Machine Learning

CSE 6363 (Fall 2016)

Lecture 10 Kernel Learning and SVR

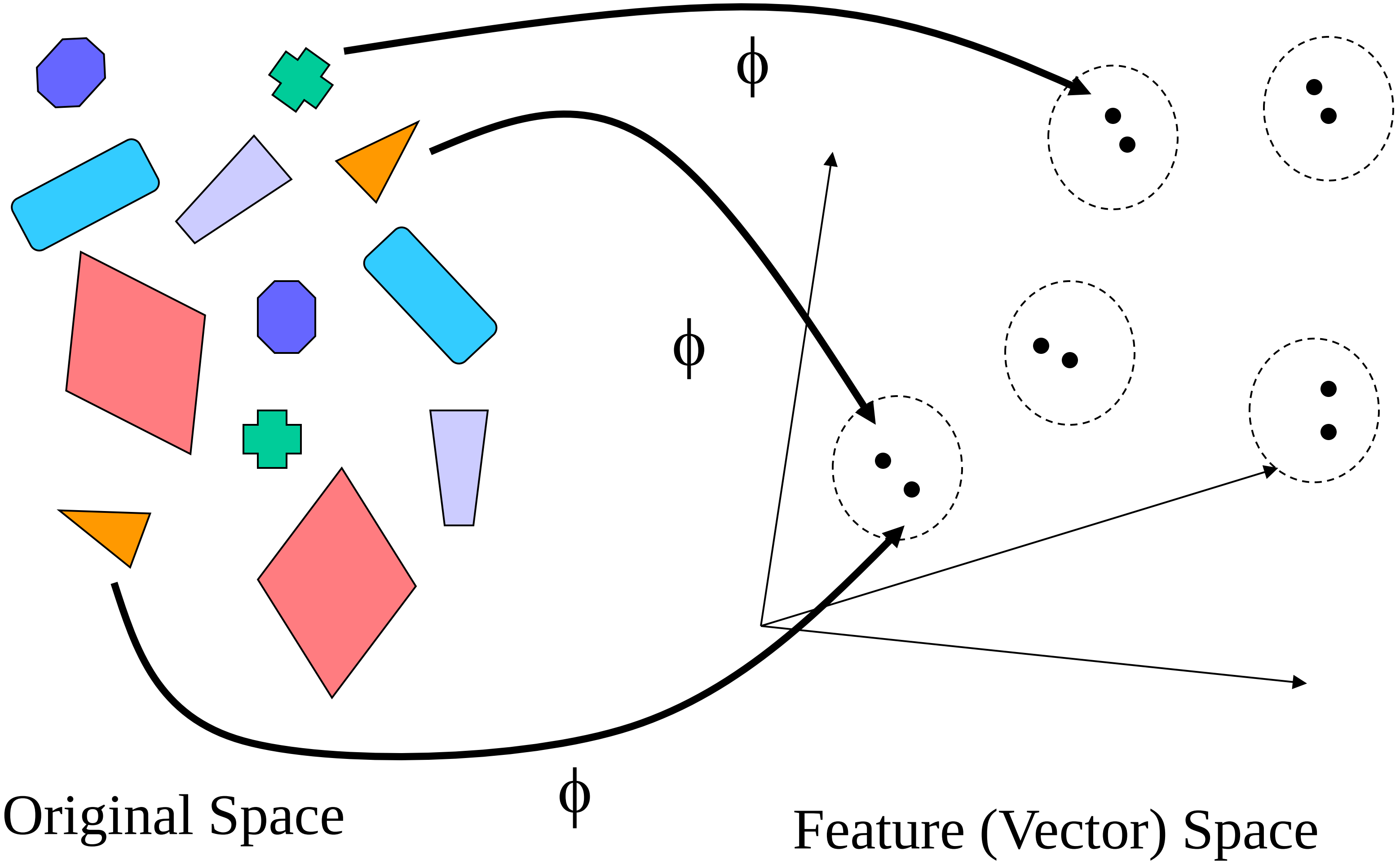
Heng Huang, Ph.D.

Department of Computer Science and Engineering

Kernel Methods : Intuitive Idea

- Find a mapping ϕ such that, in the new space, problem solving is easier (e.g. linear)
- The *kernel* represents the similarity between two objects (documents, terms, …), defined as the dot-product in this new vector space
- But the mapping is left implicit
- Easy generalization of a lot of dot-product (or distance) based pattern recognition algorithms

Kernel Methods : The Mapping



Benefits from kernels

- Generalizes (nonlinearly) pattern recognition algorithms in clustering, classification, density estimation, …
  - When these algorithms are dot-product based, by replacing the dot product $(x \bullet y)$ by $k(x,y)=\phi(x) \bullet \phi(y)$

    e.g.: linear discriminant analysis, logistic regression, perceptron, SOM, PCA, ICA, …

    This often implies to work with the "dual" form of the algo.
  - When these algorithms are distance-based, by replacing $d(x,y)$ by $k(x,x)+k(y,y)-2k(x,y)$
- Freedom of choosing ϕ implies a large variety of learning algorithms

Kernel Function in Linear Regression Model

regularized sum-of-squares error function

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) - t_n\right\}\phi(\mathbf{x}_n) = \sum_{n=1}^{N} a_n\phi(\mathbf{x}_n) = \mathbf{\Phi}^{\mathrm{T}}\mathbf{a}$$

$\mathbf{\Phi}$ is the design matrix, whose n^{th} row is given by $\phi(\mathbf{x}_n)^{\mathrm{T}}$.

