), \widehat{\Phi}(\mathbf{x}') \rangle = \frac{1}{D} \sum_{i=1}^{D} e^{-\mathbf{i}\langle \mathbf{x}-\mathbf{x}', \boldsymbol{\omega}_i \rangle} \qquad \boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_d \sim p(\boldsymbol{\omega}) = \kappa_{\boldsymbol{\omega}}^2$$

$$\Phi_{\mathrm{RFF}}(\mathbf{x}) = D^{-\frac{1}{2}} \begin{bmatrix} \cos\langle \boldsymbol{\omega}_1, \mathbf{x} \rangle & \ldots & \cos\langle \boldsymbol{\omega}_D, \mathbf{x} \rangle & \sin\langle \boldsymbol{\omega}_1, \mathbf{x} \rangle & \ldots \end{bmatrix}^\top$$

random projections drawn from
a kernel-dependent distribution

[Rahimi Recht 07]

Example: Gaussian RBF kernel

kernel

$$K(\mathbf{x}, \mathbf{z}) = \mathcal{K}(\mathbf{x} - \mathbf{z})$$

Gaussian RBF

signature

$$\mathcal{K}(\boldsymbol{\lambda}) = e^{-\frac{\|\boldsymbol{\lambda}\|^2}{2\sigma^2}}$$

Gaussian function

spectrum

$$\kappa^2_{\boldsymbol{\omega}} = \left(\frac{\sigma}{2\pi}\right)^{\frac{d}{2}} e^{-\frac{\sigma^2\|\boldsymbol{\omega}\|^2}{2}}$$

Gaussian density

Random Fourier Feature (RFF)

$$\Phi_{\mathrm{RFF}}(\mathbf{x}) = D^{-\frac{1}{2}} \begin{bmatrix} \cos\langle \boldsymbol{\omega}_1, \mathbf{x}\rangle & \dots & \cos\langle \boldsymbol{\omega}_D, \mathbf{x}\rangle & \sin\langle \boldsymbol{\omega}_1, \mathbf{x}\rangle & \dots \end{bmatrix}^\top$$

D random projections

- projection dimension = *d* as data
- distributed normally with
- null mean
- isotropic standard deviation 1/σ

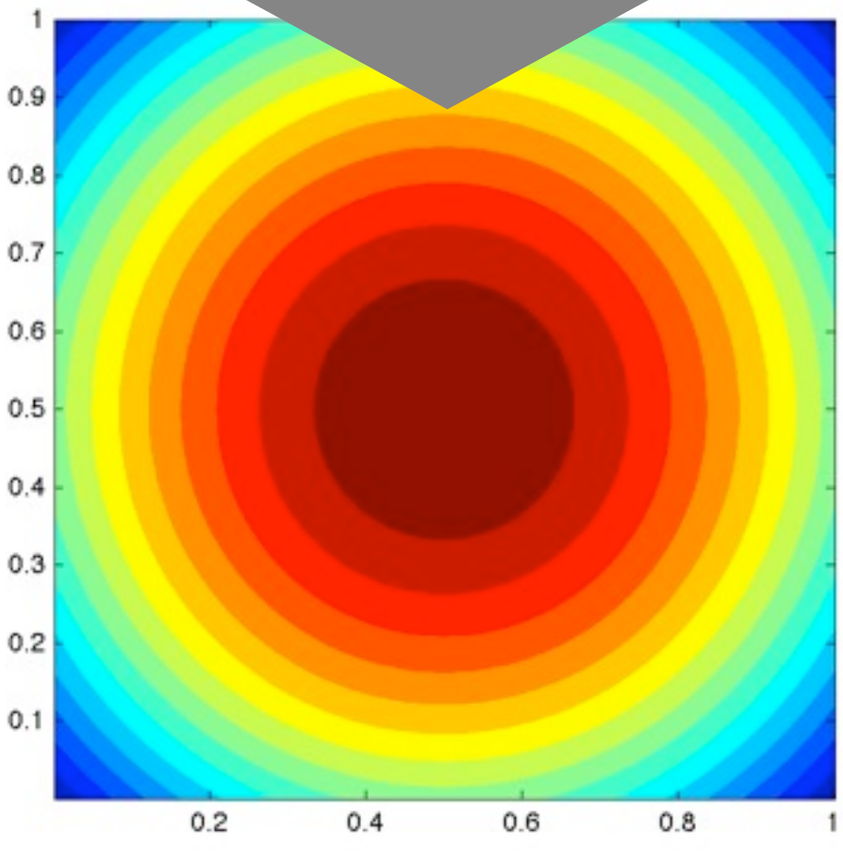
Example: RFF for Gaussian RBF kernel

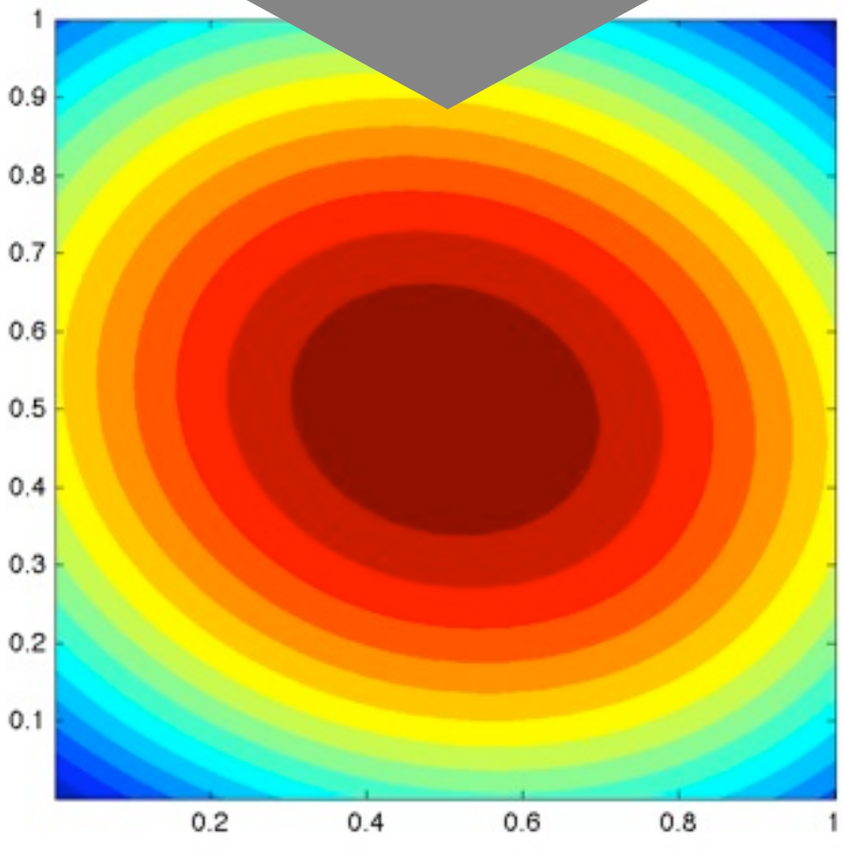
RBF kernel

```
[x1,x2] = meshgrid(range) ;
x = [x1(:) x2(:)]' ;
xp = [0.5;0.5] ;
for i=1:n*n
  sqd1 = (x(1,i) - xp(1))^2 ;

  sqd2 = (x(2,i) - xp(2))^2 ;

  K(i) = exp(-0.5*(sqd1 + sqd2)) ;
end
```

RBF w/Random Fourier Features

```
omega = randn(10,2) ;


psi = [cos(omega*x) ;
       sin(omega*x)] / sqrt(10) ;
psip = [cos(omega*xp) ;
        sin(omega*xp)] / sqrt(10) ;
K = psi'*psip ;
```





