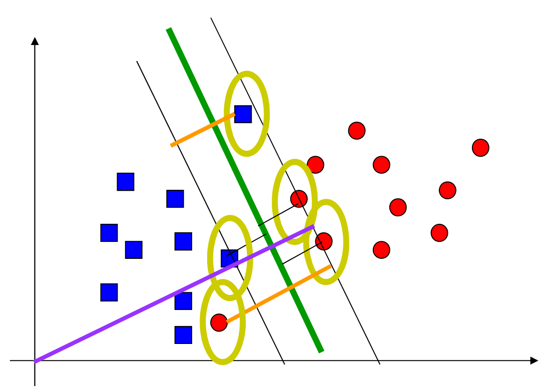
CS 2750 Machine Learning
Lecture 10

- **SVMs for regression**
- **Non-parametric/instance based classification method**

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Soft-margin SVM

- Allows some flexibility on crossing the separating hyperplane



Soft-margin SVM

minimize $\|\mathbf{w}\|^2/2 + C\sum_{i=1}^{n}\xi_i$

$\mathbf{w}^T\mathbf{x}_i + w_0 \geq 1-\xi_i$ **for** $y_i = +1$

$\mathbf{w}^T\mathbf{x}_i + w_0 \leq -1+\xi_i$ **for** $y_i = -1$

$\xi_i \geq 0$

- Rewrite $\xi_i = \max\left[0,\ \ 1 - y_i(\mathbf{w}^T\mathbf{x}_i + w_0)\right]$ in $\|\mathbf{w}\|^2/2 + C\sum_{i=1}^{n}\xi_i$

$$\|\mathbf{w}\|^2/2 + C\sum_{i=1}^{n}\max\left[0,\ \ 1 - y_i(\mathbf{w}^T\mathbf{x}_i + w_0)\right]$$

Regularization penalty

Hinge loss

Classification learning

- **General form:**

$$\min_{\mathbf{w}} \quad L(\mathbf{w}, D) + \lambda Q(\mathbf{w})$$

Loss function **Regularization penalty**

- **Loss functions:**
  - **Negative loglikelihood (used in the LR)**
  - **Hinge loss (used in SVM)**
- **Regularization terms:**
  - **L1 (lasso)**
  - **L2 (ridge)**

Support vector machines

- **The decision boundary:**

$$\hat{\mathbf{w}}^T \mathbf{x} + w_0 = \sum_{i \in SV} \hat{\alpha}_i y_i (\mathbf{x}_i{}^T \mathbf{x}) + w_0 = 0$$

- **The decision:**

$$\hat{y} = \text{sign}\left[\sum_{i \in SV} \hat{\alpha}_i y_i (\mathbf{x}_i{}^T \mathbf{x}) + w_0 \right]$$

- **(!!):**
- Decision on a new $\mathbf{x}$ requires to compute the inner product between the examples $(\mathbf{x}_i{}^T \mathbf{x})$
- Similarly, the optimization depends on $(\mathbf{x}_i{}^T \mathbf{x}_j)$

$$J(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i{}^T \mathbf{x}_j)$$

Nonlinear case

- The linear case requires to compute $(\mathbf{x}_i{}^T \mathbf{x})$
- The non-linear case can be handled by using a set of features. Essentially we map input vectors to (larger) feature vectors

$$\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$$

- It is possible to use SVM formalism on feature vectors

$$\boldsymbol{\varphi}(\mathbf{x})^T \boldsymbol{\varphi}(\mathbf{x}')$$

- **Kernel function**

$$K(\mathbf{x}, \mathbf{x}') = \boldsymbol{\varphi}(\mathbf{x})^T \boldsymbol{\varphi}(\mathbf{x}')$$

- **Crucial idea:** If we choose the kernel function wisely we can compute linear separation in the feature space implicitly such that we keep working in the original input space !!!!

