Large-Scale Kernel Methods - I

Sanjiv Kumar, Google Research, NY

EECS-6898, Columbia University - Fall, 2010

Linear Models

Popular in machine learning / Statistics due to their simplicity

Linear regression $y = w^T x + w_0 \qquad x \in \Re^d, y \in \Re$

Linear SVM $y = \mathrm{sgn}(w^T x + w_0) \quad x \in \Re^d, y \in \{-1, 1\}$

Logistic Regression $p(y = 1 | x) = \sigma(w^T x + w_0) \quad x \in \Re^d, y \in \{-1, 1\}$

- Also common in other applications e.g., dimensionality reduction
  - Principal Components Analysis (PCA)
  - Linear Discriminant Analysis (LDA)

Linear Models

Popular in machine learning / Statistics due to their simplicity

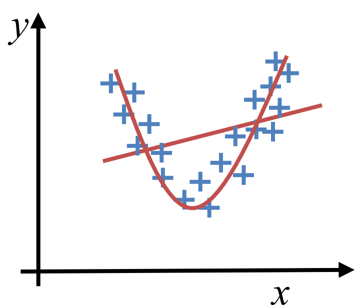
Linear regression $y = w^T x + w_0$ $\quad x \in \Re^d, y \in \Re$

Linear SVM $y = \mathrm{sgn}(w^T x + w_0)$ $\quad x \in \Re^d, y \in \{-1, 1\}$

Logistic Regression $p(y=1 \mid x) = \sigma(w^T x + w_0)$ $\quad x \in \Re^d, y \in \{-1, 1\}$

- Also common in other applications e.g., dimensionality reduction
  - Principal Components Analysis (PCA)
  - Linear Discriminant Analysis (LDA)
- For real-world data, linear models usually not sufficient

How to learn nonlinear models?



Nonlinear Models

One possible way of creating a nonlinear model

- Map the input x nonlinearly

$$x \rightarrow \Phi(x) \qquad x \in \Re^d, \Phi(x) \in \Re^D \quad \text{usually } D \geq d$$

- Learn a linear model in the new space

$$y = w^T \Phi(x)$$

Advantage of this view: Learning linear models well-known !

Nonlinear Models

One possible way of creating a nonlinear model

- Map the input x nonlinearly

$$x \rightarrow \Phi(x) \qquad x \in \Re^d, \Phi(x) \in \Re^D \quad \text{usually } D \geq d$$

- Learn a linear model in the new space

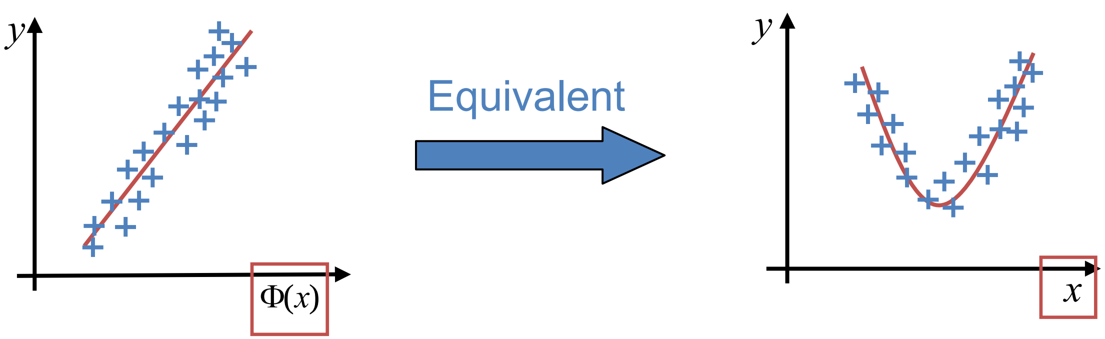
$$y = w^T \Phi(x)$$

Advantage of this view: Learning linear models well-known !

- Example: Quadratic Mapping

$$x = [x_1, x_2]^T \rightarrow \Phi(x) = [x_1, x_2, x_1^2, x_2^2, x_1 x_2]^T$$

$$y = w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2$$



Nonlinear Models

One possible way of creating a nonlinear model

- Map the input x nonlinearly

$$x \rightarrow \Phi(x) \qquad x \in \Re^d, \Phi(x) \in \Re^D \quad \text{usually } D \geq d$$

- Learn a linear model in the new space

$$y = w^T \Phi(x)$$

Advantage of this view: Learning linear models well-known !

- Example: Quadratic Mapping

$$x = [x_1, x_2]^T \rightarrow \Phi(x) = [x_1, x_2, x_1^2, x_2^2, x_1 x_2]^T$$

- Issues
  - One has to choose the degree (d_0) of mapping
  - Exponential explosion in dimension of new space

$$D = O(d^{d_0}) \qquad \text{Intractable for even moderate d and } d_0$$

Nonlinear Models

Another related way: use of nonlinear basis functions

- Map the input x nonlinearly

$$x \rightarrow \Phi(x) \qquad x \in \Re^d, \Phi(x) \in \Re^D$$

$$\Phi(x) = [\Phi_1(x), \ldots, \Phi_D(x)]^T \qquad \Phi_j(x) = f(x, \theta_j)$$

- Examples
  - Radial Basis Function $\Phi_j(x) = \exp(-\|x - \mu_j\|^2 / \sigma_j^2)$
  - Sigmoid Function $\Phi_j(x) = \sigma((x - \mu_j)/s_j)$
  - Also Fourier and wavelet bases

Nonlinear Models

Another related way: use of nonlinear basis functions

- Map the input x nonlinearly

$$x \rightarrow \Phi(x) \qquad x \in \Re^d, \Phi(x) \in \Re^D$$

$$\Phi(x) = [\Phi_1(x), \ldots, \Phi_D(x)]^T \qquad \Phi_j(x) = f(x, \theta_j)$$

- Examples
  - Radial Basis Function $\Phi_j(x) = \exp(-\|x - \mu_j\|^2 / \sigma_j^2)$
  - Sigmoid Function $\Phi_j(x) = \sigma((x - \mu_j)/s_j)$
  - Also Fourier and wavelet bases
- Learn a linear model in the new space $y = w^T \Phi(x)$
- Issues
  - Need to fix (number and parameters of) basis functions a-priori
  - With increased dimensionality, more basis functions needed

Kernel Method

A flexible method for creating nonlinear models using Mercer kernels

- Implicit (nonlinear) mapping of the input x such that

$$x \rightarrow \Phi(x)$$ feature map may be unknown

Mercer Kernel $k(x, y) \rightarrow \Phi(x)^T \Phi(y)$ represents similarity between inputs

Kernel Method

A flexible method for creating nonlinear models using Mercer kernels

- Implicit (nonlinear) mapping of the input x such that

$$x \rightarrow \Phi(x)$$

feature map may be unknown

Mercer Kernel $k(x, y) \rightarrow \Phi(x)^T \Phi(y)$ represents similarity between inputs

- Learn a linear model in the new space

$$y = w^T \Phi(x)$$

Kernel Method

A flexible method for creating nonlinear models using Mercer kernels

- Implicit (nonlinear) mapping of the input x such that

$$x \rightarrow \Phi(x)$$ feature map may be unknown

Mercer Kernel $k(x, y) \rightarrow \Phi(x)^T \Phi(y)$ represents similarity between inputs

- Learn a linear model in the new space

$$y = w^T \Phi(x)$$ but $\Phi(x)$ is not known !!