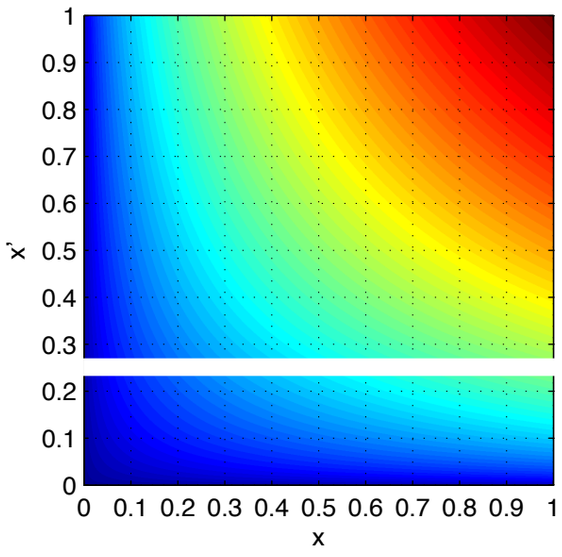
Additive *homogeneous* kernels

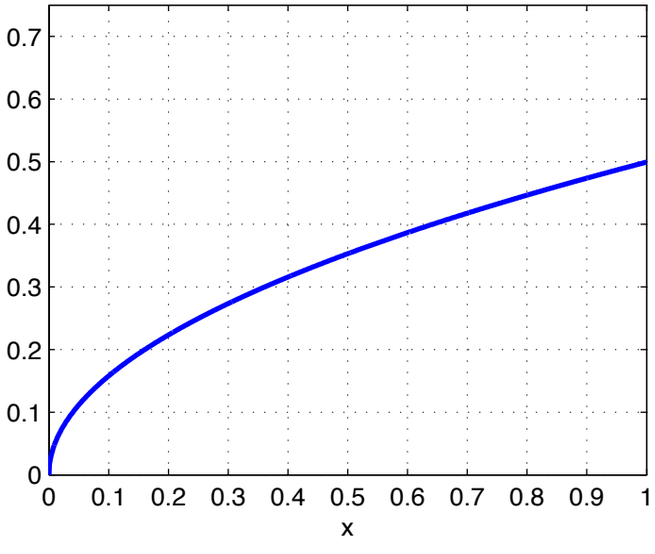
Additive kernel
Sum of 1D kernels

$$K(\mathbf{x}, \mathbf{x}') = \sum_{l=1}^{D} k(x_l, x'_l)$$

Hellinger

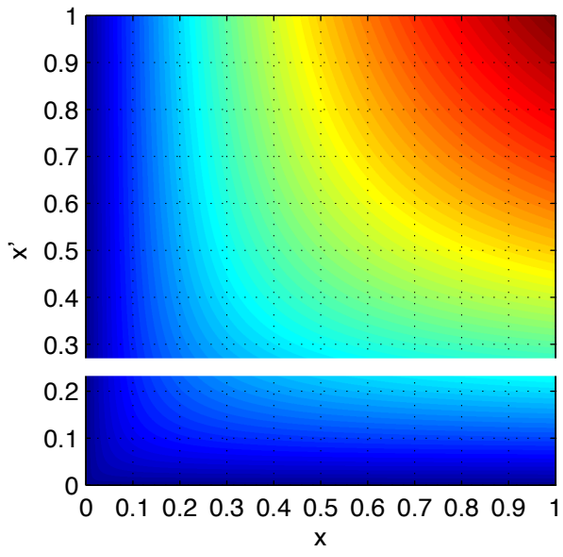
$$k(x, x') = \sqrt{xx'}$$

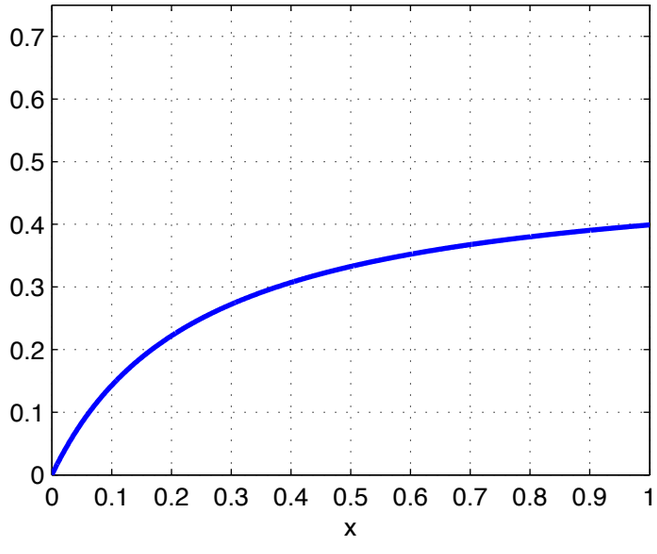




X²

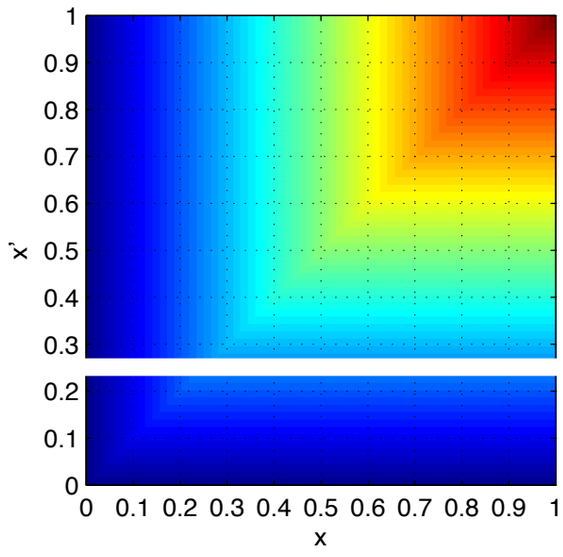
$$\frac{2xx'}{x + x'}$$

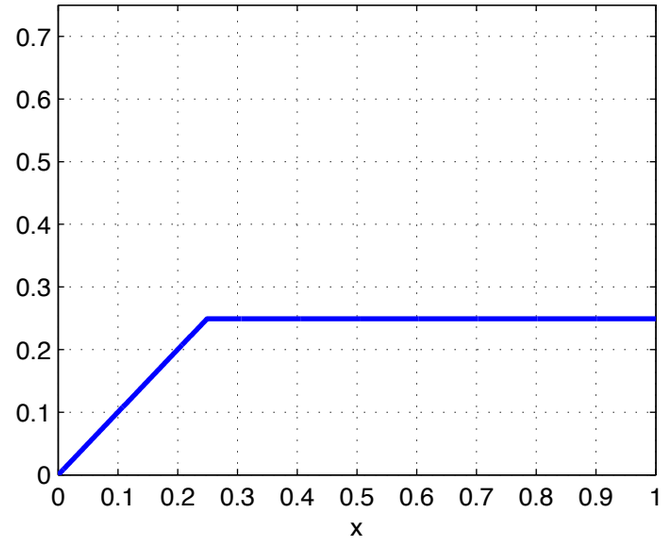




intersection

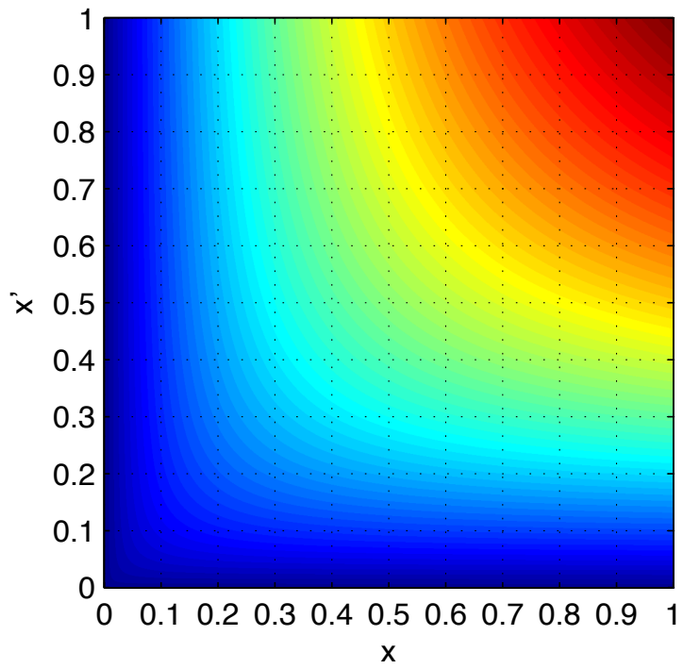
$$\min\{x, x'\}$$



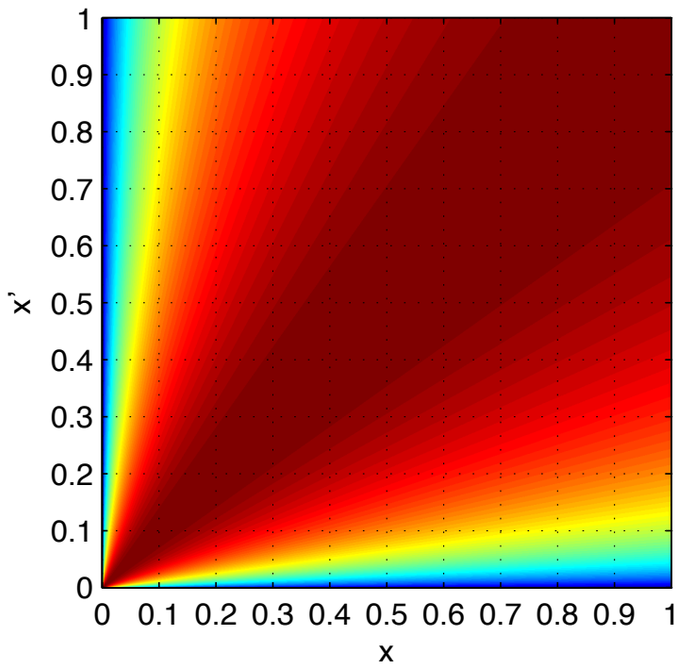


The trick

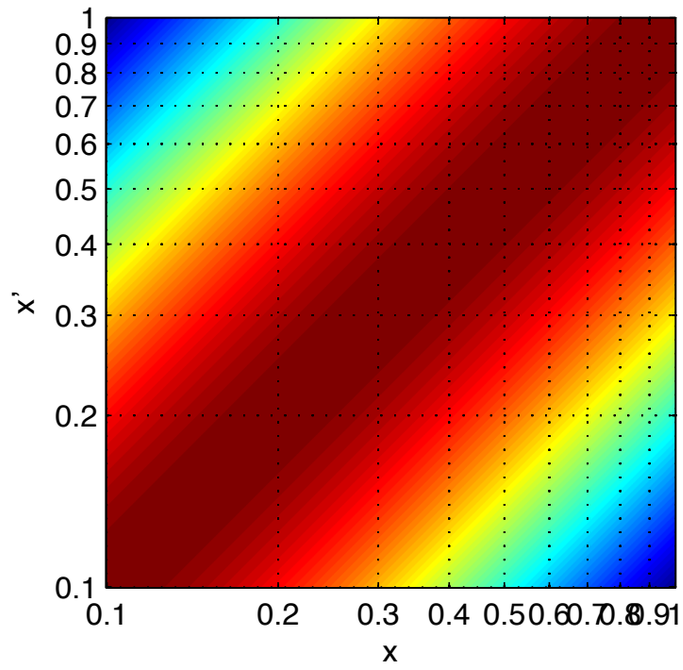
$k(x, x')$



$$\frac{k(x, x')}{\sqrt{xx'}}$$



$\log x$



Homogeneous kernel

Multiplicative constant pops out

$$\forall c \geq 0 : k(cx, cx') = ck(x, x')$$

Signature / profile

Up to a factor and a logarithm

$$k(x, x') = \sqrt{xx'}\, \mathcal{K}(\log x - \log x')$$

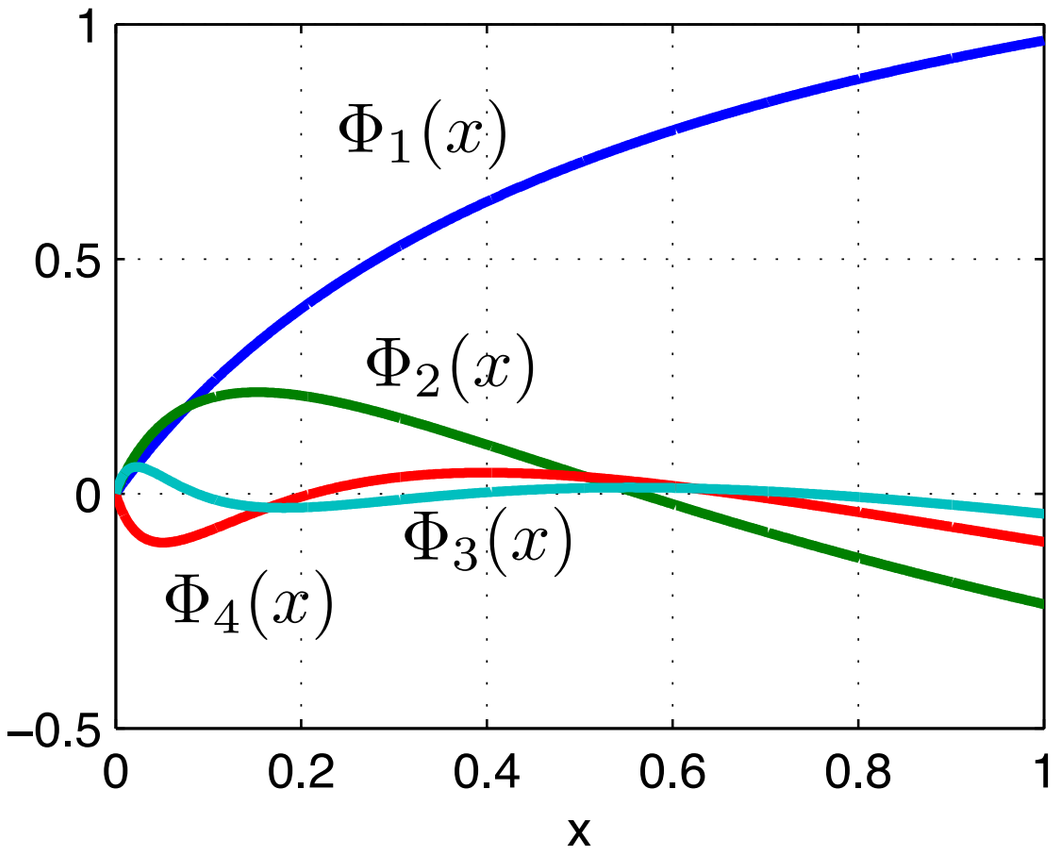
$$\Phi_\omega(x) = \kappa_\omega \sqrt{x}\, e^{-\mathbf{i}\langle \omega, \log x \rangle}$$