$$\mathbf{a} = (a_1, \ldots, a_N)^{\mathrm{T}}$$

$$a_n = -\frac{1}{\lambda}\left\{\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) - t_n\right\}$$

substitute $\mathbf{w} = \mathbf{\Phi}^{\mathrm{T}}\mathbf{a}$ into $J(\mathbf{w})$

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{a}$$

define the *Gram* matrix $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}$

$$K_{nm} = \phi(\mathbf{x}_n)^{\mathrm{T}}\phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m)$$

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}.$$

Setting the gradient of $J(\mathbf{a})$ with respect to $\mathbf{a}$ to zero

$$\mathbf{a} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}.$$

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}) = \mathbf{a}^{\mathrm{T}}\mathbf{\Phi}\phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^{\mathrm{T}}\left(\mathbf{K} + \lambda\mathbf{I}_N\right)^{-1}\mathbf{t}$$

Constructing Kernels

- Another powerful technique is to build them out of simpler kernels

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$$k(\mathbf{x}, \mathbf{x}') = c k_1(\mathbf{x}, \mathbf{x}') \quad (6.13)$$

$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') \quad (6.14)$$

$$k(\mathbf{x}, \mathbf{x}') = q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \quad (6.15)$$

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \quad (6.16)$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \quad (6.17)$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') \quad (6.18)$$

$$k(\mathbf{x}, \mathbf{x}') = k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \quad (6.19)$$

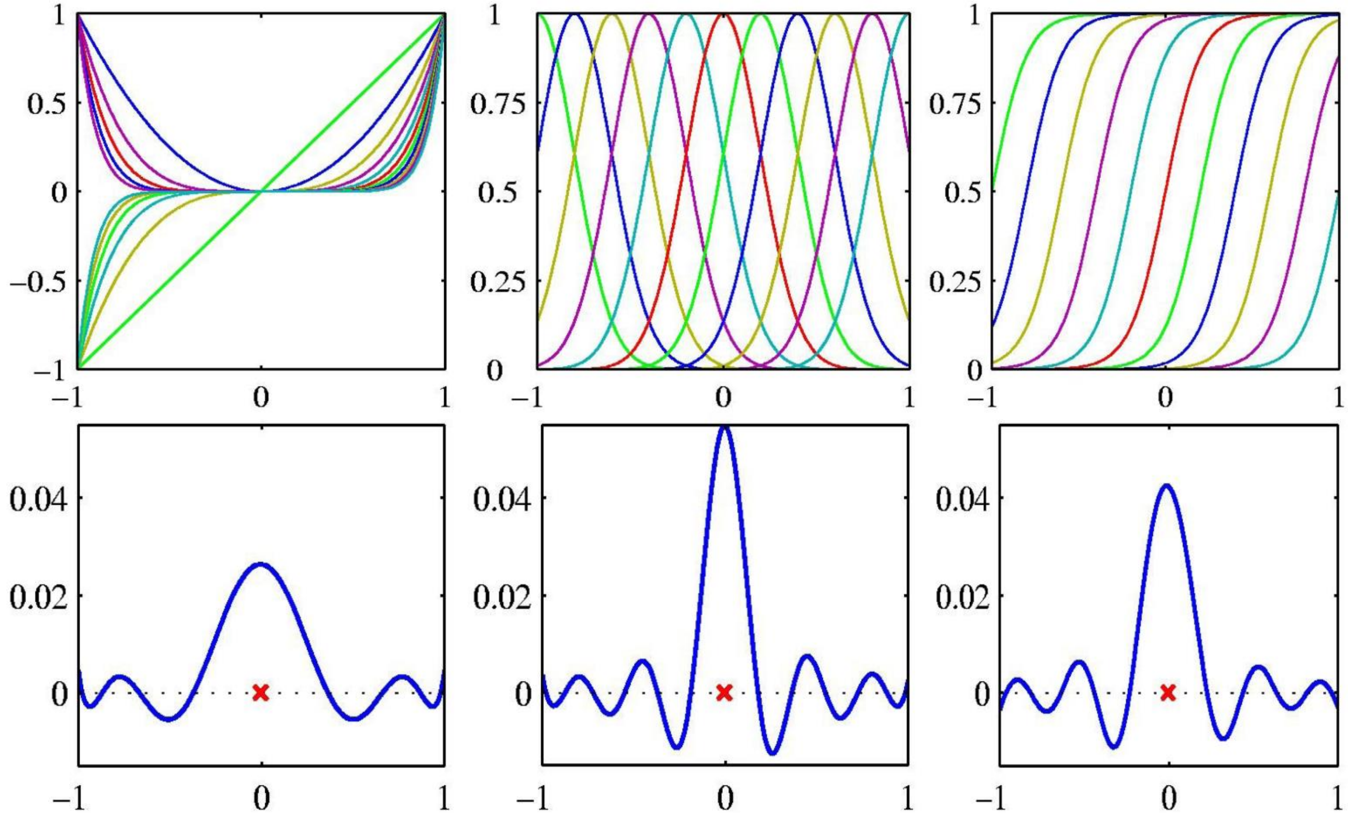
$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' \quad (6.20)$$

$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}_a') + k_b(\mathbf{x}_b, \mathbf{x}_b') \quad (6.21)$$

$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}_a') k_b(\mathbf{x}_b, \mathbf{x}_b') \quad (6.22)$$

