# Large Scale Machine Learning: SVMs

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
http://cs246.stanford.edu

# New Topic: ML!

| High dim. data | Graph data | Infinite data | Machine learning | Apps |
|---|---|---|---|---|
| Locality sensitive hashing | **PageRank,** SimRank | Filtering data streams | SVM | Recommender systems |
| Clustering | Community Detection | Web advertising | Decision Trees | Association Rules |
| Dimensionality reduction | Spam Detection | Queries on streams | Perceptron, kNN | Duplicate document detection |

# Machine Learning

**Study of algorithms that…**

- improve their performance
- at some task
- with experience

# Supervised Learning

**Given some data:**

- "Learn" a function to map from the **input** to the **output**

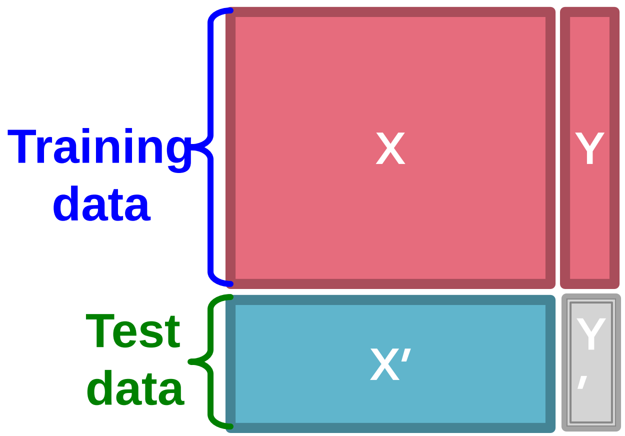
- **Given:**
  Training examples $(\boldsymbol{x_i}, \boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x_i}))$ for some unknown function $\boldsymbol{f}$
- **Find:**
  A good approximation to $\boldsymbol{f}$

# 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)

# Supervised Learning

- **Task:** Given data **(X,Y)** build a model **f()** to predict **Y'** based on **X'**
- **Strategy: Estimate $y = f(x)$ on $(X, Y)$.**
  **Hope that the same $f(x)$ also works to predict unknown $Y'$**

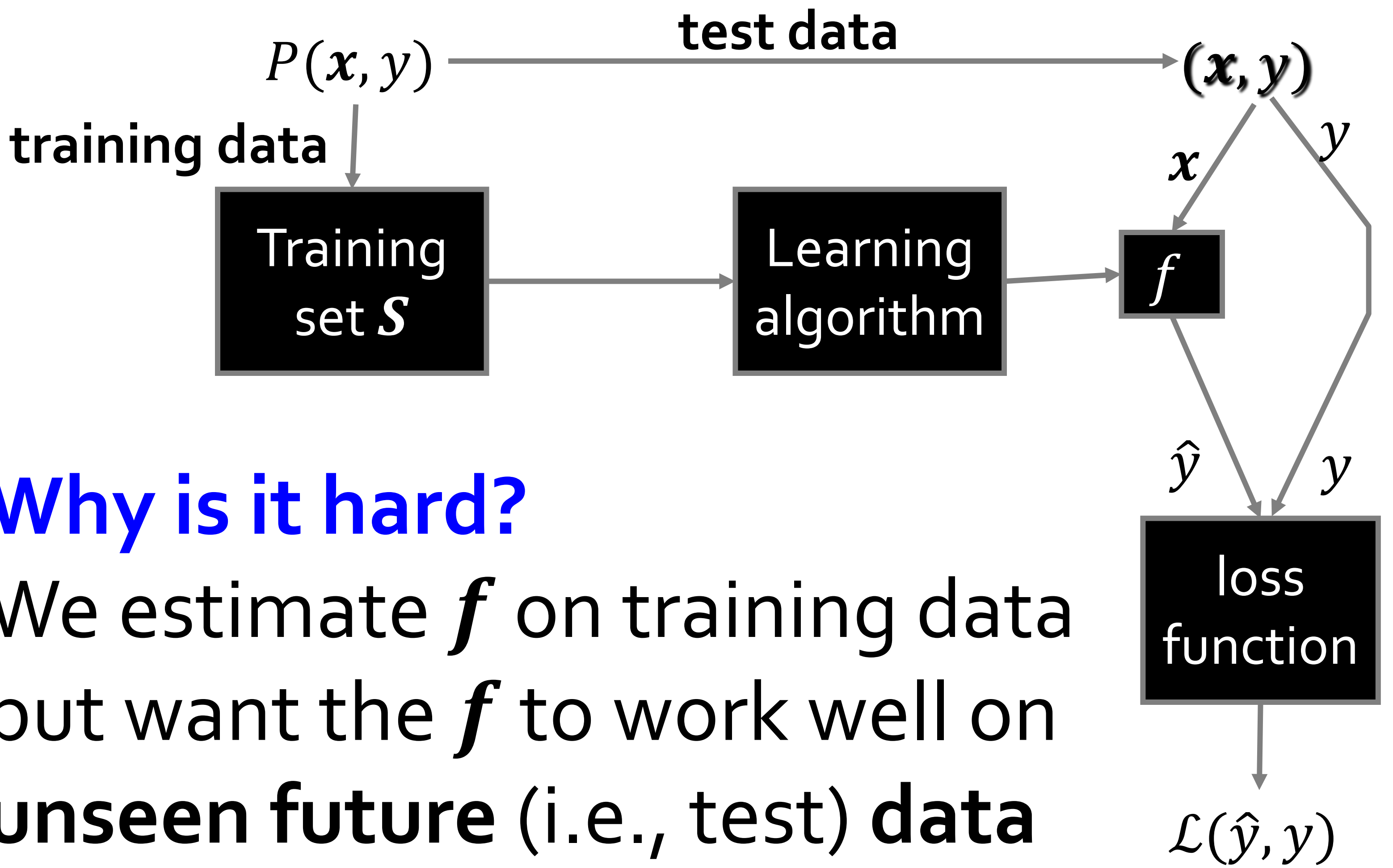



  - The **"hope"** is called **generalization**
    - **Overfitting: If f(x) predicts well Y but is unable to predict Y'**
  - **We want to build a model that <u>generalizes</u> well to unseen data**
    - But Jure, how can we well on data we have never seen before?!?

# Formal Setting

- **1)** Training data is drawn independently at random according to unknown probability distribution $P(\boldsymbol{x}, y)$
- **2)** The learning algorithm analyzes the examples and produces a classifier $\boldsymbol{f}$
- Given **new** data $(\boldsymbol{x}, y)$ drawn from $\boldsymbol{P}$, the classifier is given $\boldsymbol{x}$ and predicts $\widehat{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{x})$
- The **loss** $\boldsymbol{\mathcal{L}}(\widehat{\boldsymbol{y}}, \boldsymbol{y})$ is then measured
- **Goal of the learning algorithm:**
  Find $\boldsymbol{f}$ that minimizes **expected loss** $\boldsymbol{E_P}[\boldsymbol{\mathcal{L}}]$

# Formal Setting




**Why is it hard?**

We estimate $\boldsymbol{f}$ on training data but want the $\boldsymbol{f}$ to work well on **unseen future** (i.e., test) **data**

# Minimizing the Loss

- **Goal:** **Minimize the expected loss**

$$\min_{\mathrm{w}} \mathbb{E}_{\mathrm{P}}[\boldsymbol{\mathcal{L}}]$$

- **But, we don't have access to $P$ but only to training sample $D$:**

$$\min_{\mathrm{w}} \mathbb{E}_{\mathrm{D}}[\boldsymbol{\mathcal{L}}]$$

- **So, we minimize the average loss on the training data:**

$$\min_{\mathrm{w}} J(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(h(x), y_i)$$

