Large Scale Support Vector Machines

CS246: Mining Massive Datasets

Jure Leskovec, Stanford University

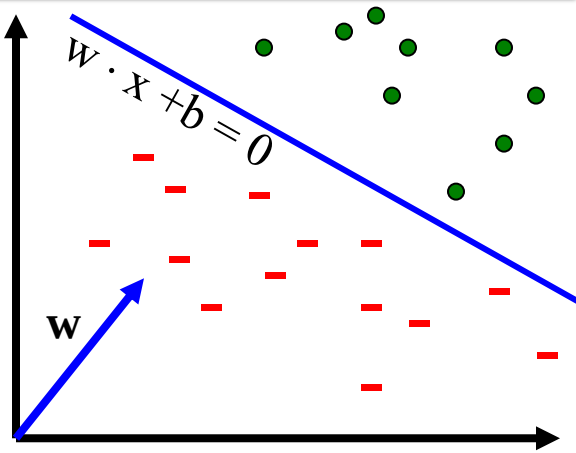
http://cs246.stanford.edu

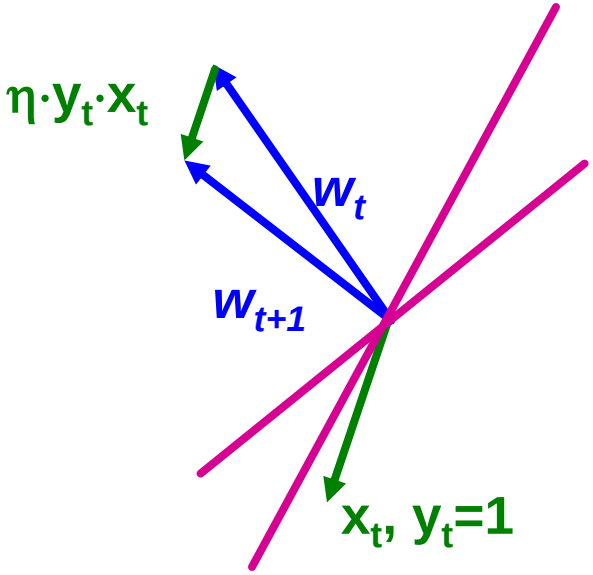
Perceptron

- **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

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

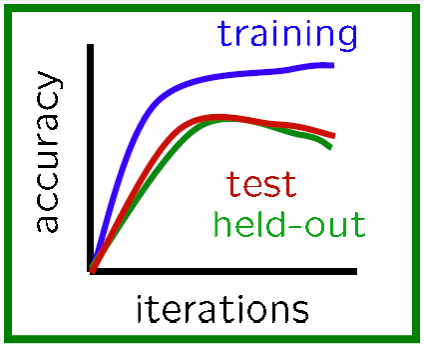
    - η is the learning rate parameter
    - x_t is the t-th training example
    - y_t is true t-th class label ({+1, -1})