Constructing Kernels

- Approach 1: Choose a feature space mapping and then use this to find the kernel



Constructing Kernels

- Approach 2: Construct kernel functions directly such that it corresponds to a scalar product in some feature space

$$k(x, x') = \phi(x)^{\mathrm{T}} \phi(x') = \sum_{i=1}^{M} \phi_i(x) \phi_i(x')$$

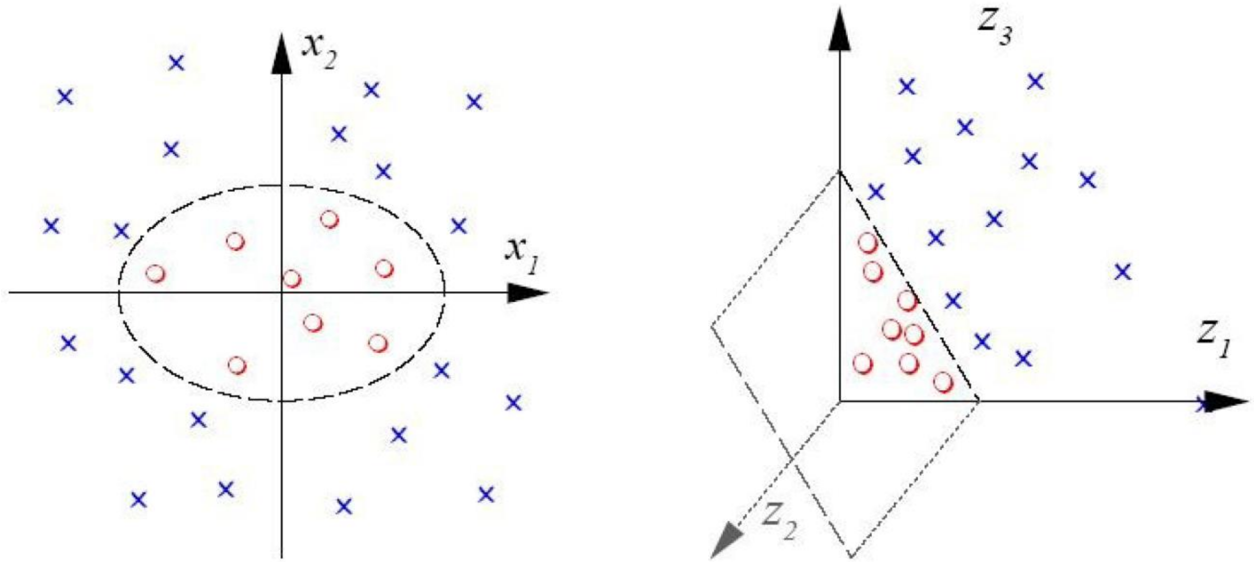
$$\begin{aligned} k(\mathbf{x}, \mathbf{z}) &= \left(\mathbf{x}^{\mathrm{T}}\mathbf{z}\right)^2 = (x_1 z_1 + x_2 z_2)^2 \\ &= x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(z_1^2, \sqrt{2} z_1 z_2, z_2^2)^{\mathrm{T}} \\ &= \phi(\mathbf{x})^{\mathrm{T}} \phi(\mathbf{z}). \end{aligned}$$

We also can use nonlinear feature mapping

Example of Kernels (I)

- Polynomial Kernels: $k(x,y)=(x \bullet y)^d$
  - Assume we know most information is contained in monomials (e.g. multiword terms) of degree *d* (e.g. d=2: x_1^2, x_2^2, x_1x_2)

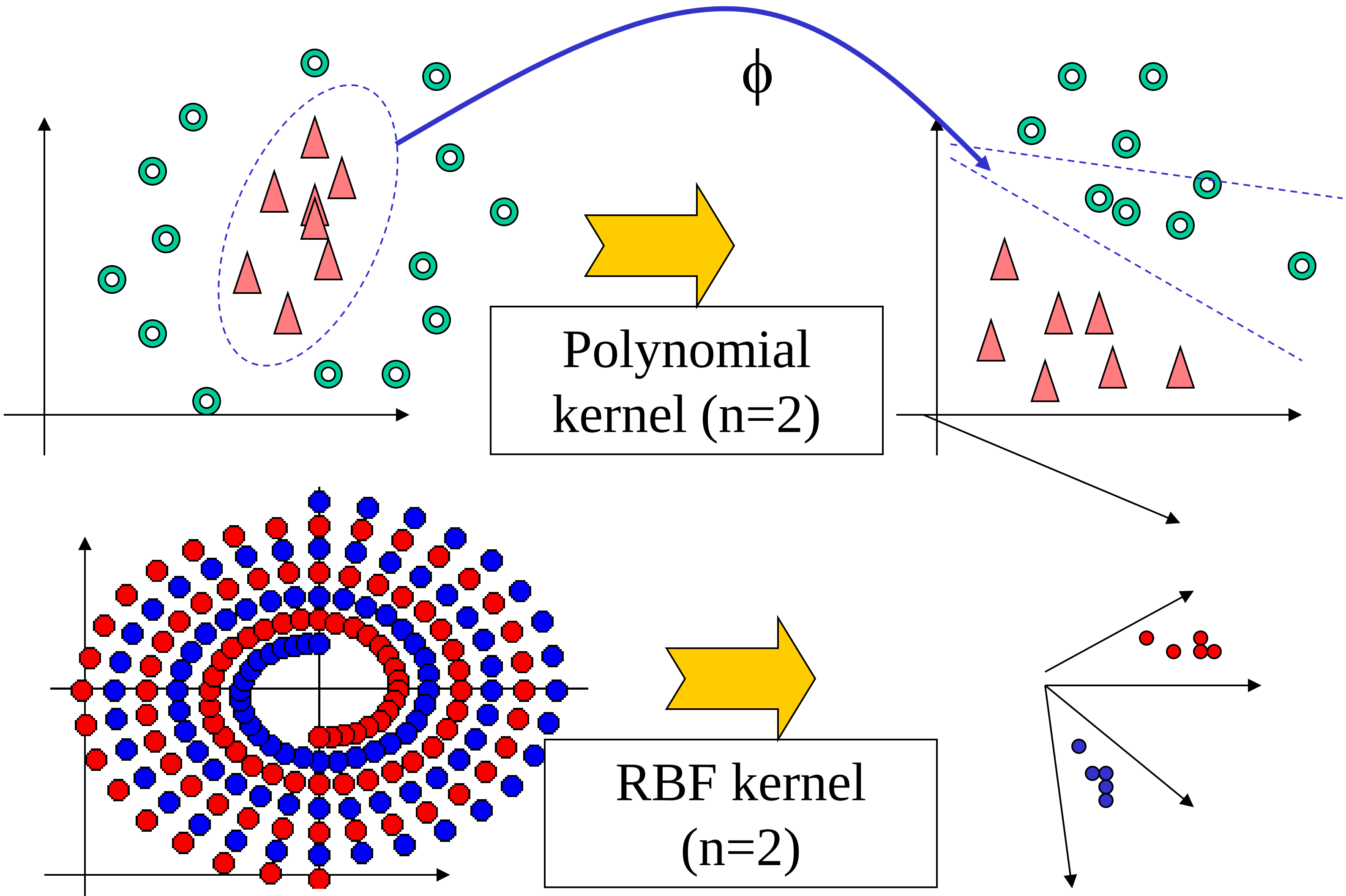
$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$