**Problem:** Just memorizing the training data gives us a perfect model (with zero loss)

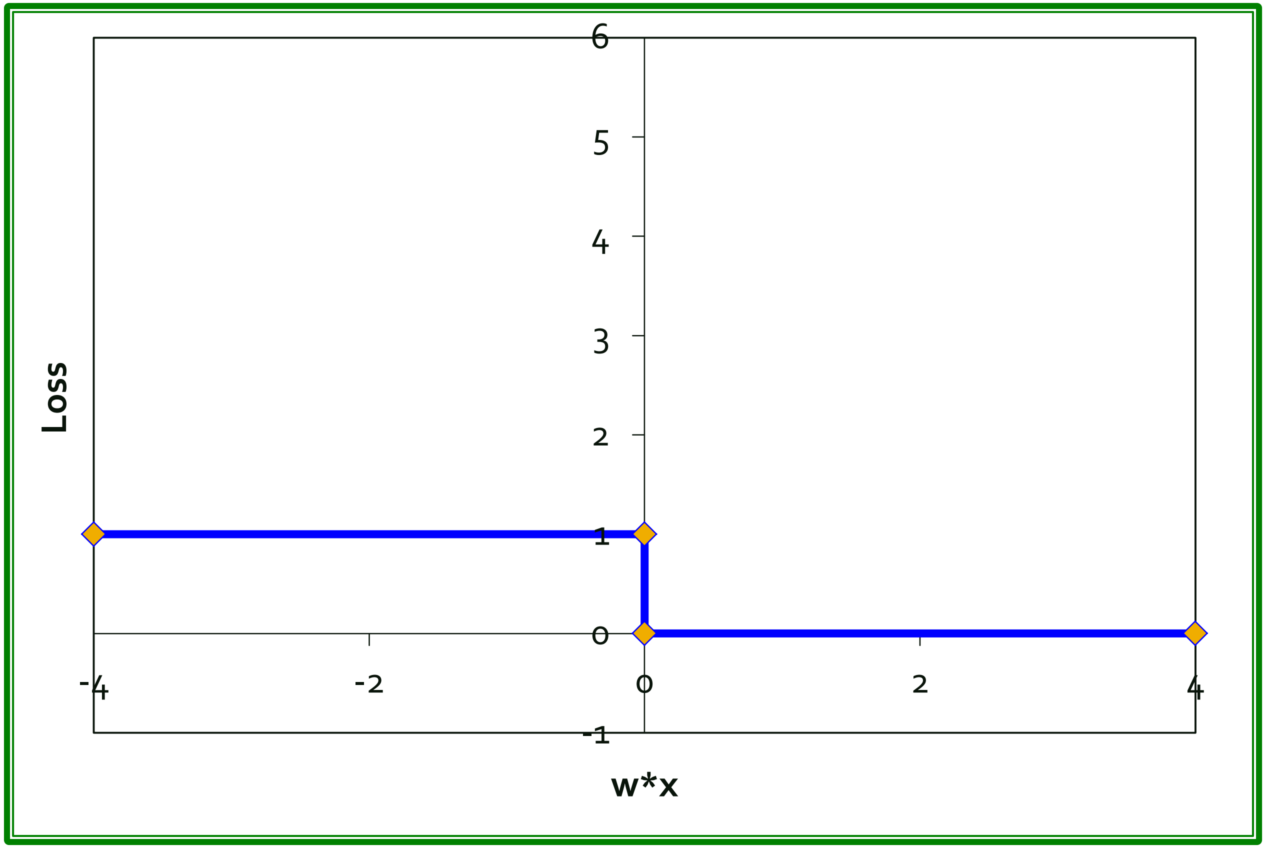
# ML == Optimization

- **Given:**
  - A set of **N** training examples
    - $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$
  - A loss function $\mathcal{L}$
- **Find:**
  - The weight vector $\boldsymbol{w}$ that minimizes the **expected loss on the training data**

$$J(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(sgn(\boldsymbol{w}^\top \boldsymbol{x}_i), y_i)$$

# Problem

- **Problem:** Step-wise Constant Loss function



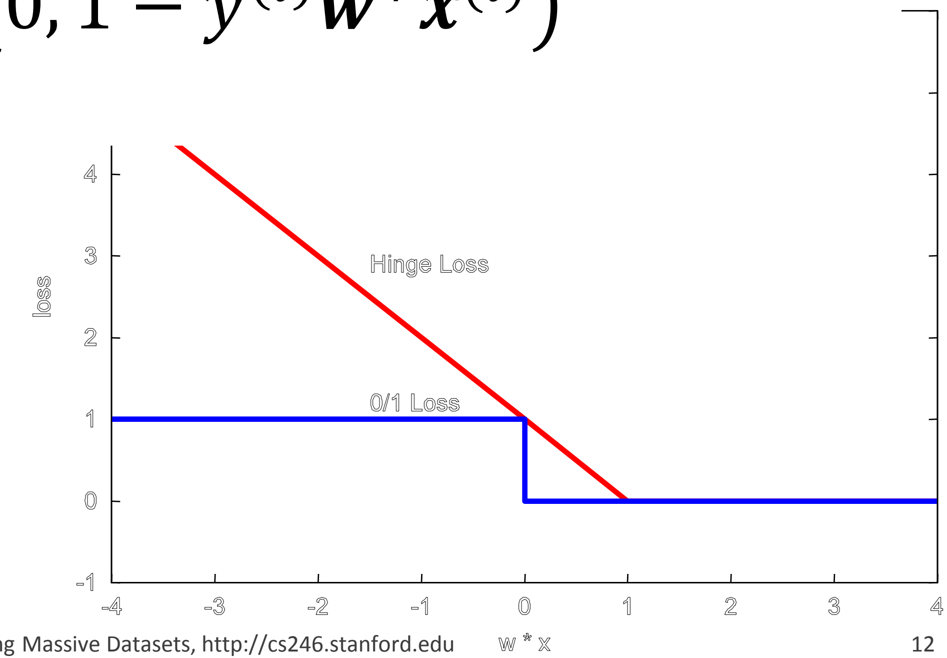

**Derivative is either 0 or ∞**

# Approximating the Loss

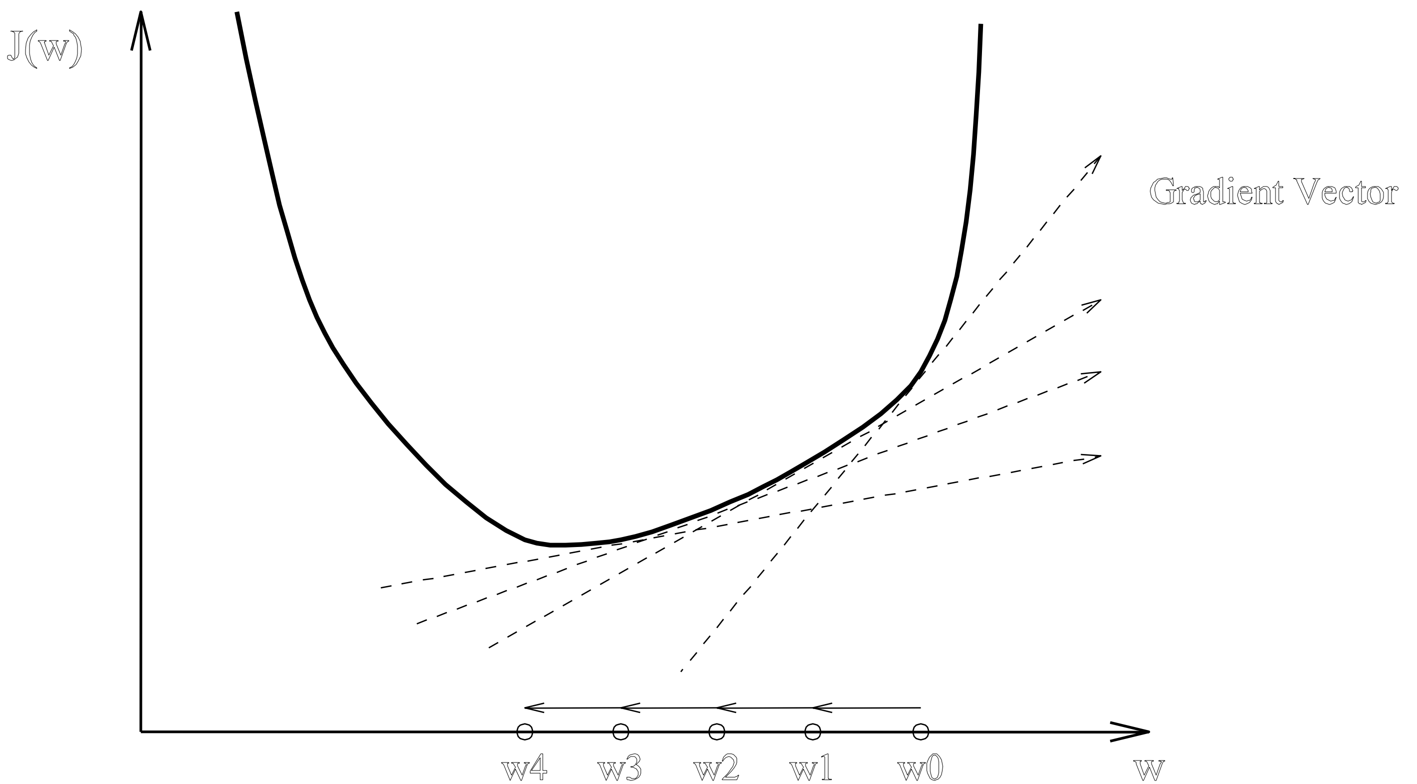
- **Approximating the expected loss by a smooth function**
  - Replace the original objective function by a surrogate loss function. E.g., **hinge loss**:

$$\tilde{J}(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N} \max\left(0, 1 - y^{(i)}\boldsymbol{w}^\top \boldsymbol{x}^{(i)}\right)$$

When $y = 1$:



# Gradient Descent



# Gradient Descent

- **Minimize $f$ by Gradient Descent**
  - Start with weight vector $\boldsymbol{w}^{(0)}$
  - Compute gradient
    $$\nabla J\left(\boldsymbol{w}^{(0)}\right) = \left(\frac{\partial J\left(\boldsymbol{w}^{(0)}\right)}{\partial w_0}, \frac{\partial J\left(\boldsymbol{w}^{(0)}\right)}{\partial w_1}, \dots, \frac{\partial J\left(\boldsymbol{w}^{(0)}\right)}{\partial w_n}\right)$$
  - Compute $\boldsymbol{w}^{(1)} = \boldsymbol{w}^{(0)} - \eta \nabla J\left(\boldsymbol{w}^{(0)}\right)$
    where $\eta$ is a "step size" parameter
  - Repeat until convergence

# Example: Spam Detection

- 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)