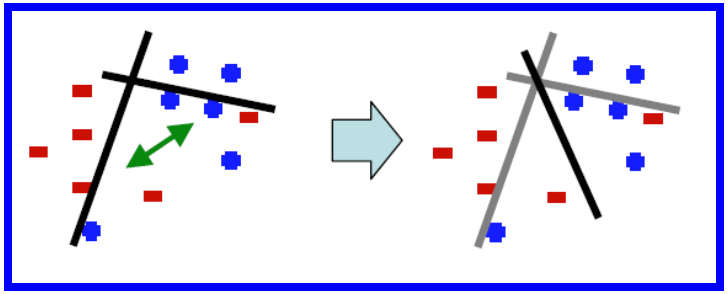


Issues with Perceptrons

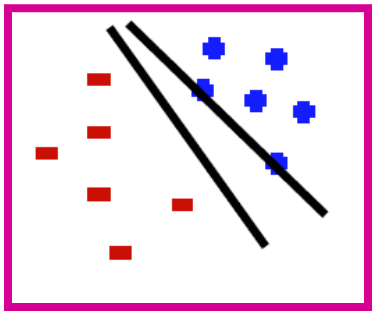
- **Overfitting:**



- **Regularization:** If the data is not separable weights dance around

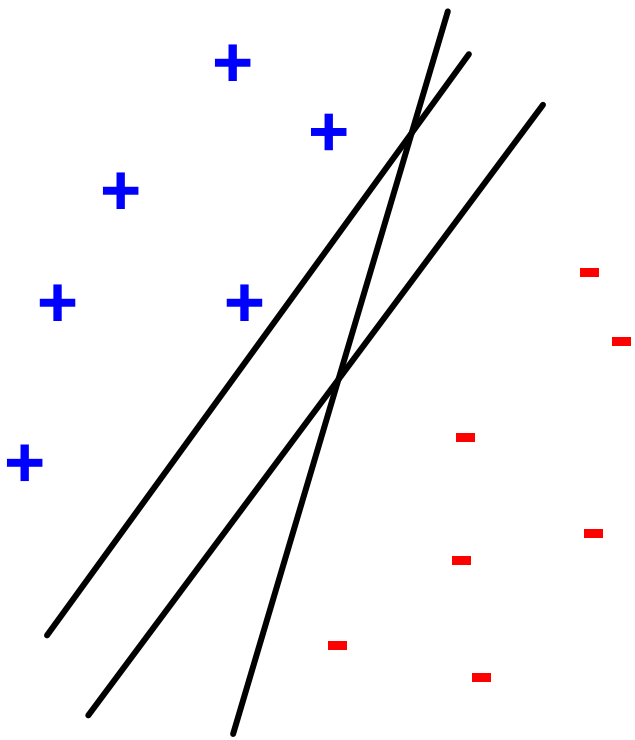


- **Mediocre generalization:**
  - Finds a "barely" separating solution



Support Vector Machines

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



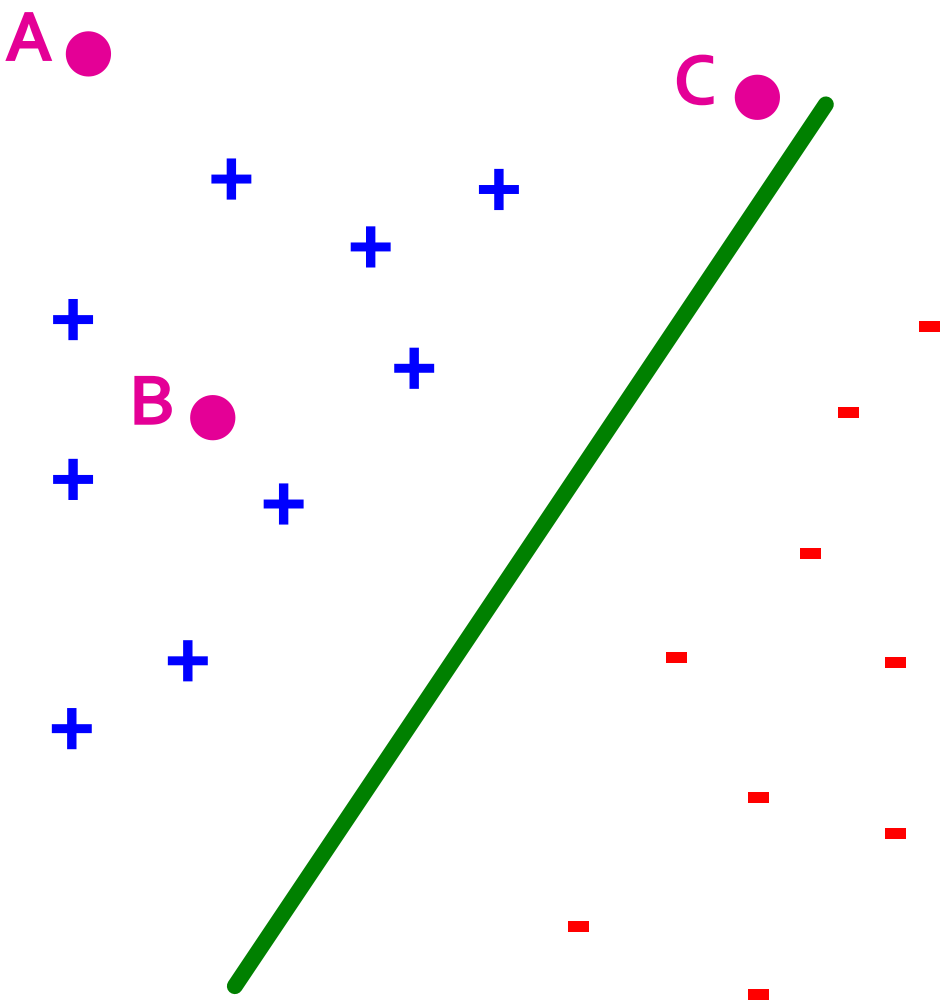
Data:

- Training examples:
  - $(x_1, y_1) \ldots (x_n, y_n)$
- Each example *i*:
  - $x_i = (x_i^{(1)}, \ldots , x_i^{(d)})$
    - $x_i^{(j)}$ 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*)?

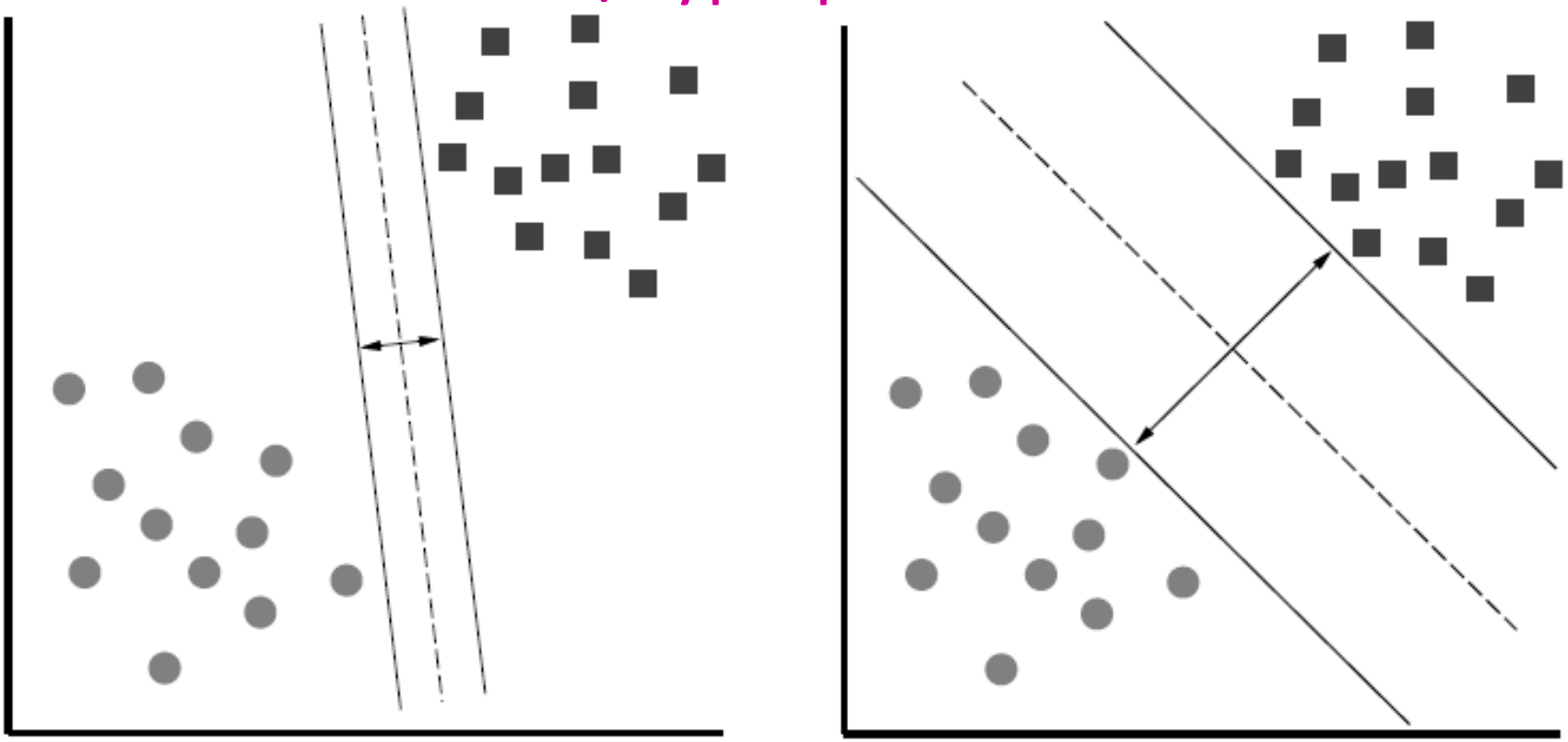
Largest Margin



- **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:** Distance of closest example from the decision line/hyperplane

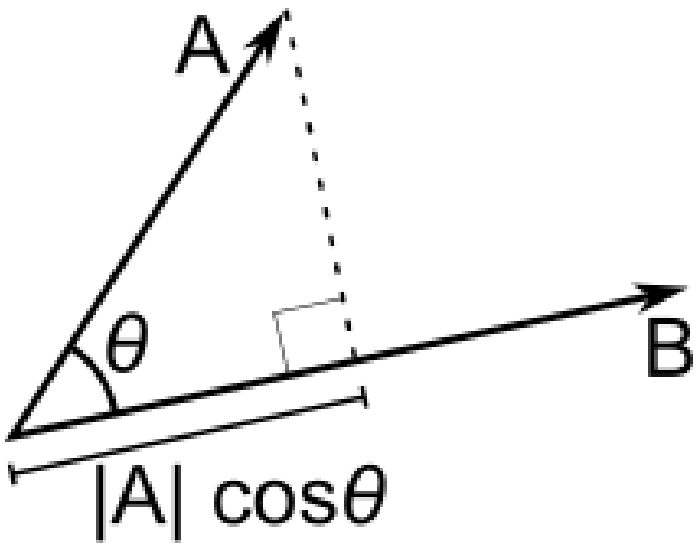


The reason we define margin this way is due to theoretical convenience and existence of generalization error bounds that depend on the value of margin.