Examples of Kernels (II)

- Stationary kernels – invariant to translations in input space
  - $k(x,x') = k(x-x')$
- Homogeneous kernels (RBF) – depend only on the magnitude of the distance
  - $k(x,x') = k(||x-x'||)$
- Simple Polynomial Kernel – terms of degree 2
  - $k(x,x') = (x^T x)^2$
- Generalized Polynomial kernel – degree M
  - $k(x,x') = (x^T x+c)^M$, $c>0$
- Gaussian Kernels – not related to gaussian pdf !
  - $k(x,x') = \exp(-||x-x'||^2/2\sigma^2)$
- Sigmoidal Kernels
  - $k(x,x') = \tanh(ax^T x+b)$

Examples of Kernels (III)



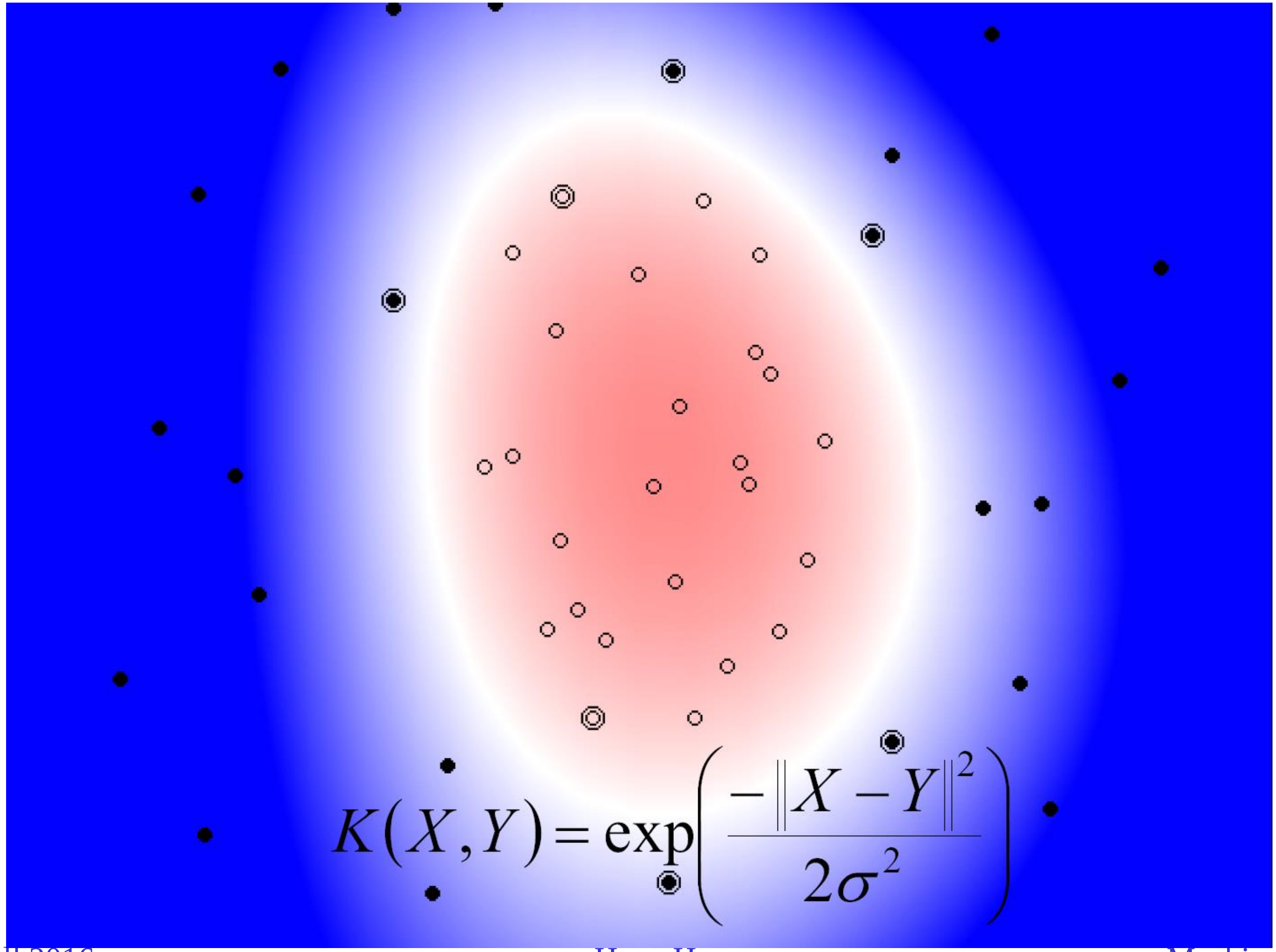
Gaussian RBF kernel

- Another popular kernel function (most powerful) is the Gaussian RBF kernel:

  $k(x,x') = \exp(-||x-x'||^2/2\sigma^2)$

- Powerful kernel as its effect is to create a small classification "hyperball" around an instance. This kernel doesn't have a projection formula since its dimension is infinite (you can create as many "balls" as you want).
- Where σ is a measure of the radius of the "hyperball" around an instance.
- You want this ball to be big enough so "hyperballs" connect with each other (pattern recognition) but not too big to overlap the other class.

Gaussian RBF kernel



$$K(X,Y) = \exp\left(\frac{-\|X-Y\|^2}{2\sigma^2}\right)$$

Construction of Probabilistic Kernels

- Kernels from probabilistic generative model
  - Can be used in discriminative setting

- Given a generative model $p(x)$, define a kernel by

$$k(x, x') = p(x)\, p(x')$$

  - Can be interpreted as inner product in the one dimensional feature space defined by mapping $p(x)$
  - Two inputs x and x' are similar if they both have highprobabilities

Construction of Probabilistic Kernels

- We can further extend this class of kernels by considering sums over products of different probability distributions with positive weighting coefficients

$$k(\mathbf{x}, \mathbf{x}') = \sum_i p(\mathbf{x}|i)p(\mathbf{x}'|i)p(i)$$

- For continuous latent variables, we have

$$k(\mathbf{x}, \mathbf{x}') = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{x}'|\mathbf{z})p(\mathbf{z})\, \mathrm{d}\mathbf{z}$$

Construction of Probabilistic Kernels

