Large-scale Classification and Regression

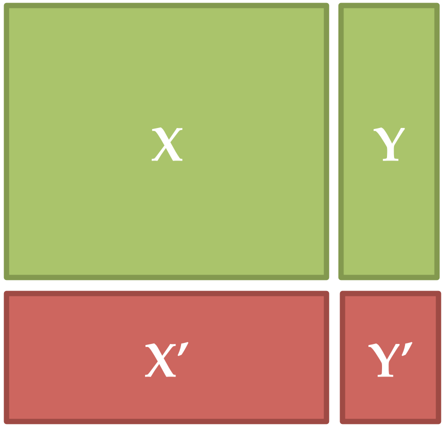
Shannon Quinn

(with thanks to J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org)

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.



Training and **test** set

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

- **Data is labeled:**
  - Have many pairs **{(x, y)}**
    - **x** … vector of binary, categorical, real valued features
    - **y** … class ({+1, -1}, or a real number)

Large Scale Machine Learning

- **We will talk about the following methods:**
  - **k**-Nearest Neighbor (Instance based learning)
  - Perceptron (neural networks)
  - Support Vector Machines
  - Decision trees

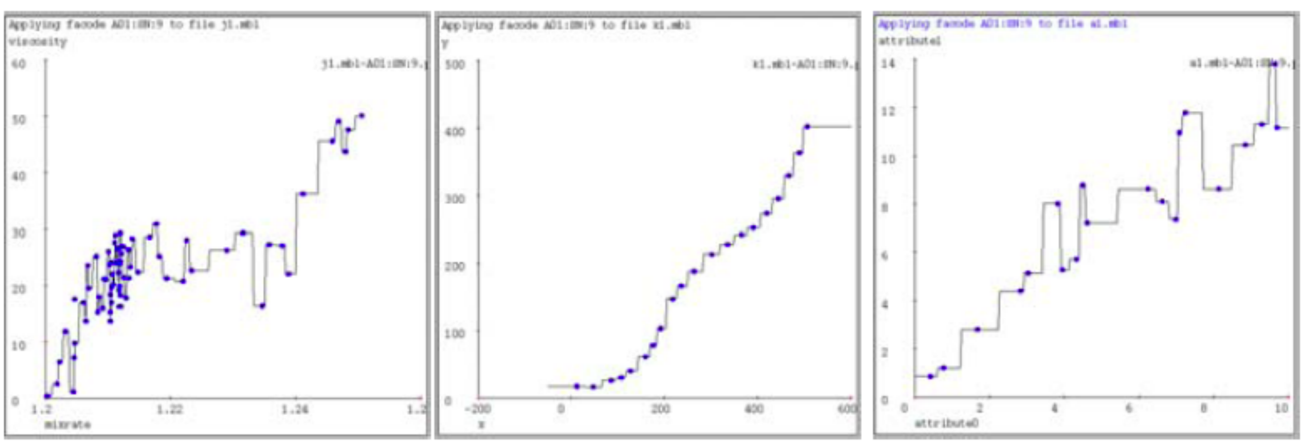
- **Main question:**
  How to efficiently train
  (build a model/find model parameters)?

Instance Based Learning

- **Instance based learning**
- **Example: Nearest neighbor**
  - Keep the whole training dataset: **{(x, y)}**
  - A query example (vector) q comes
  - Find closest example(s) $\mathbf{x}^*$
  - Predict $\mathbf{y}^*$
- **Works both for regression and classification**
  - **Collaborative filtering** is an example of k-NN classifier
    - Find k most similar people to user $\mathbf{x}$ that have rated movie $\mathbf{y}$
    - Predict rating $\mathbf{y}_\mathbf{x}$ of $\mathbf{x}$ as an average of $\mathbf{y}_\mathbf{k}$