Why maximizing γ a good idea?

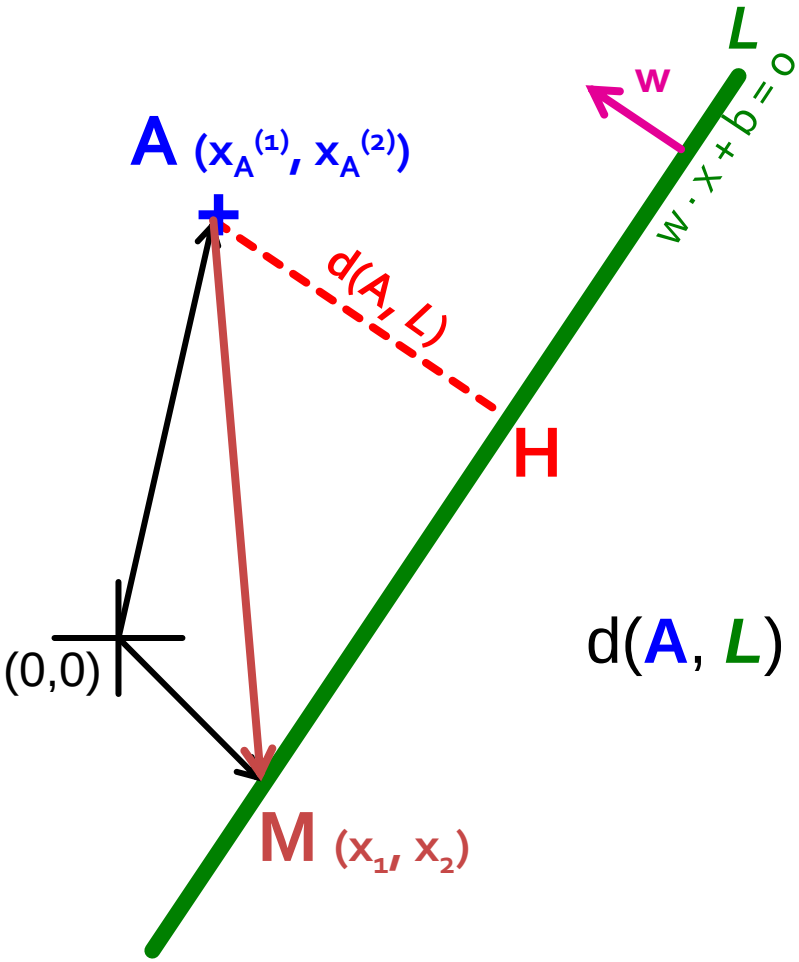
- **Remember: Dot product**

$$\boldsymbol{A} \cdot \boldsymbol{B} = \|\boldsymbol{A}\| \|\boldsymbol{B}\| \cos \boldsymbol{\theta}$$



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

What is the margin?

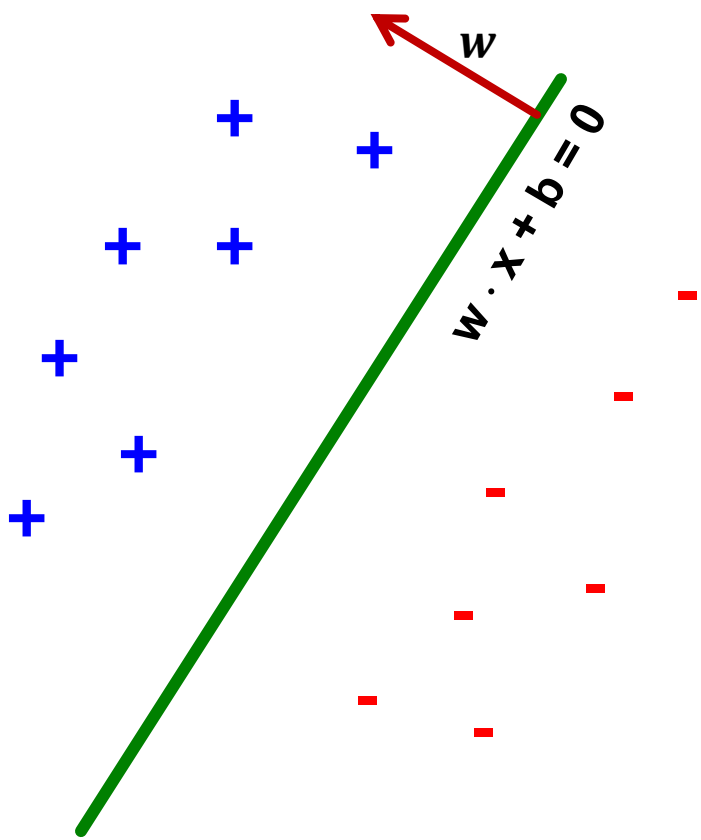


- **Let:**
  - **Line L:** $\mathbf{w \cdot x + b} = w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + b = 0$
  - $\mathbf{w} = (w^{(1)}, w^{(2)})$
  - **Point A** $= (x_A^{(1)}, x_A^{(2)})$
  - **Point M** on a line $= (x_M^{(1)}, x_M^{(2)})$

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

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

Largest Margin



- Prediction = $\mathbf{sign}(w \cdot x + b)$
- "**Confidence**" = $(w \cdot x + b)\, y$
- **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**