[Vedaldi Zisserman 10, 11]

Examples

Hellinger	**X²**	**intersection**
$k(x,x') = \sqrt{xx'}$	$\dfrac{2xx'}{x+x'}$	$\min\{x,x'\}$
$\mathcal{K}(\lambda) = 1$	$e^{-\lvert\lambda\rvert/2}$	$\operatorname{sech}(\lambda/2)$
$\kappa_\omega^2 = \delta(\omega)$	$\dfrac{2}{\pi(1+4\omega^2)}$	$\operatorname{sech}(\pi\omega)$
$\Phi_\omega(x) = \sqrt{x}$	$\sqrt{\dfrac{2x}{\pi(1+4\omega^2)}}e^{-\mathbf{i}\,\omega\log x}$	$\sqrt{x\operatorname{sech}(\pi\omega)}e^{-\mathbf{i}\,\omega\log x}$

Tabulating features

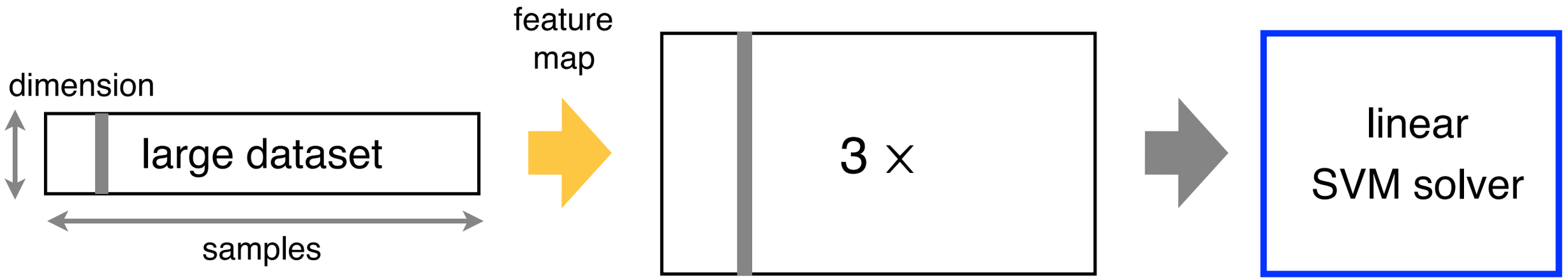


- For additive kernels, a feature decomposes as a sum of 1D functions
- The features can be easily **tabulated**
  - very fast evaluation
- additive Kernel PCA
  [Perronnin *et al.* 10]
  - apply empirical Nyström's approximation component-wise
  - tabulate the resulting functions

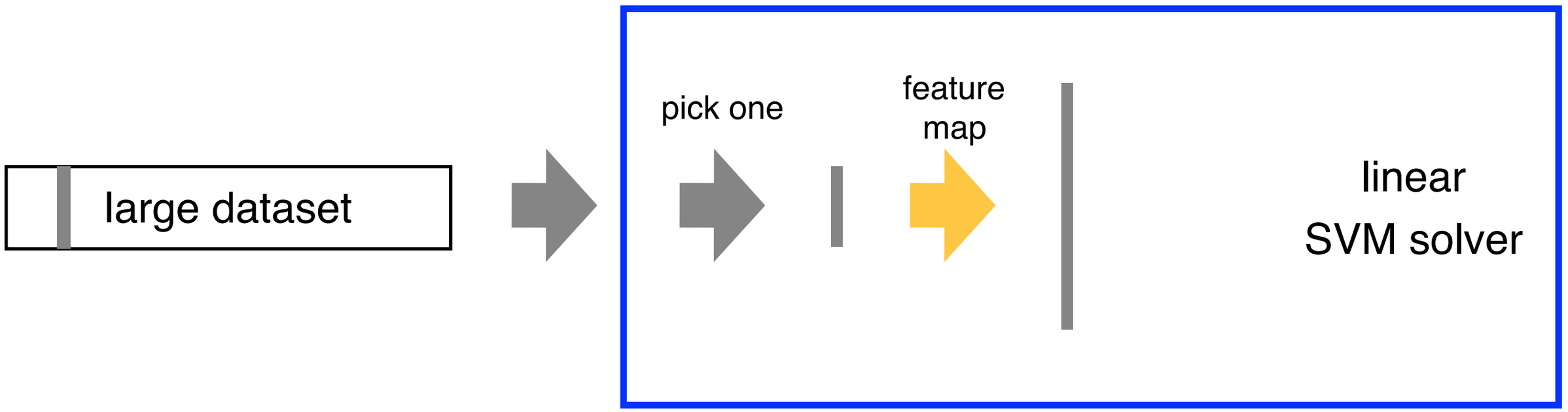
$$\Phi_i(x) = (n\kappa_i)^{-1} \sum_{i=1}^{n} k(x, x_j) u_i(x_j)$$

Space complexity: memory

- Many explicit features increase the data dimensionality



- **Solution**: expand *on-the-fly inside* the solver



Generalized RBF kernels

- Kernel → distance

$$d^2(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}, \mathbf{x}) + K(\mathbf{x}', \mathbf{x}') - 2K(\mathbf{x}, \mathbf{x}')$$

linear kernel

$$\|\mathbf{x} - \mathbf{x}'\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{x}', \mathbf{x}' \rangle - 2\langle \mathbf{x}, \mathbf{x}' \rangle$$

X^2 kernel

$$d^2_{\chi^2}(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d} \frac{2(x_i - x'_i)^2}{x_i + x'_i} = 2 - 2\sum_{i=1}^{d} \frac{2x_i x'_i}{(x_i + x'_i)}$$

- **General RBF** kernels
  - compose exp with any distance $d^2(\mathbf{x}, \mathbf{x}')$

RBF-X^2 kernel

$$K_{\text{RBF-}\chi^2}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2\sigma^2} d^2_{\chi^2}(\mathbf{x}, \mathbf{x}')\right)$$

[Sreekanth et al. 10]

Additive-RBF feature maps

- Generalized RBF ≈ Gaussian RBF in feature space

$$K(\mathbf{x}, \mathbf{x}') \xrightarrow{\text{feature}} \langle \Psi(\mathbf{x}), \Psi(\mathbf{x}') \rangle$$

$$d^2(\mathbf{x}, \mathbf{x}') \longrightarrow \|\Phi(\mathbf{x}) - \Phi(\mathbf{x}')\|^2$$

$$\exp\left(-\frac{1}{2\sigma} d^2(\mathbf{x}, \mathbf{x}')\right) \longrightarrow \exp\left(-\frac{1}{2\sigma} \|\Phi(\mathbf{x}) - \Phi(\mathbf{x}')\|^2\right)$$

- **Chain two features**: one for the distance, and random Fourier for the Gaussian

$$\Phi_{\text{RBF-}\chi^2}(\mathbf{x}) = D^{-\frac{1}{2}} \begin{bmatrix} \cos\langle \boldsymbol{\omega}_1, \Phi_{\chi^2}(\mathbf{x}) \rangle \\ \vdots \\ \cos\langle \boldsymbol{\omega}_D, \Phi_{\chi^2}(\mathbf{x}) \rangle \\ \sin\langle \boldsymbol{\omega}_1, \Phi_{\chi^2}(\mathbf{x}) \rangle \\ \vdots \end{bmatrix}^\top$$

