# Large-Scale Machine Learning: SVM

Mining of Massive Datasets
Jure Leskovec, Anand Rajaraman, Jeff Ullman
Stanford University
http://www.mmds.org

# Supervised Learning

- **Example: Spam filtering**

| | viagra | learning | the | dating | nigeria | *spam?* |
|---|---|---|---|---|---|---|
| $\vec{x}_1 = ($ | 1 | 0 | 1 | 0 | 0 $)$ | $y_1 = 1$ |
| $\vec{x}_2 = ($ | 0 | 1 | 1 | 0 | 0 $)$ | $y_2 = -1$ |
| $\vec{x}_3 = ($ | 0 | 0 | 0 | 0 | 1 $)$ | $y_3 = 1$ |

- **Instance space x ∈ X** (|**X**|= **n** data points)
  - **Binary or real-valued feature vector *x* of word occurrences**
  - ***d*** features (words + other things, **d**~100,000)
- **Class y ∈ Y**
  - ***y***: Spam (+1), Ham (-1)
- **Goal: Estimate** a function **f(x)** so that ***y = f(x)***

# More generally: Supervised Learning

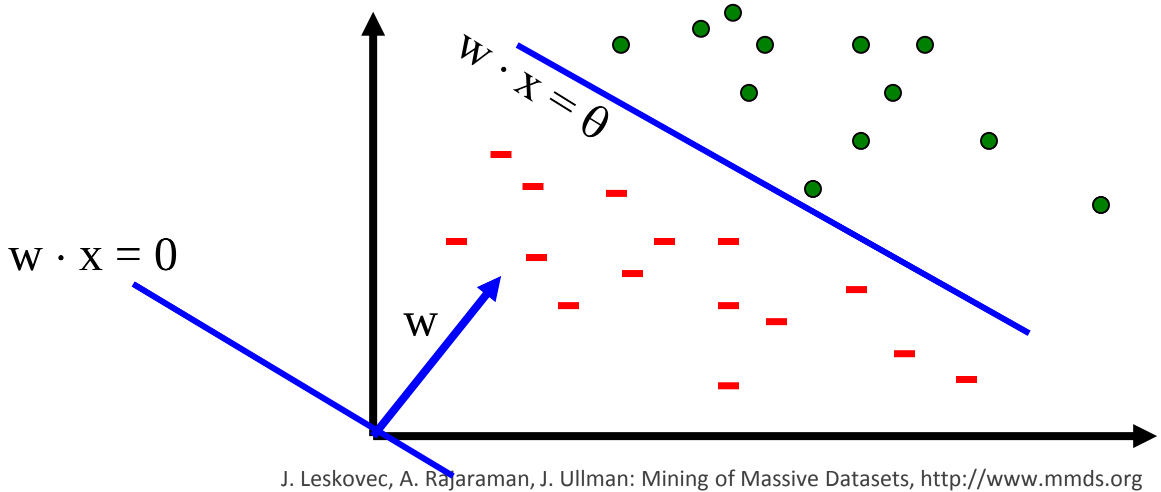
- **Would like to do prediction:**
  **estimate** a function **f(x)** so that ***y = f(x)***
- **Where *y* can be:**
  - **Real number**: Regression
  - **Categorical: Classification**
  - **Complex object:**
    - Ranking of items, Parse tree, etc.
- **Data is labeled:**
  - Have many pairs **{(x, y)}**
    - **x** … vector of binary, categorical, real valued features
    - **y** … class ({+1, -1}, or a real number)

# Linear models for classification

- **Binary classification:**

$$f(\mathbf{x}) = \begin{cases} +1 & \text{if} \quad \mathbf{w}^{(1)}\mathbf{x}^{(1)} + \mathbf{w}^{(2)}\mathbf{x}^{(2)} + \ldots \mathbf{w}^{(d)}\mathbf{x}^{(d)} \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

- **Input:** Vectors $\boldsymbol{x_j}$ and labels $\boldsymbol{y_j}$
  - Vectors $\boldsymbol{x_j}$ are real valued where $\|x\|_2 = 1$
- **Goal:** Find vector $\boldsymbol{w} = (w^{(1)}, w^{(2)}, \ldots, w^{(d)})$
  - Each $\boldsymbol{w^{(i)}}$ is a real number



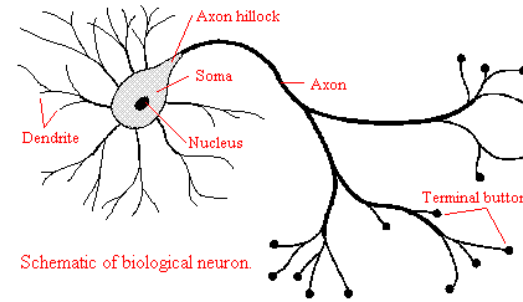

Decision boundary is **linear**

**Note:**

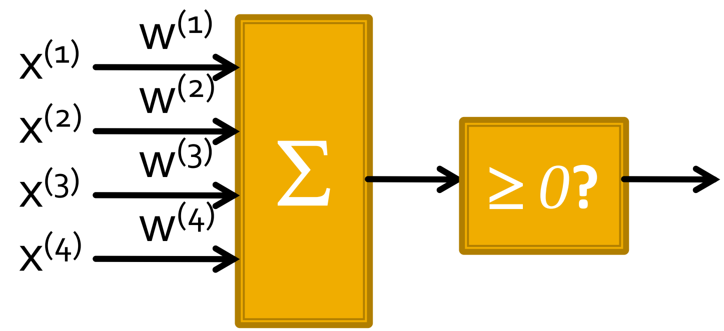
$$\mathbf{x} \rightarrow \langle \mathbf{x}, 1 \rangle \quad \forall \mathbf{x}$$

$$\mathbf{w} \rightarrow \langle \mathbf{w}, -\theta \rangle$$

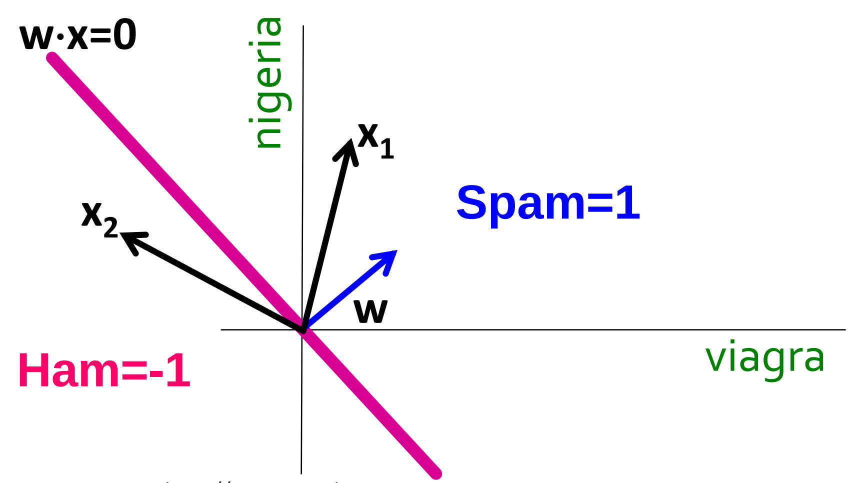
# Perceptron [Rosenblatt '58]