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

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

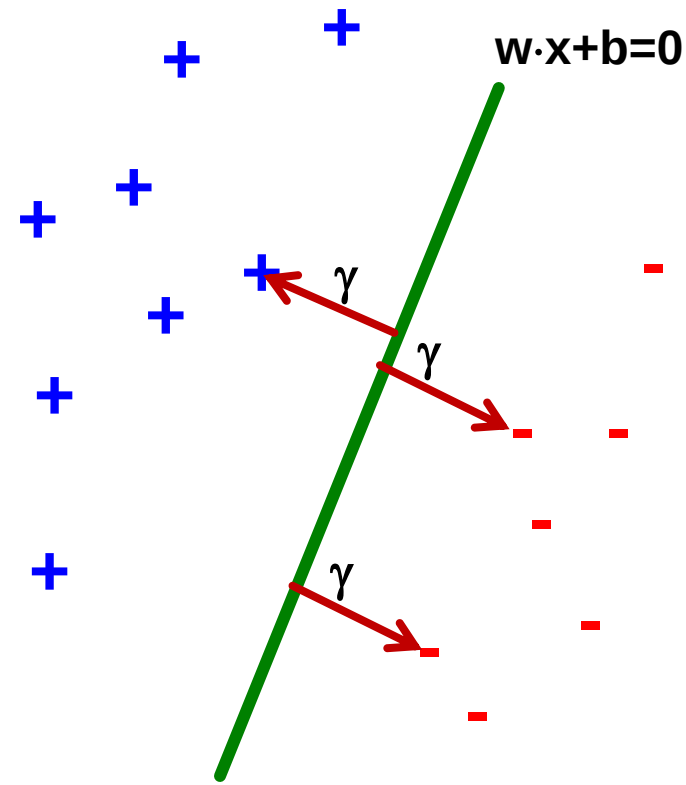
Support Vector Machine

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

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

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

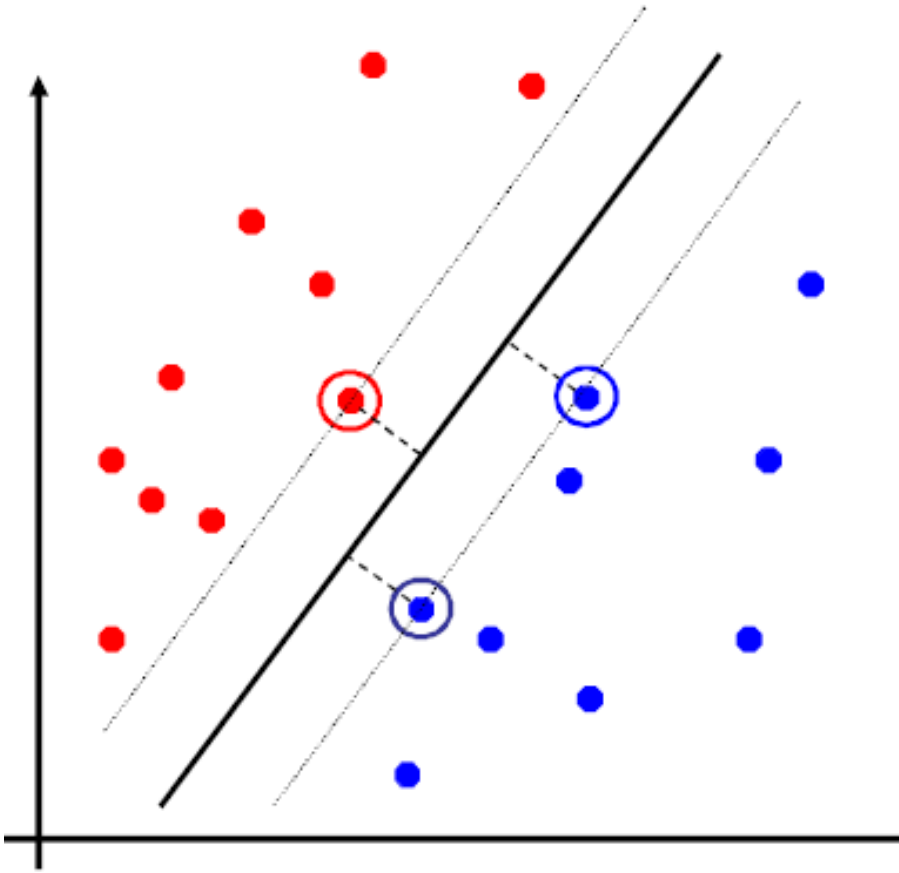
  - **γ is margin … distance from the separating hyperplane**



Maximizing the margin

Support Vector Machines

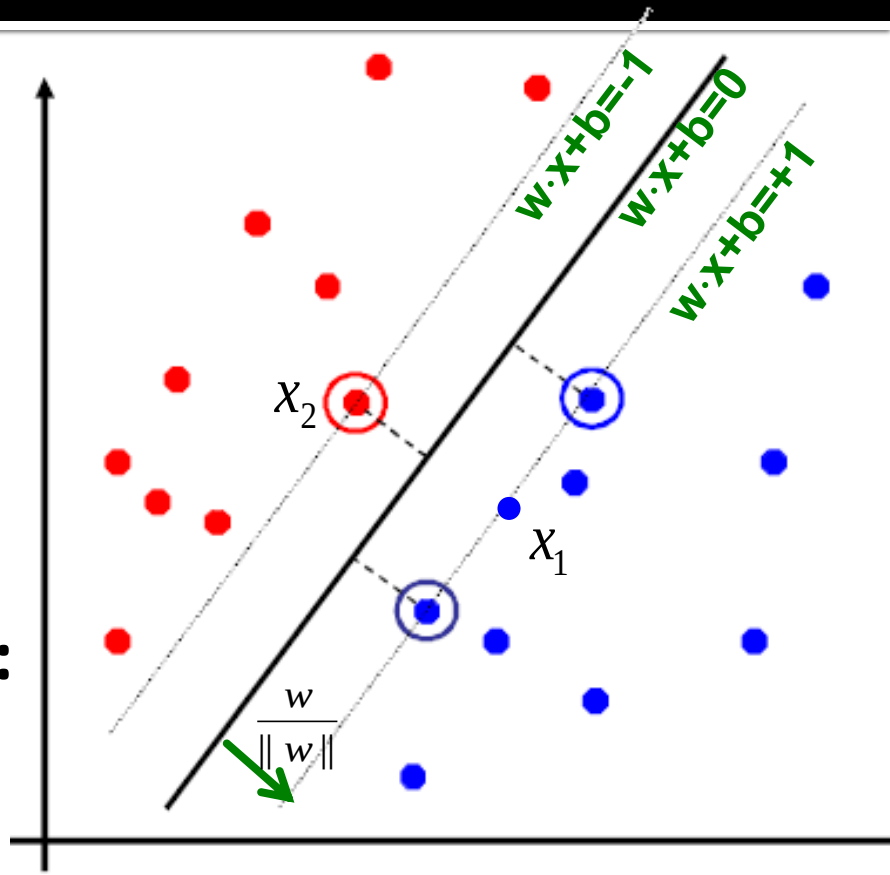
- **Separating hyperplane is defined by the support vectors**
  - Points on **+/-** planes from the solution
  - If you knew these points, you could ignore the rest
  - If no degeneracies, $d+1$ support vectors (for d dim. data)