- "Kernel Trick"
  - If possible, formulate the problem such that feature map appears only in dot products → replace these by kernel function

Kernel Method

A flexible method for creating nonlinear models using Mercer kernels

- Implicit (nonlinear) mapping of the input x such that

  $$x \rightarrow \Phi(x)$$ feature map may be unknown

  Mercer Kernel $k(x, y) \rightarrow \Phi(x)^T \Phi(y)$ represents similarity between inputs

- Learn a linear model in the new space

  $$y = w^T \Phi(x)$$ but $\Phi(x)$ is not known !!

- "Kernel Trick"
  - If possible, formulate the problem such that feature map appears only in dot products → replace these by kernel function
- Issues
  - Need to fix the family of kernels, e.g, RBF kernel, Polynomial kernel, …
  - Kernel parameters usually hand-tuned
  - Multiple kernels can be combined to define an effective single kernel

Multiple kernel learning

Example - Nonlinear (Kernel) Regression

Given: A labeled training set, $\{x_i, y_i\}_{i=1\ldots n}$ $\quad x_i \in \Re^d, y_i \in \Re$

Linear Regression $\quad y = w^T x$

Kernel Regression $\quad y = w^T \Phi(x)$

$$L(w) = \sum_{i=1..n} (w^T \Phi(x_i) - y_i)^2 + \lambda w^T w$$

$$\frac{\partial L(w)}{\partial w} = 0 \Rightarrow \quad w = \sum_i (-1/\lambda)(w^T \Phi(x_i) - y_i)\Phi(x_i)$$

Example - Nonlinear (Kernel) Regression

Given: A labeled training set, $\{x_i, y_i\}_{i=1...n} \quad x_i \in \Re^d, y_i \in \Re$

Linear Regression $\quad y = w^T x$

Kernel Regression $\quad y = w^T \Phi(x)$

$$L(w) = \sum_{i=1..n} (w^T \Phi(x_i) - y_i)^2 + \lambda w^T w$$

$$\frac{\partial L(w)}{\partial w} = 0 \Rightarrow \quad w = \sum_i \underbrace{(-1/\lambda)(w^T \Phi(x_i) - y_i)}_{\alpha_i} \Phi(x_i)$$

$$\boxed{w = \sum_i \alpha_i \Phi(x_i)}$$

solution lives in the span of feature maps !

$\alpha \in \Re^n$

Suppose $\Phi = [\Phi(x_1), \ldots, \Phi(x_n)]_{D \times n}$ Design Matrix (transposed)

$$\boxed{w = \Phi \alpha}$$

reparametrization of coefficients

$$L(w) = (y - \Phi^T w)^T (y - \Phi^T w) + \lambda w^T w$$

Example - Nonlinear (Kernel) Regression

Given: A labeled training set, $\{x_i, y_i\}_{i=1\ldots n}$ $\quad x_i \in \Re^d, y_i \in \Re$

$$y = w^T \Phi(x) = \alpha^T \Phi^T \Phi(x) = \boxed{\sum_{i=1}^{n} \alpha_i k(x, x_i)}$$

Example - Nonlinear (Kernel) Regression

Given: A labeled training set, $\{x_i, y_i\}_{i=1...n}$ $\quad x_i \in \Re^d, y_i \in \Re$

$$y = w^T \Phi(x) = \alpha^T \Phi^T \Phi(x) = \boxed{\sum_{i=1}^n \alpha_i k(x, x_i)}$$

Estimating α

$$L(\alpha) = (y - \Phi^T \Phi \alpha)^T (y - \Phi^T \Phi \alpha) + \lambda \alpha^T \Phi^T \Phi \alpha$$

$\boxed{\text{Gram or Kernel Matrix } \Phi^T \Phi = K}$ $= [k(x_i, x_j)]_{i=1,..,n}^{j=1,..,n}$

$$L(\alpha) = (y - K\alpha)^T (y - K\alpha) + \lambda \alpha^T K \alpha$$

$$\frac{\partial L(\alpha)}{\partial \alpha} = 0 \Rightarrow \quad K(K + \lambda I)\alpha = Ky$$

If K is positive definite, $\boxed{\alpha = (K + \lambda I)^{-1} y}$

Example - Nonlinear (Kernel) Regression

Given: A labeled training set, $\{x_i, y_i\}_{i=1\ldots n}$ $\quad x_i \in \Re^d, y_i \in \Re$

$$y = w^T \Phi(x) = \alpha^T \Phi^T \Phi(x) = \boxed{\sum_{i=1}^n \alpha_i k(x, x_i)}$$

Estimating α

$$L(\alpha) = (y - \Phi^T \Phi \alpha)^T (y - \Phi^T \Phi \alpha) + \lambda \alpha^T \Phi^T \Phi \alpha$$

$\boxed{\text{Gram or Kernel Matrix } \Phi^T \Phi = K}$

$$L(\alpha) = (y - K\alpha)^T (y - K\alpha) + \lambda \alpha^T K \alpha$$

$$\frac{\partial L(\alpha)}{\partial \alpha} = 0 \Rightarrow \quad K(K + \lambda I)\alpha = Ky$$

If K is positive definite, $\boxed{\alpha = (K + \lambda I)^{-1} y}$

Equivalent to doing linear ridge regression with $x' \in \Re^n$

$$x' = [k(x, x_i), \ldots, k(x, x_n)]^T$$

so $\Phi = K$

One difference: regularizer will be $\alpha^T \alpha$ instead of $\alpha^T K \alpha$

$\underbrace{\hphantom{One difference: regularizer will be}}_{\text{Empirical Kernel Map}}$

Example - Nonlinear (Kernel) Regression

Given: A labeled training set, $\{x_i, y_i\}_{i=1...n} \quad x_i \in \Re^d, y_i \in \Re$

$$y = w^T \Phi(x) = \alpha^T \Phi^T \Phi(x) = \boxed{\sum_{i=1}^n \alpha_i k(x, x_i)}$$

Estimating α

$$L(\alpha) = (y - \Phi^T \Phi \alpha)^T (y - \Phi^T \Phi \alpha) + \lambda \alpha^T \Phi^T \Phi \alpha$$

Gram or Kernel Matrix $\Phi^T \Phi = K$

$$L(\alpha) = (y - K\alpha)^T (y - K\alpha) + \lambda \alpha^T K \alpha$$

$$\frac{\partial L(\alpha)}{\partial \alpha} = 0 \Rightarrow \quad K(K + \lambda I)\alpha = Ky$$

