Instance-based Learning

Machine Learning – 10701/15781

Carlos Guestrin

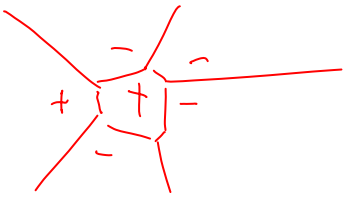
Carnegie Mellon University

October 15th, 2007

1-Nearest Neighbor

Four things make a memory based learner:

1. *A distance metric*
   Euclidian (and many more)
2. *How many nearby neighbors to look at?*
   One
3. *A weighting function (optional)*
   Unused
4. *How to fit with the local points?*
   Just predict the same output as the nearest neighbor.



Consistency of 1-NN

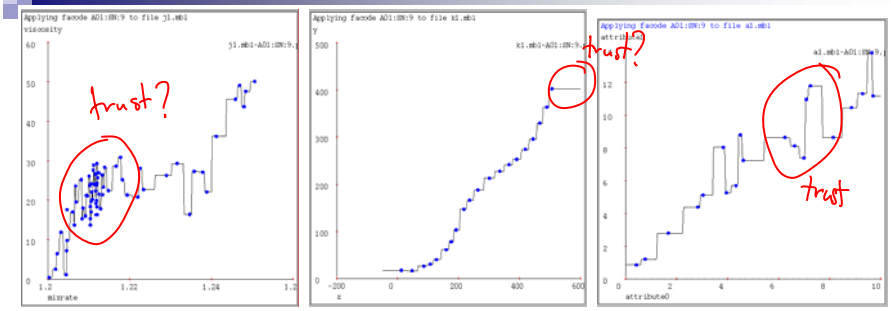
- Consider an estimator f_n trained on n examples
  - e.g., 1-NN, neural nets, regression,...
- Estimator is *consistent* if true error goes to zero as amount of data increases
  - e.g., for no noise data, consistent if:

$$\lim_{n \to \infty} MSE(f_n) = 0$$

- Regression is not consistent!
  - Representation bias
- **1-NN is consistent** (under some mild fineprint)

What about variance???

1-NN overfits?



k-Nearest Neighbor

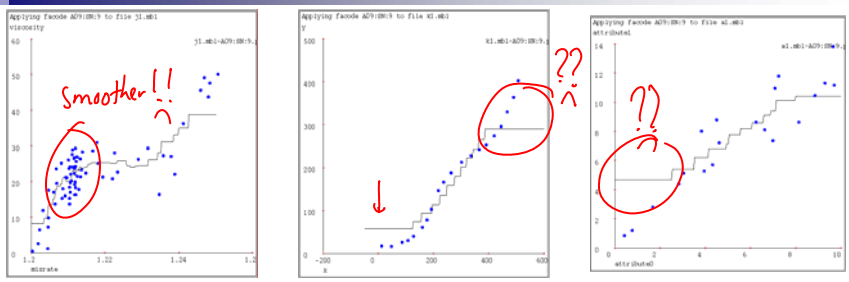
Four things make a memory based learner:

1. *A distance metric*
   Euclidian (and many more)
2. *How many nearby neighbors to look at?*
   k
1. *A weighting function (optional)*
   Unused
2. *How to fit with the local points?*
   Just predict the average output among the k nearest neighbors.

$x_q \in$ query $\quad N(x_q)$

$$\hat{y}_q = \frac{1}{k} \sum_{x_i \in N(x_q)} y_i$$

k-Nearest Neighbor (here k=9)



less variance, but more bias

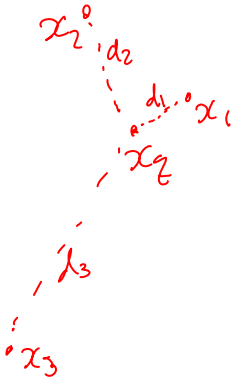
(still consistent see store for details, if k fixed as $n \to \infty$)

K-nearest neighbor for function fitting smoothes away noise, but there are clear deficiencies.

What can we do about all the discontinuities that k-NN gives us?