Canonical Hyperplane: Problem

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

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



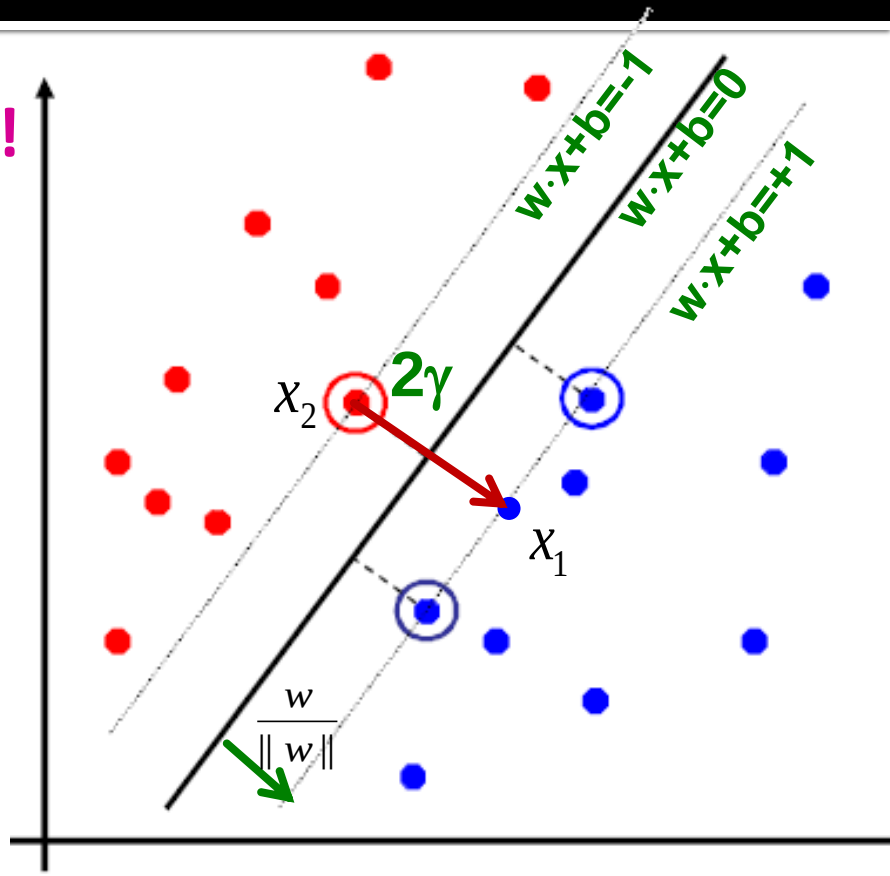
  - Let's also require **support vectors** $\boldsymbol{x_j}$ to be on the plane defined by:

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

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

Canonical Hyperplane: Solution

- Want to maximize margin γ!
- 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...**

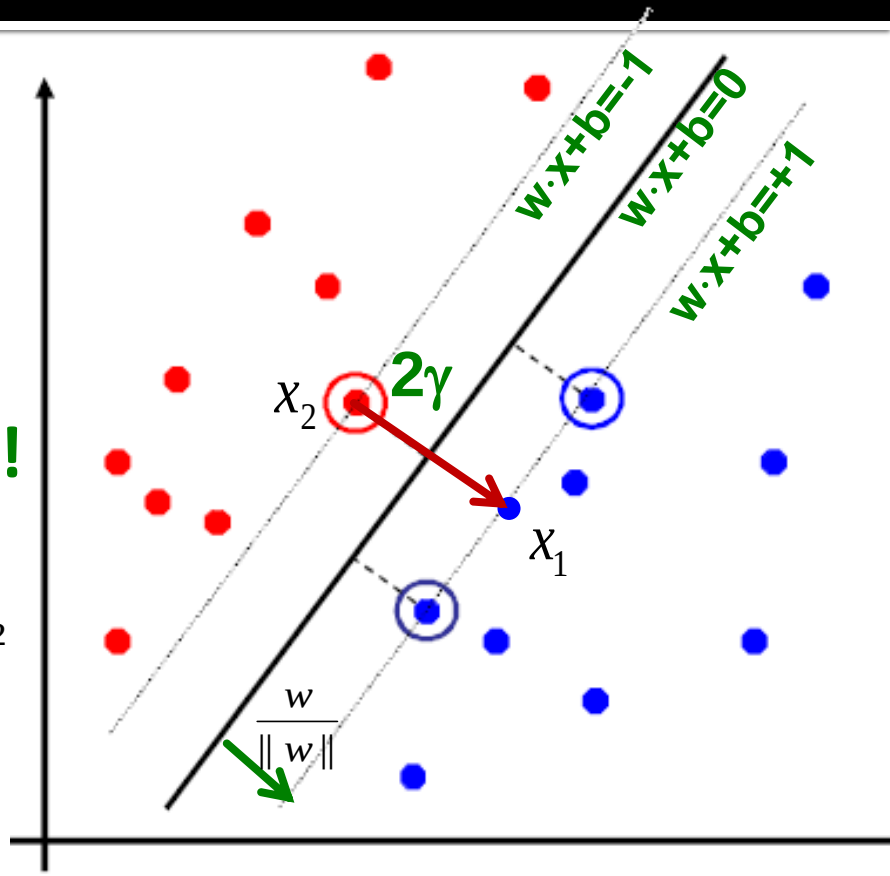
$$\max \gamma \approx \max \frac{1}{\|w\|} \approx \min \|w\| \approx \min \tfrac{1}{2}\|w\|^2$$

- **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?

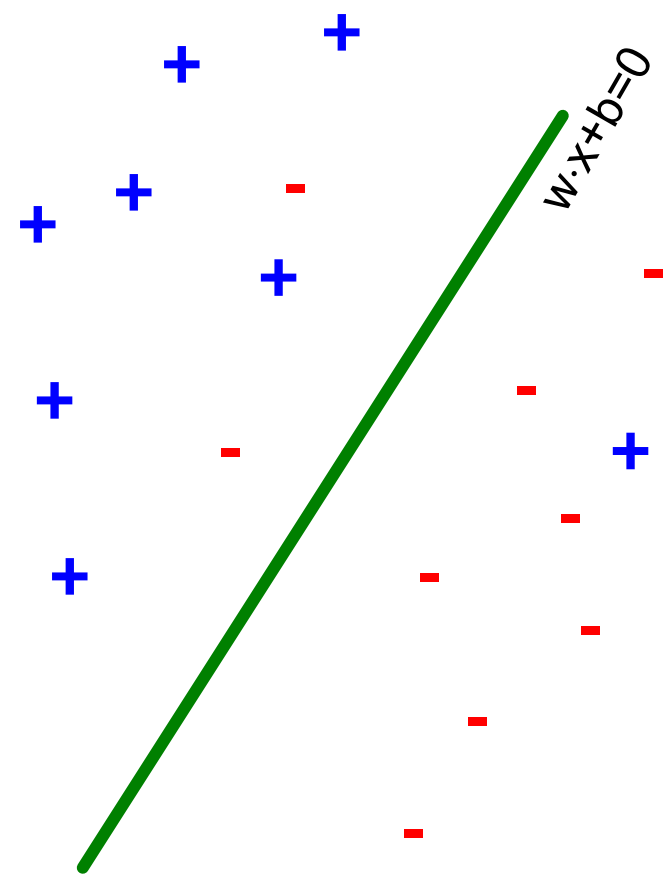
- **If data is not separable introduce penalty:**