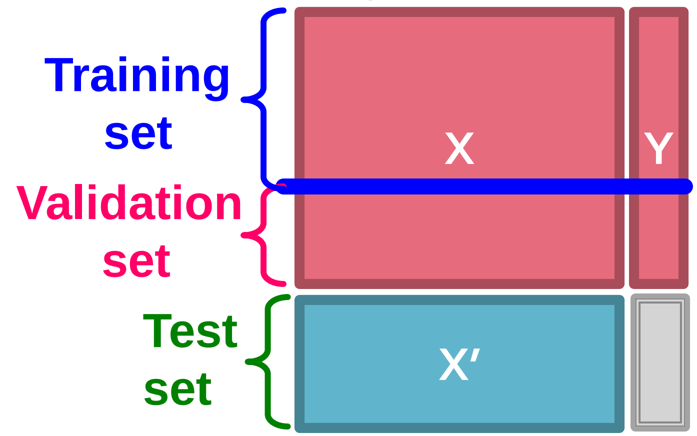
# Spam Detection

- $P(x, y)$: distribution of email messages $x$ and their true labels $y$ ("spam", "ham")
- **Training sample:** a set of email messages that have been labeled by the user
- **Learning algorithm:** What we study!
- $f$: The classifier output by the learning alg.
- **Test point:** A new email $x$ (with its true, but hidden, label $y$)
- **Loss function $\mathcal{L}(\hat{y}, y)$:**

| predicted label $\hat{y}$ | true label $y$ | |
|---|---|---|
| | spam | ham |
| spam | 0 | 10 |
| not spam | 1 | 0 |

# Supervised Learning

- **Idea: Pretend we do not know the data/labels we actually do know**
  - Build the model **f(x)** on the training data (minimize *J*)
  - See how well **f(x)** does on the validation data
    - If it does well, then apply it also to **X'**




**Estimate *y = f(x)* on *X,Y*. Hope that the same *f(x)* also works on unseen *X', Y'***

- **Refinement: Cross validation**
  - Splitting into training/validation set is brutal
  - Let's split our data **(X,Y)** into 10-folds (buckets)
  - Take out 1-fold for validation, train on remaining 9
  - Repeat this 10 times, report average performance

# Large Scale Machine Learning

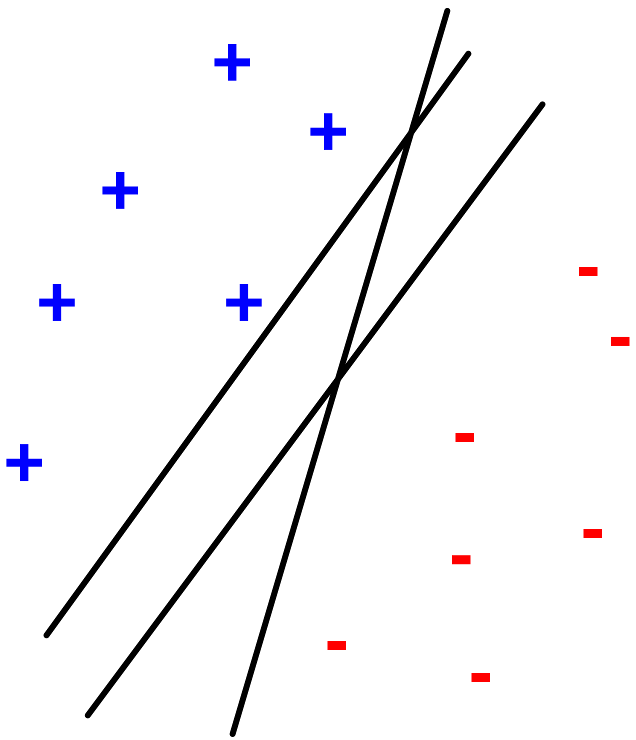
- **We will talk about the following methods:**
  - Support Vector Machines
  - Decision trees

- **Main question:**
  **How to efficiently train**
  (build a model/find model parameters)?