Weighted k-NNs

- Neighbors are not all the same (an example)

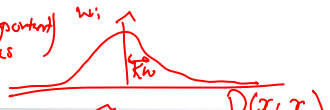


$$\hat{y}_q \overset{\text{e.g.}}{=} \frac{\sum_{x_i \in N(x_q)} \frac{1}{d_i} y_i}{\sum_{x_i \in N(x_q)} \frac{1}{d_i}}$$

normalize to get a convex combination

Kernel regression

good work for high: learn distance function (learn important features)



everything (doesn't work well in high-dim, unless lots of data)

Four things make a memory based learner:

1. *A distance metric*
   Euclidian (and many more) ← distance in any dim.
2. *How ~~many~~ nearby neighbors to look at?*
   All of them
3. *A weighting function (optional)*
   $w_i = exp(-D(x_i, query)^2 / K_w^2)$

   $$w_i = e^{-\frac{D(x_i, x_q)^2}{K_w^2}}$$

   parameter called bandwidth

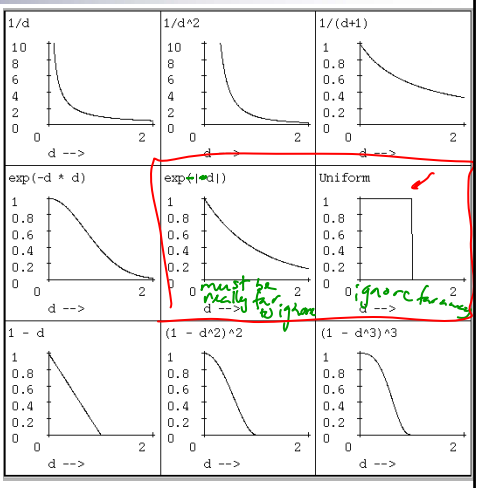
   Nearby points to the query are weighted strongly, far points weakly. The K_W parameter is the **Kernel Width**. Very important.
4. *How to fit with the local points?*
   Predict the weighted average of the outputs:
   predict = $\Sigma w_i y_i / \Sigma w_i$

$$\hat{y}_q = \frac{\sum_{i=1}^{m} w_i y_i}{\sum_{i=1}^{m} w_i}$$

Weighting functions

$w_i = exp(-D(x_i, query)^2 / K_w^2)$

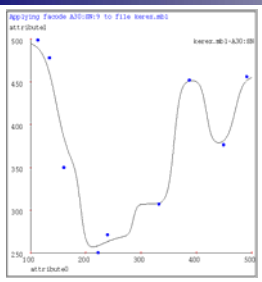
Squared exponential | Kernel
Gaussian



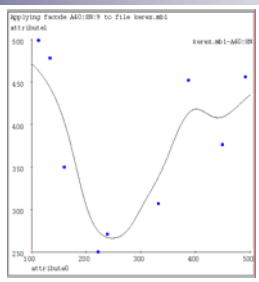
Typically optimize K_w using gradient descent

(Our examples use Gaussian)

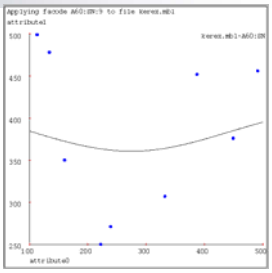
Kernel regression predictions



K_W=10



K_W=20



K_W=80

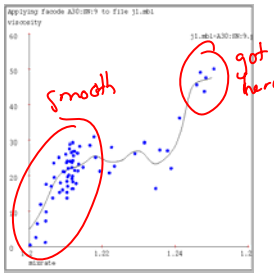
bias ↓ var ↑

bias ↑ var ↓

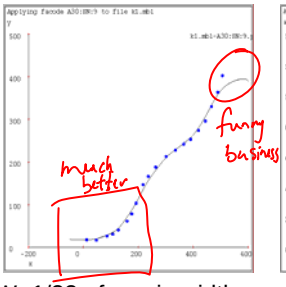
Increasing the kernel width K_w means further away points get an opportunity to influence you.

As $K_w \rightarrow \infty$, the prediction tends to the global average.

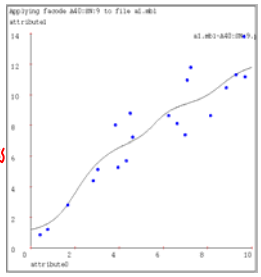
Kernel regression on our test cases



KW=1/32 of x-axis width.



