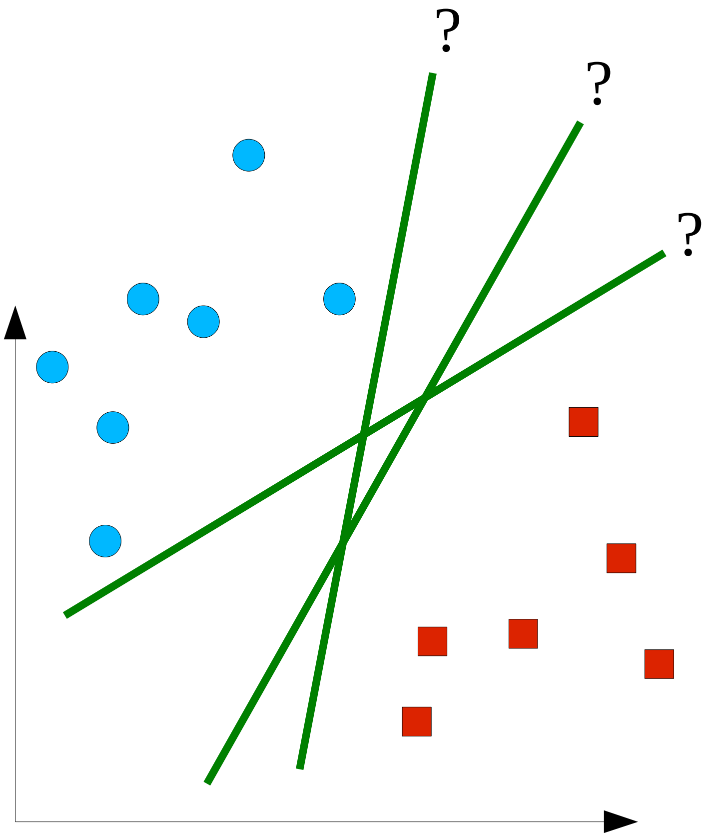
Support Vector Machines

Some material on these is slides borrowed from Andrew Moore's machine learning tutorials located at:

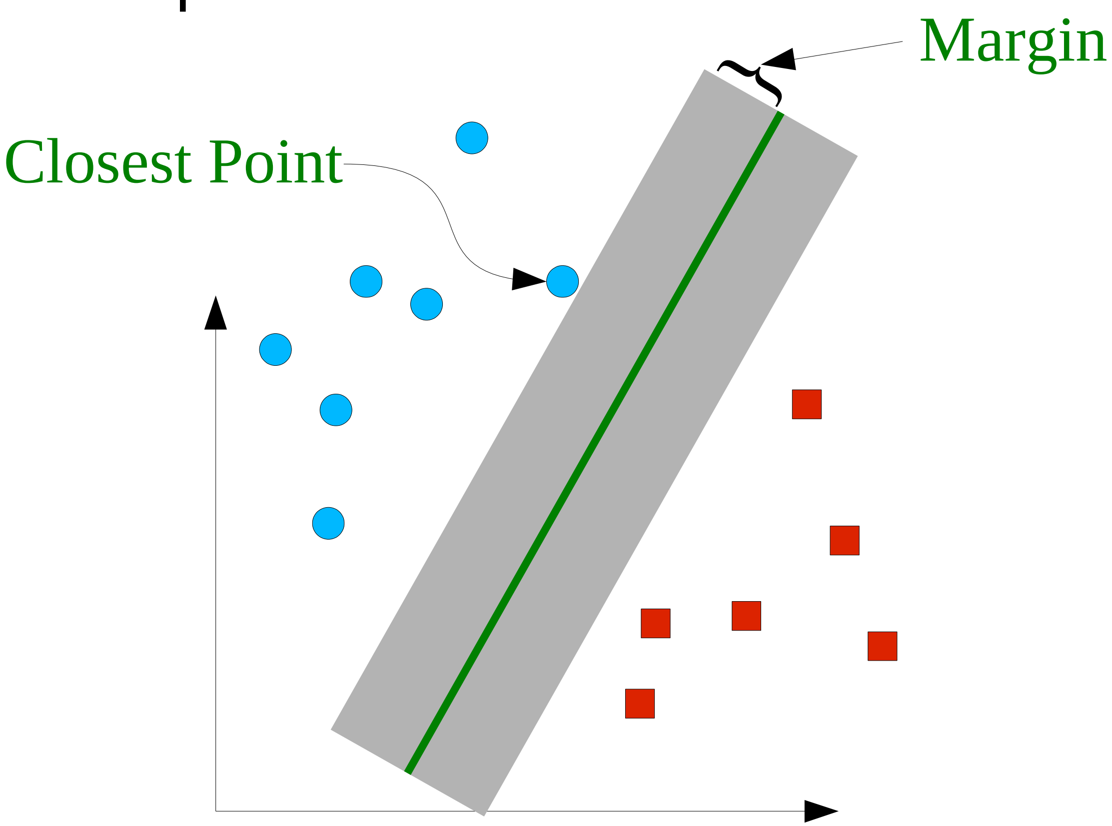
http://www.cs.cmu.edu/~awm/tutorials/

Where Should We Draw the Line?



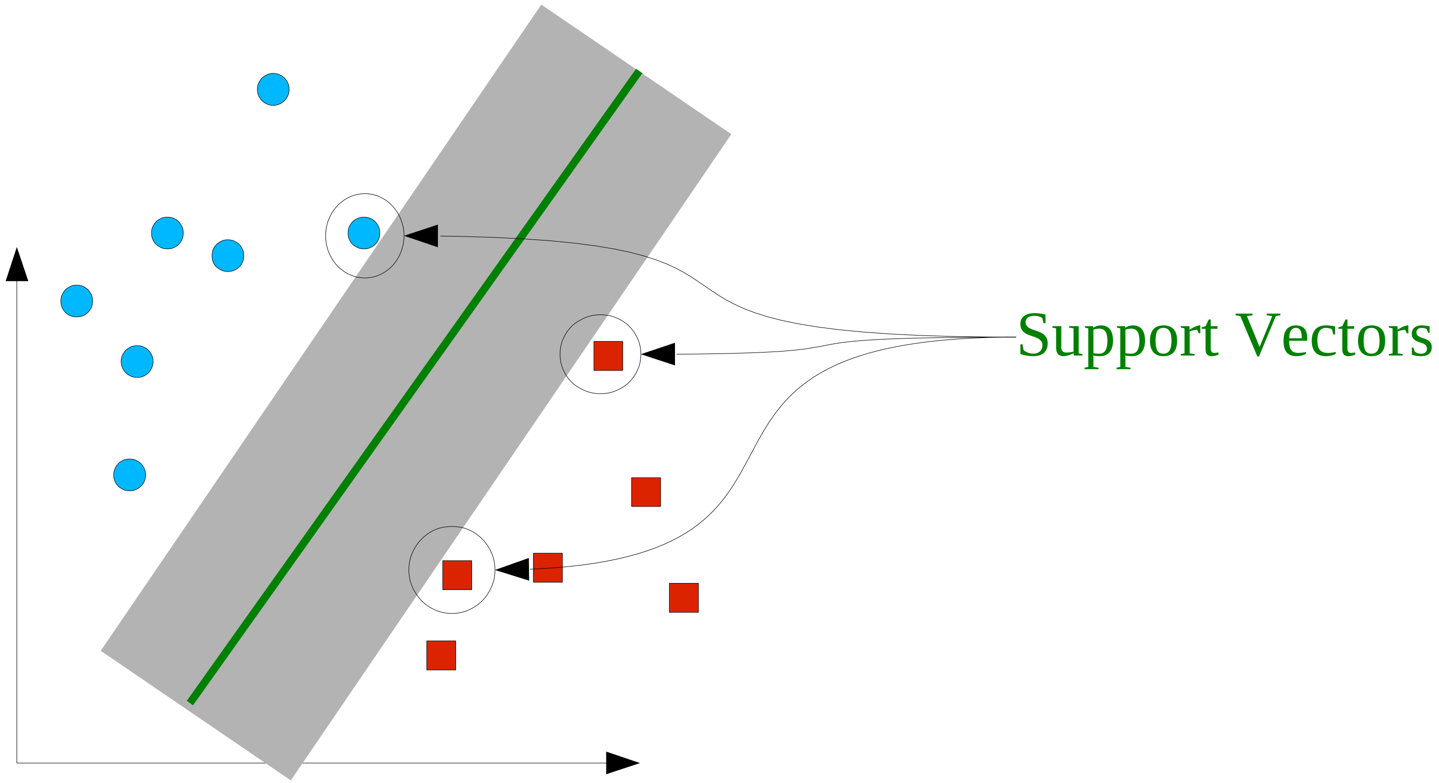
Margins

- Margin – The distance from the decision boundary to the closest point.