If K is positive definite, $\boxed{\alpha = (K + \lambda I)^{-1} y}$

Equivalent to doing linear ridge regression with $x' \in \Re^n$

$$x' = [k(x, x_i), ..., k(x, x_n)]^T$$

so $\Phi = K$

One difference: regularizer will be $\alpha^T \alpha$ instead of $\alpha^T K \alpha$

Empirical Kernel Map

Advantage of Kernel View

- Original data is not needed directly, we only need $k(x, y)$ for any pair
- Original data does not need to be a vector, only $k(x, y)$ should be defined

Example - Nonlinear (Kernel) Regression

Training

$$\boxed{\alpha = (\underbrace{K}_{O(n^2 d)} + \lambda I)^{-1} y}$$

$$\underbrace{(K + \lambda I)^{-1}}_{O(n^3)}$$

Number of parameters same as number of points!

$$n \sim O(100M), d \sim O(100K)$$

K ~ 40,000 TB!

Building K and its inversion is intractable!

Approximations, first-order optimization ?

Example - Nonlinear (Kernel) Regression

Training

$$\boxed{\alpha = \underbrace{\underbrace{(K + \lambda I)}_{O(n^2 d)}{}^{-1}}_{O(n^3)} y}$$

Number of parameters same as number of points!

$$n \sim O(100M), d \sim O(100K)$$

K ~ 40,000 TB!

Building K and its inversion is intractable!

Approximations, first-order optimization ?

Testing

$$\boxed{y = \sum_{i=1}^{n} \alpha_i k(x, x_i)}$$

Grows linearly with n

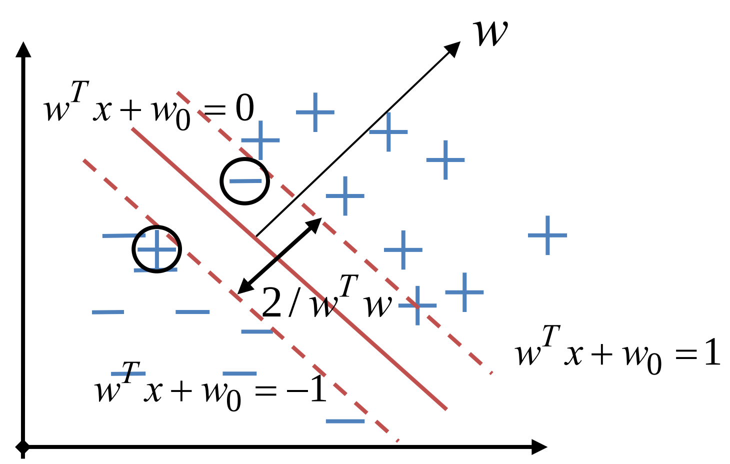
Too slow for most practical purposes

Need to induce sparsity in α - L_1 prior   Sparse kernel machines

Support Vector Machine (SVM)

Given a labeled training set, $\{x_i, y_i\}_{i=1\ldots n} \quad x_i \in \Re^d, y_i \in \{-1, 1\}$

Want to learn $f(x;w) = \mathrm{sgn}(w^T x + w_0)$



Primal $\quad \min \; w^T w + C\sum_i \xi_i$

$$\text{s.t.} \quad y_i(w^T x_i + w_0) \geq 1 - \xi_i \qquad \forall\, i$$

$$\xi_i \geq 0$$

Using Lagrange multipliers (with KKT conditions)

$$w = \sum_i \alpha_i y_i x_i$$

Dual $\quad \max \; \sum_{i=1}^{n} \alpha_i - \sum_{i,j}^{n} \alpha_i (y_i y_j x_i^T x_j) \alpha_j$

$$\sum_i \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C$$

Fast training in $O(nd)$

- cutting-plane
- stochastic gradient descent
- quasi-Newton
- coordinate descent

Testing $O(d)$

Kernel SVM

Given a labeled training set, $\{x_i, y_i\}_{i=1\ldots n} \quad x_i \in \Re^d, y_i \in \{-1, 1\}$

$$f(x\,; w) = \text{sgn}(w^T \Phi(x) + w_0) \qquad k(x, y) = \Phi(x).\Phi(y)$$

Cannot solve in primal since $\Phi(x)$ is unknown!

$$w = \sum_i \alpha_i y_i \Phi(x_i)$$

Note: Can be solved in primal if kernel SVM viewed as optimizing "regularized hinge loss" with empirical kernel map

Dual $\quad \max \ \alpha^T 1 - \alpha^T K' \alpha$

$$\alpha^T y = 0$$

$$0 \le \alpha_i \le C$$

$K' = [y_i y_j k(x_i, x_j)]_{i,j=1,\ldots,n}$

Training $\quad O(n^2) \sim O(n^3)$

Testing $\quad O(\#_{sv}) \approx O(n)$

$$\boxed{f(x\,; \alpha) = \text{sgn}(\sum_i \alpha_i y_i k(x, x_i) + \alpha_0)}$$

How to do fast training and testing ?

Approximations

1. Subsample the data
   - Randomly pick a small number of points $p << n$

$$y = \sum_{i=1}^{n} \alpha_i k(x, x_i) \approx \sum_{i=1}^{p} \alpha_i k(x, x_i)$$

   - Training: $O(npd)$ Testing: $O(pd)$
   - Better sampling for specific applications, e.g., kernel/logistic regression
     - Find p centers in the data using e.g., k-medoid
     - Use random-projection based clustering for large d
   - Selective sampling in some cases
     - Greedily pick points from the whole set based on a given criterion

Approximations

1. Subsample the data
   - Randomly pick a small number of points $p << n$

$$y = \sum_{i=1}^{n} \alpha_i k(x, x_i) \approx \sum_{i=1}^{p} \alpha_i k(x, x_i)$$

   - Training: $O(npd)$ Testing: $O(pd)$
   - Better sampling for specific applications, e.g., kernel/logistic regression
     - Find p centers in the data using e.g., k-medoid
     - Use random-projection based clustering for large d
   - Selective sampling in some cases
     - Greedily pick points from the whole set based on a given criterion
2. Low-rank approximation of kernel matrix
   - Use sampling-based methods
   - Incomplete Cholesky
3. Sparsification of kernel matrix
   - Make the kernel matrix sparse by thresholding the entries

Approximations

4. Approximate kernel matrix-vector product
   - E.g., using ANN (kd-trees)

5. Fast Optimization Methods
   - Many methods proposed for specific techniques e.g., SVM
   - Decomposition methods
     - Block coordinate-descent → slow beyond O(100K) points
   - Stochastic or online methods

6. Kernel Approximation
   - Instead of matrix speed-up, approximate kernel function directly
   - Some kernels can be computed fast fairly accurately
     - Fast Gauss Transform: Hermite or Taylor approximation of Gaussian kernels
   - Approximate linearization of kernels
     - Linear methods very fast to train and test
     - Possible for certain types of kernels

Kernel Linearization

Approximate linearization possible using empirical kernel map

$$x' = [k(x, x_i), \ldots, k(x, x_n)]^T$$

- But no gain since it is n-dim vector and requires n kernel computations

Kernel Linearization

Approximate linearization possible using empirical kernel map

$$x' = [k(x, x_i), \ldots, k(x, x_n)]^T$$

- But no gain since it is n-dim vector and requires n kernel computations

Can we approximate the feature map with a low-dim vector ?