- **(Very) loose motivation: Neuron**
- Inputs are feature values
- Each feature has a weight $w_i$
- **Activation is the sum:**
  - $f(x) = \sum_i^d w^{(i)} x^{(i)} = w \cdot x$




- If the ***f(x)*** is:
  - **Positive:** Predict **+1**
  - **Negative:** Predict **-1**



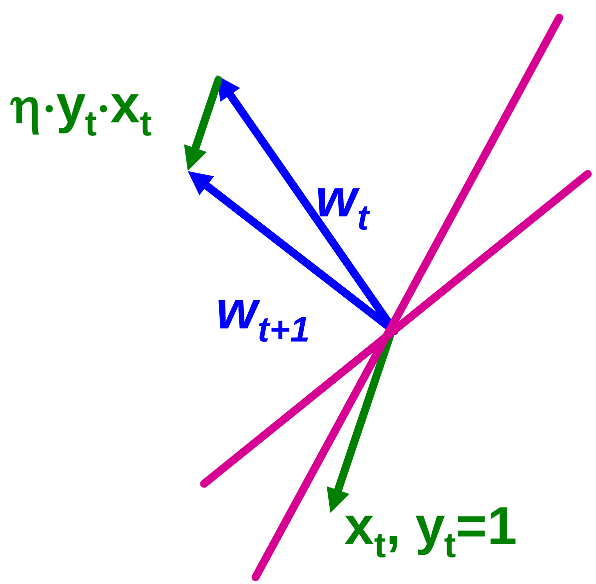
# Perceptron

Note that the Perceptron is a conservative algorithm: it ignores samples that it classifies correctly.

- **Perceptron:** $y' = sign(w \cdot x)$
- **How to find parameters $w$?**
  - Start with $w_0 = 0$
  - Pick training examples $x_t$ **one by one**
  - Predict class of $x_t$ using current $w_t$
    - $y' = sign(w_t \cdot x_t)$
  - **If $y'$ is correct (i.e., $y_t = y'$)**
    - No change: $w_{t+1} = w_t$
  - **If $y'$ is wrong:** Adjust $w_t$

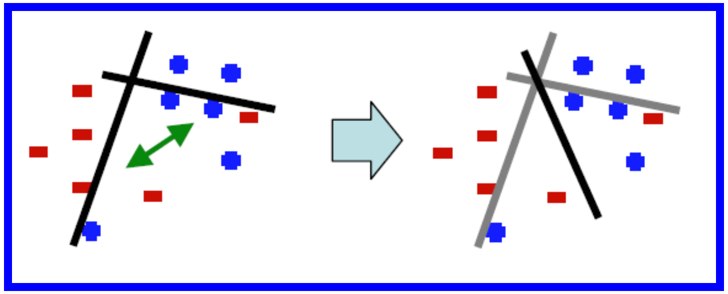
$$w_{t+1} = w_t + \eta \cdot y_t \cdot x_t$$

- $\eta$ is the learning rate parameter
- $x_t$ is the t-th training example
- $y_t$ is true t-th class label ({+1, -1})

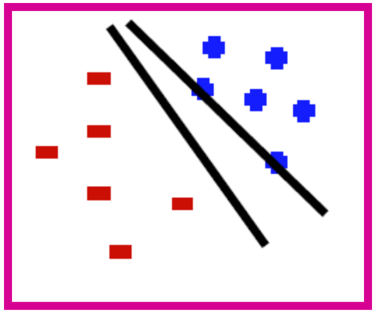


# Perceptron: The Good and the Bad

- **Good: Perceptron convergence theorem:**
  - If there exist a set of weights that are consistent (i.e., the data is linearly separable) the Perceptron learning algorithm will converge
- **Bad: Never converges:**
  If the data is not separable weights dance around indefinitely



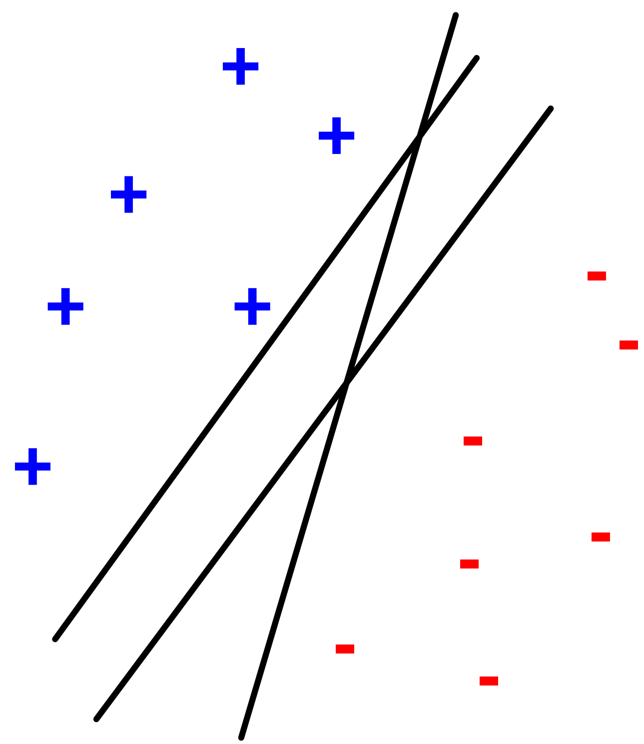
- **Bad: Mediocre generalization:**
  - Finds a "barely" separating solution