- Consider parametric generative model $p(x \mid \theta)$
- Find a kernel that measures similarity of two input vectors induced by the generative model.
- Consider the gradient w.r.t parameter θ that defines a vector in feature space having the same dimensionality as the parameter vector θ.
- Fisher Score $\mathbf{g}(\boldsymbol{\theta}, \mathbf{x}) = \nabla_{\boldsymbol{\theta}} \ln p(\mathbf{x}|\boldsymbol{\theta})$
- Fisher Kernel $k(\mathbf{x}, \mathbf{x}') = \mathbf{g}(\boldsymbol{\theta}, \mathbf{x})^{\mathrm{T}} \mathbf{F}^{-1} \mathbf{g}(\boldsymbol{\theta}, \mathbf{x}')$
- Fisher Information Matrix $\mathbf{F} = \mathbb{E}_{\mathbf{x}} \left[\mathbf{g}(\boldsymbol{\theta}, \mathbf{x}) \mathbf{g}(\boldsymbol{\theta}, \mathbf{x})^{\mathrm{T}} \right]$

Constructing Kernels in General

- A simpler way to test without having to construct $\Phi(x)$:
  - Use the necessary and sufficient condition that for a function $k(x,x')$ to be a valid kernel, the Gram matrix K, whose elements are given by $k(x_n,x_m)$, should be positive semidefinite for all possible choices of the set $\{x_n\}$

$$K_{training} = \begin{bmatrix} k(\mathbf{x}_1,\mathbf{x}_1) & k(\mathbf{x}_1,\mathbf{x}_2) & \dots & k(\mathbf{x}_1,\mathbf{x}_m) \\ k(\mathbf{x}_2,\mathbf{x}_1) & k(\mathbf{x}_2,\mathbf{x}_2) & \dots & k(\mathbf{x}_2,\mathbf{x}_m) \\ \dots & \dots & \dots & \dots \\ k(\mathbf{x}_m,\mathbf{x}_1) & k(\mathbf{x}_m,\mathbf{x}_2) & \dots & k(\mathbf{x}_m,\mathbf{x}_m) \end{bmatrix}$$

What Is A Proper Kernel

Definition: *A finitely positive semi-definite function* $k : x \times y \rightarrow R$ *is a symmetric function of its arguments for which matrices formed by restriction on any finite subset of points is positive semi-definite.*

$$\alpha^T K \alpha \geq 0 \quad \forall \alpha$$

Theorem: *A function* $k : x \times y \rightarrow R$ *can be written as* $k(x, y) = <\Phi(x), \Phi(y)>$ *where* $\Phi(x)$ *is a feature map* $x \rightarrow \Phi(x) \in F$ *iff k(x,y) satisfies the semi-definiteness property.*

Relevance: We can now check if k(x,y) is a proper kernel using only properties of k(x,y) itself,
i.e. without the need to know the feature map!