Support Vector Machine

- Find the boundary with the maximum margin.
- The points that determine the boundary are the support vectors.



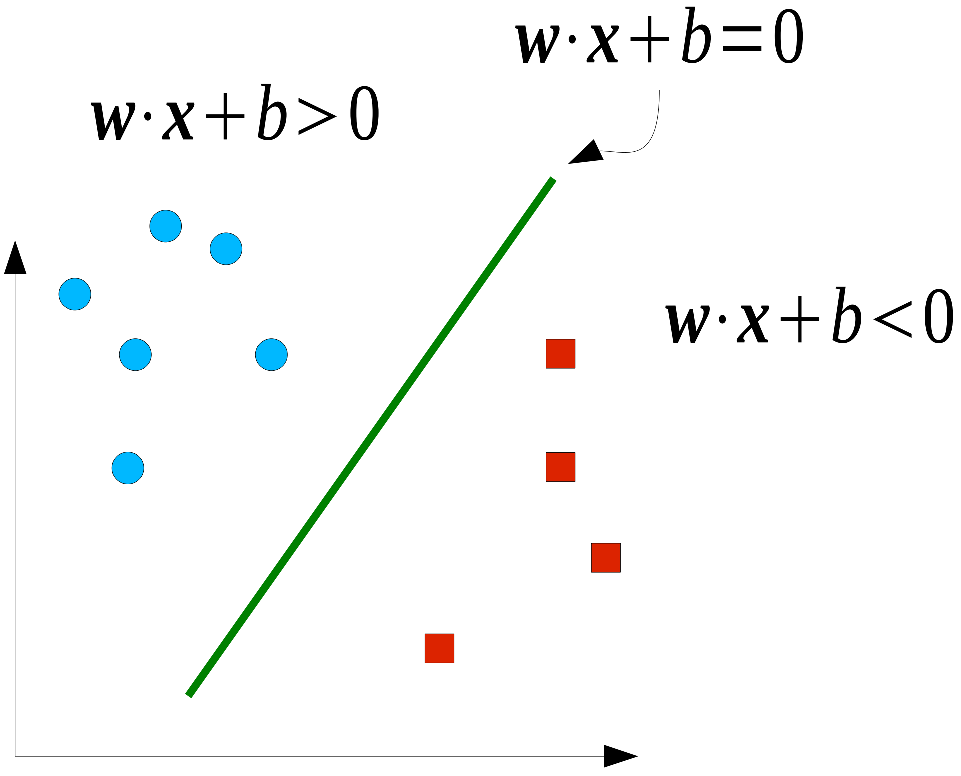
Finding the Boundary…

- The equation for a plane:

$$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$$

- Suppose we have two classes, -1 and 1, we can use this equation for classification: $c(\boldsymbol{x}) = sign(\boldsymbol{w} \cdot \boldsymbol{x} + b)$

Visualizing the Boundary...



Creating A Margin

- Input-output pairs: $(\mathbf{x}_i, t_i)$, $t_i =$ -1 or 1
- We don't just want our samples to be on the right side, we want them to be some distance from the boundary

Instead of this →

$$\mathbf{w} \cdot \mathbf{x_i} + b > 0 \quad for \quad t_i = +1$$
$$\mathbf{w} \cdot \mathbf{x_i} + b < 0 \quad for \quad t_i = -1$$