# Support Vector Machines

# Support Vector Machines

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



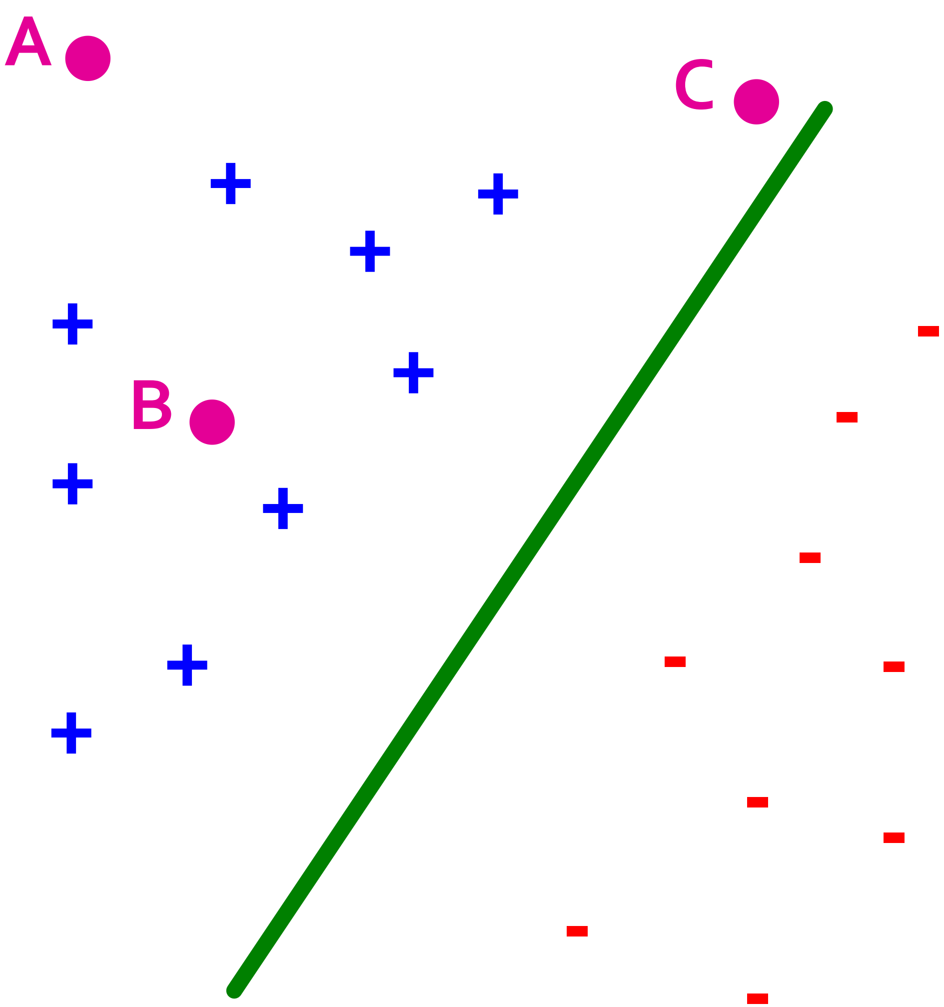

**Data:**

- **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*)?**

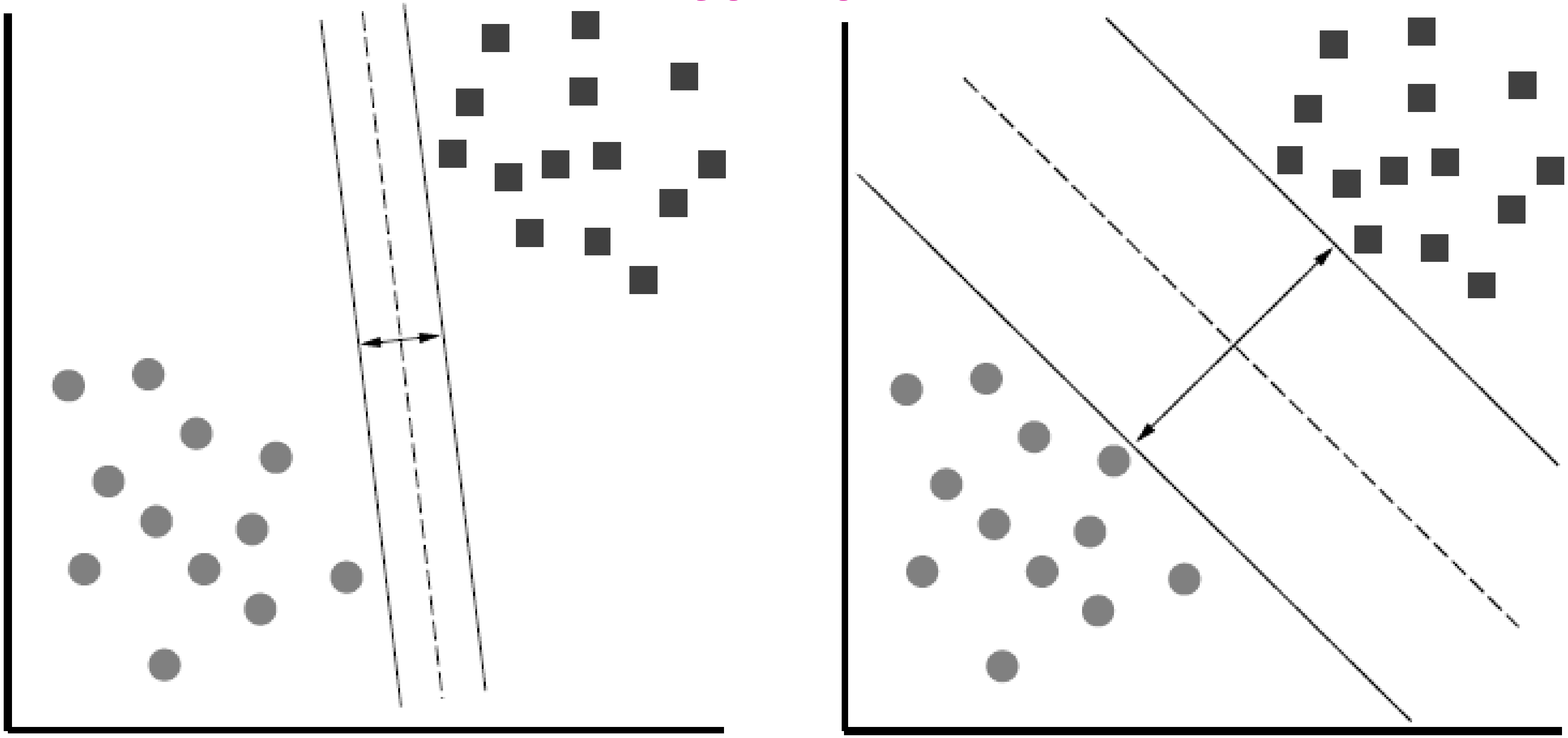
# Largest Margin




- **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**

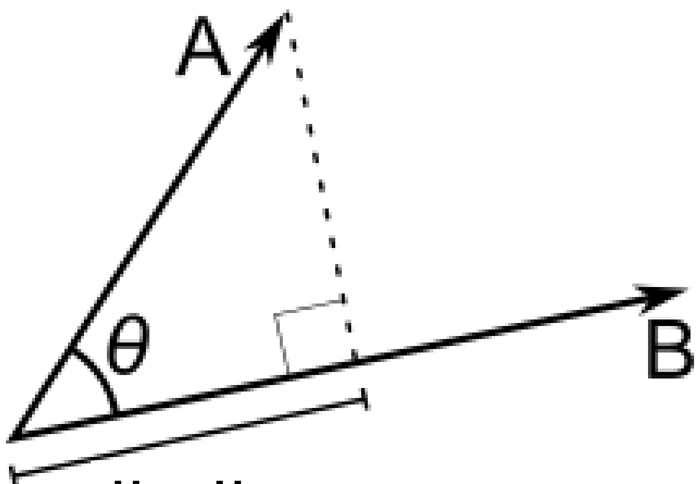


The reason we define margin this way is due to theoretical convenience and existence of generalization error bounds that depend on the value of margin.

# Why maximizing $\gamma$ a good idea?

- **Remember: the Dot product**

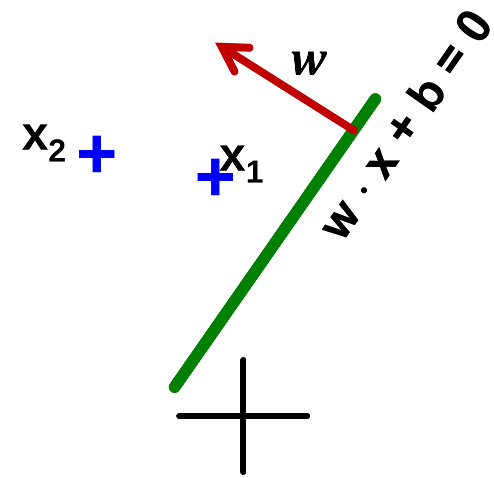
$$A \cdot B = \|A\| \cdot \|B\| \cdot \cos\theta$$




$$\|A\| cos\theta$$

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

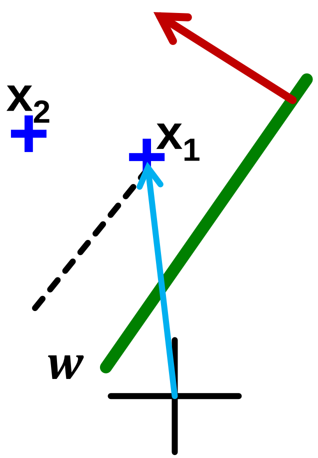
# Why maximizing $\gamma$ a good idea?



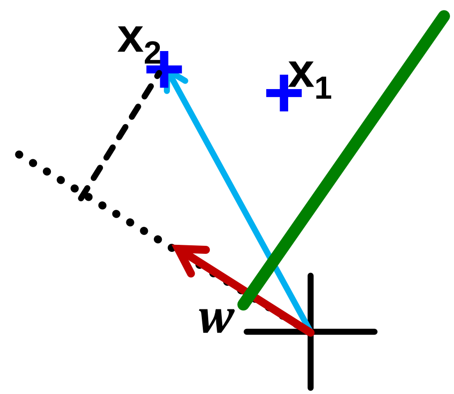

- **Dot product**

  $$A \cdot B = \|A\| \|B\| \cos\theta$$

- **What is $w \cdot x_1$ , $w \cdot x_2$?**



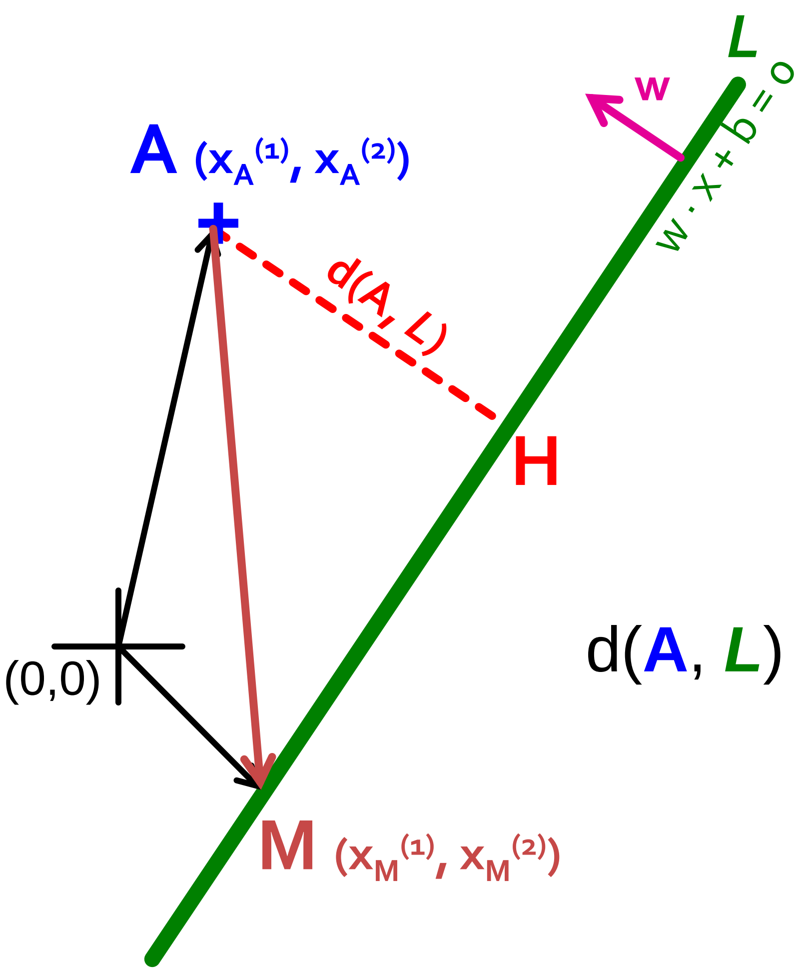

In this case
$\gamma_1 \approx \|w\|^2$




In this case
$\gamma_2 \approx 2\|w\|^2$

- **So, $\gamma$ roughly corresponds to the margin**
  - **Bottom line:** Bigger $\gamma$ bigger the separation

# What is the margin?



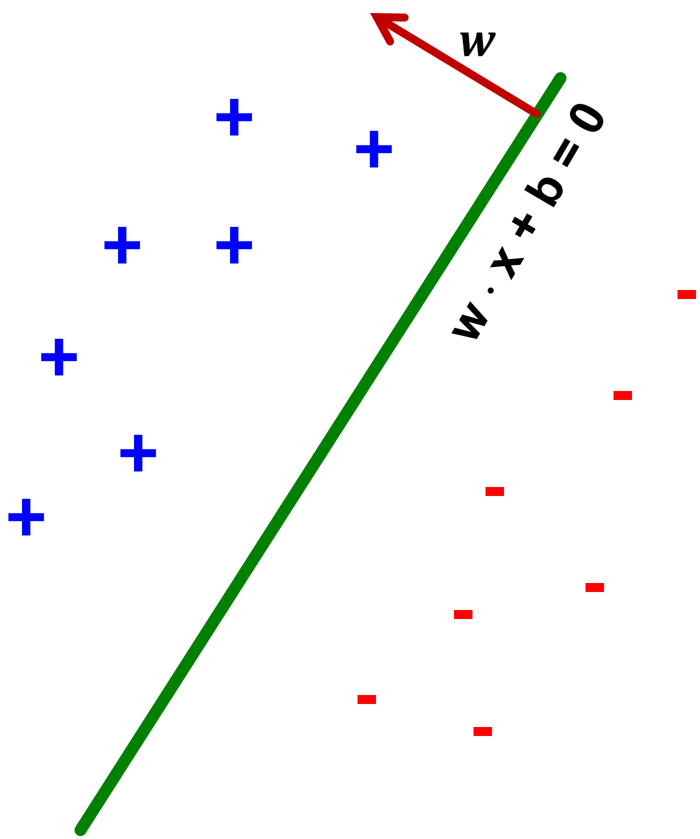

Note we assume $\|w\|_2 = 1$

- **Let:**
  - **Line L**: **w·x+b** = $w^{(1)}x^{(1)}+w^{(2)}x^{(2)}+b=0$
  - **w** = $(w^{(1)}, w^{(2)})$
  - **Point A** = $(x_A^{(1)}, x_A^{(2)})$
  - **Point M** on a line = $(x_M^{(1)}, x_M^{(2)})$

$$
\begin{aligned}
d(A, L) &= |AH| \\
&= |(A-M) \cdot w| \\
&= |(x_A^{(1)} - x_M^{(1)})\, w^{(1)} + (x_A^{(2)} - x_M^{(2)})\, w^{(2)}| \\
&= |x_A^{(1)} w^{(1)} + x_A^{(2)} w^{(2)} + b| \\
&= |w \cdot A + b|
\end{aligned}
$$