Kernel function example

- Assume $\mathbf{x} = [x_1, x_2]^T$ and a feature mapping that maps the input into a quadratic feature set

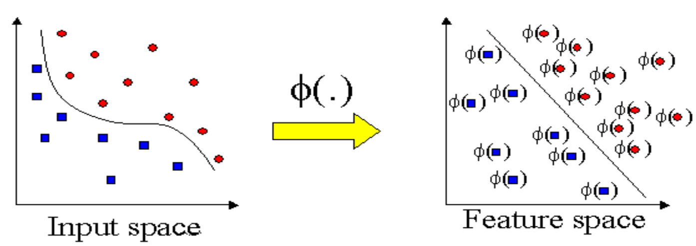
$$\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2}x_1 x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1]^T$$

- Kernel function for the feature space:

$$\begin{aligned} K(\mathbf{x'}, \mathbf{x}) &= \boldsymbol{\varphi}(\mathbf{x'})^T \boldsymbol{\varphi}(\mathbf{x}) \\ &= x_1^2 x'^2_1 + x_2^2 x'^2_2 + 2x_1 x_2 x'_1\, x'_2 + 2x_1 x'_1 + 2x_2 x'_2 + 1 \\ &= (x_1 x'_1 + x_2 x'_2 + 1)^2 \\ &= (1 + (\mathbf{x}^T \mathbf{x'}))^2 \end{aligned}$$

- The computation of the linear separation in the higher dimensional space is performed implicitly in the original input space

Nonlinear extension



Kernel trick

- Replace the inner product with a kernel
- A well chosen kernel leads to an efficient computation

Kernel functions

- **Linear kernel**

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$

- **Polynomial kernel**

$$K(\mathbf{x}, \mathbf{x}') = \left[1 + \mathbf{x}^T \mathbf{x}'\right]^k$$

- **Radial basis kernel**

$$K(\mathbf{x}, \mathbf{x}') = \exp\left[-\frac{1}{2}\|\mathbf{x} - \mathbf{x}'\|^2\right]$$

Kernels

- **Kernels define** a **similarity measure** :
  - define a distance in between two objects
- **Design criteria:** we want kernels to be
  - **valid** – Satisfy **Mercer condition** of positive semi-definiteness
  - **good** – embody the “true similarity” between objects
  - **appropriate** – generalize well
  - **efficient** – the computation of K(x,x’) is feasible
    - NP-hard problems abound with graphs

Kernels

- Research have proposed kernels for comparison of variety of objects:
  - Strings
  - Trees
  - Graphs
- **Cool thing:**
  - SVM algorithm can be now applied to classify a variety of objects

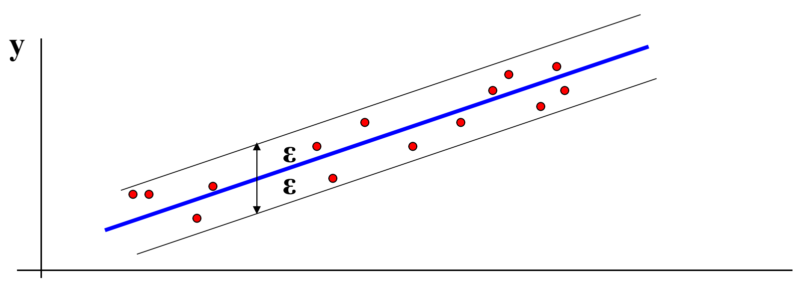
Support vector machine for regression

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.



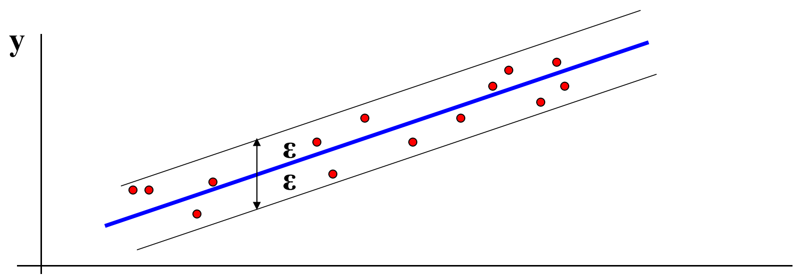
Linear model

- **Training data:**

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

- 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.