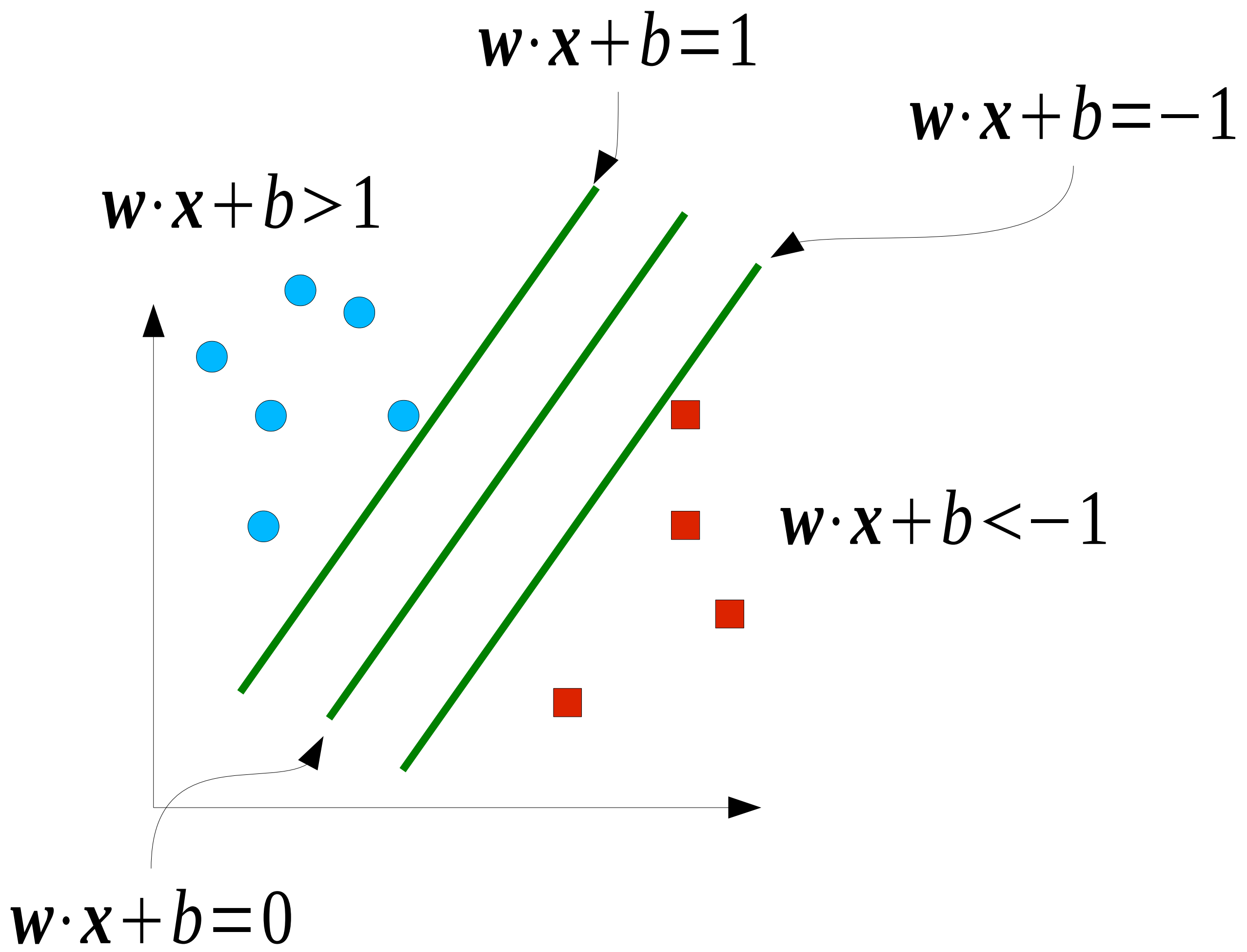
We want this →

$$\mathbf{w} \cdot \mathbf{x_i} + b \geq +1 \quad for \quad t_i = +1$$
$$\mathbf{w} \cdot \mathbf{x_i} + b \leq -1 \quad for \quad t_i = -1$$

Which is the same as this →

$$t_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq +1$$

Two Boundaries...



Minimization

- The distance from a point, $\boldsymbol{x}$, to the boundary can be expressed as:

$$\frac{|\boldsymbol{w} \cdot \boldsymbol{x} + b|}{\|\boldsymbol{w}\|}$$

- This can be maximized by minimizing $||\boldsymbol{w}||$.

- Minimize $\frac{1}{2}\|\boldsymbol{w}\|^2$ subject to $t_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq +1$, for all i.