Remember $\mathbf{x_M^{(1)}w^{(1)} + x_M^{(2)}w^{(2)} = -b}$ since **M** belongs to line **L**

# Largest Margin




- Prediction = **sign**$(\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b})$
- "**Confidence**" = $(\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b})\ \boldsymbol{y}$
- **For i-th datapoint:**

$$\gamma_i = (w \cdot x_i + b) y_i$$

- **Want to solve:**

$$\max_{w} \min_{i} \gamma_i$$

- **Can rewrite as**

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

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

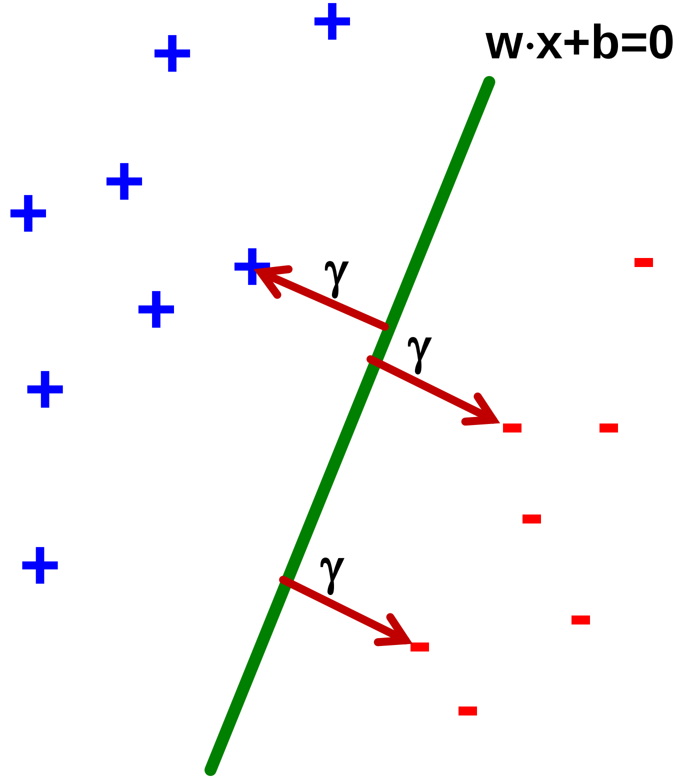
# Support Vector Machine

- **Maximize the margin:**
  - **Good according to intuition, theory (c.f. "VC dimension") and 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**

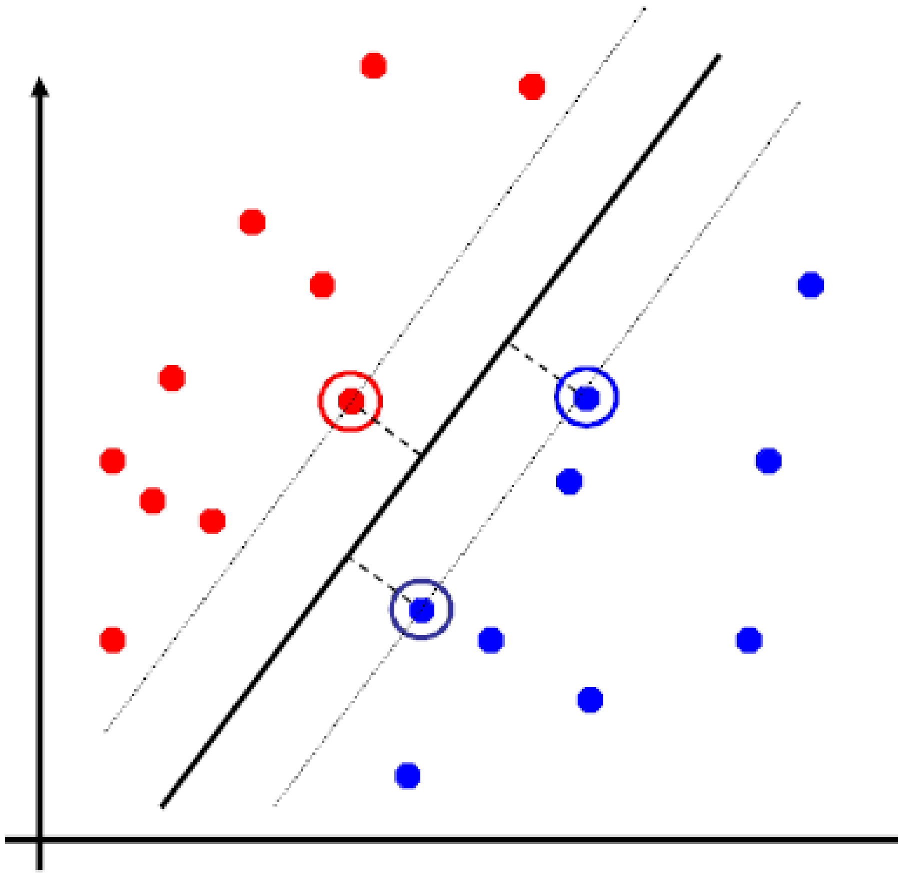



Maximizing the margin

# Support Vector Machines: Deriving the margin

# Support Vector Machines

- **Separating hyperplane is defined by the support vectors**
  - Points on **+/-** planes from the solution
  - If you knew these points, you could ignore the rest
  - Generally, $\boldsymbol{d}$+$\boldsymbol{1}$ support vectors (for $\boldsymbol{d}$ dim. data)



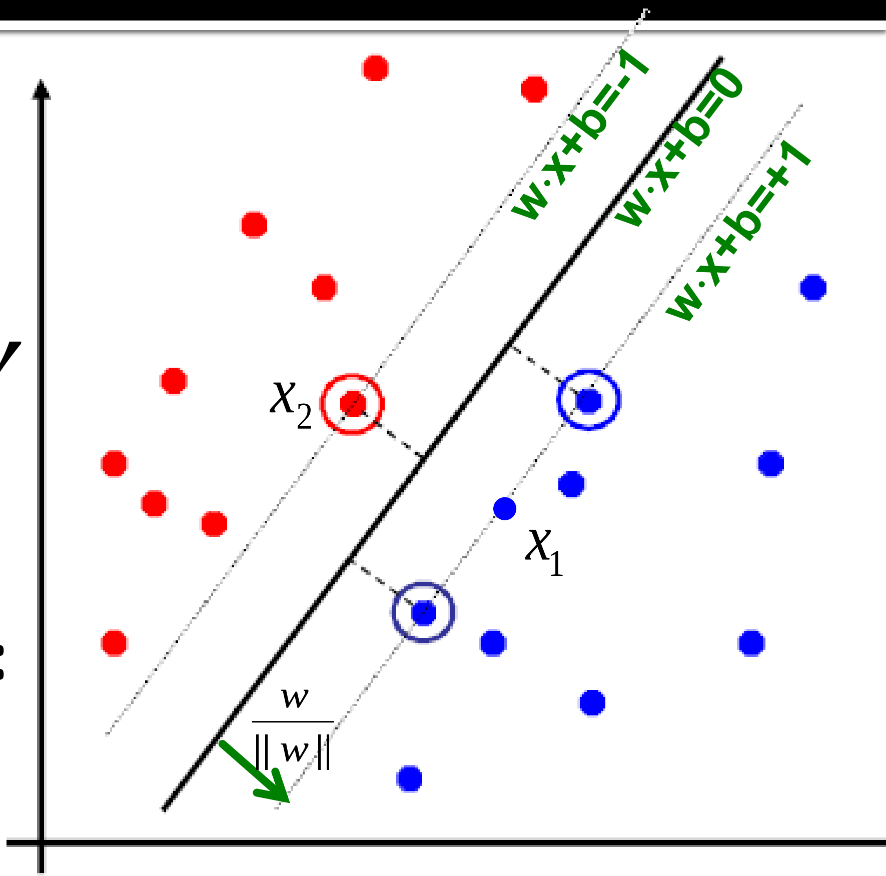
# Canonical Hyperplane: Problem

- **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:

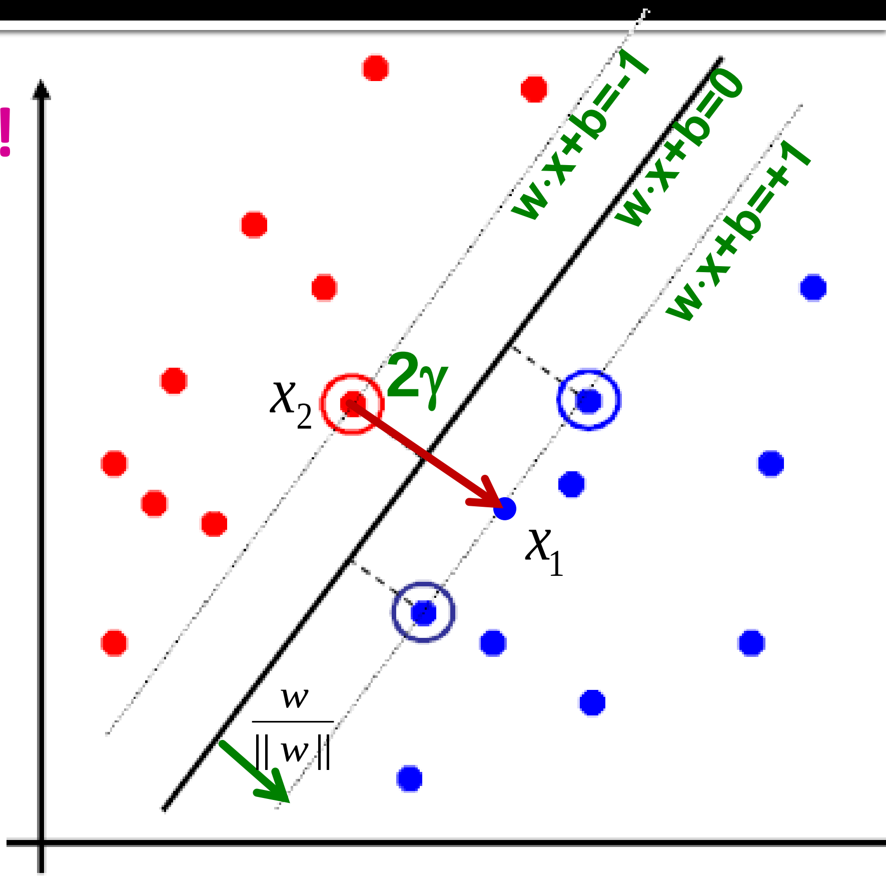
$$\boldsymbol{w} \cdot \boldsymbol{x_j} + \boldsymbol{b} = \pm \mathbf{1}$$