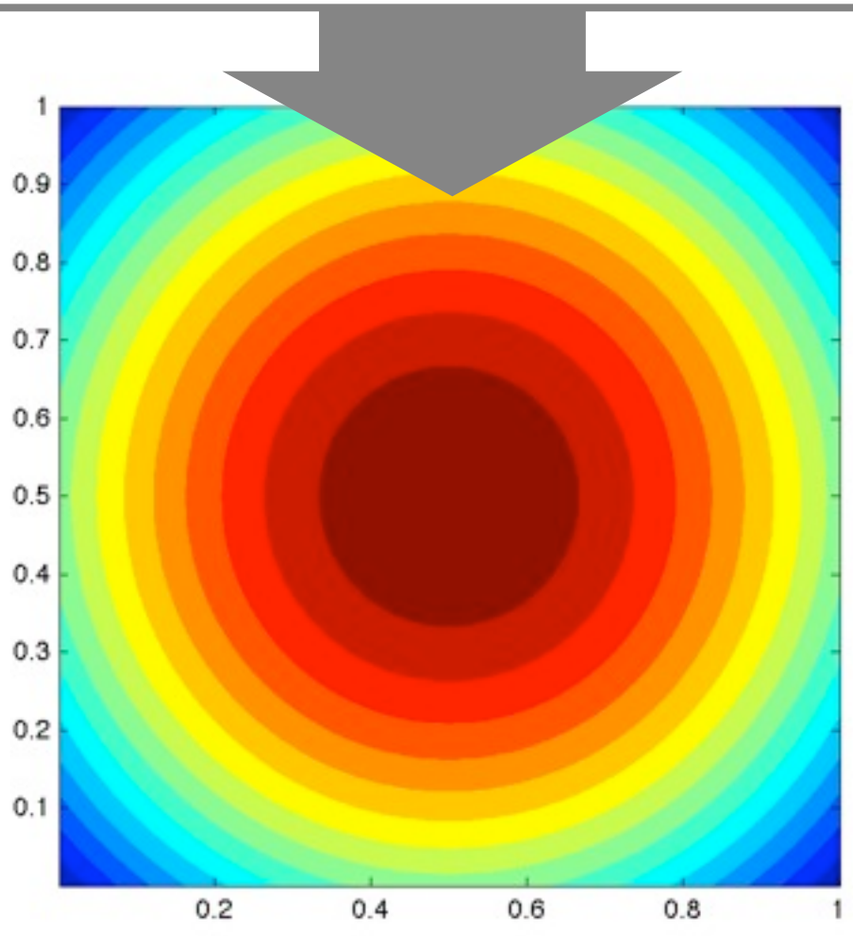
Example: RFF for Gaussian RBF kernel

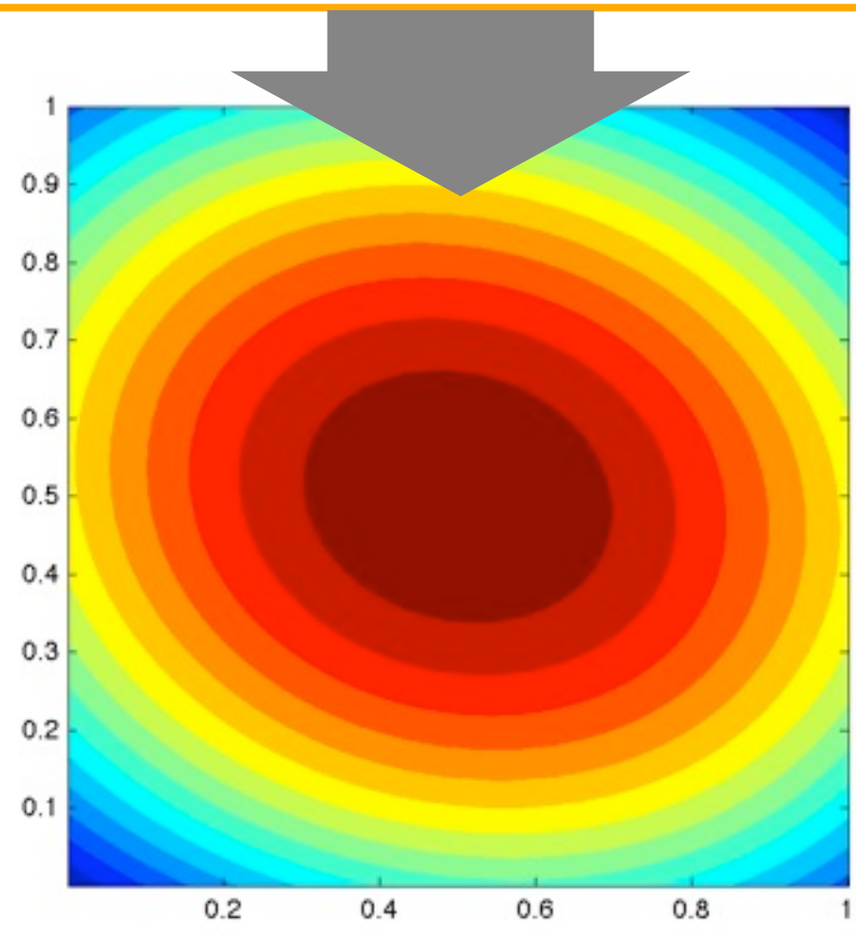
RBF kernel

```
[x1,x2] = meshgrid(range) ;
x = [x1(:) x2(:)]' ;
xp = [0.5;0.5] ;
for i=1:n*n
  sqd1 = (x(1,i) - xp(1))^2 ;

  sqd2 = (x(2,i) - xp(2))^2 ;

  K(i) = exp(-0.5*(sqd1 + sqd2)) ;
end
```

RBF w/Random Fourier Features

```
omega = randn(10,2) ;


psi = [cos(omega*x) ;
       sin(omega*x)] / sqrt(10) ;
psip = [cos(omega*xp) ;
        sin(omega*xp)] / sqrt(10) ;
K = psi'*psip ;
```





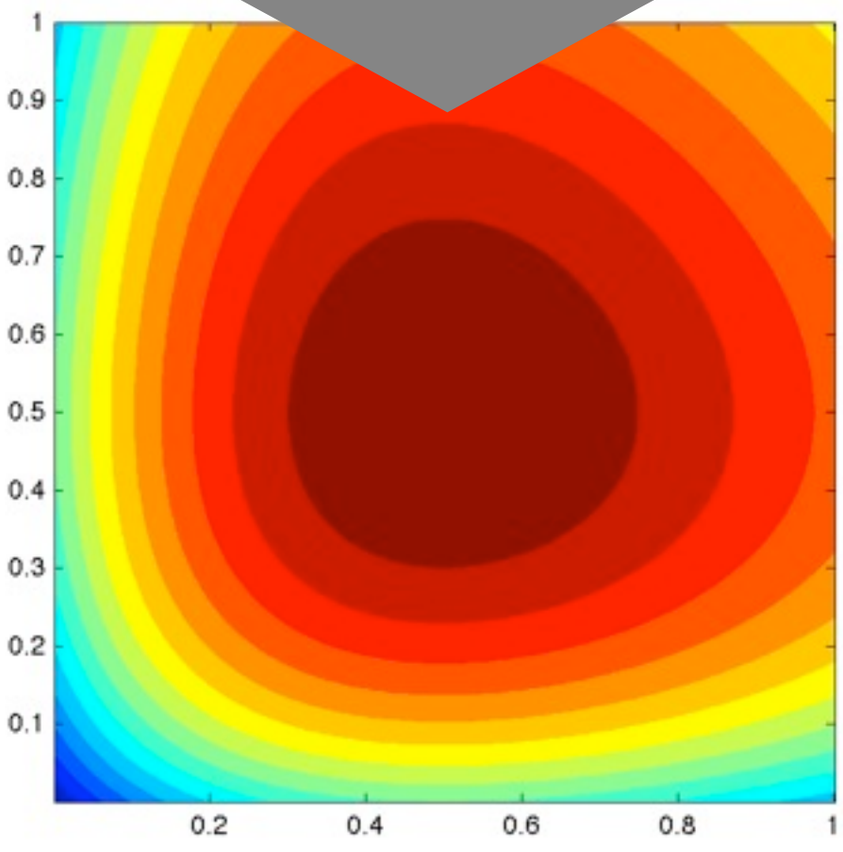
Example: RFF for X^2 RBF kernel

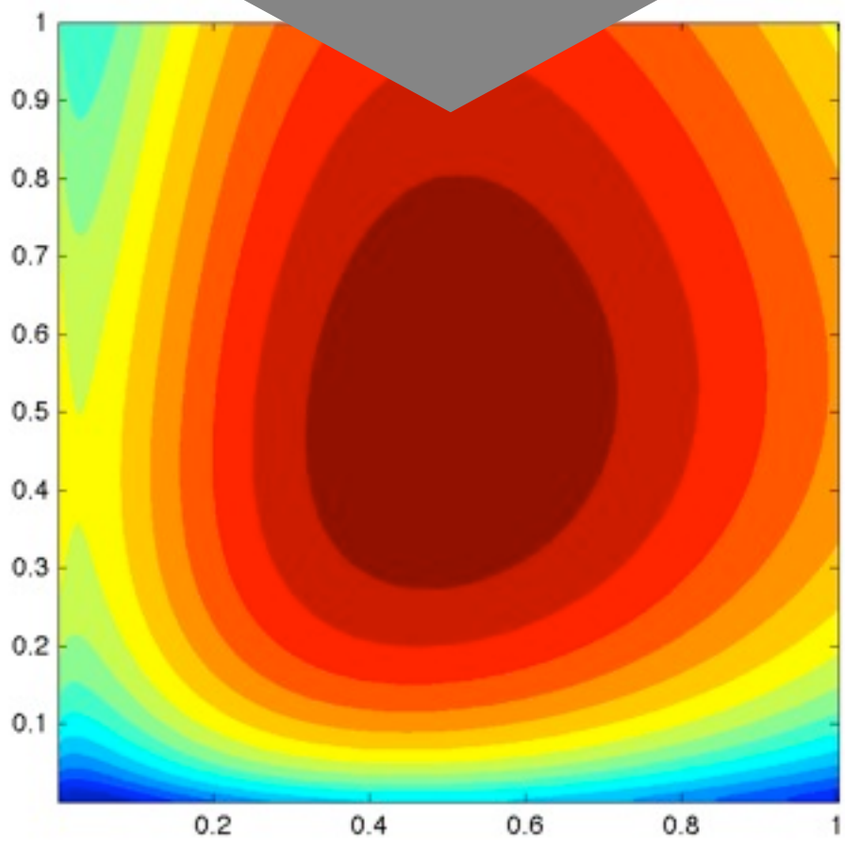
Chi2-RBF kernel

```
[x1,x2] = meshgrid(range) ;
x = [x1(:) x2(:)]' ;
xp = [0.5;0.5] ;
for i=1:n*n
  sqd1 = (x(1,i) - xp(1))^2 /
(x(1,i)+xp(1)) ;
  sqd2 = (x(2,i) - xp(2))^2 /
(x(2,i)+xp(2)) ;
  K(i) = exp(-0.5*(sqd1 + sqd2)) ;
end
```

Chi2-RBF w/Random Fourier Features

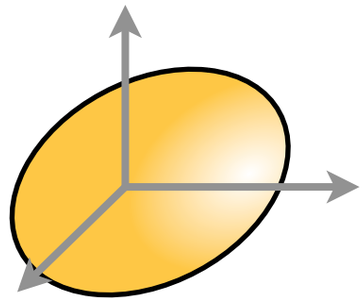
```
omega = randn(10,6) ;
x = vl_homkermap(x,1) ;
xp = vl_homkermap(xp,1) ;
psi = [cos(omega*x) ;
       sin(omega*x)] / sqrt(10) ;
psip = [cos(omega*xp) ;
        sin(omega*xp)] / sqrt(10) ;
K = psi'*psip ;
```





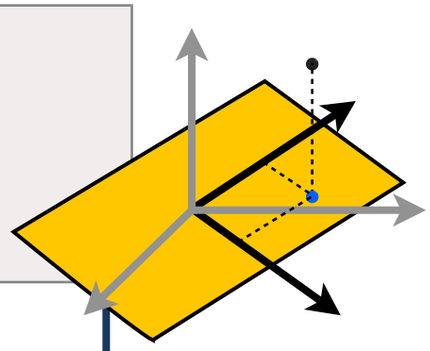
kernel $K(\mathbf{x}, \mathbf{x}')$

∞ dimensional
feature space



finite dimensional
subspaces

dense
(PCA)



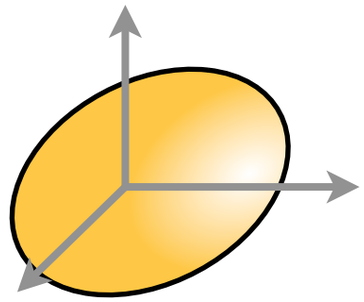
homogeneous
kernel map

addKPCA

additive kernels

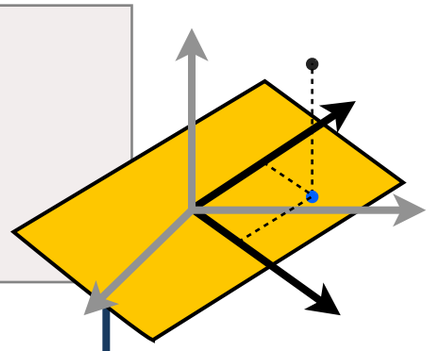
kernel $K(\mathbf{x}, \mathbf{x}')$

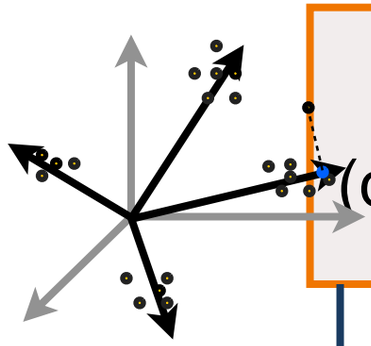
∞ dimensional
feature space



finite dimensional
subspaces

dense
(PCA)





sparse
(dictionary learning)