# Support Vector Machines

# Support Vector Machines

- Want to separate "+" from "-" using a line



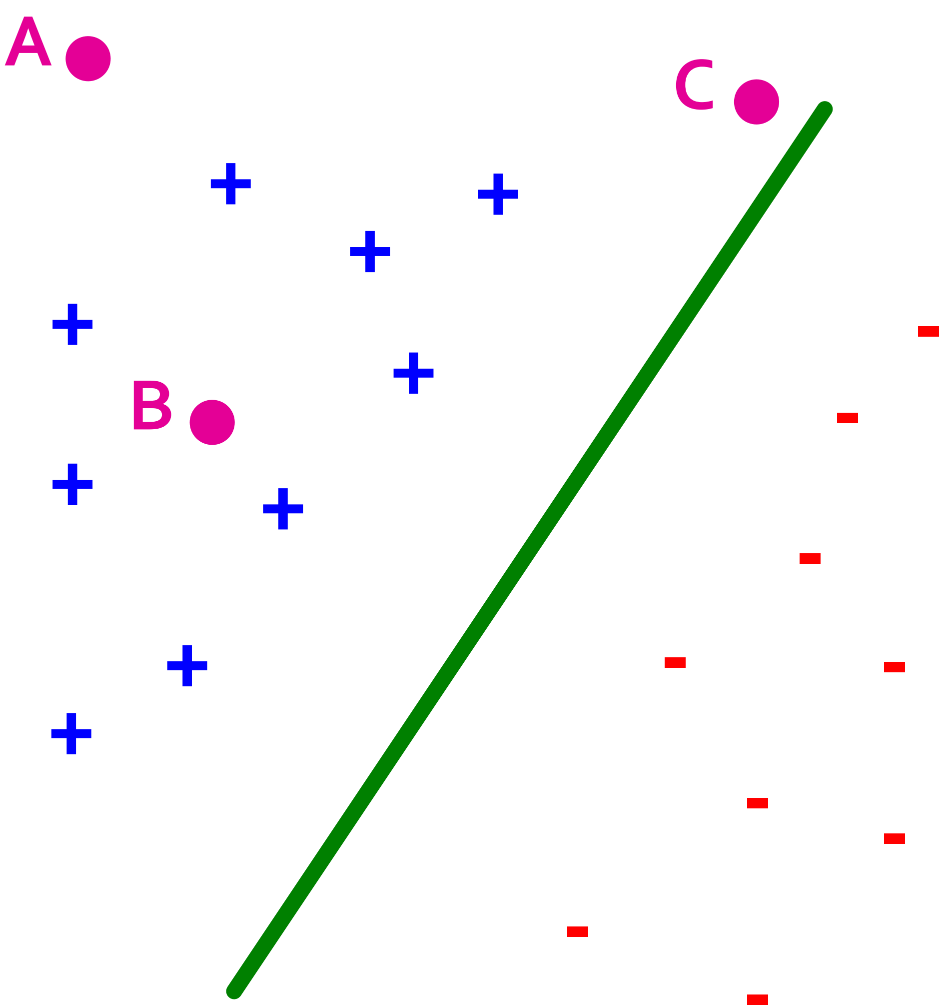

**Data:**

- **Training examples:**
  - $(x_1, y_1) \dots (x_n, y_n)$
- **Each example *i*:**
  - $x_i = ( x_i^{(1)}, \dots , x_i^{(d)} )$
    - $x_i^{(j)}$ is real valued
  - $y_i \in \{ -1, +1 \}$
- **Inner product:**

$$\boldsymbol{w} \cdot \boldsymbol{x} = \sum_{j=1}^{d} w^{(j)} \cdot x^{(j)}$$

**Which is best linear separator (defined by *w*)?**

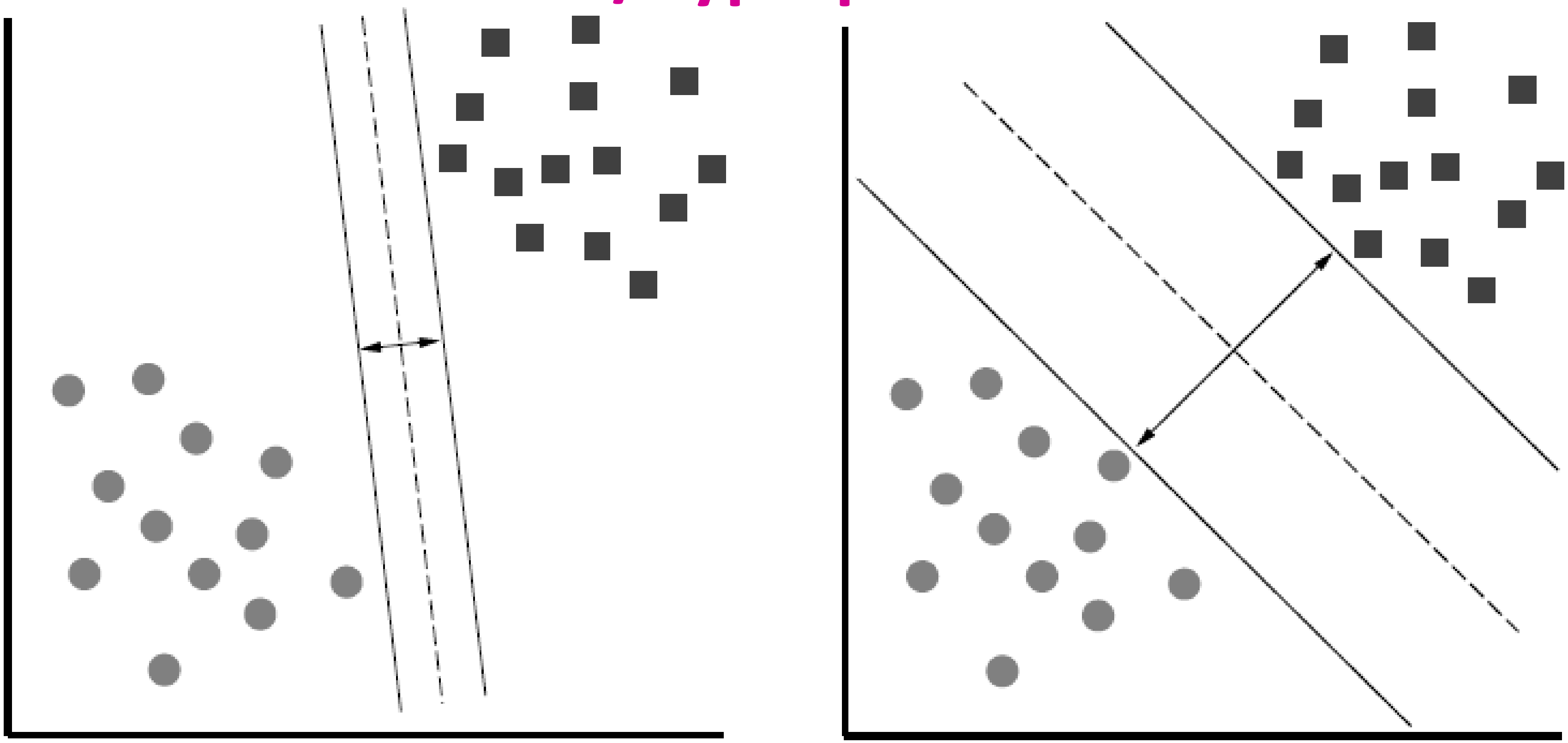
# Largest Margin




- **Distance from the separating hyperplane corresponds to the "confidence" of prediction**
- **Example:**
  - We are more sure about the class of **A** and **B** than of **C**

# Largest Margin

- **Margin $\gamma$: Distance of closest example from the decision line/hyperplane**



The reason we define margin this way is due to theoretical convenience and existence of generalization error bounds that depend on the value of margin.