Kernel Linearization $k(x, y) = \Phi(x)^T \Phi(y) \approx z(x)^T \underbrace{z(y)}_{\in \Re^D,\ D << n}$

Kernel Linearization

Approximate linearization possible using empirical kernel map

$$x' = [k(x, x_i), \ldots, k(x, x_n)]^T$$

- But no gain since it is n-dim vector and requires n kernel computations

Can we approximate the feature map with a low-dim vector ?

Kernel Linearization $k(x, y) = \Phi(x)^T \Phi(y) \approx z(x)^T \underbrace{z(y)}_{\in \Re^D, \, D << n}$

Suppose the kernel is shift-invariant:

$$k(x, y) = k'(x - y) = k'(\Delta)$$

Gaussian

$$k(x, y) = \exp\{-\|x - y\|_2^2 / 2\sigma^2\}$$
$$k'(\Delta) = \exp\{-\|\Delta\|_2^2 / 2\sigma^2\}$$

Laplacian

$$k(x, y) = \exp\{-\|x - y\|_1 / \lambda\}$$
$$k'(\Delta) = \exp\{-\|\Delta\|_1 / \lambda\}$$

Random Fourier Features

$$z(x) = [z_j(x)]_{D\times 1}$$

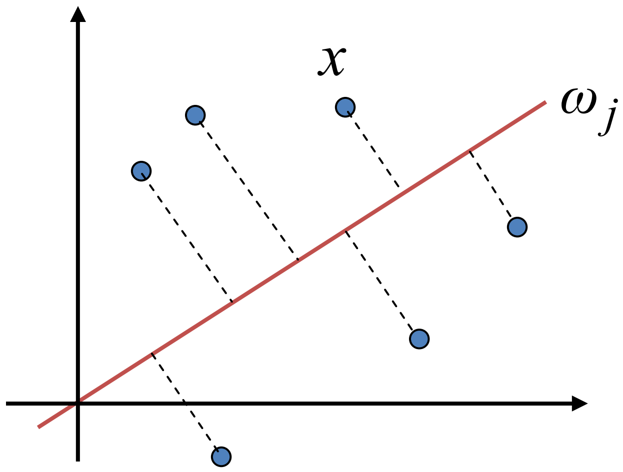
$$\boxed{z_j(x) = \sqrt{2/D}\cos(\omega_j x + b)} \quad \omega_j \sim P(\omega) \quad b \sim U(0, 2\pi)$$

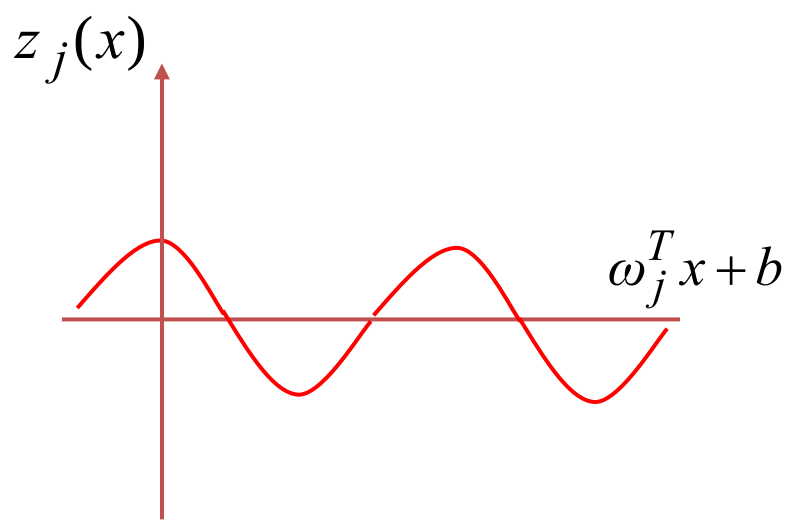
Random Fourier Features

$$z(x) = [z_j(x)]_{D \times 1}$$

$$\boxed{z_j(x) = \sqrt{2/D}\cos(\omega_j x + b)} \quad \omega_j \sim P(\omega) \quad b \sim U(0, 2\pi)$$

Gaussian $\omega_{jk} \sim N(0,1)$ Laplacian $\omega_{jk} \sim Cauchy(0,1)$





Main Theory

A continuous shift-invariant kernel is positive definite if and only if $k'(\Delta)$ is the Fourier transform of a non-negative measure. [Bochner]

$$k'(x-y) = \int p(\omega) e^{j\omega^T(x-y)} d\omega$$

$p(\omega)$ - Inverse Fourier Transform of $k'(\Delta)$

Main Theory

A continuous shift-invariant kernel is positive definite if and only if $k'(\Delta)$ is the Fourier transform of a non-negative measure. [Bochner]

$$k'(x-y) = \int p(\omega) e^{j\omega^T (x-y)} d\omega$$

- since $k'(.)$ and $p(.)$ both are real, use real part of complex exponentials

$$k(x,y) = E\left[z_\omega(x).z_\omega(y)\right] \text{ if } z_\omega(x) = \sqrt{2}\cos(\omega^T x + b)$$

- Reduce variance by concatenating many (D) dimensions in $z_\omega(.)$

$$z_\omega(x)^T z_\omega(y) = (1/D)\sum_{j=1}^{D} z_{\omega_j}(x) z_{\omega_j}(y) \qquad z_{\omega_j}(x) = \sqrt{(2/D)}\cos(\omega^T x + b)$$

Main Theory

A continuous shift-invariant kernel is positive definite if and only if $k'(\Delta)$ is the Fourier transform of a non-negative measure. [Bochner]

$$k'(x-y) = \int p(\omega) e^{j\omega^T (x-y)} d\omega$$

- since $k'(.)$ and $p(.)$ both are real, use real part of complex exponentials

$$k(x,y) = E[z_\omega(x).z_\omega(y)] \;\text{ if }\; z_\omega(x) = \sqrt{2}\cos(\omega^T x + b)$$

- Reduce variance by concatenating many (D) dimensions in $z_\omega(.)$

$$z_\omega(x)^T z_\omega(y) = (1/D)\sum_{j=1}^{D} z_{\omega_j}(x) z_{\omega_j}(y) \qquad z_{\omega_j}(x) = \sqrt{(2/D)}\cos(\omega^T x + b)$$