homogeneous
kernel map

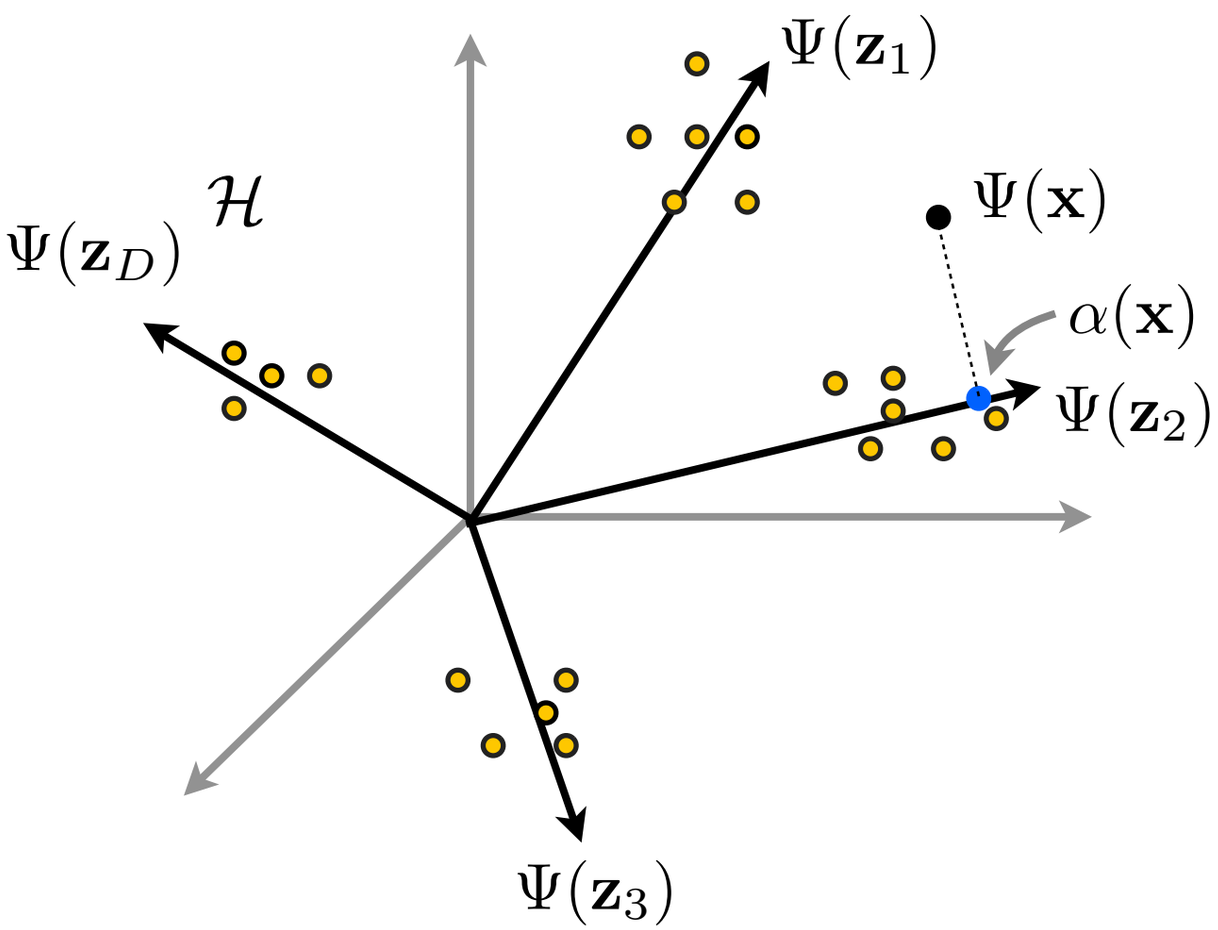
addKPCA

sparse add.
(fast IKSVM)

PQ

additive kernels

Sparse kernel maps



Sparse projection

$$\Psi(\mathbf{x}) \approx \sum_{i=1}^{D} \Psi(\mathbf{z}_i)\Phi_i(\mathbf{x}) \qquad \Phi(\mathbf{x}) = \begin{bmatrix} 0 \\ \alpha(\mathbf{x}) \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- **non-diagonal** inner product

$$K(\mathbf{x}, \mathbf{x}') \approx \Phi(\mathbf{x})^\top K_{ZZ} \Phi(\mathbf{x}')$$

- **The goal is still to "fit" the data**
  - kernel function (= exact feature)

$$K(\mathbf{x}, \mathbf{x}') = \langle \Psi(\mathbf{x}), \Psi(\mathbf{x}') \rangle_{\mathcal{H}}$$

  - empirical data distribution

$$\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$$

- **Redundant basis**
  - find representative points

$$Z = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_D\}$$

  - obtain "over-complete" basis

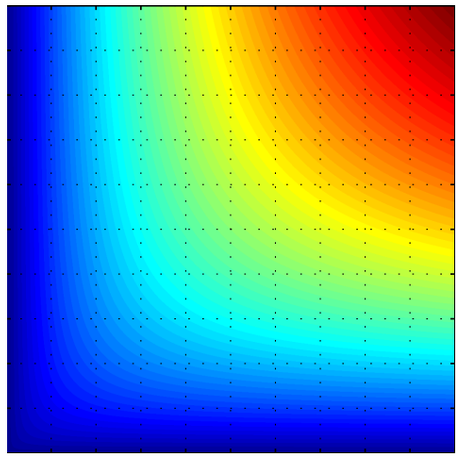
$$\Psi_i = \Psi(\mathbf{z}_i)$$

✔ → arbitrarily good approximation

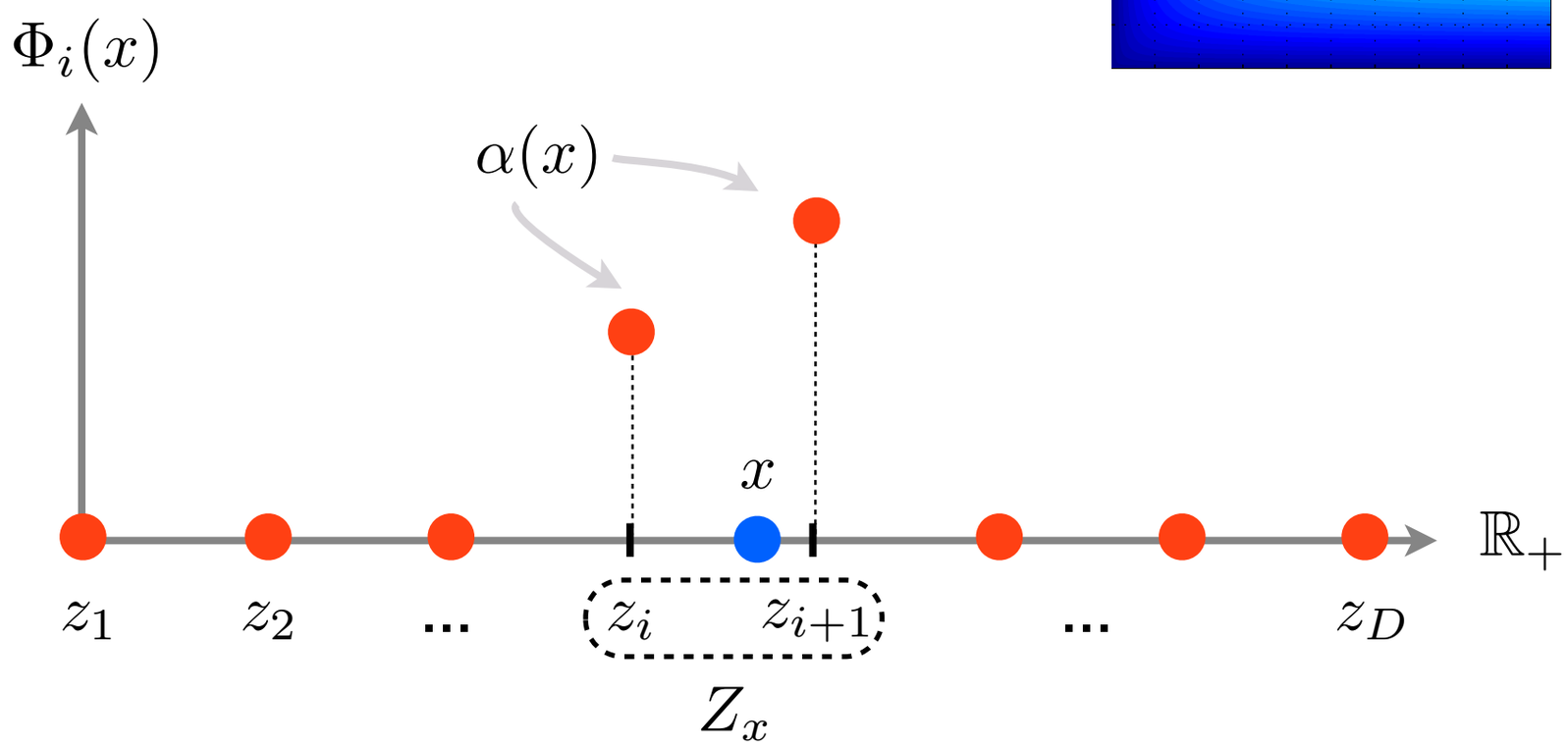
[Snelson Ghaharamani 07, Vedaldi Zisserman 12]
Not the same as [*Kernel Matching Pursuit*, Vincente Bengio 02].

Sparse kernel maps: additive kernels

$$k(x,x') \approx \Phi(x)^\top K_{ZZ} \Phi(x') = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \end{bmatrix}$$



$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$



Fast Intersection Kernel

$$\alpha(x) = \begin{bmatrix} i+1-x \\ x-1 \end{bmatrix}$$

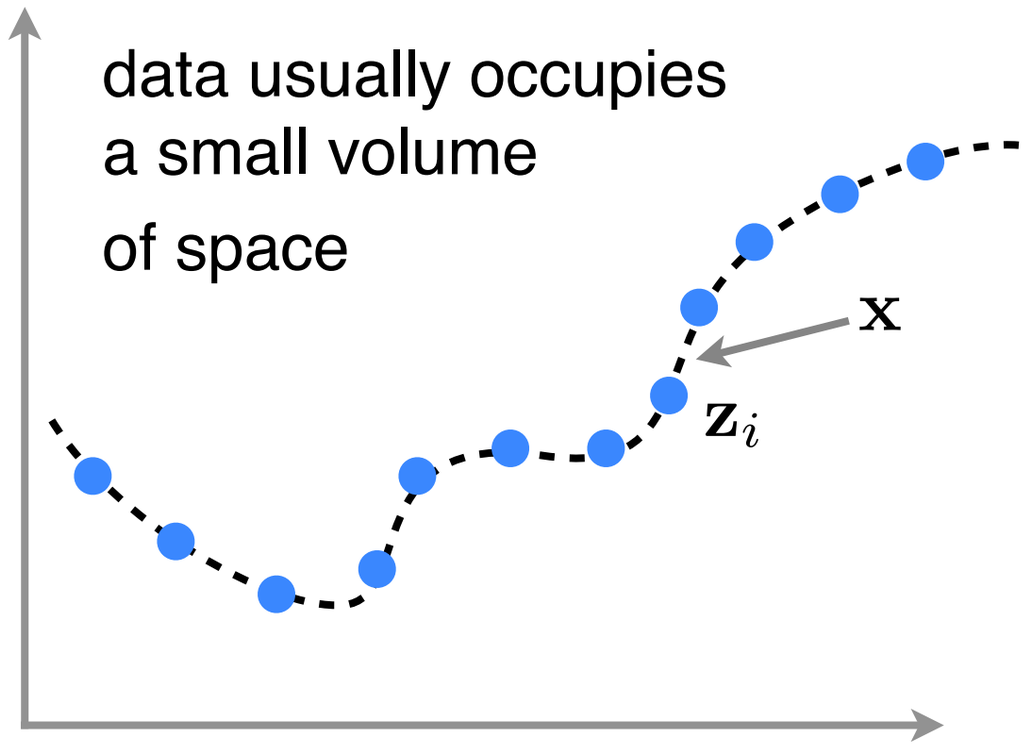
[Maji Berg 09]