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

# 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**

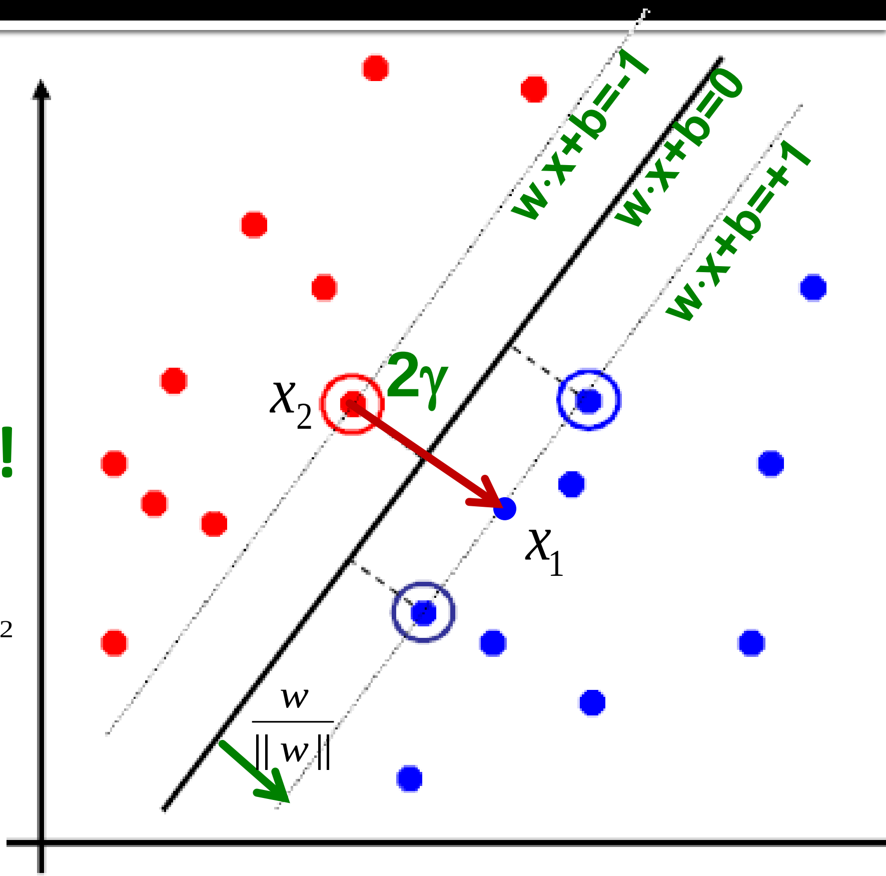
$$\max_{w,\gamma} \gamma$$

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

**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**

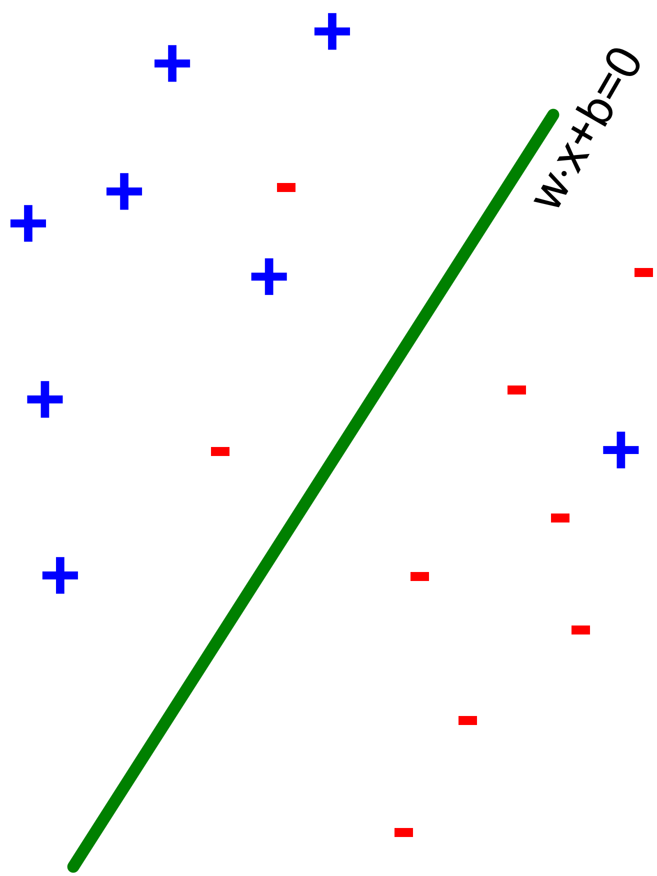
# Non-linearly Separable Data

- **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!**



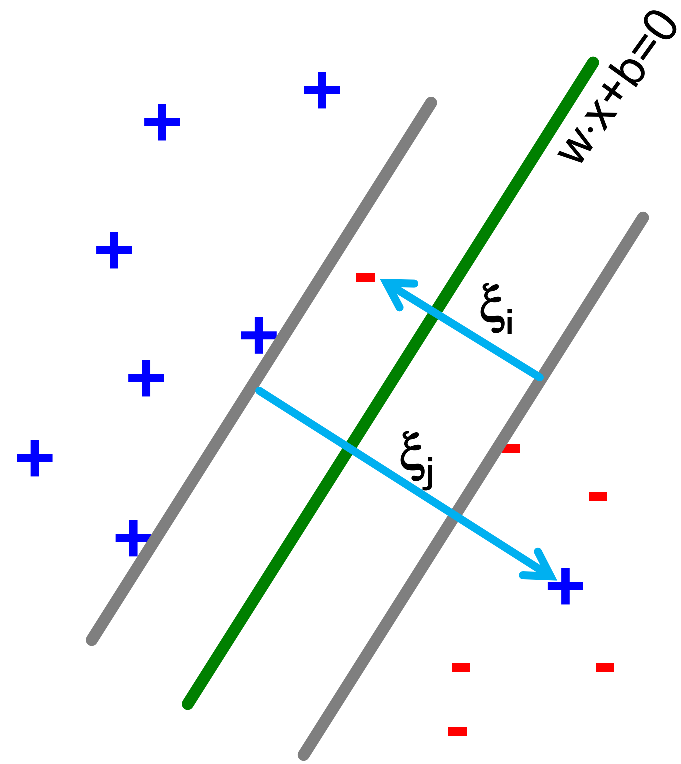
# Support Vector Machines

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

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

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

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

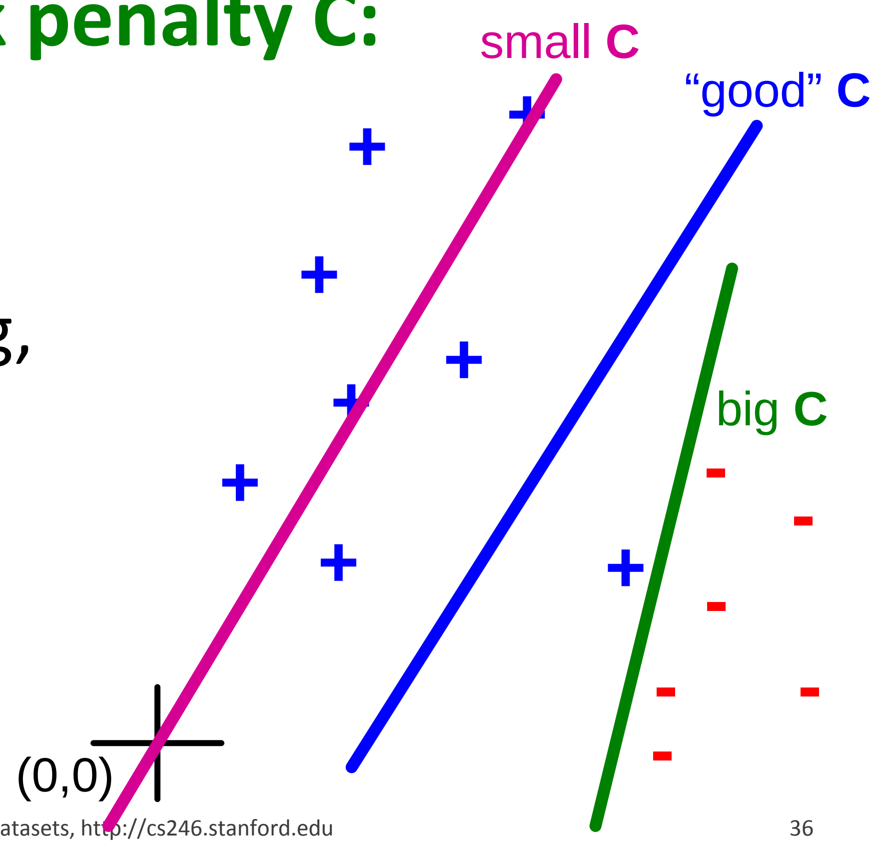



**For each data point:**
If margin ≥ 1, don't care
If margin < 1, pay linear penalty

# Slack Penalty *C*

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

- **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)