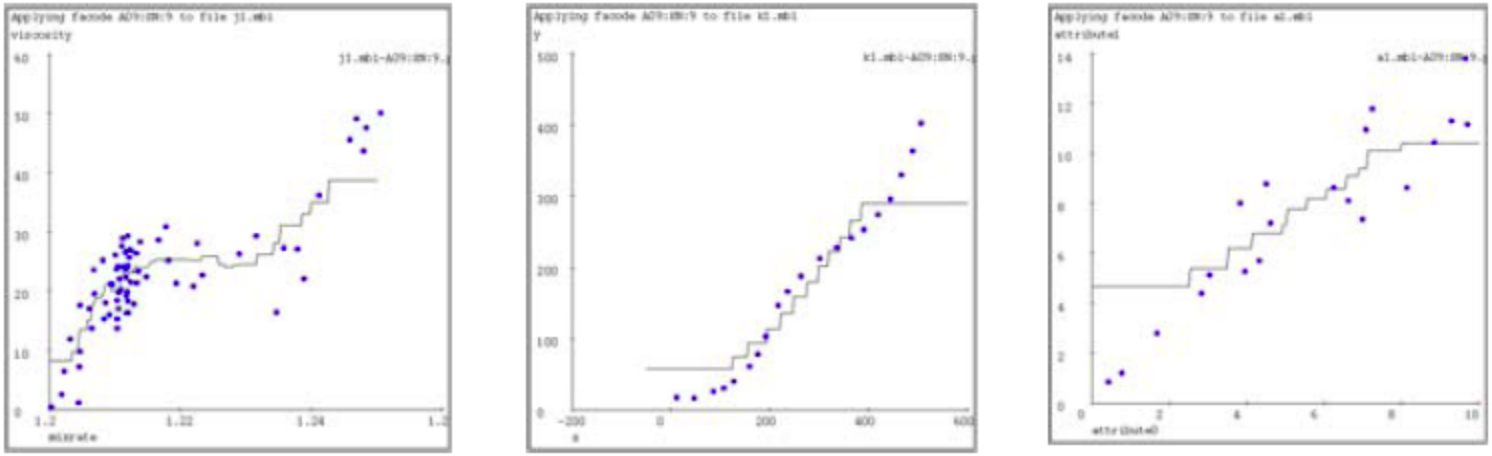
1-Nearest Neighbor

- **To make Nearest Neighbor work we need 4 things:**
  - **Distance metric:**
    - Euclidean
  - **How many neighbors to look at?**
    - One
  - **Weighting function (optional):**
    - Unused
  - **How to fit with the local points?**
    - Just predict the same output as the nearest neighbor



k-Nearest Neighbor

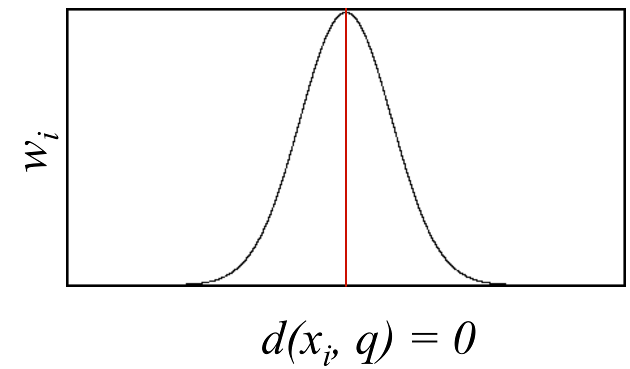
- **Distance metric:**
  - Euclidean
- **How many neighbors to look at?**
  - *k*
- **Weighting function (optional):**
  - Unused
- **How to fit with the local points?**
  - Just predict the average output among *k* nearest neighbors