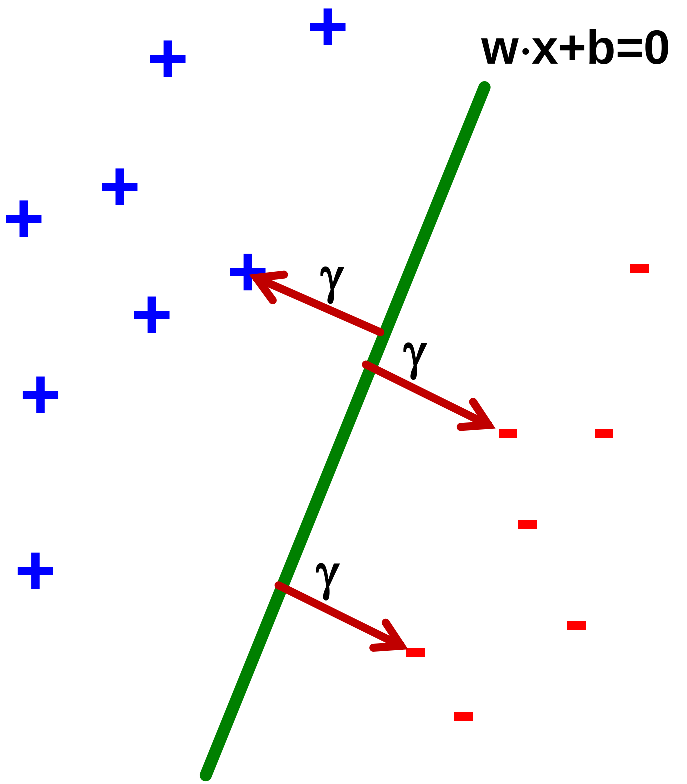
# Support Vector Machine

- **Maximize the margin:**
  - **Good according to intuition, theory (VC dimension) & practice**

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

$$s.t. \forall i, y_i(w \cdot x_i + b) \geq \gamma$$

  - **$\gamma$ is margin … distance from the separating hyperplane**

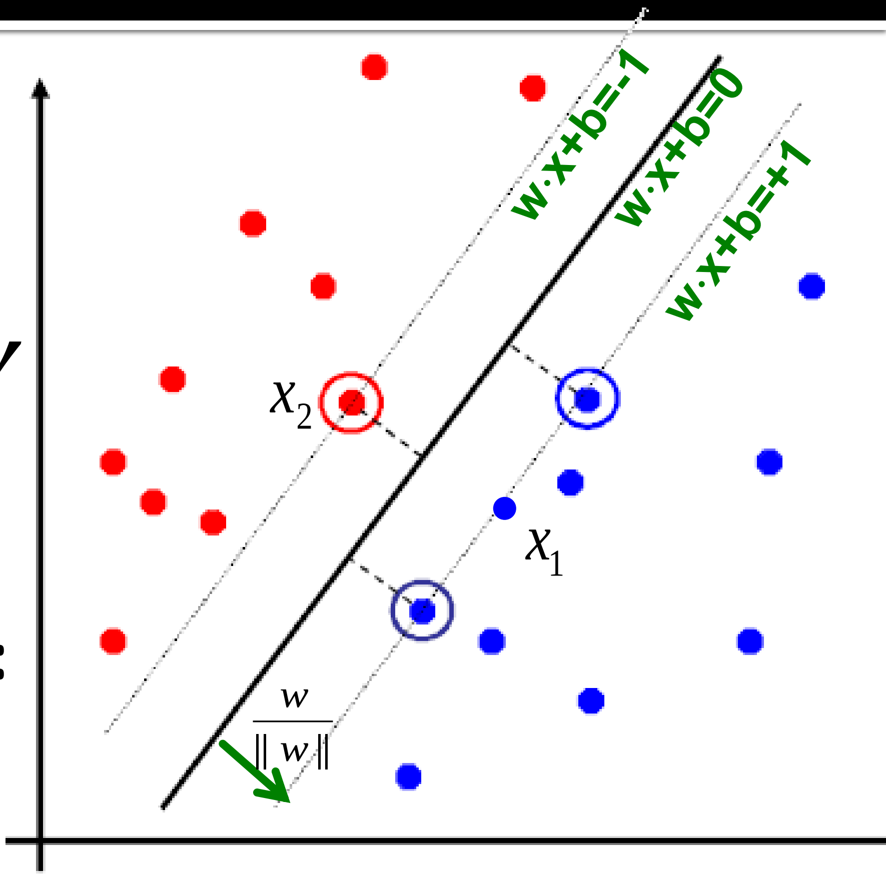



Maximizing the margin

# Canonical Hyperplane: Problem

- **Problem:**
  - Let $(\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b})\boldsymbol{y} = \boldsymbol{\gamma}$ then $(\mathbf{2}\boldsymbol{w} \cdot \boldsymbol{x} + \mathbf{2}\boldsymbol{b})\boldsymbol{y} = \mathbf{2}\boldsymbol{\gamma}$
    - Scaling ***w*** increases margin!
- **Solution:**
  - **Work with normalized *w*:**

$$\boldsymbol{\gamma} = \left(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \cdot \boldsymbol{x} + \boldsymbol{b}\right)\boldsymbol{y}$$

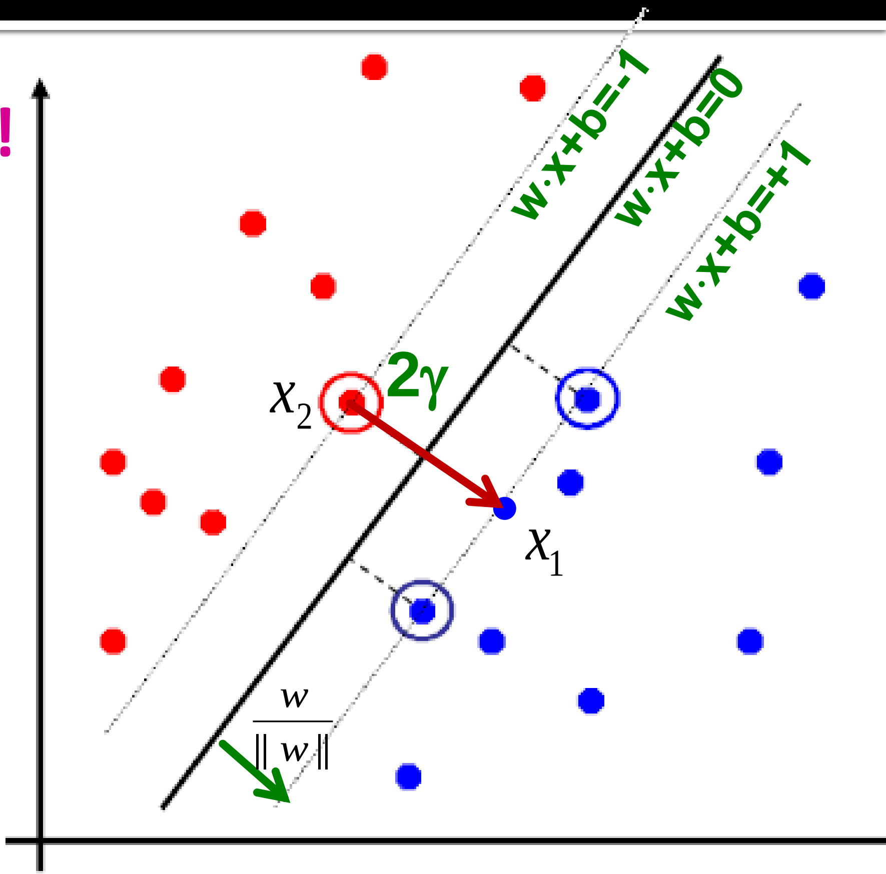



  - Let's also require **support vectors $\boldsymbol{x_j}$** to be on the plane defined by: $\boldsymbol{w} \cdot \boldsymbol{x_j} + \boldsymbol{b} = \pm \mathbf{1}$