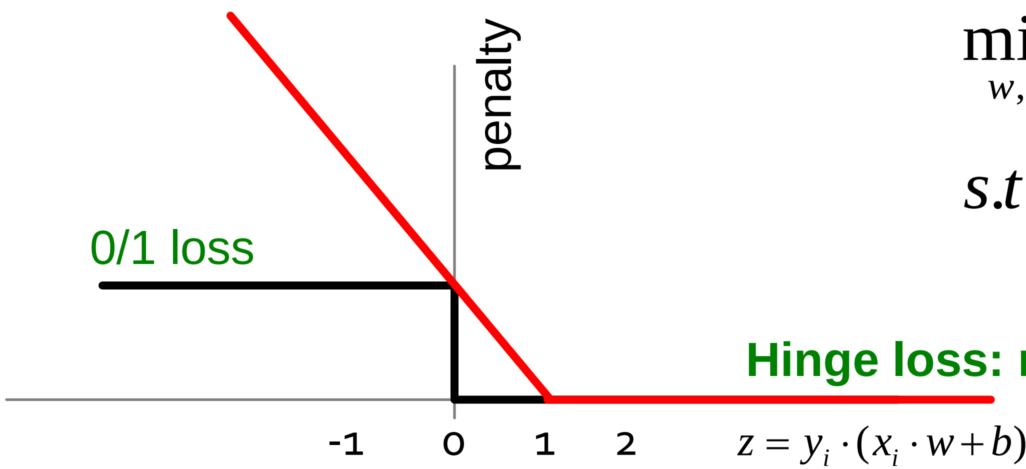
# Support Vector Machines

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

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

Regularization parameter

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




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

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

# Support Vector Machines: How to compute the margin?

# 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*?

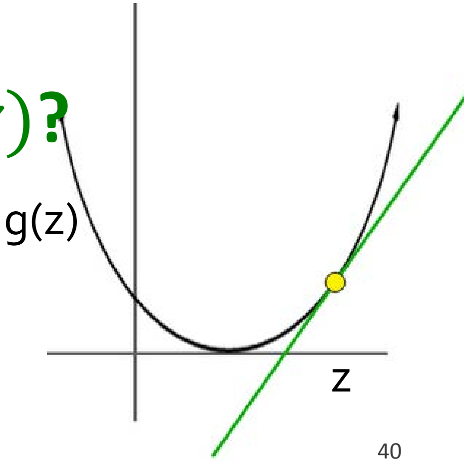
$$\min_{w,b} \ \tfrac{1}{2} w \cdot w + C \sum_{i=1}^{n} \xi_i$$

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

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

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

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



# SVM: How to estimate w?

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

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

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

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

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

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

# SVM: How to estimate w?

- **Gradient descent:**

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

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

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

# SVM: How to estimate w?

**We just had:**

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

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

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

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

- **Stochastic gradient descent:**

**Iterate until convergence:**

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

# Support Vector Machines: Example

# 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

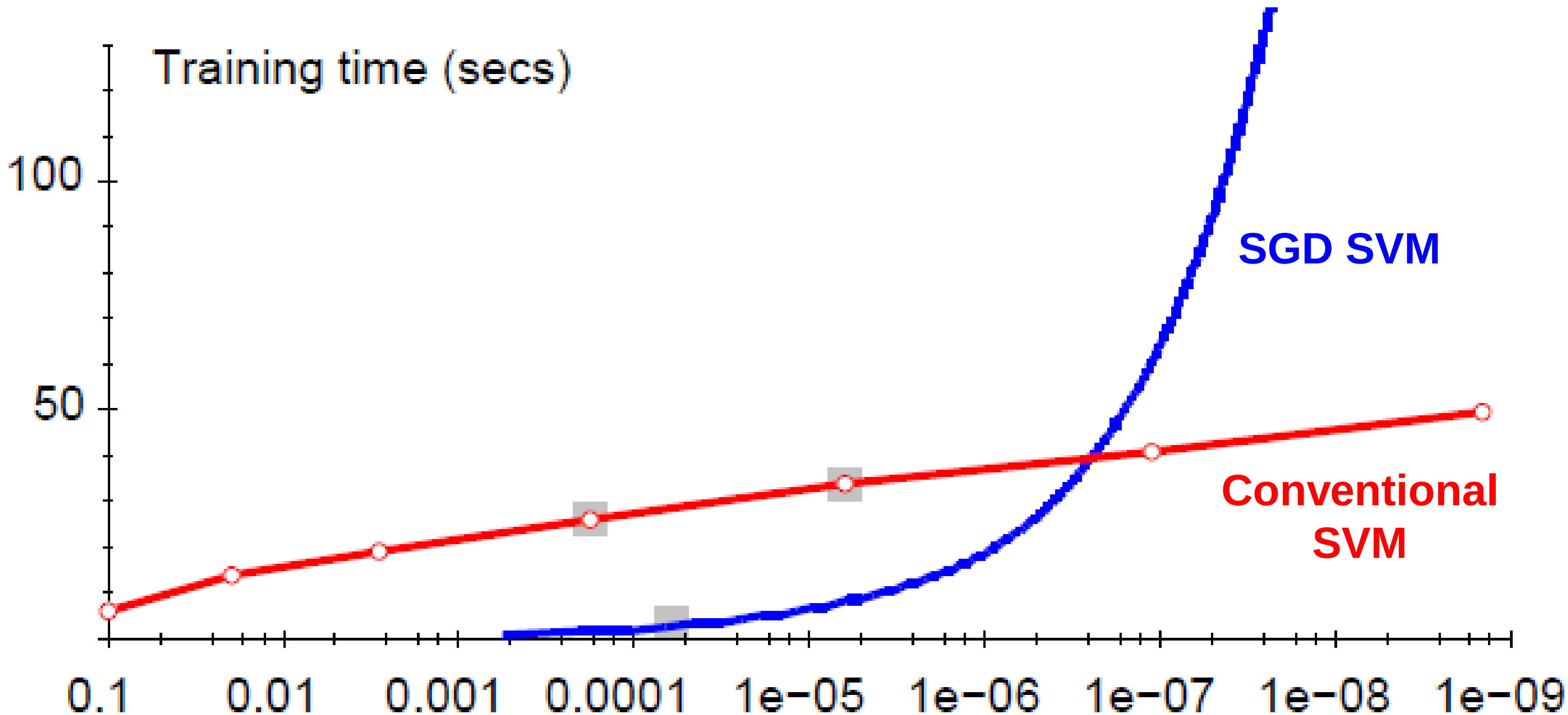
# Example: Text categorization

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

| | ***Training time*** | ***Value of J(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 ***J(w,b)***
**(2)** SGD-SVM is super fast
**(3)** SGD-SVM test set error is comparable

# Optimization "Accuracy"

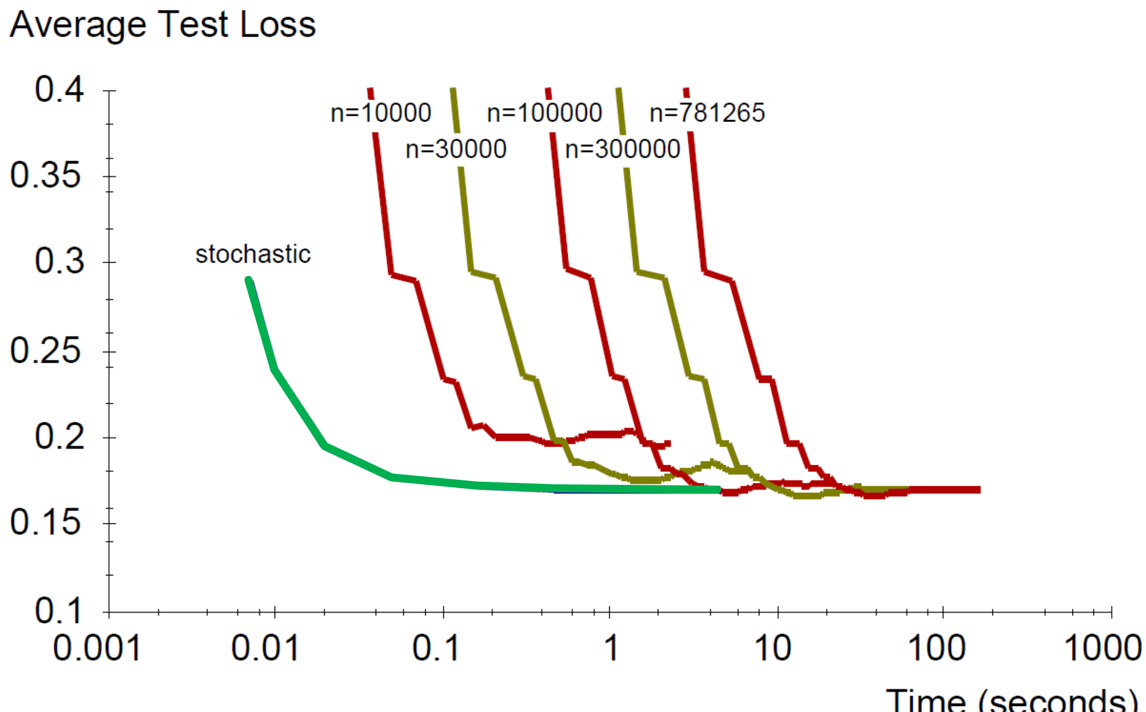



**Optimization quality: | $J(w,b) - J(w^{opt},b^{opt})$ |**

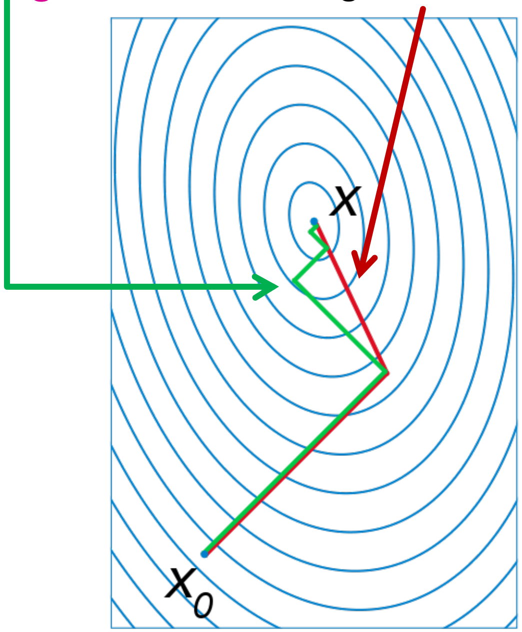
For optimizing *J(w,b) within reasonable* quality
*SGD-SVM* is super fast