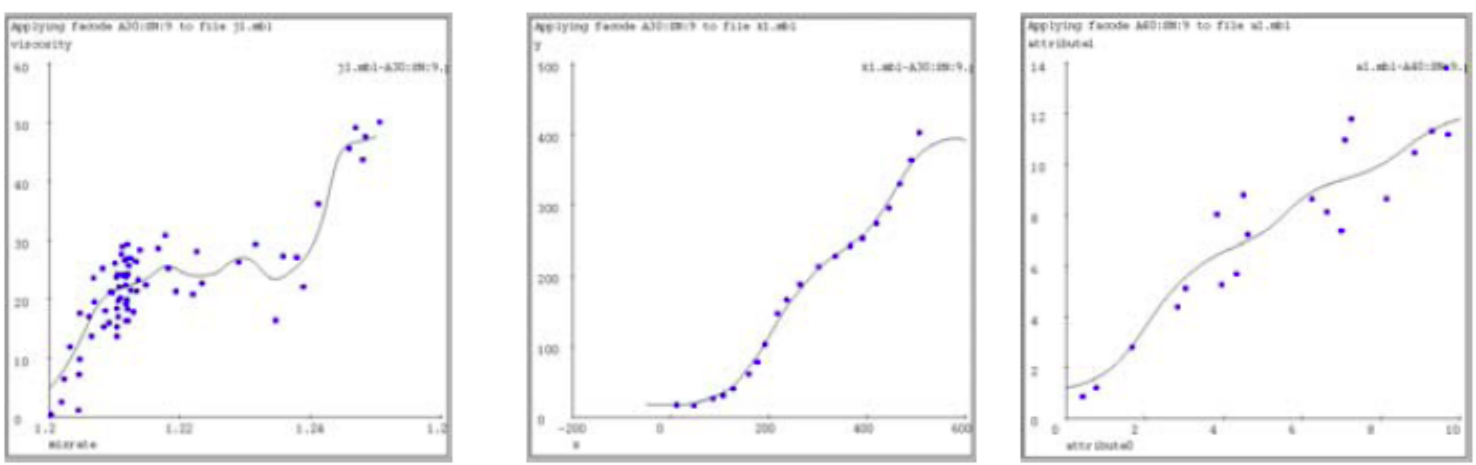
k=9

Kernel Regression

- **Distance metric:**
  - Euclidean
- **How many neighbors to look at?**
  - **All of them (!)**
- **Weighting function:**
  - $w_i = \exp(-d(x_i, q)^2 / K_w)$
    - Nearby points to query q are weighted more strongly. $\mathbf{K_w}$… kernel width.

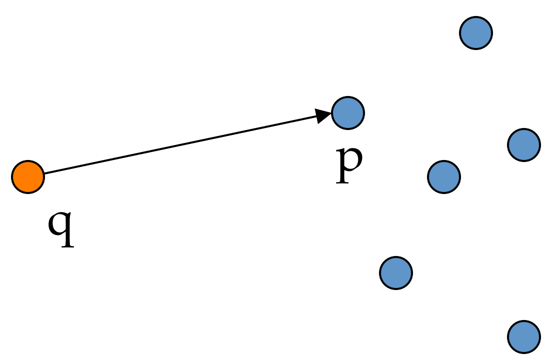


- **How to fit with the local points?**
  - **Predict weighted average:** $\sum_i w_i y_i / \sum_i w_i$



How to find nearest neighbors?

- **Given:** a set P of n points in R^d
- **Goal: Given a query point** q
  - **NN:** Find the *nearest neighbor* p of q in P
  - **Range search:** Find one/all points in P within distance r from q



Algorithms for NN

- **Main memory:**
  - **Linear scan**
  - **Tree based:**
    - Quadtree
    - kd-tree
  - **Hashing:**
    - Locality-Sensitive Hashing

(1958)
F. Rosenblatt

The perceptron: a probabilistic model
for information storage and organization in the brain
Psychological Review 65:386–408

Perceptron

Linear models: Perceptron

- **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)