Determines the size of the margin

Enforces correct classification

Quadratic Programming

- Minimize $\frac{1}{2}\|\boldsymbol{w}\|^2$ subject to $t_i(\boldsymbol{w}\cdot\boldsymbol{x_i}+b)\geq+1$, for all i.

- Minimizing a quadratic function subject to linear constraints… So What?
- This is a (convex) quadratic programming problem.
- What does that mean?
  - No local minima.
  - Good solvers exist.

Lagrange Multipliers

- Minimize $\frac{1}{2}\|\mathbf{w}\|^2$ subject to $t_i(\mathbf{w}\cdot\mathbf{x_i}+b)\geq+1$ for all i.

- Lagrange multipliers are a tool for converting a constrained optimization problem into an unconstrained problem with additional variables...

$$L_P=\frac{1}{2}\|\mathbf{w}\|^2 - \frac{1}{2}\sum_i \lambda_i\left(t_i\left(\mathbf{w}^T\mathbf{x_i}+b\right)-1\right)$$

Dual Formulation

- Maximize:

$$L_D = \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j t_i t_j \left(\mathbf{x_i} \cdot \mathbf{x_j} \right)$$

$$\text{subject to } \lambda_i \geq 0 \text{ and } \sum_i \lambda_i t_i = 0$$

- Once this is done we can get our weights according to:

$$\mathbf{w} = \sum_i \lambda_i t_i \mathbf{x_i}$$

Two Things to Notice

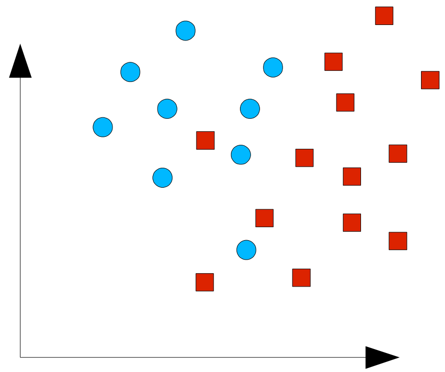
$$\mathbf{w} = \sum_i \lambda_i t_i \mathbf{x_i}$$

- Most of the λ_i will be 0. Those that are non-zero correspond to support vectors.

$$L_D = \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j t_i t_j (\mathbf{x_i} \cdot \mathbf{x_j})$$

- The inputs only show up in the form of dot products.

What About This Case?

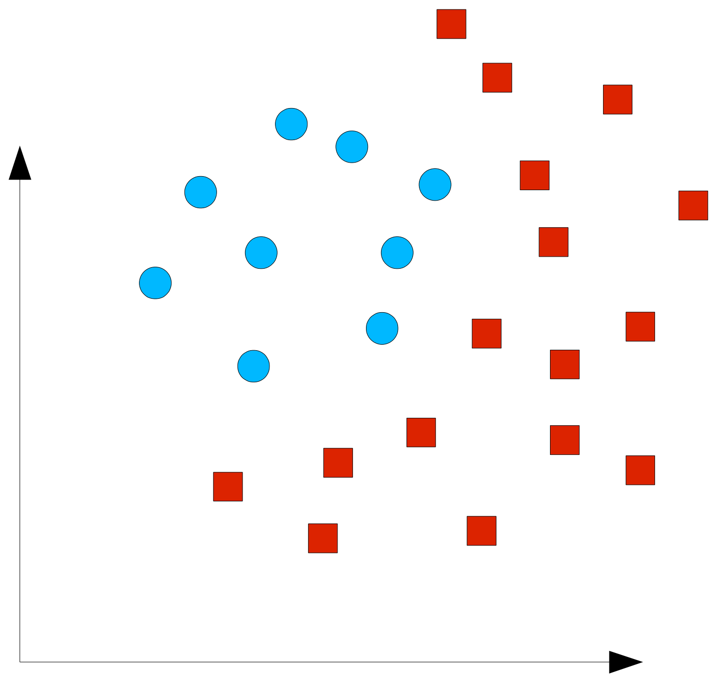


- We can introduce "slack variables" that penalize points on the wrong side of the boundary.
- Good news is that it barely changes the optimization process:

$$L_D = \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j t_i t_j (\boldsymbol{x_i} \cdot \boldsymbol{x_j})$$