$$||w|| = \sqrt{\sum_{j=1}^{d} (w^{(j)})^2}$$

# Canonical Hyperplane: Solution

- **Want to maximize margin $\gamma$!**
- **What is the relation between $x_1$ and $x_2$?**
  - $x_1 = x_2 + 2\gamma \frac{w}{||w||}$
  - **We also know:**
    - $w \cdot x_1 + b = +1$
    - $w \cdot x_2 + b = -1$
- **So:**
  - $w \cdot x_1 + b = +1$
  - $w\left(x_2 + 2\gamma \frac{w}{||w||}\right) + b = +1$
  - $\underbrace{w \cdot x_2 + b}_{-1} + 2\gamma \frac{w \cdot w}{||w||} = +1$




$$\Rightarrow \gamma = \frac{\|w\|}{w \cdot w} = \frac{1}{\|w\|}$$

**Note:** $w \cdot w = \|w\|^2$

# Maximizing the Margin

- We started with

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

$$s.t. \forall i, y_i (w \cdot x_i + b) \geq \gamma$$

**But *w* can be arbitrarily large!**

- We normalized and...

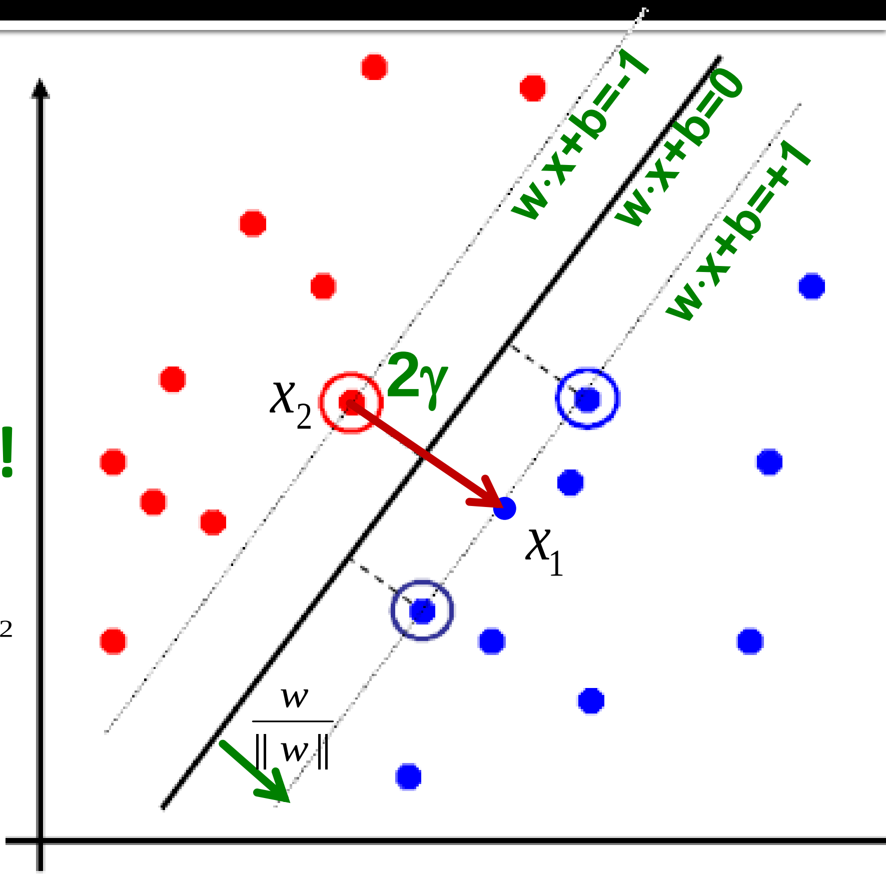
$$\arg\max \gamma = \arg\max \frac{1}{\|w\|} = \arg\min \|w\| = \arg\min \tfrac{1}{2}\|w\|^2$$

- Then:

$$\min_w \tfrac{1}{2} \| w \|^2$$

$$s.t. \forall i, y_i (w \cdot x_i + b) \geq 1$$

**This is called SVM with "hard" constraints**

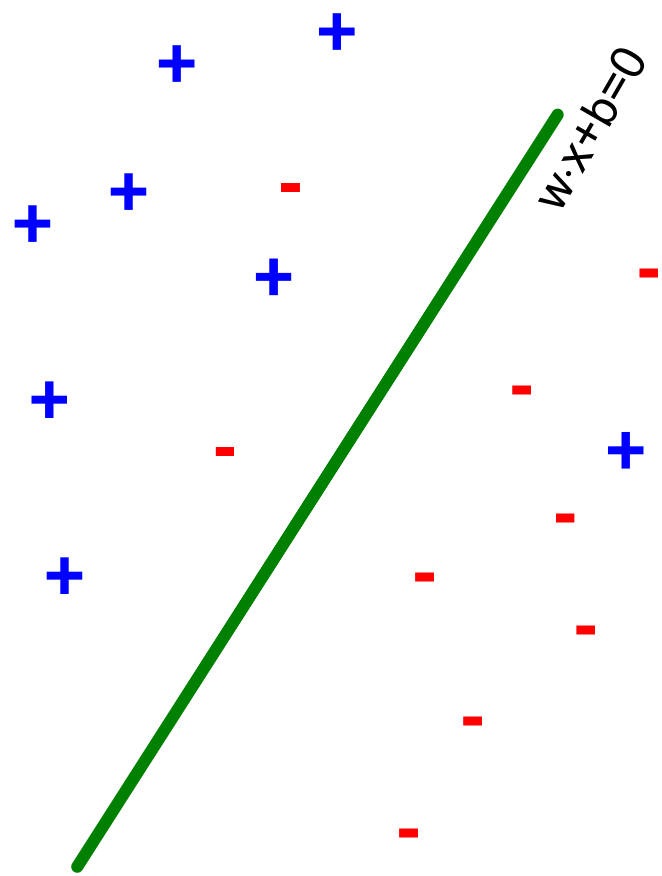


# Non-linearly Separable Data

- **If data is not separable introduce penalty:**

$$\min_w \tfrac{1}{2}\|w\|^2 + C \cdot (\#\text{number of mistakes})$$
$$s.t. \forall i, y_i (w \cdot x_i + b) \geq 1$$

  - Minimize $\|w\|^2$ plus the number of training mistakes
  - Set *C* using cross validation