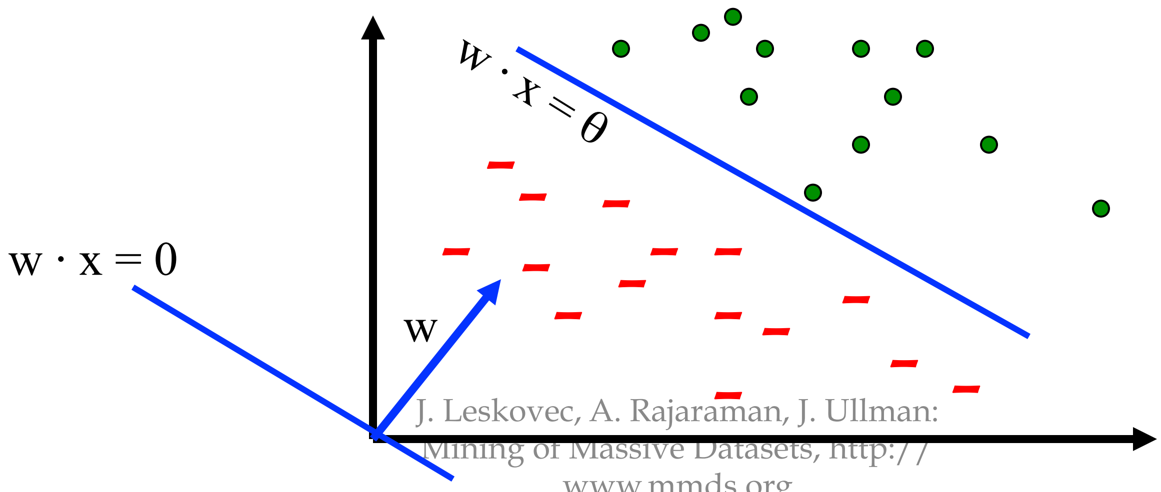
Linear models for classification

Decision boundary y is **linear**

- **Binary classification:**

$$f(\mathbf{x}) = \begin{cases} +1 & \text{if } \mathbf{w}_1 \mathbf{x}_1 + \mathbf{w}_2 \mathbf{x}_2 + \ldots \mathbf{w}_d \mathbf{x}_d \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

- **Input:** Vectors $x^{(j)}$ and labels $y^{(j)}$
  – Vectors $x^{(j)}$ are real valued where
- **Goal:** Find vector $w = (w_1, w_2, \ldots, w_d)$
  – Each w_i is a real number



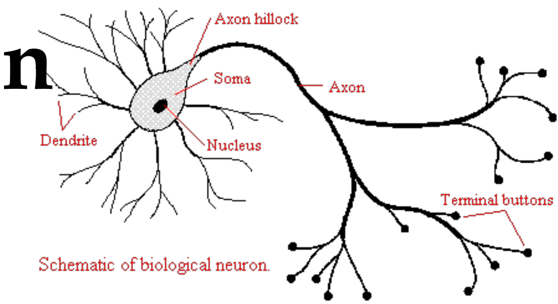
Note:

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

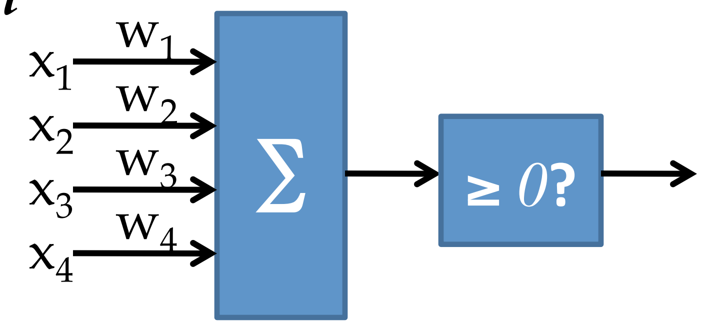
$$\mathbf{w} \Leftrightarrow \langle \mathbf{w}, -\theta \rangle$$

Perceptron [Rosenblatt '58]

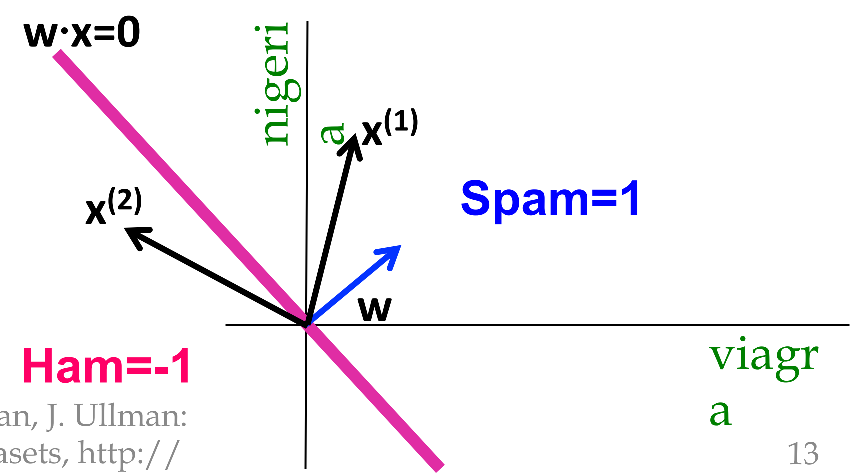
- (very) Loose motivation: **Neuron**



- Inputs are feature values
- Each feature has a weight w_i
- **Activation is the sum:**
  - $f(x) = \Sigma_i w_i x_i = w \cdot x$