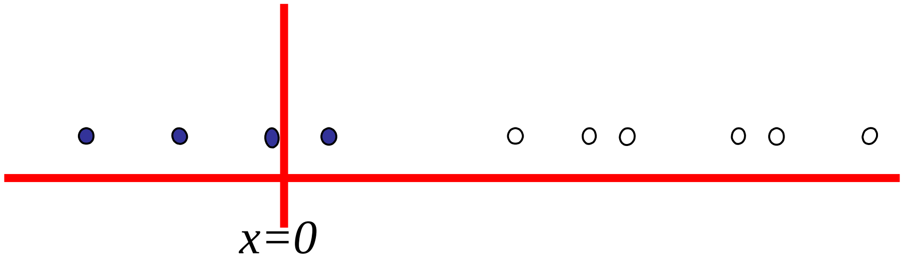
$$\text{subject to } 0 \leq \lambda_i \leq C \text{ and } \sum_i \lambda_i t_i = 0$$

What About This Case?



A 1-D Classification Problem

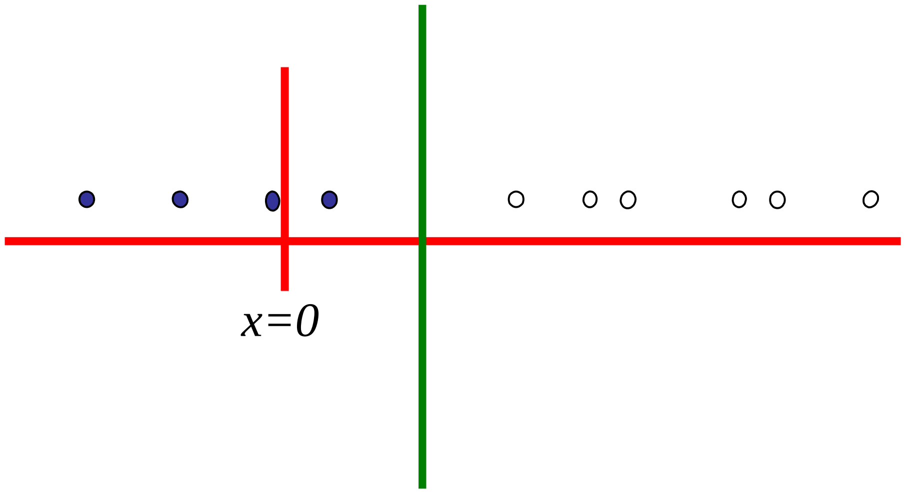
- Where will an SVM put the decision boundary?



http://www.cs.cmu.edu/~awm/tutorials/

1-D Problem Continued

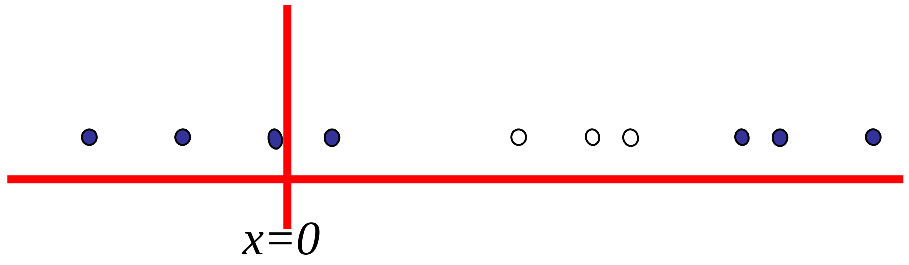
- No problem.
- Equidistant from the two classes.