X² kernel

$$\alpha(x) = \frac{2(1+2i)x}{(i+x)(1+i+x)} \begin{bmatrix} i+1-x \\ x-1 \end{bmatrix}$$

[Vedaldi Zisserman 12]

Quantisation for compression

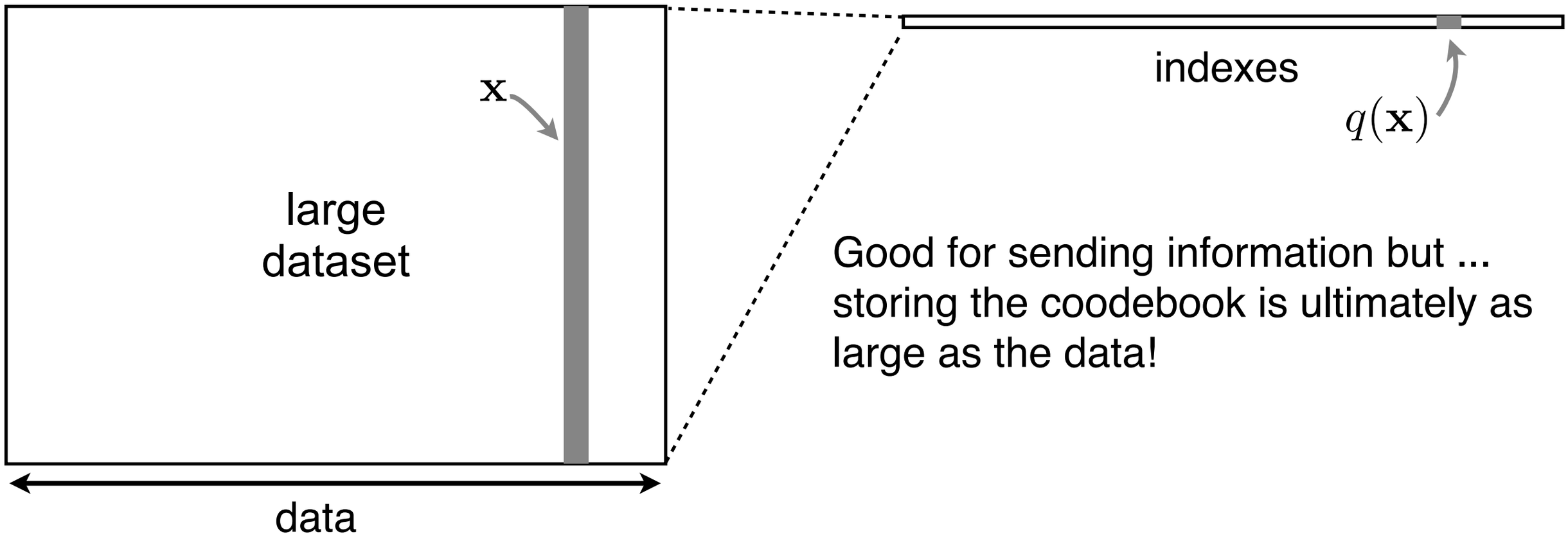


- $\mathbf{z}_i$ **codewords**

$Z = \{\mathbf{z}_1, \ldots, \mathbf{z}_D\}$ **codebook**

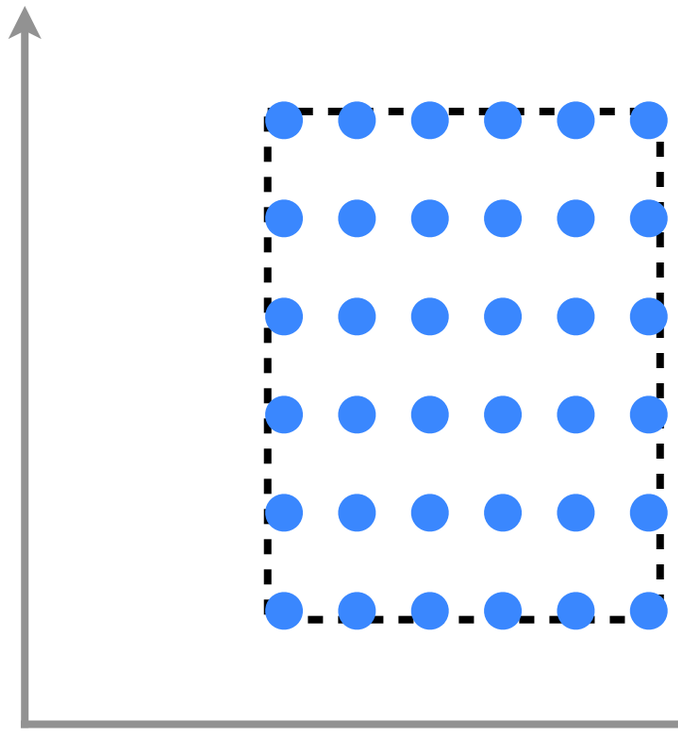
$\mathbf{x} \approx \mathbf{z}_{q(\mathbf{x})}$ **quantisation**: represent a data point by the closest codeword

saving: store only the index *q*, using $\log_2(D)$ bits



Good for sending information but ... storing the coodebook is ultimately as large as the data!

Product Quantisation (PQ)

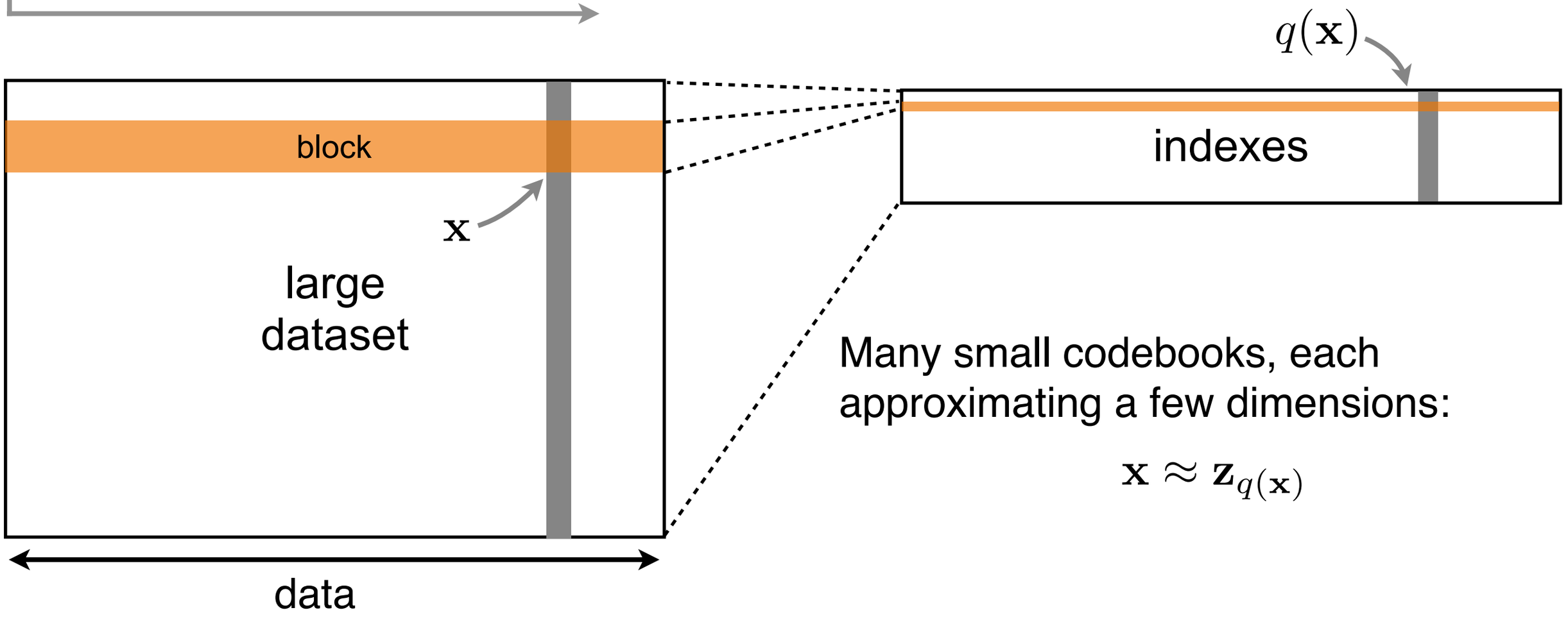


correlation
may be weak

Idea

Apply quantisation to blocks

[Gray Neuhoff 98, Jégou et al. 11, Sánchez and Perronnin 11]



Many small codebooks, each approximating a few dimensions:

$$\mathbf{x} \approx \mathbf{z}_{q(\mathbf{x})}$$

Quantisation as sparse coding

- **Data** = codebook × sparse code

x is approximated by the q(x)-th codeword

code selects q(x) codeword

$$\mathbf{x} \approx \mathbf{z}_{q(x)} = \begin{bmatrix} \mathbf{z}_1 & \dots & \mathbf{z}_D \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = Z\Phi(\mathbf{x})$$

- **Kernel** = inner product of sparse codes

$$K(\mathbf{x}, \mathbf{x}') \approx \Phi(\mathbf{x})^\top Z^\top Z \Phi(\mathbf{x}') = \Phi(\mathbf{x})^\top K_{ZZ} \Phi(\mathbf{x}')$$

The two key operations in learning

(I) Computing many **inner products**
[Jégou et al. 11]

PQ as sparse feature

$$\mathbf{w}^\top \mathbf{x}_i, \qquad i = 1, \ldots, n \longrightarrow (\mathbf{w}^\top Z)\Phi(\mathbf{x}_i)$$

O(*dn*) p ≪ n O(*pn*)