- **How to penalize mistakes?**
  - **All mistakes are not equally bad!**

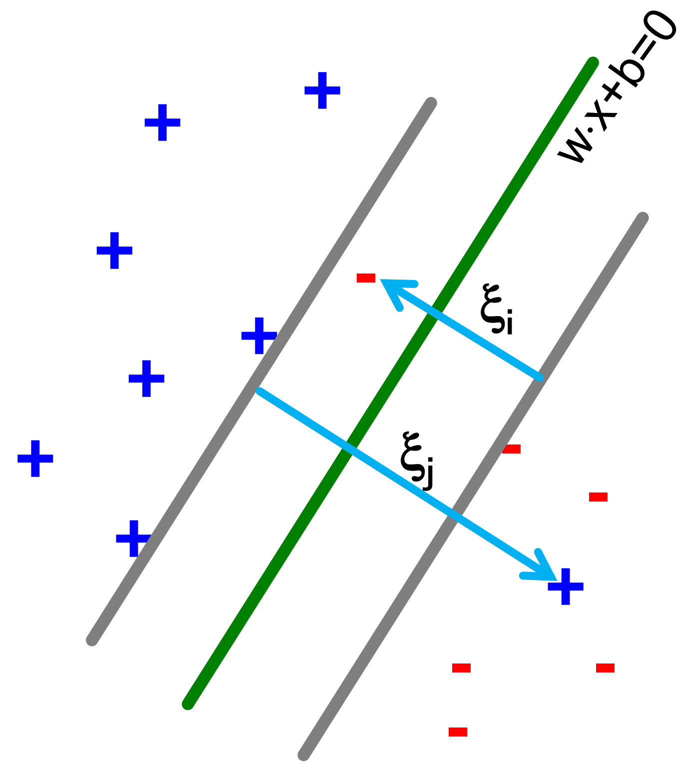
# Support Vector Machines

- **Introduce** **slack variables** **$\xi_i$**

$$\min_{w,b,\xi_i \geq 0} \ \tfrac{1}{2}\|w\|^2 + C \cdot \sum_{i=1}^{n} \xi_i$$

$$s.t. \forall i, y_i (w \cdot x_i + b) \geq 1 - \xi_i$$

- If point $\boldsymbol{x_i}$ is on the wrong side of the margin then get penalty $\xi_i$



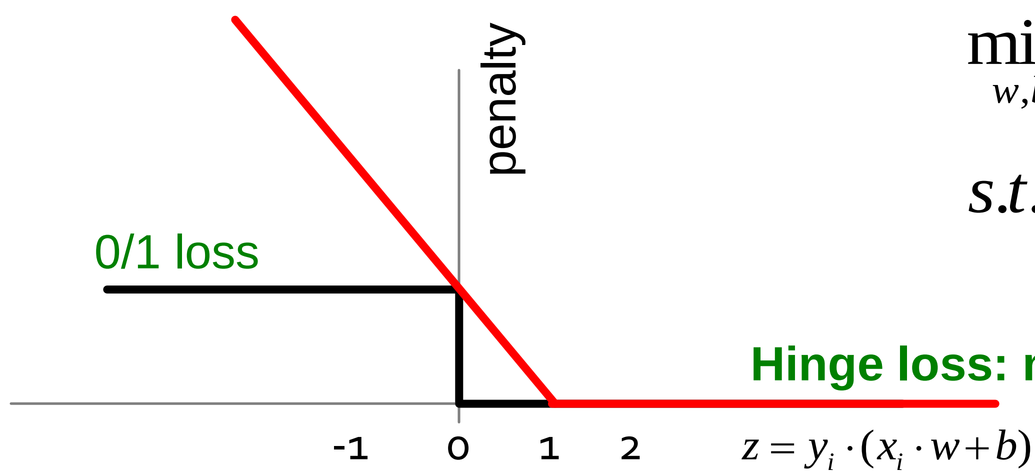

**For each data point:**
If margin $\geq$ 1, don't care
If margin < 1, pay linear penalty

# Support Vector Machines

- **SVM in the "natural" form**

$$\arg\min_{w,b} \underbrace{\tfrac{1}{2}\, w \cdot w}_{\text{Margin}} + \underset{\substack{\uparrow\\ \text{Regularization parameter}}}{C} \cdot \underbrace{\sum_{i=1}^{n} \max\left\{0, 1 - y_i (w \cdot x_i + b)\right\}}_{\text{Empirical } \textbf{loss L} \text{ (how well we fit training data)}}$$

- **SVM uses "Hinge Loss":**




$$\min_{w,b} \tfrac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

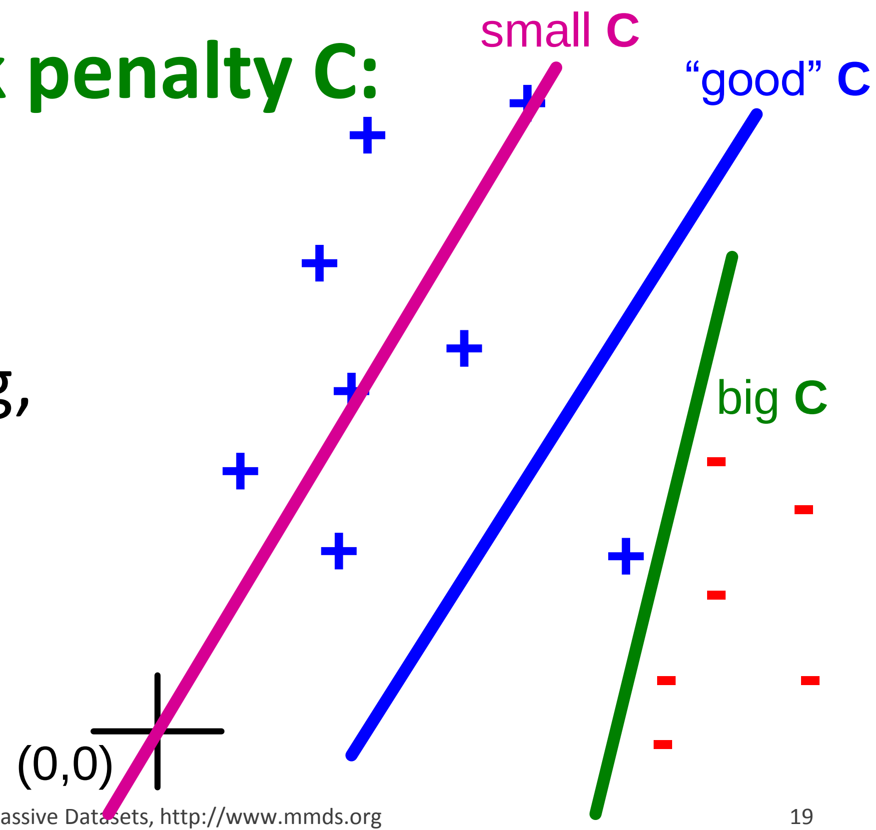
$$s.t. \forall i, y_i \cdot (w \cdot x_i + b) \geq 1 - \xi_i$$

# Slack Penalty $C$

$$\min_{w,b,\xi_i \geq 0} \ \tfrac{1}{2}\|w\|^2 + C \cdot \sum_{i=1}^{n} \xi_i$$

$$s.t.\forall i, y_i(w \cdot x_i + b) \geq 1 - \xi_i$$

- **What is the role of slack penalty C:**
  - **C=∞:** Only want to **w, b** that separate the data
  - **C=0:** Can set $\xi_i$ to anything, then **w=0** (basically ignores the data)



# SVM: How to estimate *w*?

$$\min_{w,b} \ \tfrac{1}{2} w \cdot w + C \cdot \sum_{i=1}^{n} \xi_i$$

$$s.t. \forall i, y_i \cdot (x_i \cdot w + b) \geq 1 - \xi_i$$

- **Want to estimate *w* and *b*!**
  - **Standard way:** Use a solver!
    - **Solver:** software for finding solutions to "common" optimization problems
- **Use a quadratic solver:**
  - Minimize quadratic function
  - Subject to linear constraints
- **Problem:** Solvers are inefficient for big data!