# SGD vs. Batch Conjugate Gradient

- **SGD** on full dataset vs. **Conjugate Gradient** on a sample of ***n*** training examples




**Theory says: Gradient descent** converges in linear time $k$. **Conjugate gradient** converges in $\sqrt{k}$.




$k$… condition number

**Bottom line:** Doing a simple (but fast) SGD update many times is better than doing a complicated (but slow) CG update a few times

# Practical Considerations

- **Sparse Linear SVM:**
  - **Feature vector $x_i$ is sparse (contains many zeros)**
    - Do not do: $\boldsymbol{x_i} = [0,0,0,1,0,0,0,0,5,0,0,0,0,0,0,\ldots]$
    - But represent $\mathbf{x_i}$ as a sparse vector $\mathbf{x_i}$**=[(4,1), (9,5), …]**
  - **Can we do the SGD update more efficiently?**

$$w \leftarrow w - \eta\left(w + C\frac{\partial L(x_i, y_i)}{\partial w}\right)$$

  - **Approximated in 2 steps:**

$$w \leftarrow w - \eta C\frac{\partial L(x_i, y_i)}{\partial w}$$

**cheap**: $\boldsymbol{x_i}$ is sparse and so few coordinates $\boldsymbol{j}$ of $\boldsymbol{w}$ will be updated

$$w \leftarrow w(1-\eta)$$

**expensive**: $\boldsymbol{w}$ is not sparse, all coordinates need to be updated

# Practical Considerations

- **Solution 1:** $\boldsymbol{w} = \boldsymbol{s} \cdot \boldsymbol{v}$
  - Represent vector $\boldsymbol{w}$ as the product of scalar $\boldsymbol{s}$ and vector $\boldsymbol{v}$
  - Then the update procedure is:
    - *(1)* $\boldsymbol{v} = \boldsymbol{v} - \eta C \frac{\partial L(x_i, y_i)}{\partial w}$
    - *(2)* $\boldsymbol{s} = \boldsymbol{s}(1 - \eta)$
- **Solution 2:**
  - Perform only step **(1)** for each training example
  - Perform step **(2)** with lower frequency and higher $\eta$

**Two step update procedure:**

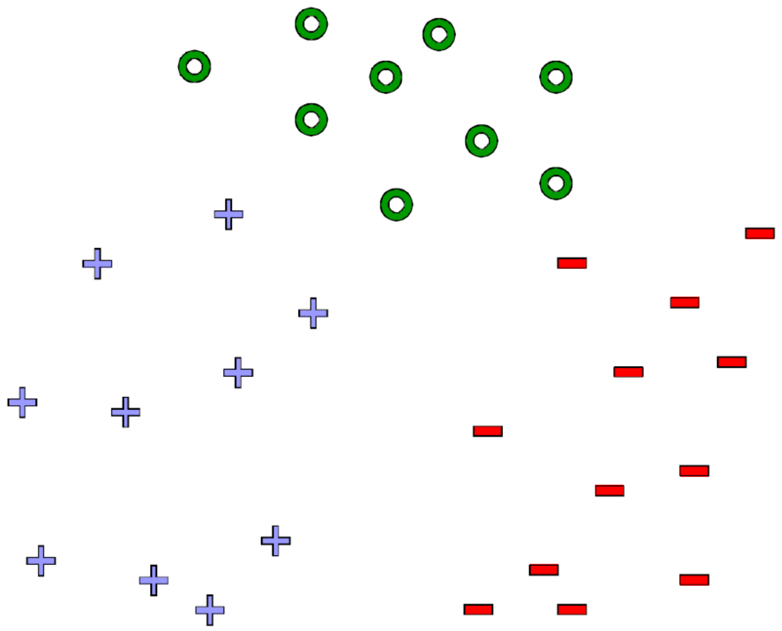
**(1)** $w \leftarrow w - \eta C \frac{\partial L(x_i, y_i)}{\partial w}$

**(2)** $w \leftarrow w(1 - \eta)$

# Practical Considerations

- **Stopping criteria:**
  **How many iterations of SGD?**
  - **Early stopping with cross validation**
    - Create a validation set
    - Monitor cost function on the validation set
    - Stop when loss stops decreasing
  - **Early stopping**
    - Extract two (very) small sets of training data **A** and **B**
    - Train on **A**, stop by validating on **B**
    - Number of training epochs on **A** is an estimate of ***k***
    - Train for ***k*** epochs on the **full** dataset

# What about multiple classes?



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

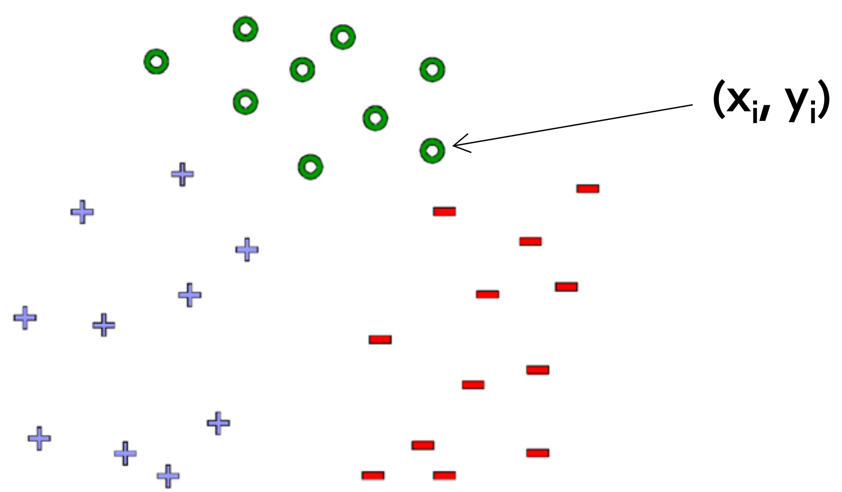
  Obtain:

  $\mathbf{w_+ b_+,\ \ w_- b_-,\ \ w_o b_o}$
- **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 simoultaneously!**
  - For each class ***c*** estimate $\boldsymbol{w_c}$, $\boldsymbol{b_c}$
  - **Want the correct class $\boldsymbol{y_i}$ 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:**

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

$$w_{y_i}\cdot x_i + b_{y_i} \geq w_c \cdot x_i + b_c + 1 - \xi_i \qquad \xi_i \geq 0, \forall i$$

  - 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**

# Support Vector Machines: Example

# Online Learning

- **New setting: Online Learning**
  - Allows for modeling problems where we have a continuous stream of data
  - We want an algorithm to learn from it and slowly adapt to the changes in data
- **Idea: Do slow updates to the model**
  - SGD-SVM makes updates if misclassifying a datapoint
  - **So:** First train the classifier on training data. Then for every example from the stream, if we misclassify, update the model (using a small learning rate)

# Example: Shipping Service

- **Protocol:**
  - User comes and tell us origin and destination
  - We offer to ship the package for some money (\$10 - \$50)
  - Based on the price we offer, sometimes the user uses our service (**y = 1**), sometimes they don't (**y = -1**)
- **Task: Build an algorithm to optimize what price we offer to the users**
- **Features *x* capture:**
  - Information about user
  - Origin and destination
- **Problem: Will user accept the price?**

# Example: Shipping Service

- **Model whether user will accept our price:**

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x};\ \boldsymbol{w})$$

  - **Accept: y =1**, **Not accept: y=-1**
  - Build this model with say Perceptron or SVM
- **The website that runs continuously**
- **Online learning algorithm would do something like**
  - User comes
  - User is represented as an **(x,y)** pair where
    - **x:** Feature vector including price we offer, origin, destination
    - **y:** If they chose to use our service or not
  - The algorithm updates **w** using just the **(x,y)** pair
  - Basically, we update the **w** parameters every time we get some new data

# Example: Shipping Service

- We discard this idea of a data "set"
- Instead we have a continuous stream of data
- **Further comments:**
  - For a major website where you have a massive stream of data then this kind of algorithm is pretty reasonable
  - Don't need to deal with all the training data
  - If you had a small number of users you could save their data and then run a normal algorithm on the full dataset
    - Doing multiple passes over the data

# Online Algorithms

- An online algorithm can adapt to changing user preferences
- For example, over time users may become more price sensitive
- **The algorithm adapts and learns this**
- So the system is dynamic