Hoeffding Bound $\quad \Pr(|z(x)^T z(y) - k(x,y)| \geq \varepsilon) \leq 2\exp(-D\varepsilon^2/4)$

Example Results

Regression and Classification errors

Training $\min_{w} \left(\left\| Z^T w - y \right\|_2^2 + \lambda \left\| w \right\|_2^2 \right)$ Testing $f(x) = w^T z(x)$

Dataset	Fourier+LS	CVM	Exact SVM
CPU regression 6500 instances 21 dims	3.6% 20 secs $D = 300$	5.5% 51 secs	11% 31 secs ASVM
Census regression 18,000 instances 119 dims	5% 36 secs $D = 500$	8.8% 7.5 mins	9% 13 mins SVMTorch
Adult classification 32,000 instances 123 dims	14.9% 9 secs $D = 500$	14.8% 73 mins	15.1% 7 mins $\text{SVM}^{\text{light}}$
Forest Cover classification 522,000 instances 54 dims	11.6% 71 mins $D = 5000$	2.3% 7.5 hrs	2.2% 44 hrs libSVM
KDDCUP99 (see footnote) classification 4,900,000 instances 127 dims	7.3% 1.5 min $D = 50$	6.2% (18%) 1.4 secs (20 secs)	8.3% < 1 s SVM+sampling

Rahimi & Recht [7]

Learning Low Dimensional Features

Instead of randomization, can we learn low-dim features directly?

Key Idea: project (high-dim) implicit features $\Phi(x)$ on D basis vectors

Learning Low Dimensional Features

Instead of randomization, can we learn low-dim features directly?

Key Idea: project (high-dim) implicit features $\Phi(x)$ on D basis vectors

Given a set of basis vectors $\{h_i\}_{i=1..,D}$ $h_i \in \Re^d$ and $\{\Phi(h_i)\}_{i=1..,D}$

Low-dim representation using implicit feature space

$$\hat{v}_x = \arg\min_{v_x} \|\Phi(x) - Hv_x\|^2 \qquad H = [\Phi(h_1),...\Phi(h_D)]$$

$$= (H^T H)^{-1}(H^T \Phi(x))$$

Learning Low Dimensional Features

Instead of randomization, can we learn low-dim features directly?

Key Idea: project (high-dim) implicit features $\Phi(x)$ on D basis vectors

Given a set of basis vectors $\{h_i\}_{i=1..,D}$ $h_i \in \Re^d$ and $\{\Phi(h_i)\}_{i=1..,D}$

Low-dim representation using implicit feature space

$$\hat{v}_x = \arg\min_{v_x} \left\| \Phi(x) - H v_x \right\|^2 \qquad H = [\Phi(h_1),\ldots\Phi(h_D)]$$

$$= (H^T H)^{-1} (H^T \Phi(x))$$

To approximate kernel $k(x,y) = \Phi(x)^T \Phi(y) \approx (Hv_x)^T (Hv_y) = v_x^T \underbrace{H^T H}_{D\times D} v_y$

$$= (H^T \Phi(x))^T (H^T H)^{-1} (H^T \Phi(y))$$

$$= (k_h(x))^T K_{hh}^{-1} (k_h(y))$$

$$k_h(x) = [k(h_1,x),\ldots k(h_D,x)]^T$$
$$K_{hh} = [k(h_i,h_j)]_{i,j=1,\ldots,D}$$
$$K_{hh}^{-1} = G^T G$$

Learning Low Dimensional Features

Instead of randomization, can we learn low-dim features directly?

Key Idea: project (high-dim) implicit features $\Phi(x)$ on D basis vectors

Given a set of basis vectors $\{h_i\}_{i=1..,D}$ $h_i \in \Re^d$ and $\{\Phi(h_i)\}_{i=1..,D}$

Low-dim representation using implicit feature space

$$\hat{v}_x = \arg\min_{v_x} \|\Phi(x) - Hv_x\|^2 \qquad H = [\Phi(h_1), \ldots \Phi(h_D)]$$

$$= (H^T H)^{-1}(H^T \Phi(x))$$

To approximate kernel $k(x, y) = \Phi(x)^T \Phi(y) \approx (Hv_x)^T (Hv_y) = v_x^T \underbrace{H^T H}_{D \times D} v_y$

$$= (H^T \Phi(x))^T (H^T H)^{-1}(H^T \Phi(y))$$

$$= (k_h(x))^T K_{hh}^{-1}(k_h(y))$$

$$k_h(x) = [k(h_1, x), \ldots k(h_D, x)]^T$$

$$K_{hh} = [k(h_i, h_j)]_{i,j=1,\ldots,D}$$

$$K_{hh}^{-1} = G^T G$$

Desired linearization $\boxed{z(x) = Gk_h(x)}$

How to get h's ?

Learning Low Dimensional Features

Learning the basis vectors using a few sampled points

$$\hat{h}, \hat{v} = \arg\min_{h,v} \sum_{i=1}^{p} \left\| \Phi(x_i) - Hv_i \right\|^2$$

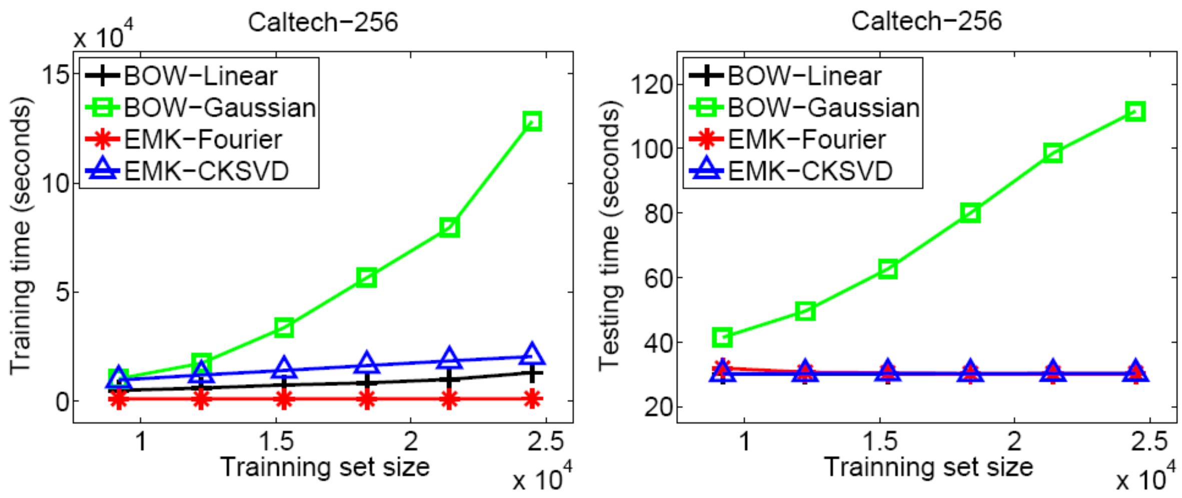
Given H, $\hat{v}_i = (H^T H)^{-1}(H^T \Phi(x_i))$

$$\boxed{\arg\min_{h} [-\sum_{i=1}^{p} (k_h(x_i))^T K_{hh}^{-1} (k_h(x_i))]}$$

Use Stochastic Gradient Descent to obtain $\{h_i\}$

Experiment

d = 1000, 256-class classification



Randomized Learned

Algorithms	BOW-Linear	BOW-Gaussian	EMK-Fourier	EMK-CKSVD
15 training	17.4±0.7	19.1±0.8	22.6±0.7	23.2±0.6
30 training	22.7±0.4	24.4±0.6	30.1±0.5	30.5±0.4
45 training	26.9±0.3	28.3±0.5	34.1±0.5	34.4±0.4
60 training	29.3±0.6	30.9±0.4	37.4±0.6	37.6±0.5