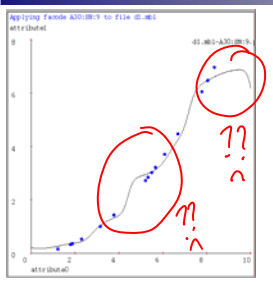
KW=1/32 of x-axis width.



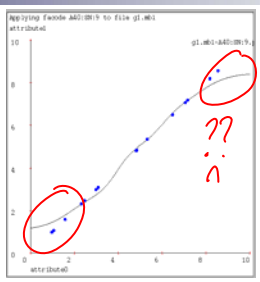
KW=1/16 axis width.

Choosing a good K_w is important. Not just for Kernel Regression, but for all the locally weighted learners we're about to see.

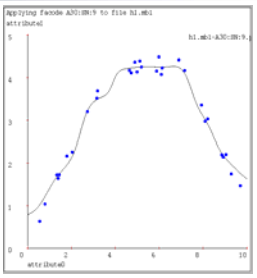
Kernel regression can look bad



KW = Best.



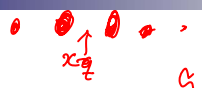
KW = Best.



KW = Best.

Time to try something more powerful…

Locally weighted regression



Kernel regression:

Take a very very conservative function approximator called AVERAGING. Locally weight it.

Locally weighted regression:

Take a conservative function approximator called LINEAR REGRESSION. Locally weight it.

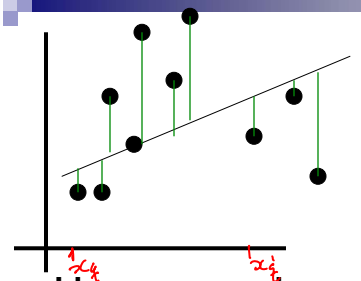
Locally weighted regression

- **Four things make a memory based learner:**
- *A distance metric*
  - **Any**
- *How many nearby neighbors to look at?*
  - **All of them**
- *A weighting function (optional)*
  - **Kernels**
    - ***wi = exp(-D(xi, query)² / Kw²)***
- *How to fit with the local points?*

  General weighted regression:

$$\hat{\beta} = \arg\min_{\beta} \sum_{k=1}^{N} w_k^2 \left(y_k - \beta^T x_k\right)^2$$

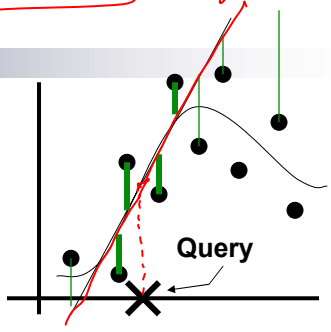
How LWR works



Linear regression

- Same parameters for all queries

$$\hat{\beta} = \left(X^T X\right)^{-1} X^T Y$$



Locally weighted regression

- Solve weighted linear regression for each query

$$\hat{\beta} = \left((WX)^T WX\right)^{-1} (WX)^T WY$$

$$W = \begin{pmatrix} w_1 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & w_n \end{pmatrix}$$

Another view of LWR

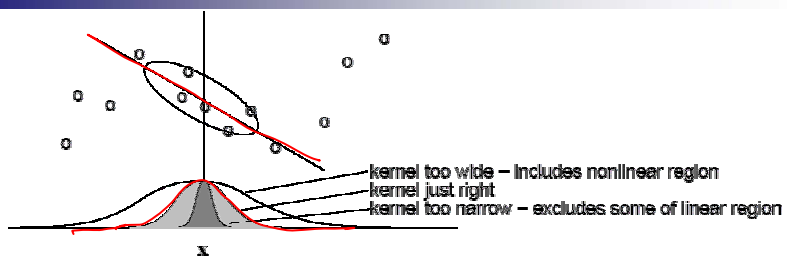
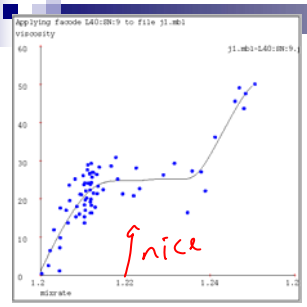
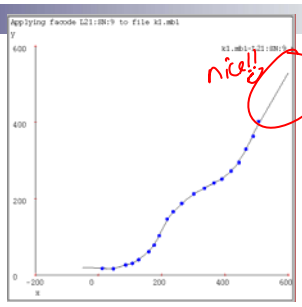


Image from Cohn, D.A., Ghahramani, Z., and Jordan, M.I. (1996). "Active Learning with Statistical Models", JAIR Volume 4, pages 129-145.