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



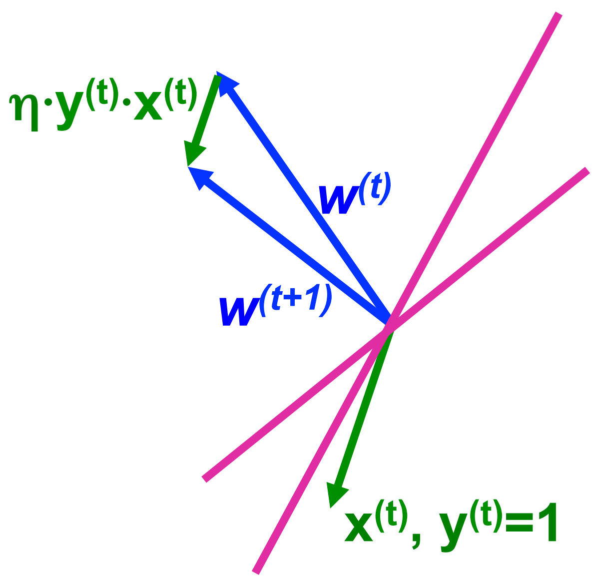
Perceptron: Estimating *w*

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 (from disk)**
  - Predict class of $x^{(t)}$ using current weights
    - $y' = sign(w^{(t)} \cdot x^{(t)})$
  - **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})



Perceptron Convergence

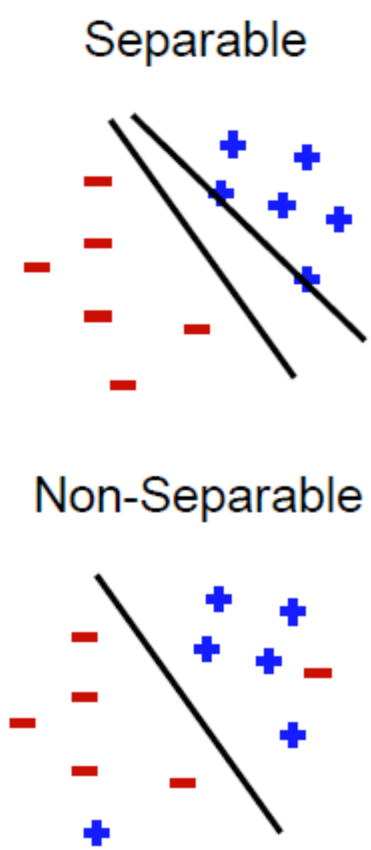
- **Perceptron Convergence Theorem:**
  - If there exist a set of weights that are consistent (i.e., the data is linearly separable) the Perceptron learning algorithm will converge
- **How long would it take to converge?**
- **Perceptron Cycling Theorem:**
  - If the training data is not linearly separable the Perceptron learning algorithm will eventually repeat the same set of weights and therefore enter an infinite loop
- **How to provide robustness, more expressivity?**

Properties of Perceptron

- **Separability:** Some parameters get training set perfectly
- **Convergence:** If training set is separable, perceptron will converge
- **(Training) Mistake bound:** Number of mistakes
  - **where**

    and

    - Note we assume **x** Euclidean length **1**, then γ is the minimum distance of any example to plane **u**



Updating the Learning Rate

- **Perceptron will oscillate and won't converge**
- **When to stop learning?**
- **(1)** Slowly decrease the learning rate η
  - A classic way is to: $\eta = c_1/(t + c_2)$
    - But, we also need to determine constants c_1 and c_2
- **(2)** Stop when the training error stops chaining
- **(3)** Have a small test dataset and stop when the test set error stops decreasing
- **(4)** Stop when we reached some maximum number of passes over the data