# SVM: How to estimate *w*?

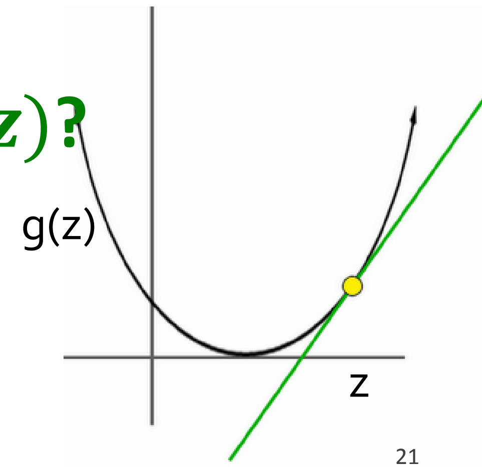
$$\min_{w,b} \ \tfrac{1}{2} w \cdot w + C \sum_{i=1}^{n} \xi_i$$

$$s.t. \forall i, y_i \cdot (x_i \cdot w + b) \geq 1 - \xi_i$$

- **Want to estimate w, b!**
- **Alternative approach:**
  - **Want to minimize *f(w,b)*:**

$$f(w,b) = \tfrac{1}{2} w \cdot w + C \cdot \sum_{i=1}^{n} \max\left\{ 0, 1 - y_i \left( \sum_{j=1}^{d} w^{(j)} x_i^{(j)} + b \right) \right\}$$

- **Side note:**
  - **How to minimize convex functions $g(z)$?**
  - Use gradient descent: $\mathbf{\min_z g(z)}$
  - Iterate: $\mathbf{z_{t+1} \leftarrow z_t - \eta \nabla g(z_t)}$



# SVM: How to estimate *w*?

- **Want to minimize *f(w,b)*:**

$$f(w,b) = \tfrac{1}{2}\sum_{j=1}^{d}\left(w^{(j)}\right)^2 + C\sum_{i=1}^{n}\underbrace{\max\left\{0, 1 - y_i\left(\sum_{j=1}^{d} w^{(j)} x_i^{(j)} + b\right)\right\}}_{\text{Empirical loss } L(x_i\, y_i)}$$

- **Compute the gradient ∇(j) w.r.t. $w^{(j)}$**

$$\nabla f^{(j)} = \frac{\partial f(w,b)}{\partial w^{(j)}} = w^{(j)} + C\sum_{i=1}^{n}\frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$

$$\frac{\partial L(x_i, y_i)}{\partial w^{(j)}} = 0 \qquad \text{if } y_i(\mathrm{w}\cdot x_i + b) \geq 1$$

$$= -y_i x_i^{(j)} \qquad \text{else}$$

# SVM: How to estimate *w*?

- **Gradient descent:**

**Iterate until convergence:**
- **For j = 1 … d**
  - **Evaluate:** $\nabla f^{(j)} = \frac{\partial f(w,b)}{\partial w^{(j)}} = w^{(j)} + C\sum_{i=1}^{n}\frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$
  - **Update:**
    $w^{(j)} \leftarrow w^{(j)} - \eta \nabla f^{(j)}$

$\eta$…learning rate parameter
C… regularization parameter

- **Problem:**
  - **Computing $\nabla f^{(j)}$ takes O(n) time!**
    - **n** … size of the training dataset

# SVM: How to estimate *w*?

We just had:

$$\nabla f^{(j)} = w^{(j)} + C\sum_{i=1}^{n} \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$

- **Stochastic Gradient Descent**
  - Instead of evaluating gradient over all examples evaluate it for each **individual** training example

$$\nabla f^{(j)}(x_i) = w^{(j)} + C \cdot \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$

Notice: no summation over ***i*** anymore

- **Stochastic gradient descent:**

**Iterate until convergence:**
- **For i = 1 … n**
  - **For j = 1 … d**
    - **Compute: $\nabla f^{(j)}(x_i)$**
    - **Update: $w^{(j)} \leftarrow w^{(j)} - \eta \nabla f^{(j)}(x_i)$**

# Example: Text categorization

- **Example by Leon Bottou:**
  - **Reuters RCV1** document corpus
    - Predict a category of a document
      - One **vs.** the rest classification
  - ***n* = 781,000** training examples (documents)
  - 23,000 test examples
  - ***d* = 50,000** features
    - One feature per word
    - Remove stop-words
    - Remove low frequency words