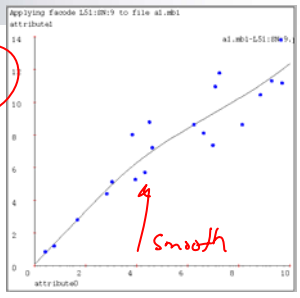
LWR on our test cases



KW = 1/16 of x-axis width.

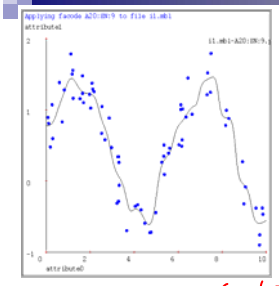


KW = 1/32 of x-axis width.



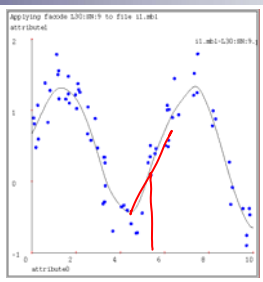
KW = 1/8 of x-axis width.

Locally weighted polynomial regression



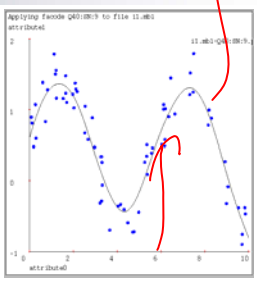
Kernel Regression
Kernel width K_W at optimal level.

KW = 1/100 x-axis



LW Linear Regression
Kernel width K_W at optimal level.

KW = 1/40 x-axis



LW Quadratic Regression
Kernel width K_W at optimal level.

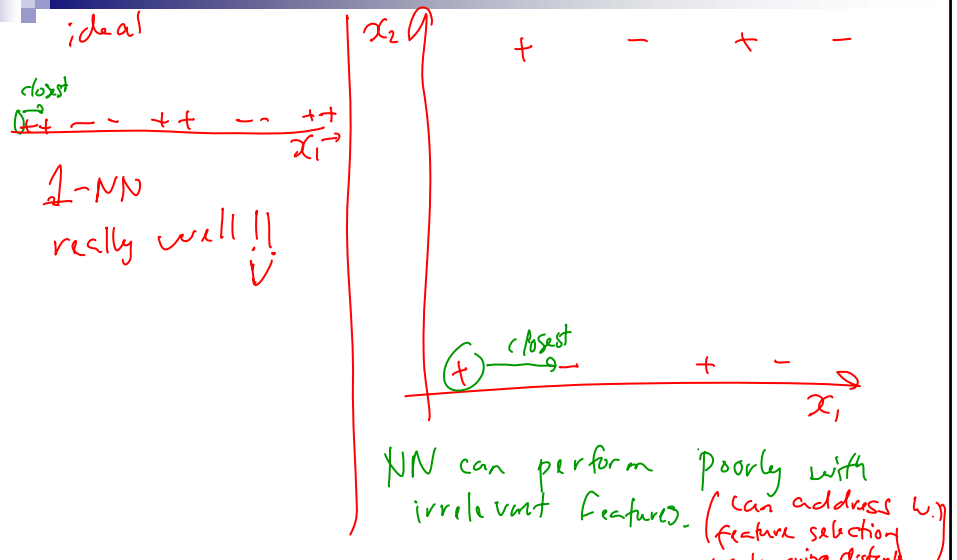
KW = 1/15 x-axis

Local quadratic regression is easy: just add quadratic terms to the WX^TWX matrix. As the regression degree increases, the kernel width can increase without introducing bias.