$$\min_w \tfrac{1}{2}\|w\|^2 + C\ (\#\text{number of mistakes})$$

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

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

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



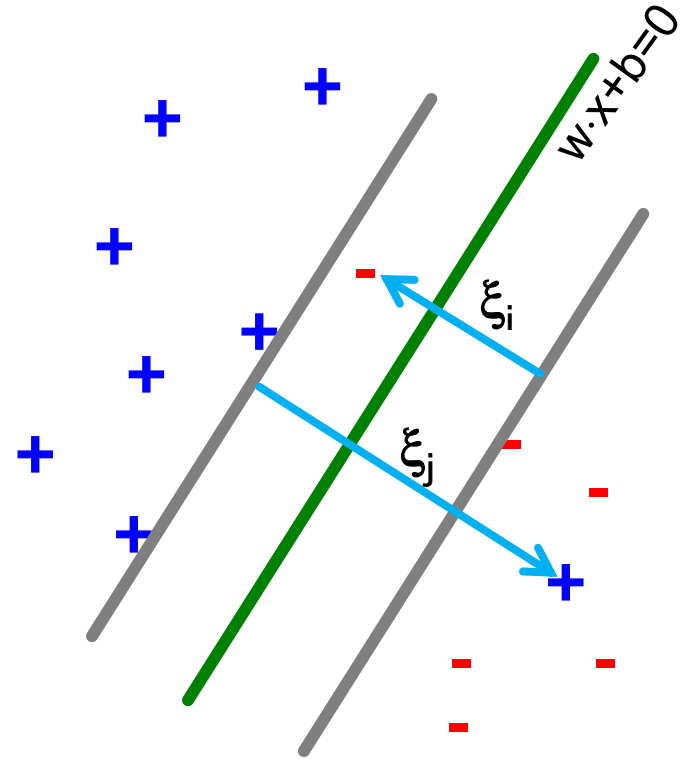
Support Vector Machines

- **Introduce slack variables ξ_i**

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

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

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



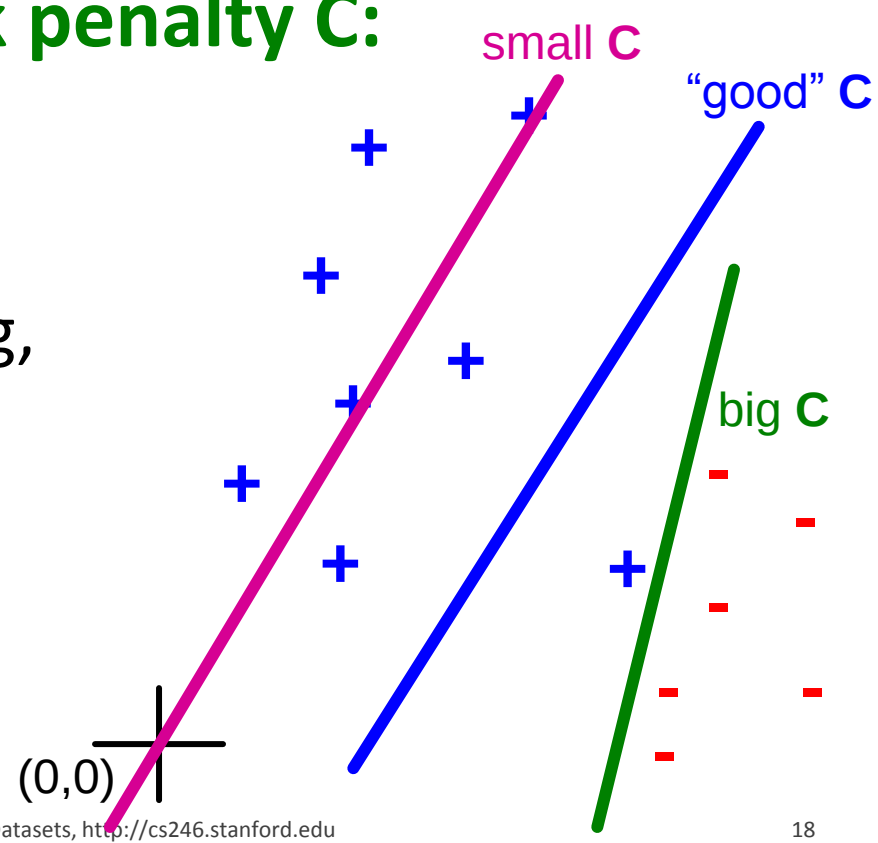
For each datapoint:
If margin ≥ 1, don't care
If margin < 1, pay linear penalty

Slack Penalty *C*

$$\min_w \tfrac{1}{2}\|w\|^2 + \mathrm{C}\ (\#\text{number of mistakes})$$

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

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



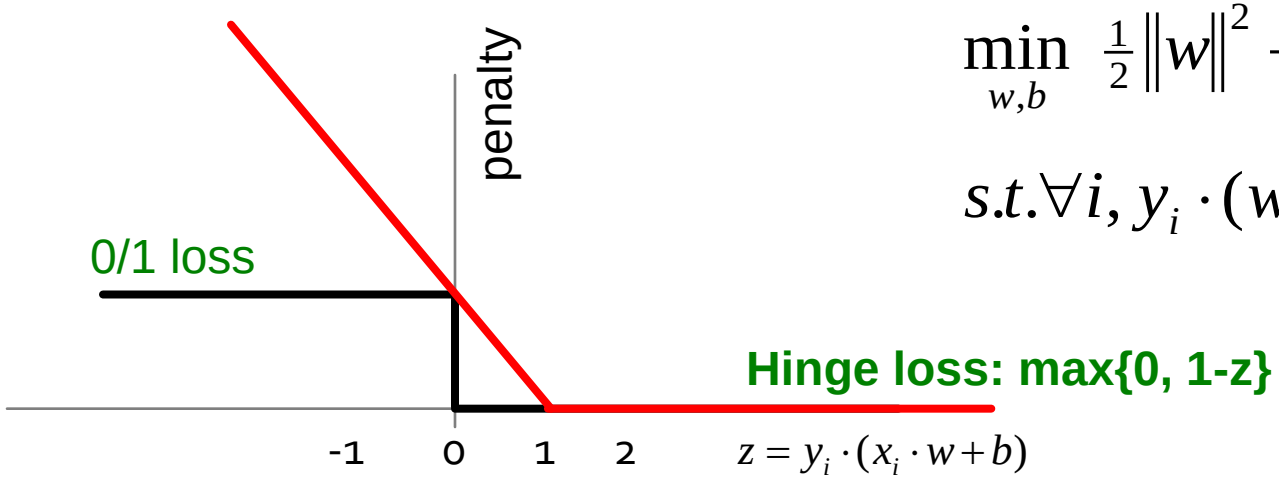
Support Vector Machines

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

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

Regularization parameter

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



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

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

Announcement: HW2 is graded. We sorted it alphabetically into several piles. Please don't mess the piles.

SVM: How to estimate *w*,*b*?

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 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*?