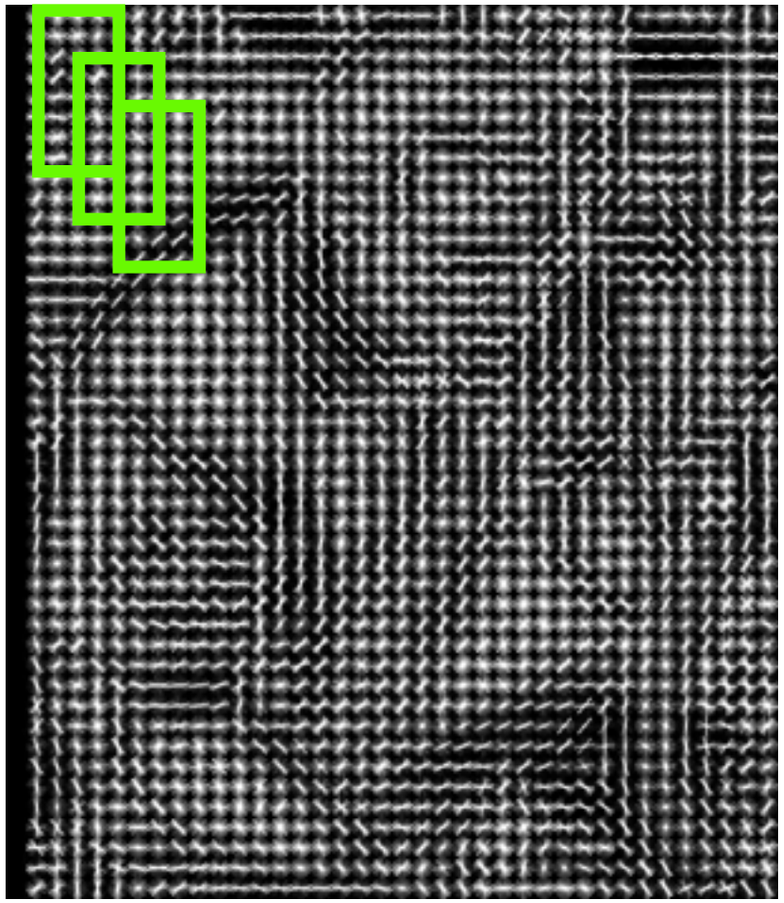
(II) **Accumulating** many vectors
[Vedaldi Zisserman 12]

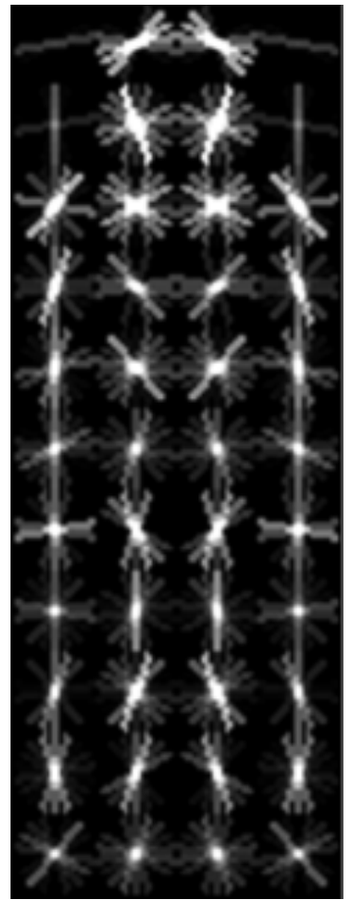
PQ as sparse feature

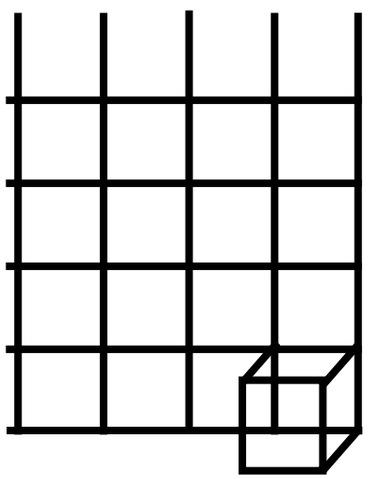
$$\sum_{i=1}^{n} \mathbf{x}_i \longrightarrow Z \sum_{i=1}^{n} \Phi(\mathbf{x}_i)$$

O(*dn*) p ≪ n O(*pn*)

Faster testing: HOG detection example







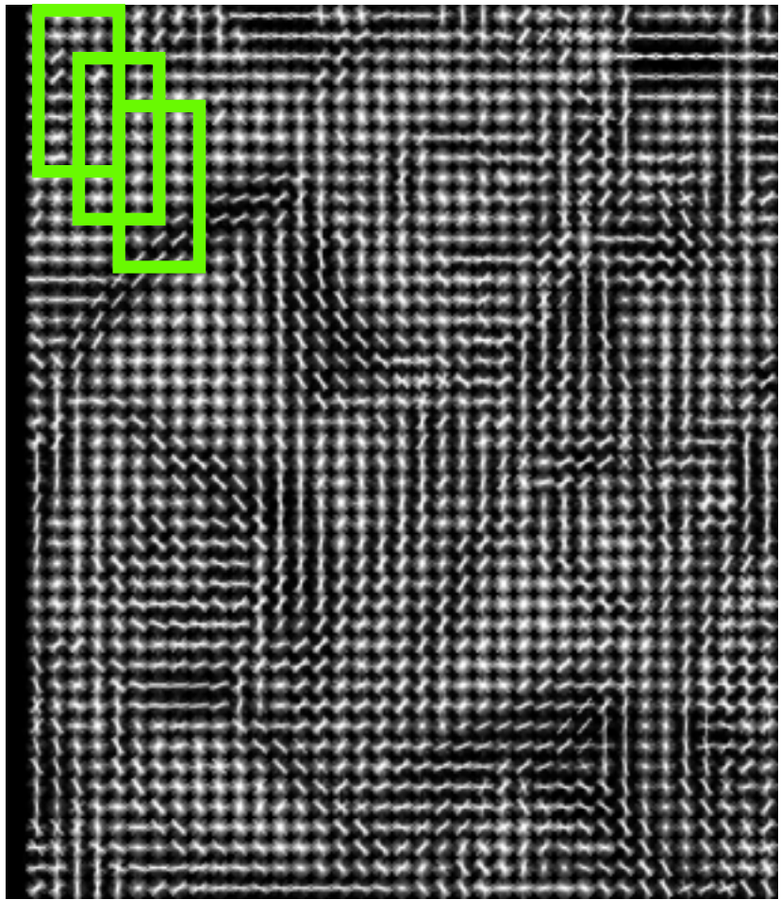
each HOG cell is a
31 dimensional vector

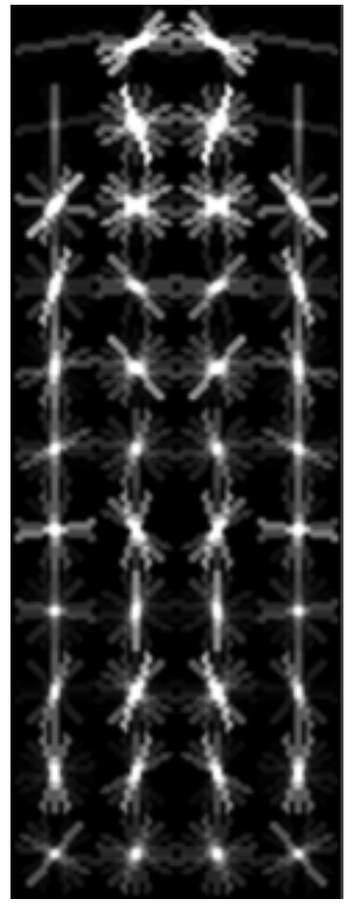
score of one window

$$S(x,y) = \sum_{i=1}^{h}\sum_{j=1}^{w}\sum_{k=1}^{d} w_{ijk}x_{i+x,j+y,k}$$

$$= \sum_{i=1}^{h}\sum_{j=1}^{w} \mathbf{w}_{ij}^{\top}\mathbf{x}_{i+x,j+y}$$

Faster testing: HOG detection example





$$S(x,y) = \sum_{i=1}^{h}\sum_{j=1}^{w}\sum_{k=1}^{d} w_{ijk} x_{i+x,j+y,k} \qquad O(hwd)$$

$$= \sum_{i=1}^{h}\sum_{j=1}^{w} \mathbf{w}_{ij}^{\top} \mathbf{x}_{i+x,j+y} \qquad p \ll n$$

$$= \sum_{i=1}^{h}\sum_{j=1}^{w} (\mathbf{w}_{ij}^{\top} Z)\Phi(\mathbf{x}_{i+x,j+y}) \qquad O(hwp)$$

[Vedaldi Zisserman 12]

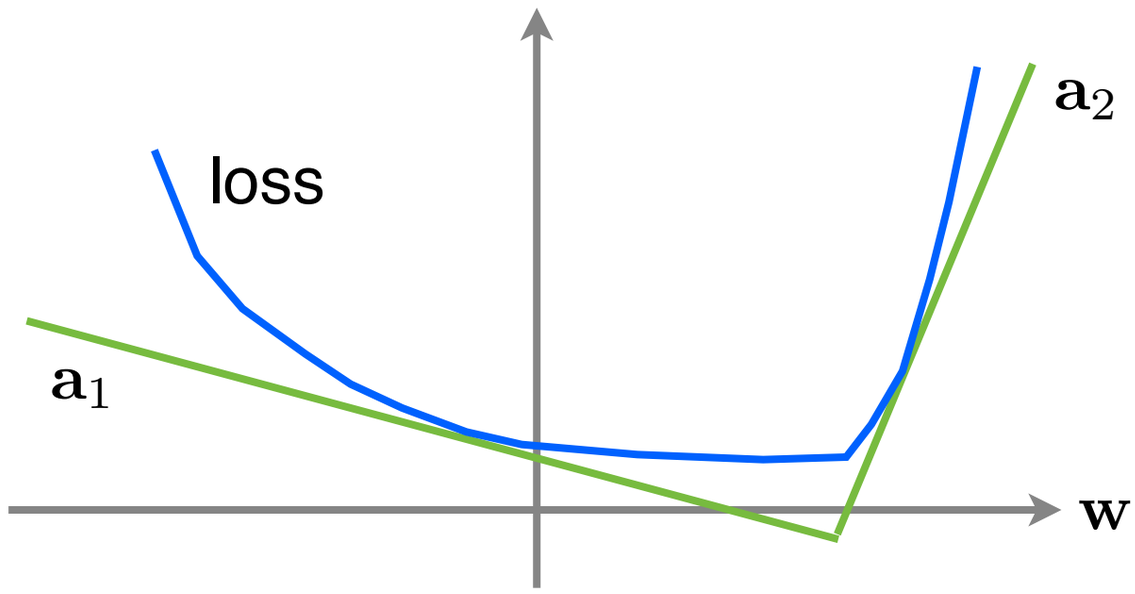
Faster learning with cutting planes

- The model **w** is *high dimensional and dense*

$$\frac{\lambda}{2}\mathbf{w}^\top K_{ZZ}\mathbf{w} + \frac{1}{n}\sum_{i=1}^{n}\max\{0, 1 - y_i\mathbf{w}^\top K_{ZZ}\phi(\mathbf{x}_i)\}$$

- Use *cuts* to lower bound the loss [SVM^{perf}]

$$\frac{\lambda}{2}\mathbf{w}^\top K_{ZZ}\mathbf{w} + \sup_{t=1,\ldots,T} b_t - \langle \mathbf{a}_t, \mathbf{w}\rangle$$