Curse of dimensionality for instance-based learning

- Must store and retreve all data!
  - Most real work done during testing
  - For every test sample, must search through all dataset – very slow!
  - We'll see fast methods for dealing with large datasets → KD-trees ?? -trees
- Instance-based learning often poor with noisy or irrelevant features

Curse of the irrelevant feature



What you need to know about instance-based learning

- k-NN
  - Simplest learning algorithm
  - With sufficient data, very hard to beat "strawman" approach
  - Picking k?
- Kernel regression
  - Set k to n (number of data points) and optimize weights by gradient descent
  - Smoother than k-NN
- Locally weighted regression
  - Generalizes kernel regression, not just local average
- Curse of dimensionality
  - Must remember (very large) dataset for prediction
  - Irrelevant features often killers for instance-based approaches

Acknowledgment

- This lecture contains some material from Andrew Moore's excellent collection of ML tutorials:
  - http://www.cs.cmu.edu/~awm/tutorials

Support Vector Machines

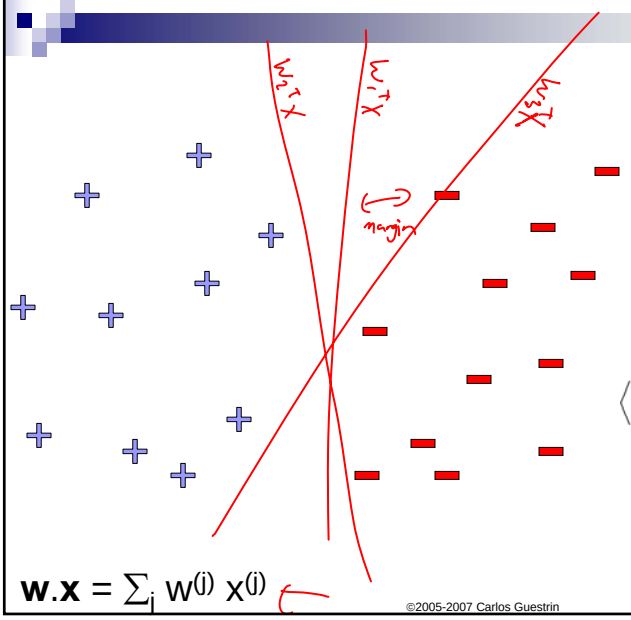
Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

October 15th, 2007

Linear classifiers – Which line is better?



Data:

$$\left\langle x_1^{(1)}, \ldots, x_1^{(m)}, y_1 \right\rangle$$
$$\vdots$$
$$\left\langle x_n^{(1)}, \ldots, x_n^{(m)}, y_n \right\rangle$$

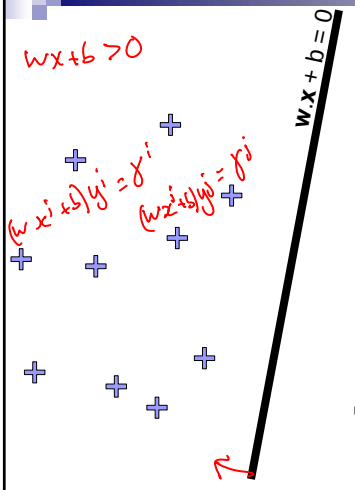
Example i:

$\left\langle x_i^{(1)}, \ldots, x_i^{(m)} \right\rangle$ — m features

$y_i \in \{-1, +1\}$ — class

w.x = $\sum_j w^{(j)} x^{(j)}$

Pick the one with the largest margin!



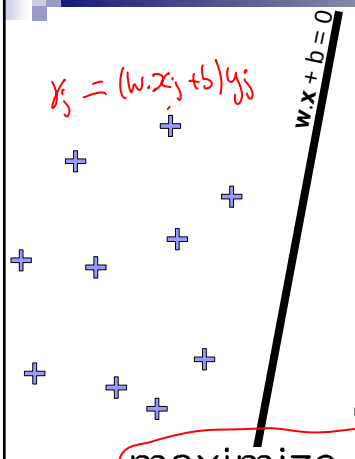
"confidence" $= \left(\mathbf{w}.\mathbf{x}_j + b\right) y_j$