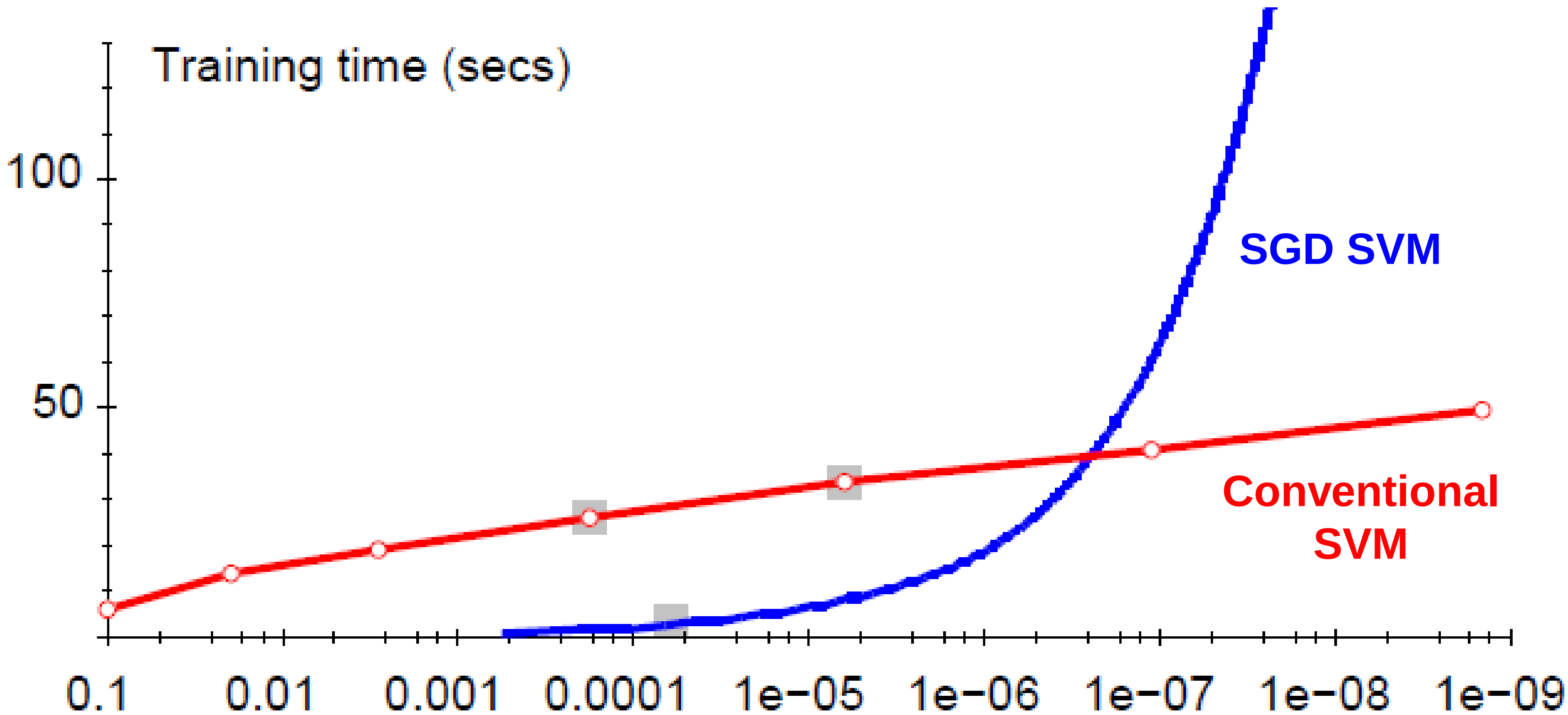
# Example: Text categorization

- **Questions:**
  - **(1)** Is **SGD** successful at minimizing ***f(w,b)***?
  - **(2)** How quickly does **SGD** find the min of ***f(w,b)***?
  - **(3)** What is the error on a test set?

| | *Training time* | *Value of f(w,b)* | *Test error* |
|---|---|---|---|
| Standard SVM | 23,642 secs | 0.2275 | 6.02% |
| "Fast SVM" | 66 secs | 0.2278 | 6.03% |
| **SGD SVM** | 1.4 secs | 0.2275 | 6.02% |

**(1)** SGD-SVM is successful at minimizing the value of ***f(w,b)***
**(2)** SGD-SVM is super fast
**(3)** SGD-SVM test set error is comparable

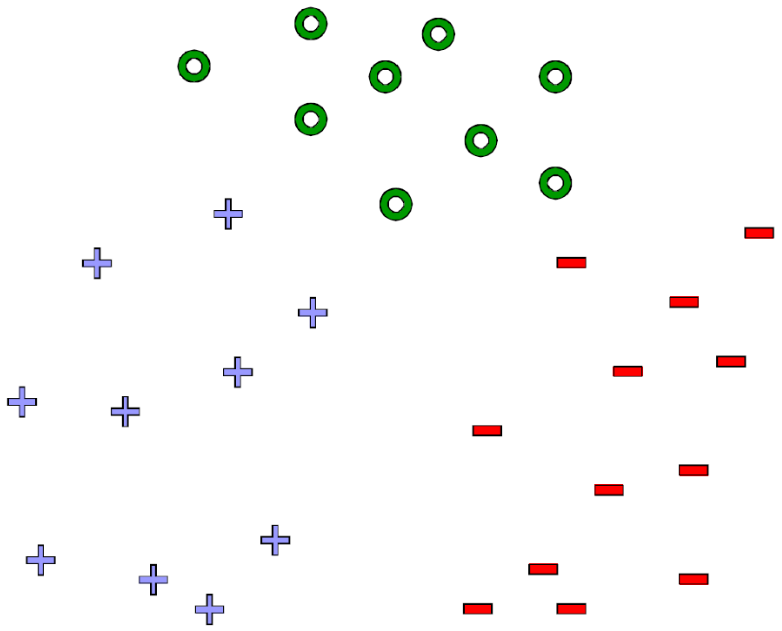
# Optimization "Accuracy"




**Optimization quality: | $f(w,b) - f(w^{opt},b^{opt})$ |**

For optimizing *f(w,b) within reasonable* quality
*SGD-SVM* is super fast

# What about multiple classes?



- **Idea 1:**
  **One against all**
  Learn 3 classifiers
  - **+ vs. {o, -}**
  - **- vs. {o, +}**
  - **o vs. {+, -}**

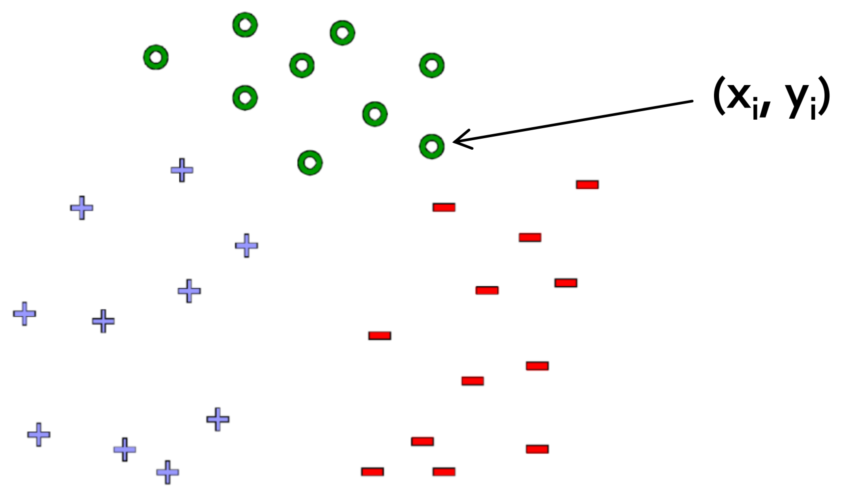
  Obtain:

  $\mathbf{w_+ \, b_+, \;\; w_- \, b_-, \;\; w_o \, b_o}$
- **How to classify?**
- Return class *c*
  $\mathbf{arg\ max_c\ w_c\, x + b_c}$

# Learn 1 classifier: Multiclass SVM

- **Idea 2: Learn 3 sets of weights simultaneously!**
  - For each class ***c*** estimate $\boldsymbol{w_c}$, $\boldsymbol{b_c}$
  - **Want the correct class to have highest margin:**

$$\mathbf{w}_{y_i}\, \mathbf{x}_i + \mathbf{b}_{y_i} \geq 1 + \mathbf{w}_c\, \mathbf{x}_i + \mathbf{b}_c \quad \forall c \neq y_i\ , \forall i$$



# Multiclass SVM

- **Optimization problem:**

$$\min_{w,b} \ \tfrac{1}{2}\sum_{c}\|w_c\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$w_{y_i}\cdot x_i + b_{y_i} \geq w_c\cdot x_i + b_c + 1 - \xi_i \qquad \forall c \neq y_i, \forall i$$

$$\xi_i \geq 0, \forall i$$

  - To obtain parameters $\boldsymbol{w_c}$, $\boldsymbol{b_c}$ (for each class $\boldsymbol{c}$) we can use similar techniques as for 2 class **SVM**

- **SVM is widely perceived a very powerful learning algorithm**