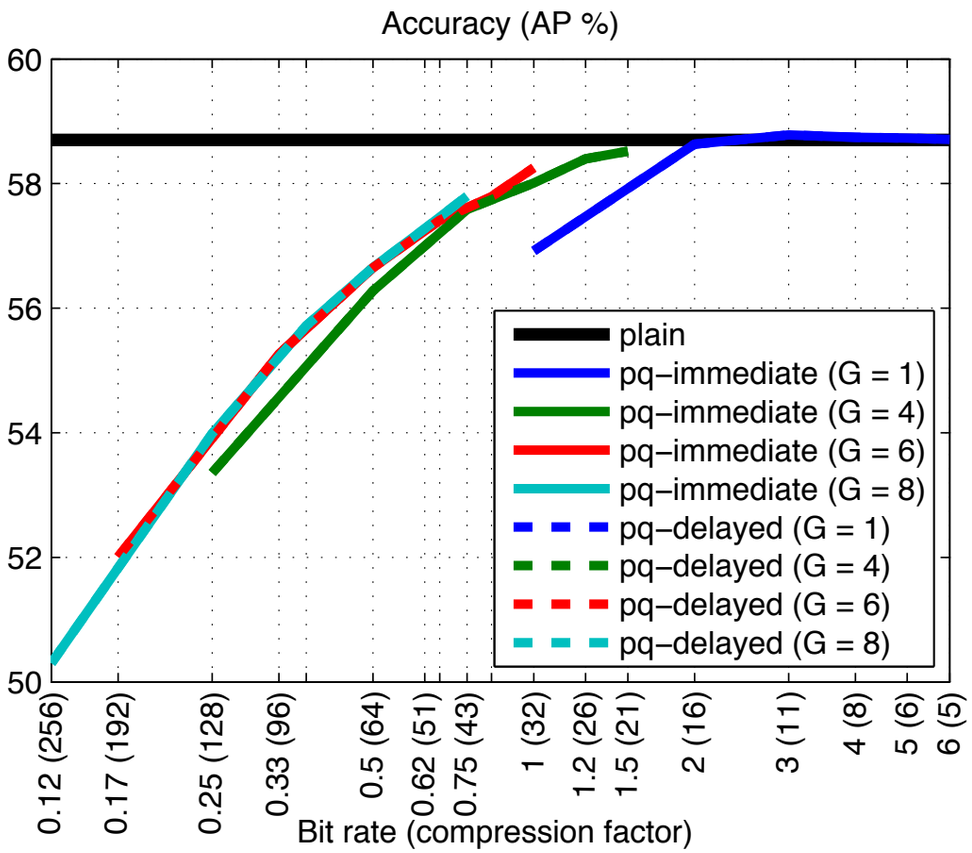
- *How fast can we compute a cut?*

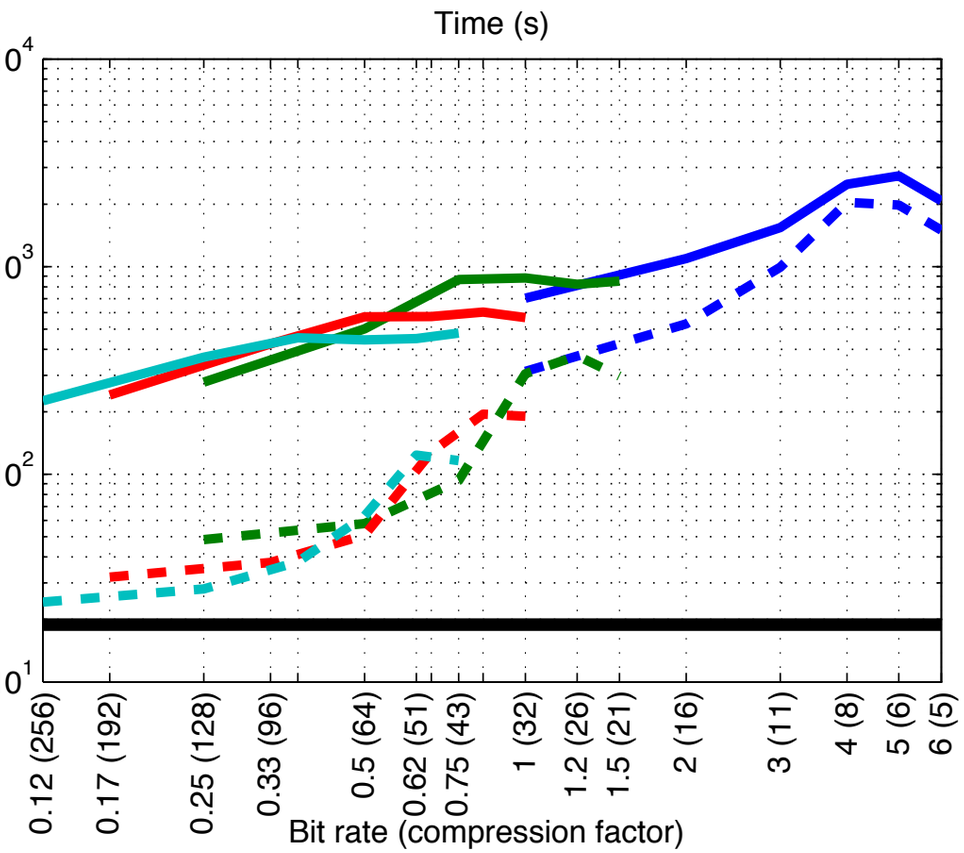
$$\mathbf{a}_t = \frac{1}{n}\sum_{i=1}^{n}\gamma_i(\mathbf{w}_t)K_{ZZ}\phi(\mathbf{x}_i) \quad \textbf{immediate expansion} \quad O(nDP)$$

$$\mathbf{a}_t = \frac{1}{n}K_{ZZ}\sum_{i=1}^{n}\gamma_i(\mathbf{w}_t)\phi(\mathbf{x}_i) \quad \textbf{delayed expansion} \quad O(D^2 + nP)$$

[Vedaldi Zisserman 12]

Fast and compact learning: PASCAL VOC example





for about same performance

58.5% mAP → 58% mAP

40-fold compression factor

2GB → 5 MB

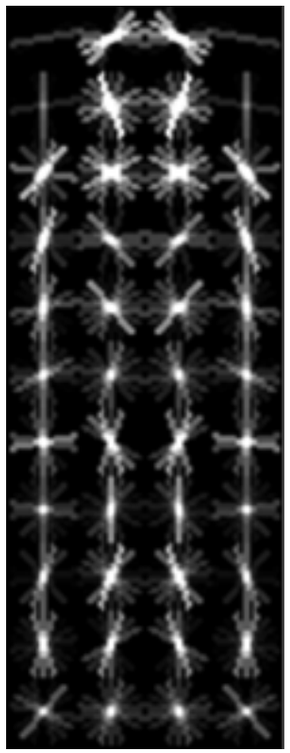
10-fold speedup

1 hour → > 1 minute

[Vedaldi Zisserman 12]

PQ example: *detection*

- **PASCAL VOC 2007**
  - HOG part-based detector
  - 32-dimensional HOG cells
  - block size G = 32
  - dictionary size D = 512



- **Memory reduction**
  - proper code: **113 times**
  - lazy MATLAB code: **32 times**
- **Speedup**
  - Training speedup = testing speedup
  - theoretical: **32 times**
  - lazy MATLAB code: **2 times**

	aerop	bicyc	bird	boat	bottl	bus	car	cat	chair	cow	dinin	dog	horse	motor	perso	potte	sheep	sofa	train	tvmon	mean
AP [%]	30.6	58.1	10.0	12.7	21.2	52.1	55.0	20.1	19.4	22.1	20.5	11.3	56.4	43.6	38.8	10.8	14.8	25.6	43.8	43.8	30.5
neg. mining [min]	173	173	96	169	171	174	143	127	115	160	156	96	161	163	42	136	172	152	176	168	146
solver [min]	6.1	6.2	12.5	6.7	6.8	5.4	8.9	11.2	13.9	7.2	8.3	14.0	7.1	9.1	17.9	9.8	6.3	9.5	6.5	8.2	9.1
neg. mined [GB]	1.02	1.09	1.27	1.05	1.06	1.05	1.05	1.10	1.15	1.06	1.10	1.11	1.13	1.10	1.19	1.04	1.03	1.13	1.11	1.09	1.10
data [GB]	14.0	13.6	15.0	13.2	12.5	13.6	17.8	15.8	14.5	13.1	13.6	16.6	15.1	14.3	34.6	13.3	13.1	13.7	14.7	12.6	15.2
AP [%]	27.9	55.2	9.5	10.4	16.4	47.6	52.0	16.0	13.5	18.6	20.7	10.7	53.4	39.7	37.3	10.4	12.7	19.7	41.7	40.9	27.7
neg. mining [min]	97	100	68	75	89	103	84	63	63	75	88	64	90	93	28	58	90	82	104	94	80
solver [min]	2.4	2.1	3.9	2.1	2.8	1.8	2.3	4.2	4.8	2.7	2.7	4.8	2.6	3.9	4.8	3.2	2.0	3.1	2.3	2.3	3.0
neg. mined [GB]	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.05	0.05	0.04	0.05	0.05	0.04	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04
data [GB]	0.9	0.9	1.0	0.9	0.8	0.9	1.2	1.1	1.0	0.9	0.9	1.1	1.0	1.0	2.4	0.9	0.9	0.9	1.0	0.8	1.0

[Vedaldi Zisserman 12]

Conclusions

- **Feature maps**
  - Explicit linear embeddings reproducing a kernel
  - Allow tremendous speed-ups in learning
  - Particularly simple & efficient for additive kernels
- **Dense low-dimensional features**
  - Similar to PCA
  - *Homogeneous kernel map* (analytical for homogeneous additive)
  - *addKPCA* (empirical Nyström for additive)
  - *Random Fourier features* (Gaussian)
  - *Generalised Random Fourier Features* (Gaussian + additive)
- **Sparse high-dimensional features**
  - Similar to sparse coding
  - *Intersection kernel map* (sparse version)
  - *Product quantisation* for compression
  - Computation in the compressed domain
    - inner products and accumulation

Some references

- S. Maji and A. C. Berg. Max-margin additive classifiers for detection. In Proc. ICCV, 2009.
- S. Maji, A. C. Berg, and J. Malik. Classification using intersection kernel support vector machines is efficient. In Proc. CVPR, 2008.
- F. Perronnin, J. Sánchez, and Y. Liu. Large-scale image categorization with explicit data embedding. In Proc. CVPR, 2010.
- A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. PAMI, 34(3), 2012.
- A. Rahimi and B. Recht. Random features for large-scale kernel machines. In Proc. NIPS, 2007.
- H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. PAMI, 33(1), 2011.
- R. M. Gray and D. L. Neuhoff. Quantization. IEEE Trans. on Information Theory, 1998.
- A. Rahimi and B. Recht. Random features for large-scale kernel machines. In Proc. NIPS, 2007.
- V. Sreekanth, A. Vedaldi, C. V. Jawahar, and A. Zisserman. Generalized RBF feature maps for efficient detection. In Proc. BMVC, 2010.