$\max_{w,b} \min_{i \text{ fixing}} \gamma^i$

pick plane where worst point has highest confidence

$\mathbf{w}.\mathbf{x} = \sum_j w^{(j)} x^{(j)}$

Maximize the margin



$\gamma_1 = 0.1 \geq \gamma$
$\gamma_2 = 0.2 \geq \gamma$
$\gamma_3 = 0.05 \geq \gamma$

$\max_{w,b} \min_i (w.x_i + b) y_i$

$= \max_{w,b} \min_i \gamma_i$

$(w.x_i + b) y_i \geq \gamma_i$

$= \max_{w,b} \gamma$

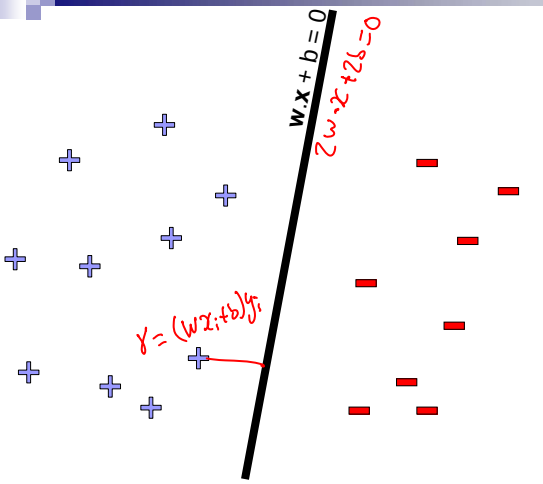
$(w.x_i + b) y_i \geq \gamma \ \forall i$

pick linear classifier

$\text{maximize}_{\gamma,\mathbf{w},b} \ \gamma$

$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq \gamma, \ \forall j \in \text{Dataset}$

But there are a many planes…

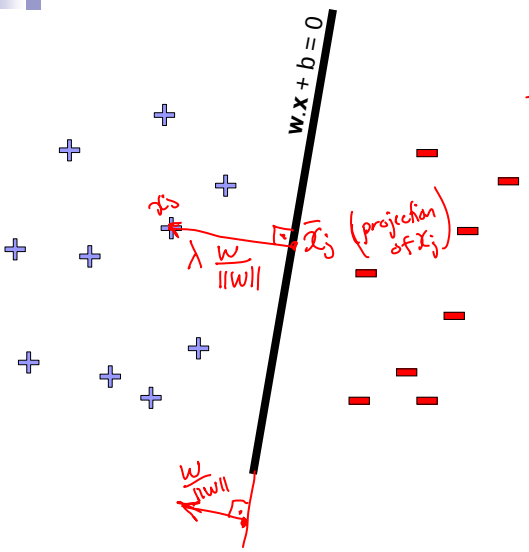


what if:
new plane
$\bar{w} = 2w$
$\bar{b} = 2b$
↳ now $\bar{\gamma} = 2\gamma$

I can be as confident as I want by increasing w

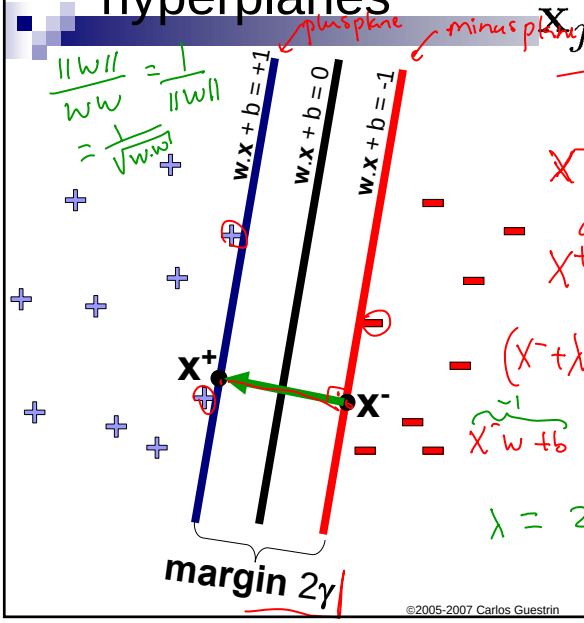
Review: Normal to a plane

$$\mathbf{x}_j = \bar{\mathbf{x}}_j + \lambda_j \frac{\mathbf{w}}{||\mathbf{w}||}$$