Mercer's Theorem

<u>Theorem:</u> *X is compact, k(x,y) is symmetric continuous function s.t.* $T_k f \equiv \int k(.,x) f(x)\,dx$ *is a positive semi-definite operator:* $T_k \geq 0$
i.e.

$$\int\int k(x,y) f(x) f(y)\,dxdy \geq 0 \quad \forall f \in L_2(X)$$

then there exists an orthonormal feature basis of eigen-functions such that:

$$k(x,y) = \sum_{i=1}^{\infty} \Phi_i(x)\Phi_j(y)$$

Hence: k(x,y) is a proper kernel.

A Bad Kernel …

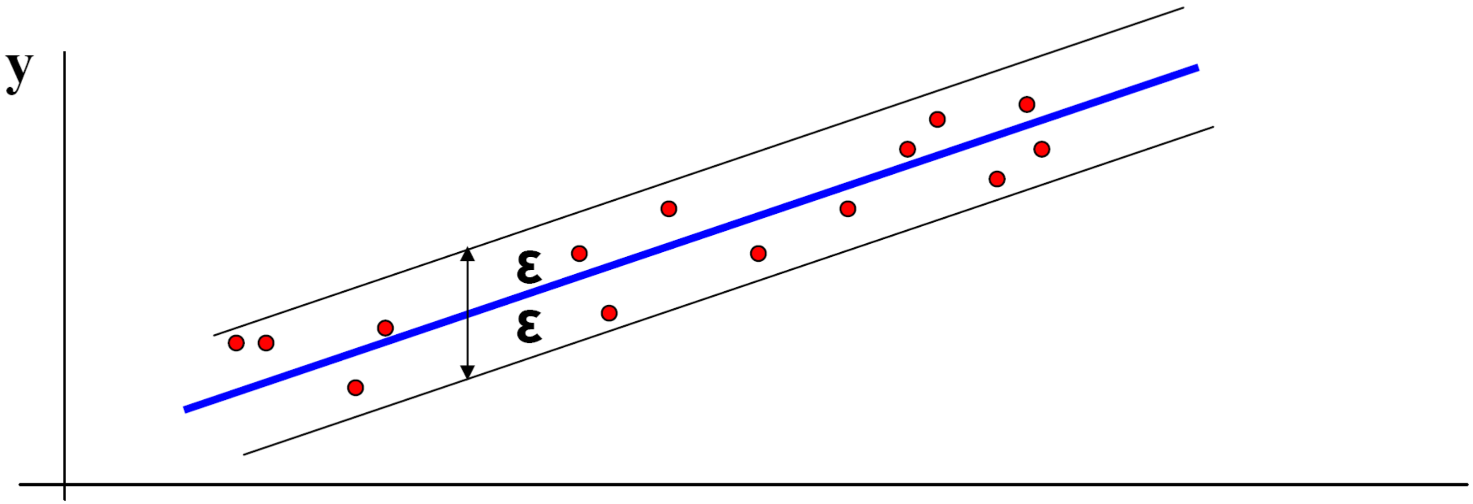
- … would be a kernel whose kernel matrix is mostly diagonal: all points orthogonal to each other, no clusters, no structure …

1	0	0	…	0
0	1	0	…	0
		1		
…	…	…	…	…
0	0	0	…	1

- If mapping in a space with too many irrelevant features, kernel matrix becomes diagonal

Support Vector Machine for Regression

- **Regression** = find a function that fits the data.
- A data point may be wrong due to the noise
- **Idea:** Error from points which are close **should count as a valid noise**
- Line should be influenced by the real data not the noise.