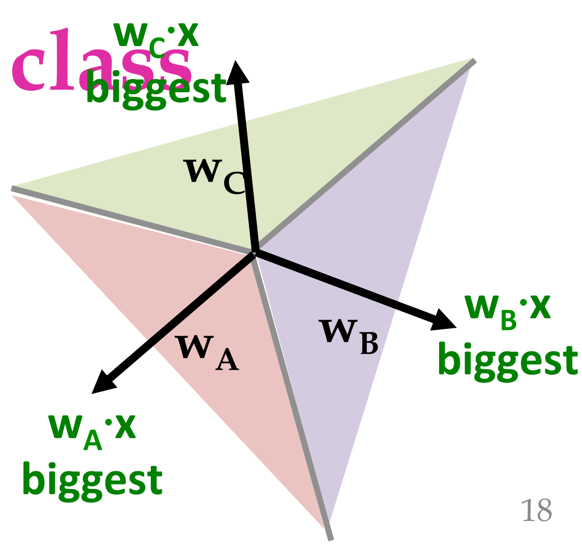
Multiclass Perceptron

- **What if more than 2 classes?**
- Weight vector w_c for each class **c**
  - **Train one class vs. the rest:**
    - <u>Example:</u> 3-way classification **y = {A, B, C}**
    - Train 3 classifiers: $\mathbf{w_A}$: A vs. B,C; $\mathbf{w_B}$: B vs. A,C; $\mathbf{w_C}$: C vs. A,B
- **Calculate activation for each class**