D = 1000

Bo & Sminchisescu [10]

Linearization of Additive Kernels

Additive homogeneous kernels for $x, y \in \Re^d$ defined as,

$$k(x,y) = \sum_{j=1}^{d} k_d(x_j, y_j) \quad \text{suppose } x_j, y_j \geq 0, \;\; \forall\, j$$

$$k_d(ca, cb) = c k_d(a,b) \qquad a, b \text{ are scalars}$$

- Intersection kernel $k(x,y) = \sum_{j=1}^{d} \min(x_j, y_j)$
- Bhattacharya (Hellinger) kernel $k(x,y) = \sum_{j=1}^{d} \sqrt{x_j y_j}$
- Chi-square kernel $k(x,y) = \sum_{j=1}^{d} x_j y_j / (x_j + y_j)$

Linearization of Additive Kernels

Additive homogeneous kernels for $x, y \in \Re^d$ defined as,

$$k(x,y) = \sum_{j=1}^{d} k_d(x_j, y_j) \quad \text{suppose } x_j, y_j \geq 0,\ \forall\ j$$

$$k_d(ca, cb) = c k_d(a,b) \qquad a, b \text{ are scalars}$$

- Intersection kernel $k(x,y) = \sum_{j=1}^{d} \min(x_j, y_j)$
- Bhattacharya (Hellinger) kernel $k(x,y) = \sum_{j=1}^{d} \sqrt{x_j y_j}$
- Chi-square kernel $k(x,y) = \sum_{j=1}^{d} x_j y_j / (x_j + y_j)$

Signature of a homogeneous kernel

$$k_d(a,b) = k_d\left(\sqrt{ab}\sqrt{\frac{a}{b}}, \sqrt{ab}\sqrt{\frac{b}{a}}\right) = \sqrt{ab}\, k_d\left(\sqrt{\frac{a}{b}}, \sqrt{\frac{b}{a}}\right) = \sqrt{ab}\, \mathbf{K}\left(\log\frac{b}{a}\right)$$

Kernel Signature $\mathbf{K}(\omega) = k_d(e^{-\omega/2}, e^{\omega/2}),\ \ \omega = \log\frac{b}{a}$

Linearization of Additive Kernels

Signature for homogeneous kernels can be written as Fourier Transform,

$$\mathbf{K}(\omega) = \int_{-\infty}^{\infty} e^{-i\lambda\omega} \kappa(\lambda) d\lambda$$

$$k_d(a,b) = \Psi(a)^T \Psi(b) = \int_{-\infty}^{+\infty} [\Psi(a)]_\lambda^* [\Psi(b)]_\lambda \, d\lambda \qquad k_d(a,b) = \sqrt{ab}\,\mathbf{K}(\log\frac{b}{a})$$

Linearization of Additive Kernels

Signature for homogeneous kernels can be written as Fourier Transform,

$$\mathbf{K}(\omega) = \int_{-\infty}^{\infty} e^{-i\lambda\omega}\kappa(\lambda)d\lambda$$

$$k_d(a,b) = \Psi(a)^T\Psi(b) = \int_{-\infty}^{+\infty}[\Psi(a)]_\lambda^*[\Psi(b)]_\lambda d\lambda$$

$$k_d(a,b) = \sqrt{ab}\,\mathbf{K}(\log\frac{b}{a})$$

infinite dimensional vector $[\Psi(a)]_\lambda = e^{-i\lambda\log a}\sqrt{a\kappa(\lambda)}$

inverse Fourier Transform $\kappa(\lambda) = (1/2\pi)\int_{-\infty}^{\infty} e^{i\lambda\omega}\mathbf{K}(\omega)d\omega$

Linearization of Additive Kernels

Signature for homogeneous kernels can be written as Fourier Transform,

$$\mathbf{K}(\omega) = \int_{-\infty}^{\infty} e^{-i\lambda\omega} \kappa(\lambda) d\lambda$$

$$k_d(a,b) = \Psi(a)^T \Psi(b) = \int_{-\infty}^{+\infty} [\Psi(a)]_\lambda^* [\Psi(b)]_\lambda d\lambda$$

$$k_d(a,b) = \sqrt{ab}\, \mathbf{K}(\log \frac{b}{a})$$

infinite dimensional vector $[\Psi(a)]_\lambda = e^{-i\lambda \log a} \sqrt{a\kappa(\lambda)}$

inverse Fourier Transform $\kappa(\lambda) = (1/2\pi) \int_{-\infty}^{\infty} e^{i\lambda\omega} \underbrace{\mathbf{K}(\omega)}_{} d\omega$

can be computed explicitly for many kernels

How to get finite linear map?

Use finite number of samples with a certain period – determined empirically

Example Kernels

kernel	$k(x,y)$	$\mathcal{K}(\omega)$	$\kappa(\lambda)$
Hellinger's	$\sqrt{xy}$	1	$\delta(\lambda)$
χ^2	$2\frac{xy}{x+y}$	$\operatorname{sech}(\omega/2)$	$\operatorname{sech}(\pi\lambda)$
intersection	$\min\{x,y\}$	$e^{-\lvert\omega\rvert/2}$	$\frac{2}{\pi}\frac{1}{1+4\lambda^2}$

Vedaldi & Zisserman [11]

Experiment

n = 1500, d = 1200, 101-class classification

mthd.	dm.	χ^2 kernel acc.	χ^2 kernel time	inters. kernel acc.	inters. kernel time
kernel	–	$64.2_{\pm 1.7}$	$388.4_{\pm 8.7}$	$62.2_{\pm 1.8}$	$354.7_{\pm 24.4}$
appr.	1	$62.4_{\pm 1.6}$	$\mathbf{20.7}_{\pm 0.3}$	$62.0_{\pm 1.4}$	$22.9_{\pm 0.7}$
appr.	3	$64.2_{\pm 1.5}$	$58.4_{\pm 7.2}$	$63.9_{\pm 1.2}$	$66.5_{\pm 2.3}$
appr.	5	$64.0_{\pm 1.6}$	$101.3_{\pm 0.7}$	$64.0_{\pm 1.7}$	$105.8_{\pm 6.5}$