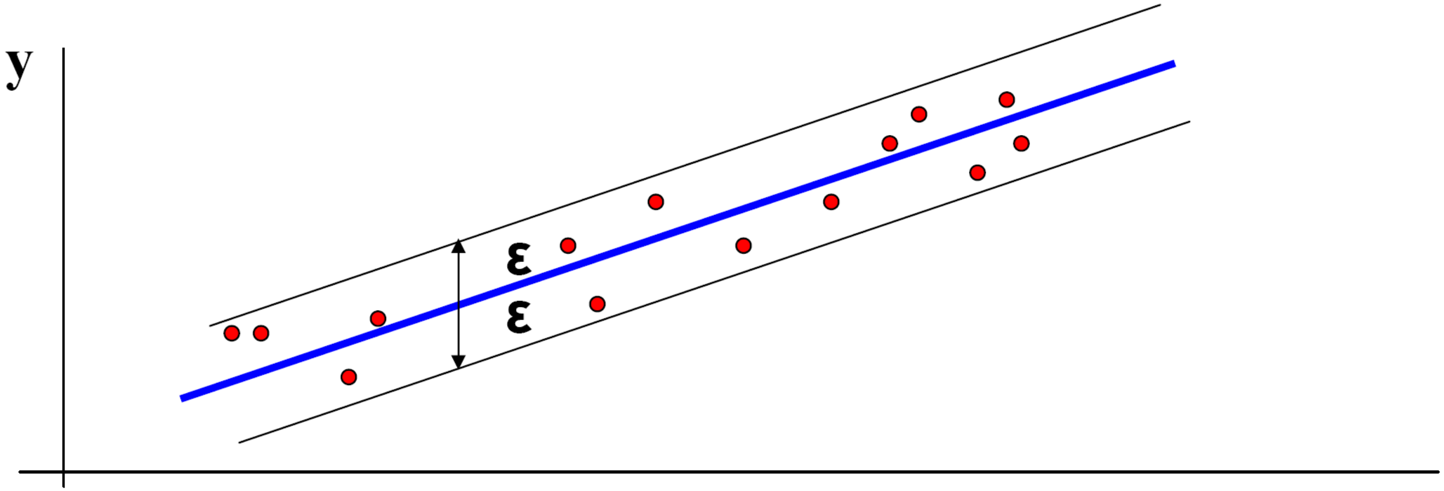
Linear Model

- **Training data:**

$$\{(x_1, y_1), \ldots, (x_l, y_l)\}, \; x \in R^n, \; y \in R$$

- Our goal is to find a function f(x) that has at most ε deviation from the actually obtained target for all the training data.

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b = \langle \mathbf{w}, \mathbf{x} \rangle + b$$



Linear Model

Linear function:

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

We want a function that is:

- **flat:** means that one seeks small **w**
- all data points are within its ε neighborhood

The problem can be formulated as a **convex optimization problem:**

$$\text{minimize} \quad \frac{1}{2}\|w\|^2$$

$$\text{subject to} \quad \begin{cases} \mathrm{y_i} - \langle w_i, x_i \rangle - b \le \varepsilon \\ \langle w_i, x_i \rangle + b - \mathrm{y_i} \le \varepsilon \end{cases}$$

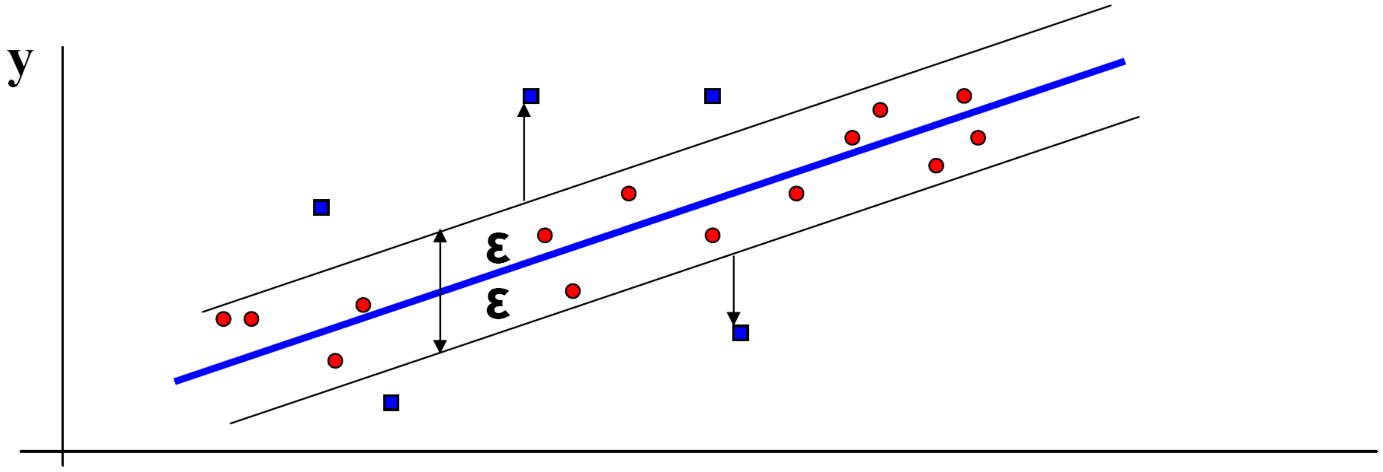
All data points are assumed to be in the ε neighborhood

Linear Model

- **Real data: not all data points always fall into the ε** neighborhood

$$f(\mathbf{x}) = \mathbf{w}^{T}\mathbf{x} + b = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

- **Idea:** penalize points that fall outside the ε neighborhood



Linear Model

Linear function:

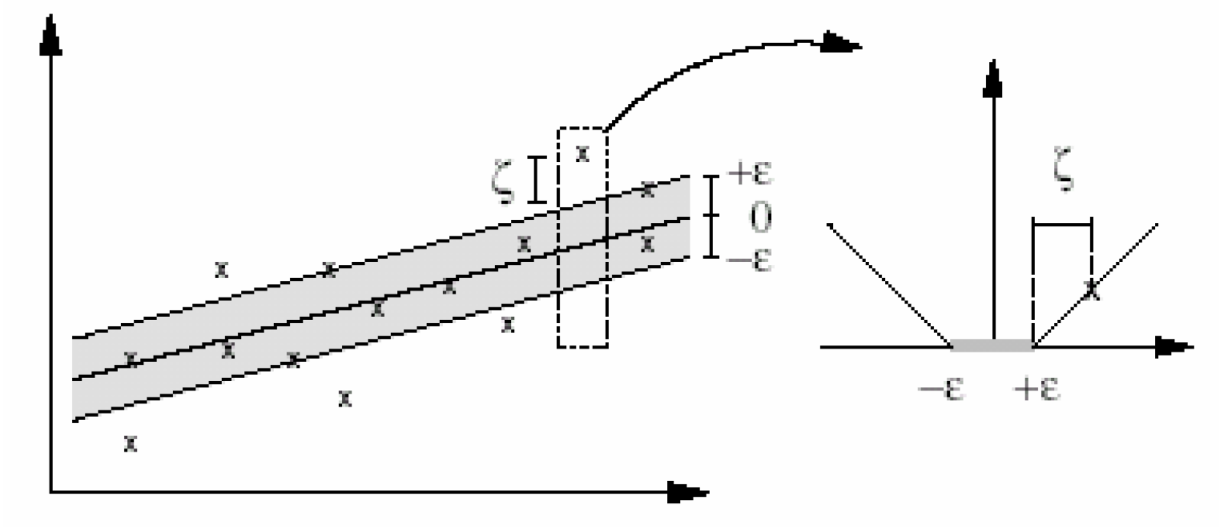
$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