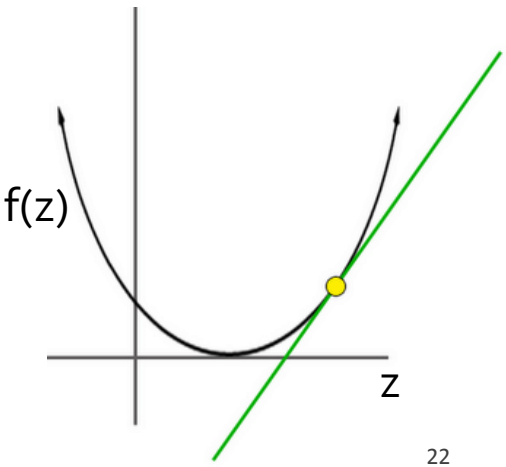
$$\min_{w,b} \ \tfrac{1}{2} w \cdot w + C \sum_{i=1}^{n} \xi_i$$

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

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

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

  - **How to minimize convex functions f(z)?**
  - Use gradient descent: **min$_z$ f(z)**
  - Iterate: $\mathbf{z_{t+1} \leftarrow z_t - \eta\, f'(z_t)}$



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*$^{(j)}$**

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

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

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

SVM: How to estimate *w*?

- **Gradient descent:**

Iterate until convergence:

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

    $w^{(j)} \leftarrow w^{(j)} - \eta\nabla(j)$

η…learning rate parameter
C… regularization parameter

- **Problem:**
  - **Computing ∇(j) takes O(n) time!**
    - **n** … size of the training dataset

SVM: How to estimate *w*?

We just had:

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

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

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

- **Stochastic gradient descent:**

Iterate until convergence:

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

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

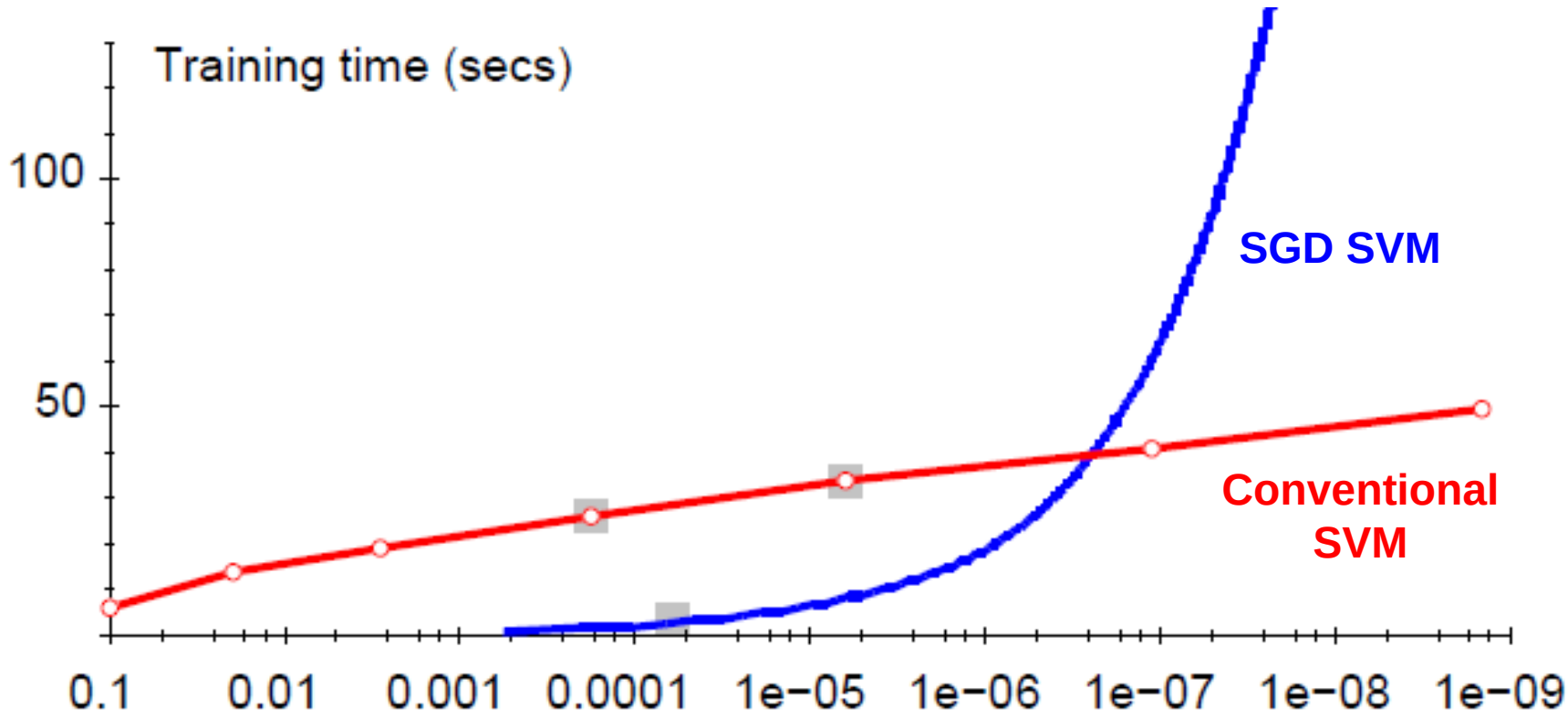
Example: Text categorization

- **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"