Normalized margin – Canonical hyperplanes

$$\mathbf{x}_j = \bar{\mathbf{x}}_j + \lambda \frac{\mathbf{w}}{||\mathbf{w}||}$$



$x^+ = x^- + \lambda \frac{w}{||w||}$

plug

$x^+ w + b = 1$

$(x^- + \lambda \frac{w}{||w||}) \cdot w + b = 1$

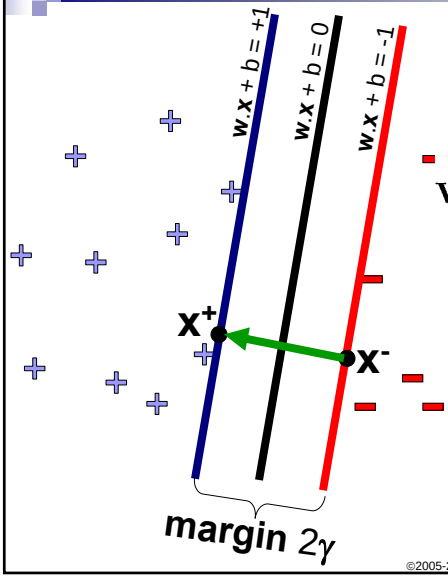
$x^- w + b + \lambda \frac{ww}{||w||} = 1$

$\lambda = 2\gamma$

$\lambda = 2\frac{||w||}{ww} \Rightarrow \gamma = \frac{1}{\sqrt{ww}}$

Normalized margin – Canonical hyperplanes

$$\mathbf{x}^+ = \mathbf{x}^- + \lambda \mathbf{w}$$



$$\mathbf{w}.\mathbf{x}^+ + b = 1$$

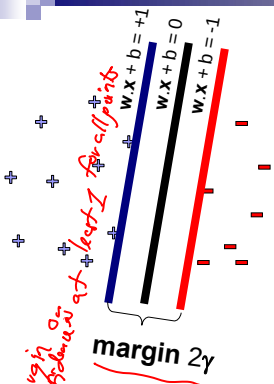
$$\mathbf{w}.(\mathbf{x}^- + \lambda \frac{\mathbf{w}}{||\mathbf{w}||}) + b = 1$$

$$\lambda = \frac{2}{||\mathbf{w}||}$$

$$\gamma = \frac{1}{\sqrt{\mathbf{w}.\mathbf{w}}}$$

Margin maximization using canonical hyperplanes

$$\gamma = \frac{1}{\sqrt{\mathbf{w}.\mathbf{w}}}$$



$$\text{maximize}_{\gamma,\mathbf{w},b} \quad \gamma$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq \gamma, \quad \forall j \in \text{Dataset}$$

$$\max \frac{1}{\sqrt{w.w}} \quad (w.x_j+b)\,y_j \geq \gamma, \quad \gamma \leq 1$$

$$= \min w.w \quad (w.x_j+b)\,y_j \geq \gamma, \quad \gamma \leq 1$$

$$= \min \sqrt{w.w}$$

$$= \min w.w$$

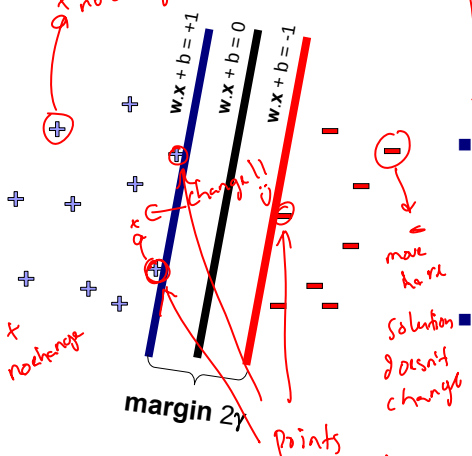
is a monotonic function

maximum achieved when $\gamma = 1$

SVM

$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w}$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1, \quad \forall j \in \text{Dataset}$$

Support vector machines (SVMs)



$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w}$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1, \quad \forall j$$

- Solve efficiently by quadratic programming (QP)
  - Well-studied solution algorithms
- Hyperplane defined by support vectors

points where $(w.x_j+b)\,y_j = 1$ (support vectors)