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



Linear model

Linear function:

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

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:**

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

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

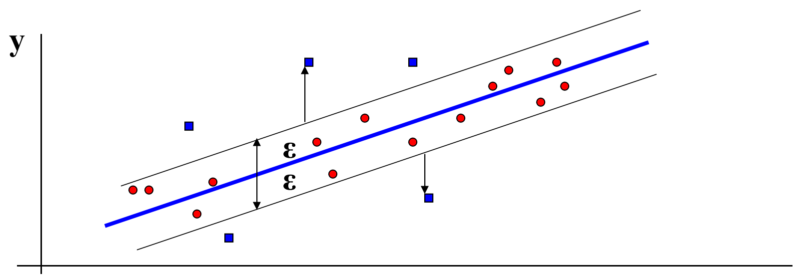
All data points are assumed to be in the ε neighborhood

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:

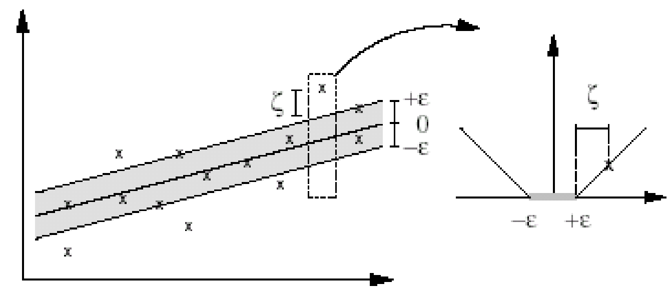
$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

Idea: penalize points that fall outside the ε neighborhood

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

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

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

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

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

$$\frac{\partial L}{\partial \xi_i^{(*)}} = C - a_i^{(*)} - \eta_i^{(*)} = 0$$

$$\frac{\partial L}{\partial \xi_i} = C - a_i - \eta_i = 0$$

Optimization

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

Optimization

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

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

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

Optimization

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

Maximize the dual

Inner product

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

$$- \sum_{i=1}^{l} (a_i + a_i^*)\varepsilon - \sum_{i=1}^{l} (a_i + a_i^*) y_i$$

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

SVM 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:

Inner product

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

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

Nonparametric vs Parametric Methods

Nonparametric models:

- More flexibility – no parametric model is needed
- But require storing the entire dataset
- and the computation is performed with all data examples.

Parametric models:

- Once fitted, only parameters need to be stored
- They are much more efficient in terms of computation
- But the model needs to be picked in advance

Non-parametric Classification methods

- Given a data set with N_k data points from class C_k and $\sum_k N_k = N$, we have

$$p(\mathbf{x}) = \frac{K}{NV}$$

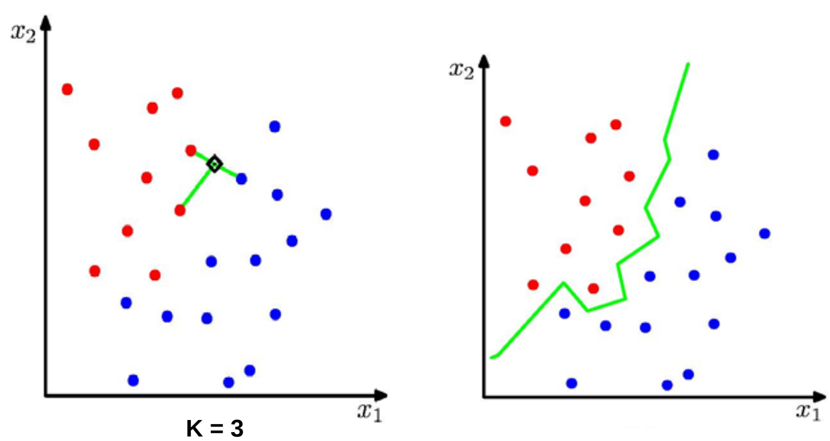
- and correspondingly

$$p(\mathbf{x}|\mathcal{C}_k) = \frac{K_k}{N_k V}.$$

- Since $p(\mathcal{C}_k) = N_k/N$, Bayes' theorem gives

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})} = \frac{K_k}{K}.$$

K-Nearest-Neighbours for Classification



Nonparametric kernel-based classification

- **Kernel function: k(x,x')**
  - Models similarity between x, x'
  - **Example:** Gaussian kernel we used in the kernel density estimation

$$k(x, x') = \frac{1}{(2\pi h^2)^{D/2}} \exp\left(- \frac{(x - x')^2}{2h^2} \right)$$

$$p(x) = \frac{1}{N} \sum_{i=1}^{N} k(x, x_i)$$

- **Kernel for classification**

$$p(y = C_k \mid x) = \frac{\sum_{x':y'=C_k} k(x, x')}{\sum_{x'} k(x, x')}$$