Vedaldi & Zisserman [11]

Linearization of Intersection Kernel

Intersection kernel

$$k(x,y) = \sum_{j=1}^{d} \min(x_j, y_j) \qquad x_j, y_j \in [0,1]$$

normalized feature vectors

$$\min(x_j, y_j) \approx \Phi(x_j)^T \Phi(y_j)$$

$$\Phi(a) = \sqrt{1/N}\, \underbrace{U(Na)}_{\text{pseudo-binary representation}}$$

Example $N = 10, a = 0.25, U(Na)$

$$U(Na) = U(2.5) = [1, 1, 0.5, 0, 0, \ldots, 0]^T$$

For high accuracy, N should be large

Issue: Original dimension gets blown by a factor of N

Experiment

Encoding	Training Algorithm	15 examples	
		Training Time(s)	Accuracy(%)
identity	LIBLINEAR	**18.57 (0.87)**	41.19 (0.94)
identity	LIBSVM (int kernel)	844.13 (2.10)	**50.15 (0.61)**
snow=ϕ_1	LIBLINEAR	45.22 (1.17)	46.02 (0.58)
ϕ_2	LIBLINEAR	42.31 (1.43)	48.70 (0.61)
ϕ_2	PWLSGD	238.98 (2.49)	**49.89 (0.45)**

Maji & Berg [9]

Prediction with Intersection Kernel

SVM prediction $f(x) = \text{sgn}[\sum_{i=1}^{m} \alpha_i y_i k(x, x_i) + \alpha_0]$ sum is over m support vectors

$O(md)$

Intersection kernel $k(x, v) = \sum_{j=1}^{d} \min(x(j), v(j))$

Prediction with Intersection Kernel

SVM prediction $f(x) = \mathrm{sgn}[\sum_{i=1}^{m} \alpha_i y_i k(x, x_i) + \alpha_0]$ sum is over m support vectors

$O(md)$

Intersection kernel $k(x, v) = \sum_{j=1}^{d} \min(x(j), v(j))$

$$f(x) = \mathrm{sgn}[\sum_{i=1}^{m} \alpha_i y_i \sum_{j=1}^{d} \min(x(j), x_i(j)) + \alpha_0]$$

$$f(x) = \mathrm{sgn}[\sum_{j=1}^{d} \sum_{i=1}^{m} \alpha_i y_i \min(x(j), x_i(j)) + \alpha_0]$$ swap summation

Prediction with Intersection Kernel

SVM prediction $f(x) = \mathrm{sgn}[\sum_{i=1}^{m} \alpha_i y_i k(x, x_i) + \alpha_0]$ sum is over m support vectors

$O(md)$

Intersection kernel $k(x, v) = \sum_{j=1}^{d} \min(x(j), v(j))$

$$f(x) = \mathrm{sgn}[\sum_{i=1}^{m} \alpha_i y_i \sum_{j=1}^{d} \min(x(j), x_i(j)) + \alpha_0]$$

$$f(x) = \mathrm{sgn}[\sum_{j=1}^{d} \underbrace{\sum_{i=1}^{m} \alpha_i y_i \min(x(j), x_i(j))}_{f_j(x(j))} + \alpha_0]$$ swap summation

$$f_j(s) = \sum_{i=1}^{m} \alpha_i y_i \min(s, x_i(j))$$

Prediction with Intersection Kernel

SVM prediction $f(x) = \text{sgn}[\sum_{i=1}^{m} \alpha_i y_i k(x, x_i) + \alpha_0]$ sum is over m support vectors

$O(md)$

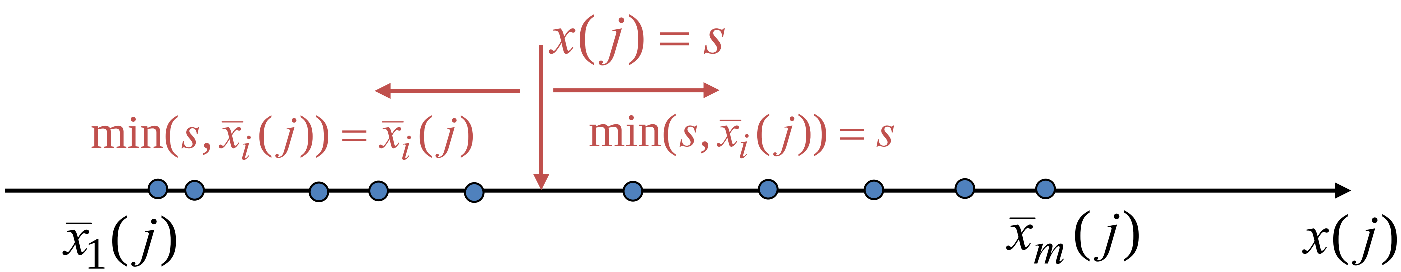
Intersection kernel $k(x, v) = \sum_{j=1}^{d} \min(x(j), v(j))$

$$f(x) = \text{sgn}[\sum_{i=1}^{m} \alpha_i y_i \sum_{j=1}^{d} \min(x(j), x_i(j)) + \alpha_0]$$

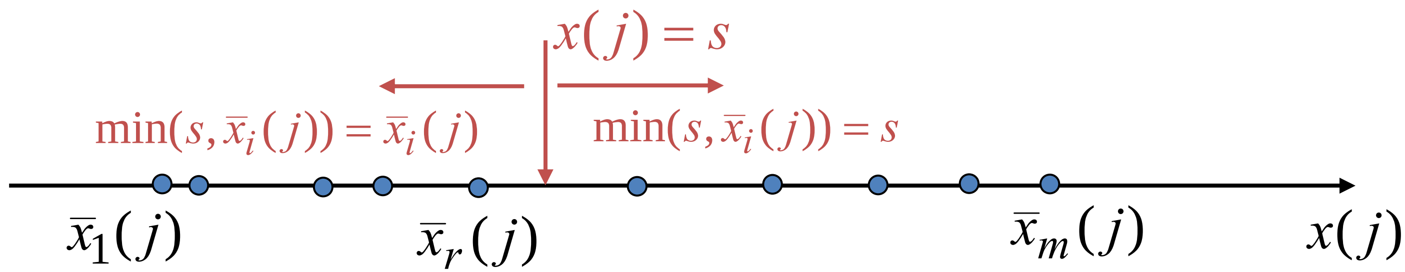
$$f(x) = \text{sgn}[\sum_{j=1}^{d} \underbrace{\sum_{i=1}^{m} \alpha_i y_i \min(x(j), x_i(j))}_{f_j(x(j))} + \alpha_0]$$ swap summation