http://www.cs.cmu.edu/~awm/tutorials/

The Non-Separable Case

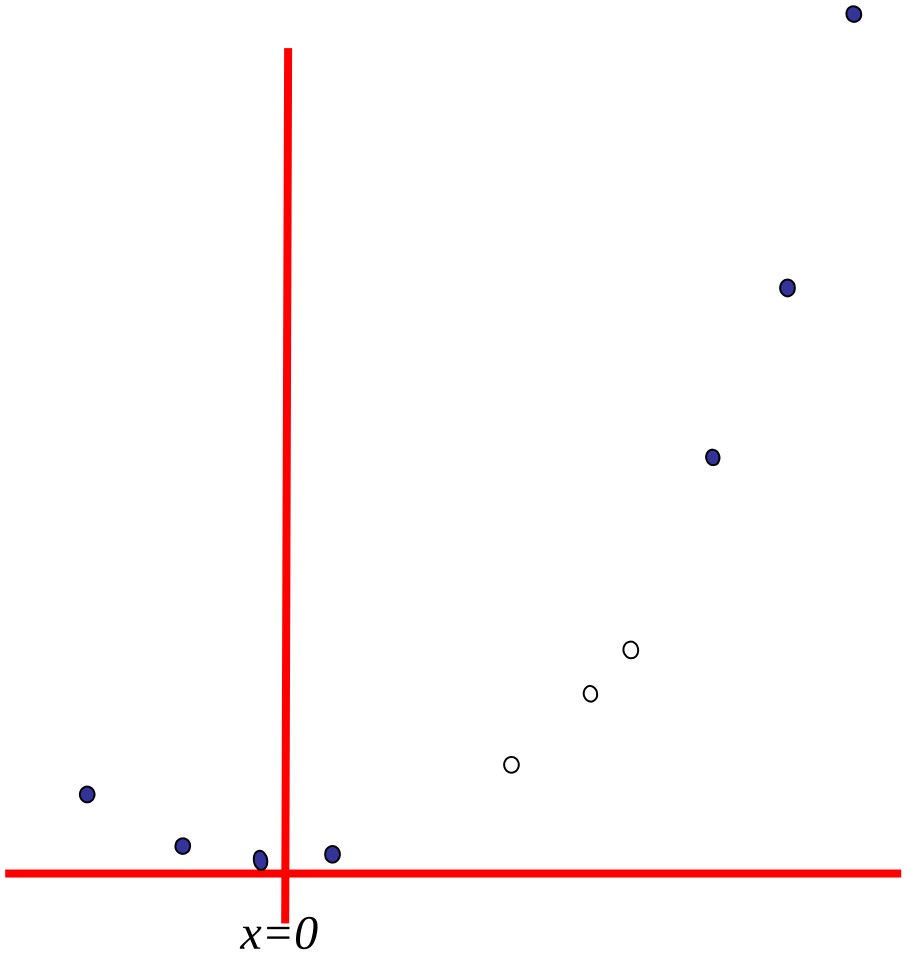
- Now we have a problem...