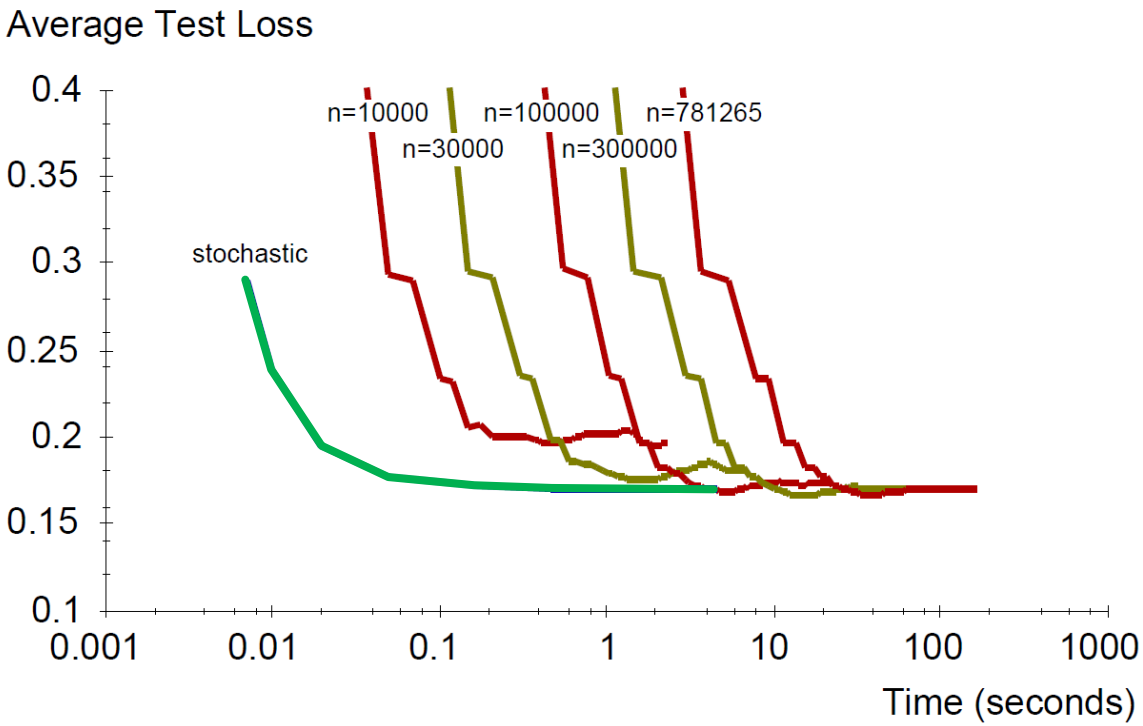


Optimization quality: | *f(w,b) – f (w^opt,b^opt)* |

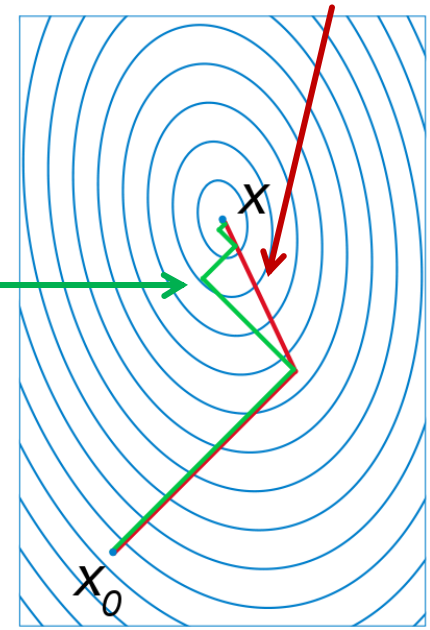
For optimizing *f(w,b) within reasonable* quality
SGD-SVM is super fast

SGD vs. Batch Conjugate Gradient

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



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



$\boldsymbol{k}$… condition number

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

Practical Considerations

- **Need to choose learning rate η and t_0**

$$w_{t+1} \leftarrow w_t - \frac{\eta_t}{t+t_0}\left(w_t + C\frac{\partial L(x_i, y_i)}{\partial w}\right)$$

- **Leon suggests:**
  - Choose $\mathbf{t_0}$ so that the expected initial updates are comparable with the expected size of the weights
  - Choose η:
    - Select a **small subsample**
    - Try various rates η (e.g., 10, 1, 0.1, 0.01, …)
    - Pick the one that most reduces the cost
    - Use η for next 100k iterations on the full dataset

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 $\mathbf{x}_i = [(4,1), (9,5), \ldots]$
  - **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:
    - $(1)\ v = v - \eta C \frac{\partial L(x_i, y_i)}{\partial w}$
    - $(2)\ s = s(1 - \eta)$
- **Solution 2:**
  - Perform only step **(1)** for each training example
  - Perform step **(2)** with lower frequency and higher η

Two step update procedure:

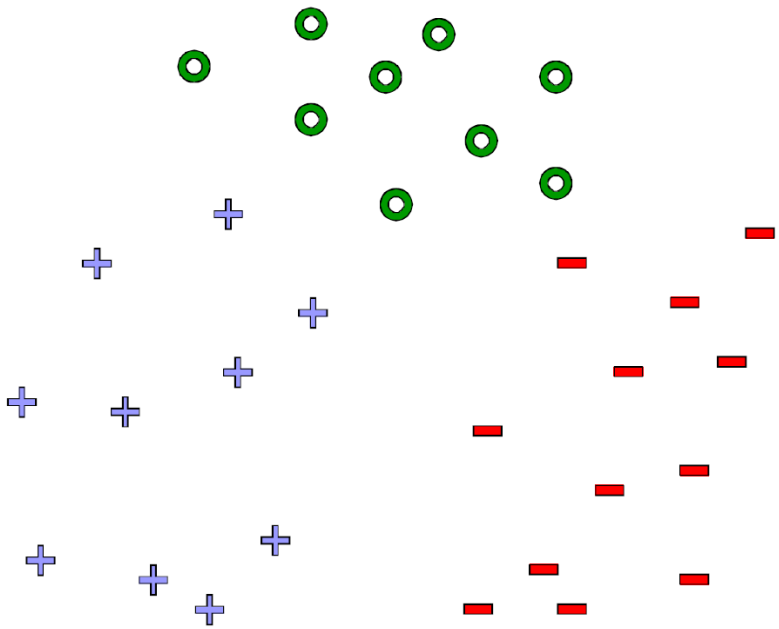
$$(1)\ w \leftarrow w - \eta C \frac{\partial L(x_i, y_i)}{\partial w}$$

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

Practical Considerations

- **Stopping criteria:**
  How many iterations of SGD?
  - **Early stopping with cross validation**
    - Create validation set
    - Monitor cost function on the validation set
    - Stop when loss stops decreasing
  - **Early stopping**
    - Extract two disjoint subsamples **A** and **B** of training data
    - Train on **A**, stop by validating on **B**
    - Number of epochs 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:

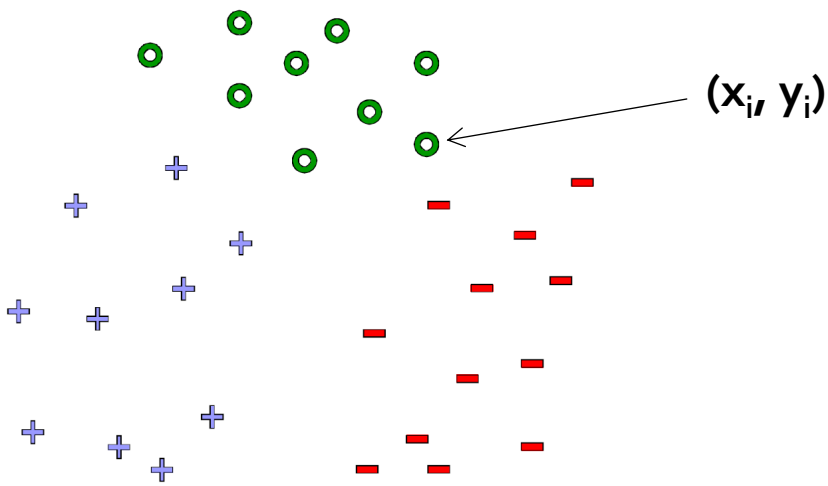
  $\mathbf{w_+ b_+,\ w_- b_-,\ w_o b_o}$
- **How to classify?**
- Return class *c*

  $\mathbf{\arg\max_c\ w_c x + b_c}$

Learn 1 classifier: Multiclass SVM

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

$$\mathbf{w_{y^i}\, x_i + b_{y^i} \;\geq\; 1 + w_c\, x_i + b_c \quad \forall c \neq y_i \;,\; \forall i}$$



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**

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**
  - All our methods SVM and Perceptron make updates if they misclassify an example
  - **So:** First train the classifier on training data. Then for every example from the stream, if we misclassify, update the model (using 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: y = f(x; w)**
  - **Accept: y =1, Not accept: y=-1**
  - Build this model with say Perceptron or Winnow
- **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