$$f_j(s) = \sum_{i=1}^{m} \alpha_i y_i \min(s, x_i(j))$$

sort the j^{th} dim of all support vectors



Prediction with Intersection Kernel

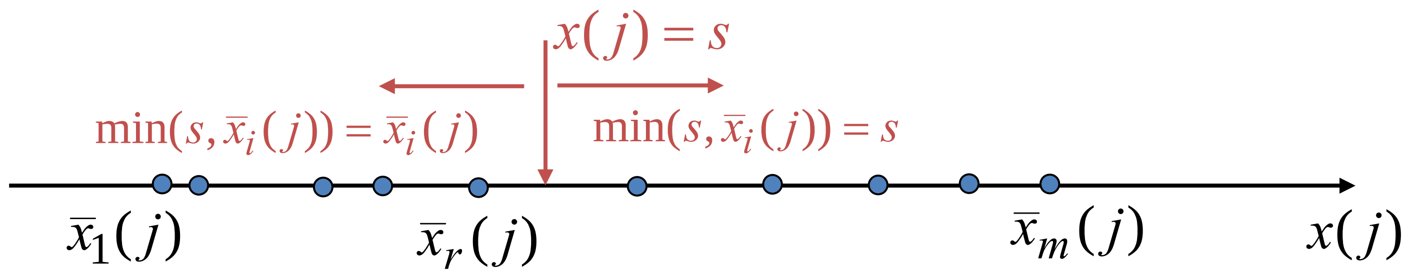


Suppose r is the largest integer such that $\bar{x}_r(j) \le s$

$$f_j(s) = \sum_{i=1}^{m} \bar{\alpha}_i \bar{y}_i \min(s, \bar{x}_i(j))$$

$$= \sum_{1 \le i \le r} \bar{\alpha}_i \bar{y}_i \bar{x}_i(j) + s \sum_{r < i \le m} \bar{\alpha}_i \bar{y}_i \; = A(r) + sB(r)$$

Prediction with Intersection Kernel



Suppose r is the largest integer such that $\bar{x}_r(j) \le s$

$$f_j(s) = \sum_{i=1}^{m} \bar{\alpha}_i \bar{y}_i \min(s, \bar{x}_i(j))$$

$$= \sum_{1 \le i \le r} \bar{\alpha}_i \bar{y}_i \bar{x}_i(j) + s \sum_{r < i \le m} \bar{\alpha}_i \bar{y}_i \;\; = A(r) + sB(r)$$

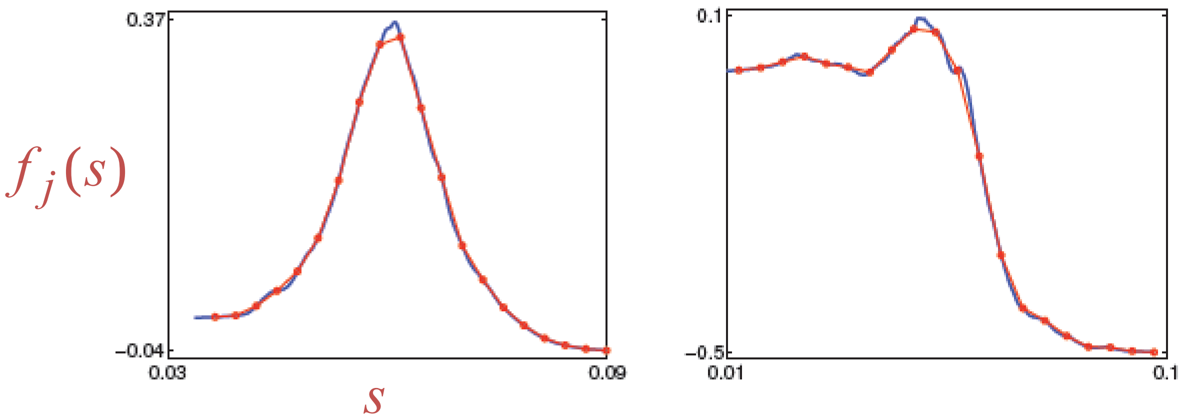
Simple procedure:

1. Sort each dimension of m support vectors – can be done offline
2. For each test point, find the location of its j^th-dim value in the j^th-sorted list using binary search $O(\log m)$
3. Keep cumulative sum of $\sum_{i=1}^{r} \bar{\alpha}_i \bar{y}_i \bar{x}_i(j)$ and $\sum_{i=r+1}^{m} \bar{\alpha}_i \bar{y}_i$ $2m$ extra storage

Time complexity $O(d \log m)$ instead of $O(d\, m)$ Exact computation !

Approximate Prediction

Key Idea: Instead of keeping m values of $A(r)$ and $B(r)$, store values at much reduced (equidistant) locations and make piecewise linear or piecewise constant approximation



Traditional function approximation: Can be done for any univariate function

Time complexity $O(d)$ instead of $O(d\,m)$ Approximate computation !

Maji et al. [8]

Experiment

Dataset	Model parameters		SVM kernel type		fast IKSVMs		
	#SVs	#features	linear	intersection	binary search	piecewise-const	piecewise-lin
INRIA Ped	3363	1360	0.07 ± 0.00	659.1 ± 1.92	2.57 ± 0.03	0.34 ± 0.01	0.43 ± 0.01
DC Ped	5474 ± 395	656	0.03 ± 0.00	459.1 ± 31.3	1.43 ± 0.02	0.18 ± 0.01	0.22 ± 0.00
Caltech 101	175 ± 46	1360	0.07 ± 0.01	24.77 ± 1.22	1.63 ± 0.12	0.33 ± 0.03	0.46 ± 0.03

m d

Exact methods

100 knots

30 knots

Maji et al. [8]

References

1. G. Wahba, Spline Models for Observational Data. SIAM, 1990.
2. B. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifiers," Computational Learning Theory, 1992.
3. C. Cortes and V. Vapnik, "Support Vector Networks," Machine Learning, 1995.
4. B. Scholkopf and A. Smola, "Learning with Kernels", MIT Press, Cambridge, MA, 2002.
5. M. Hein and O. Bousquet, "Hilbertian Metrics and Positive Definite Kernels on Probability Measures," AISTAT, 2005.
6. *Large Scale Kernel Machines*, Edited by L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, Neural Information Processing Series, MIT Press, Cambridge, MA., 2007.
7. A. Rahimi & B. Recht, "Random Features for Large-Scale Kernel Machines," NIPS, 2007.
8. S. Maji, A. Berg & J. Malik, "Classification using Intersection Kernel Support Vector Machines is Efficient", CVPR 2008.
9. S. Maji & A. Berg, "Max-Margin Additive Classifiers for Detection ", ICCV, 2009.
10. L. Bo & C. Sminchisescu, "Efficient Match Kernels between Sets of Features for Visual Recognition," NIPS 2009.
11. A. Vedaldi & A. Zisserman, "Efficient Additive Kernels via Explicit Feature Maps", CVPR 2010.