http://www.cs.cmu.edu/~awm/tutorials/

Increase the Dimensionality

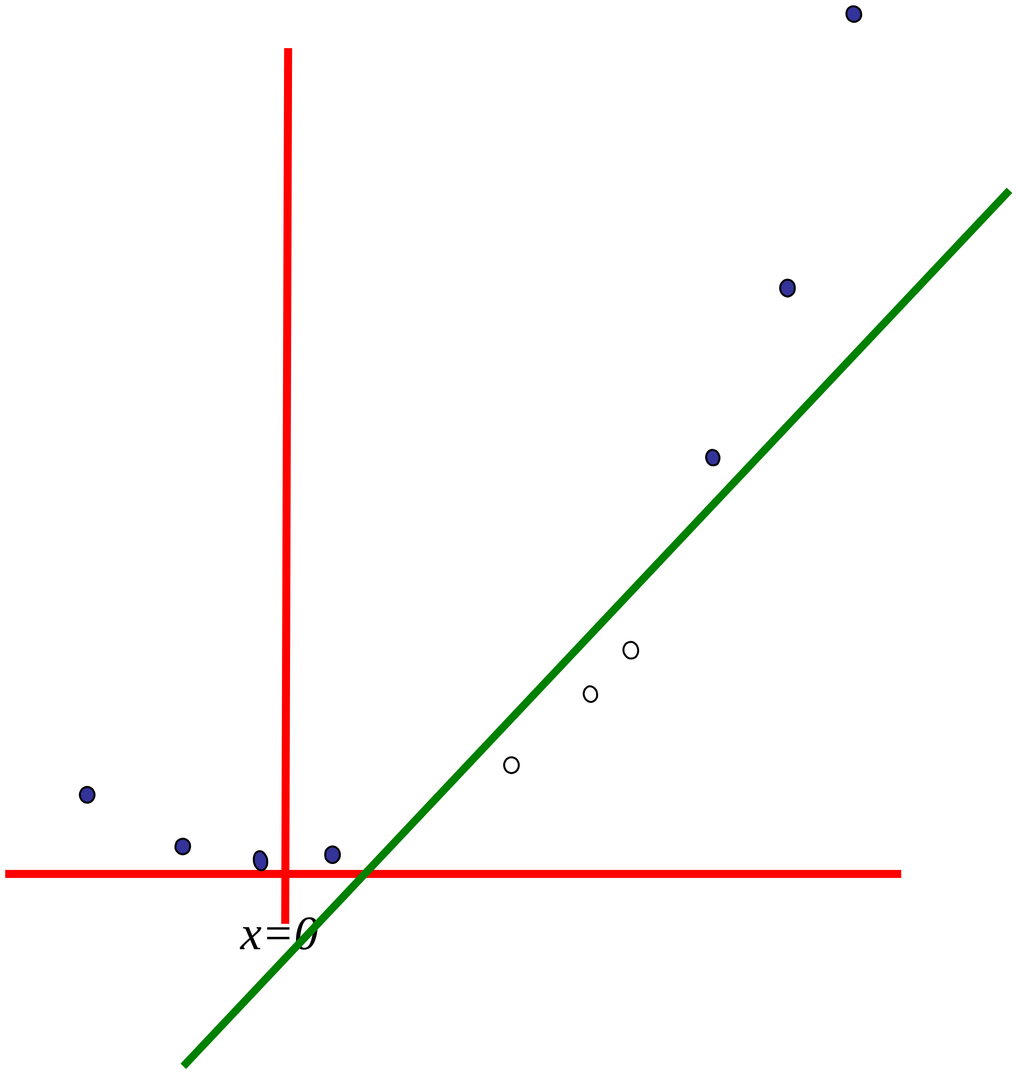
- Use our old data points x_i to create a new set of data points $\boldsymbol{z}_i$.
- $\boldsymbol{z}_i = (\boldsymbol{x}_i\,,\ \boldsymbol{x}_i^2)$



http://www.cs.cmu.edu/~awm/tutorials/

Increase the Dimensionality

- Now the data is separable.



http://www.cs.cmu.edu/~awm/tutorials/

The Blessing of Dimensionality (?)

- This works in general.
- When you increase the dimensionality of your data, you increase the chance that it will be linearly separable.
- In an N-1 dimensional space you should always be able to separate N data points. (Unless you are unlucky.)

Let's do it!

- Define a function $\phi(\boldsymbol{x})$ that maps our low dimensional data into a very high dimensional space.
- Now we can just rewrite our optimization to use these high dimensional vectors:

$$L_D = \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j t_i t_j [\phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j})]$$

$$\text{subject to } 0 \leq \lambda_i \leq C \text{ and } \sum_i \lambda_i t_i = 0$$

- What's the problem?

The Kernel Trick

- It turns out we can often find a kernel function *K* such that: $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \phi(x_i) \cdot \phi(x_j)$
- In fact, almost any kernel function corresponds to a dot product in *some* space.
- Now we have:

$$L_D = \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j t_i t_j K(\boldsymbol{x_i}, \boldsymbol{x_j})$$

$$\text{subject to } 0 \leq \lambda_i \leq C \text{ and } \sum_i \lambda_i t_i = 0$$

- Support vector machines are also called kernel machines.

The Kernel Trick

- We get to perform classification in very high dimensional spaces for almost no additional cost.
- Some Kernels:
  - Polynomial: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i \cdot \boldsymbol{x}_j + 1)^q$
  - Radial Basis Function: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left[\frac{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{\sigma^2}\right]$
  - Sigmoidal: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(2\,\boldsymbol{x}_i \cdot \boldsymbol{x}_j + 1)$

Nice Things about SVM's

- Good generalization because of margin maximization.
- Not many parameters to pick.
  - No learning rate, no hidden layer size.
  - Just C, and possibly some parameters for kernel function.
  - You also have to pick a kernel function.
- No problems with local minima.
- What about SVM regression? It's possible, but we won't talk about it.