Idea: penalize points that fall outside the ε neighborhood

$$\text{minimize} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$

$$\text{subject to} \quad \begin{cases} \mathrm{y}_\mathrm{i} - \langle w_i, x_i \rangle - b \le \varepsilon + \xi_i \\ \langle w_i, x_i \rangle + b - \mathrm{y}_\mathrm{i} \le \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0 \end{cases}$$

Linear Model



$$|\xi|_{\varepsilon} = \begin{cases} 0 & \text{for } |\xi| \leq \varepsilon \\ |\xi| - \varepsilon & \text{otherwise} \end{cases}$$

ε-intensive loss function

Optimization

Lagrangian that solves the optimization problem

$$L = \frac{1}{2}\langle w, w\rangle + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$
$$-\sum_{i=1}^{l} a_i(\varepsilon - \xi_i - y_i + \langle w, x_i\rangle + b) - \sum_{i=1}^{l} a_i^*(\varepsilon + \xi_i^* + y_i - \langle w, x_i\rangle - b)$$
$$-\sum_{i=1}^{l}(\eta_i \xi_i + \eta_i^* \xi_i^*)$$

Subject to $a_i, a_i^*, \eta_i, \eta_i^* \geq 0$

Primal variables w, b, ξ_i, ξ_i^*

Optimization

Derivatives with respect to primal variables

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{l} (a_i^* - a_i) = 0 \qquad \frac{\partial L}{\partial \xi_i^{(*)}} = C - a_i^{(*)} - \eta_i^{(*)} = 0$$

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{l} (a_i^* - a_i)\mathbf{x}_i = \mathbf{0} \qquad \frac{\partial L}{\partial \xi_i} = C - a_i - \eta_i = 0$$

$$L = \frac{1}{2}\langle w, w\rangle + \sum_{i=1}^{l} C\xi_i + \sum_{i=1}^{l} C\xi_i^*$$

$$-\sum_{i=1}^{l} a_i\varepsilon - \sum_{i=1}^{l} a_i\xi_i - \sum_{i=1}^{l} a_i y_i - \sum_{i=1}^{l} a_i\langle \omega, x_i\rangle + \sum_{i=1}^{l} a_i b$$

$$-\sum_{i=1}^{l} a_i^*\varepsilon - \sum_{i=1}^{l} a_i^*\xi_i^* - \sum_{i=1}^{l} a_i^* y_i + \sum_{i=1}^{l} a_i^*\langle \omega, x_i\rangle + \sum_{i=1}^{l} a_i^* b$$

$$-\sum_{i=1}^{l} \eta_i\xi_i - \sum_{i=1}^{l} \eta_i^*\xi_i^*$$

Optimization

$$L = \frac{1}{2}\langle w, w\rangle + \sum_{i=1}^{l} \xi_i \underbrace{(C - \eta_i - a_i)}_{=0\,(C-\eta_i^{(*)}-a_i^{(*)}=0\,)} +$$

$$\sum_{i=1}^{l} \xi_i^* \underbrace{(C - \eta_i^* - a_i^*)}_{=0\,(C-\eta_i^{(*)}-a_i^{(*)}=0\,)} - \sum_{i=1}^{l} (a_i + a_i^*)\varepsilon - \sum_{i=1}^{l} (a_i + a_i^*) y_i$$

$$- \sum_{i=1}^{l} \underbrace{(a_i - a_i^*)\langle \omega, x_i\rangle}_{=\langle w,w\rangle\,(\,\omega=\sum_{i=1}^{l}(a_i+a_i^*)x_i\,)} + \sum_{i=1}^{l} \underbrace{(a_i^* - a_i) b}_{=0\,(\,\sum_{i=1}^{l}(a_i^*-a_i)=0)}$$

$$= -\frac{1}{2}\langle w, w\rangle - \sum_{i=1}^{l} (a_i + a_i^*)\varepsilon - \sum_{i=1}^{l} (a_i + a_i^*) y_i$$

Maximize the dual

$$L(a, a^*) = -\frac{1}{2}\sum_{i=1}^{l} (a_i - a_i^*)(a_j - a_j^*)\langle x_i, x_j\rangle - \sum_{i=1}^{l} (a_i + a_i^*)\varepsilon - \sum_{i=1}^{l} (a_i + a_i^*) y_i$$

subject to $: \begin{cases} \sum_{i=1}^{l} (a_i - a_i^*) = 0 \\ a_i, a_i^* \in [0, C] \end{cases}$

Solution

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{l} (a_i^* - a_i)\mathbf{x}_i = \mathbf{0}$$

$$\mathbf{w} = \sum_{i=1}^{l} (a_i - a_i^*)\mathbf{x}_i$$

We can get:

$$f(\mathbf{x}) = \sum_{i=1}^{l} (a_i - a_i^*)\langle \mathbf{x_i}, \mathbf{x} \rangle + b$$

at the optimal solution the Lagrange multipliers are non-zero only for points outside the ε band.