$$f(x,c) = \Sigma_i \; w_{c,i} \, x_i \; = \; w_c \cdot x$$

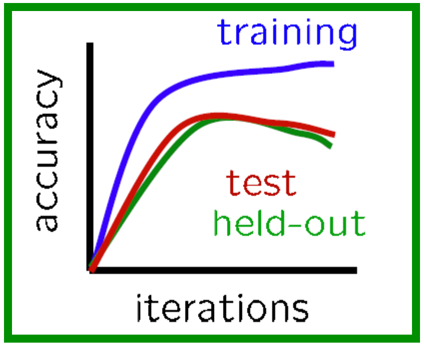
- **Highest activation wins**

$$c = \arg\max_c f(x,c)$$

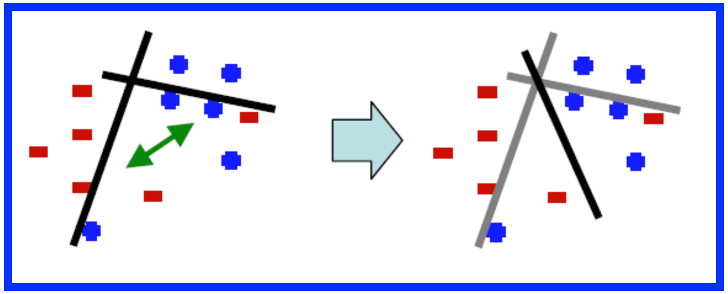


Issues with Perceptrons

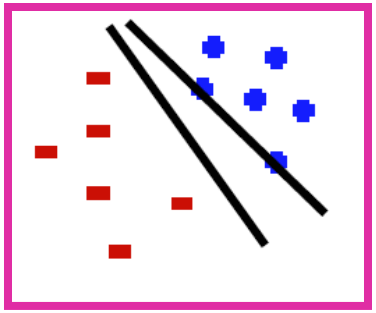
- **Overfitting:**



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

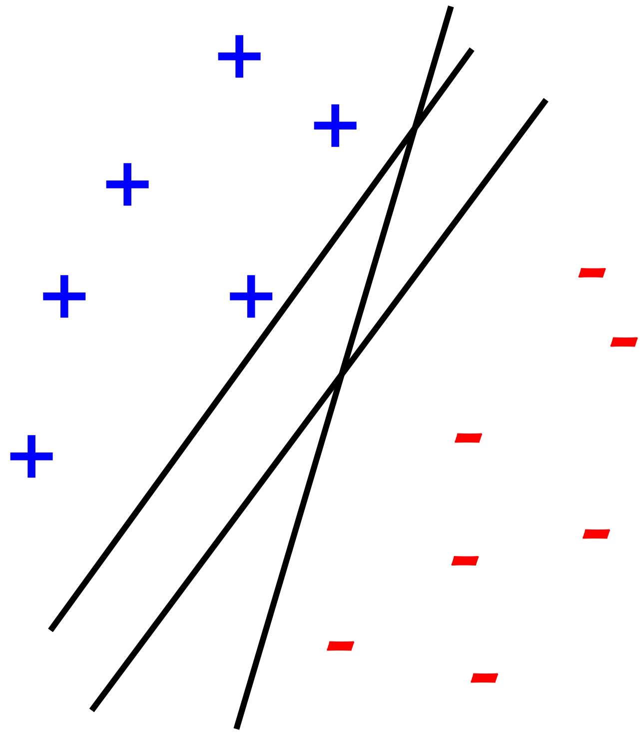


- **Mediocre generalization:**
  - Finds a "barely" separating solution



Support Vector Machines

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

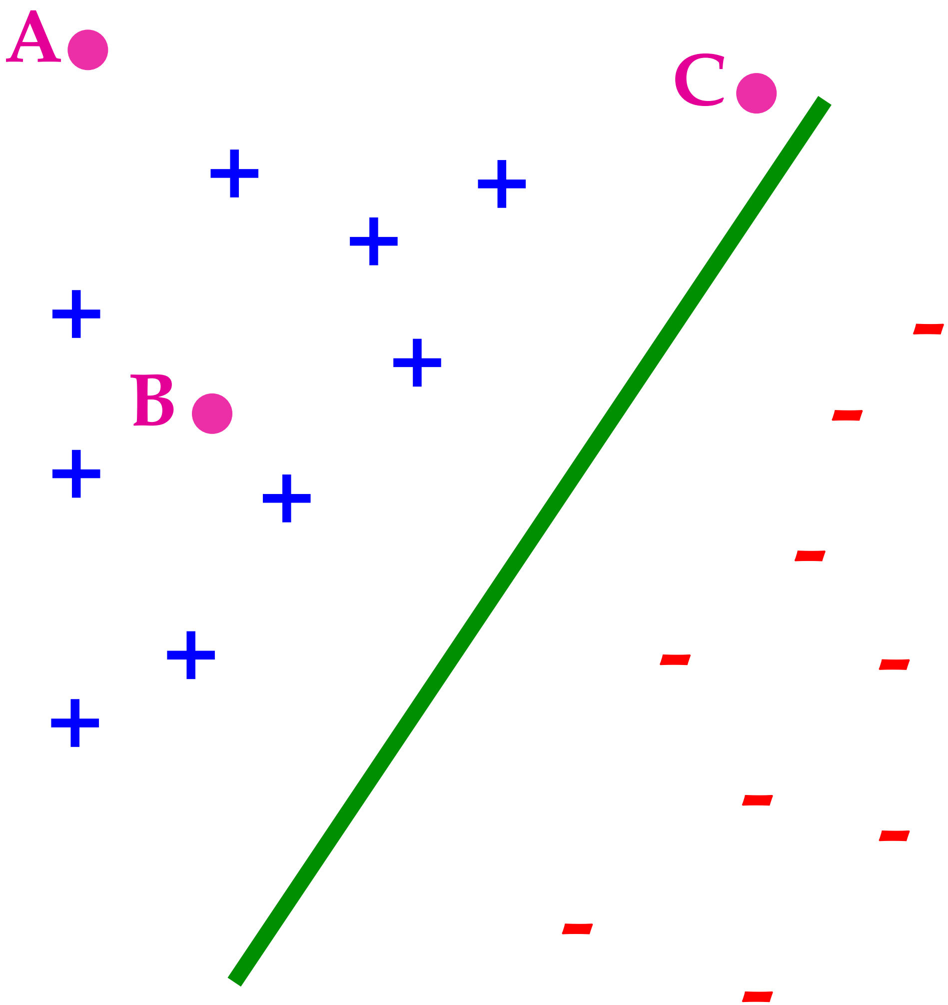


Data:

- **Training examples:**
  - $(x_1, y_1) \dots (x_n, y_n)$
- **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:**

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

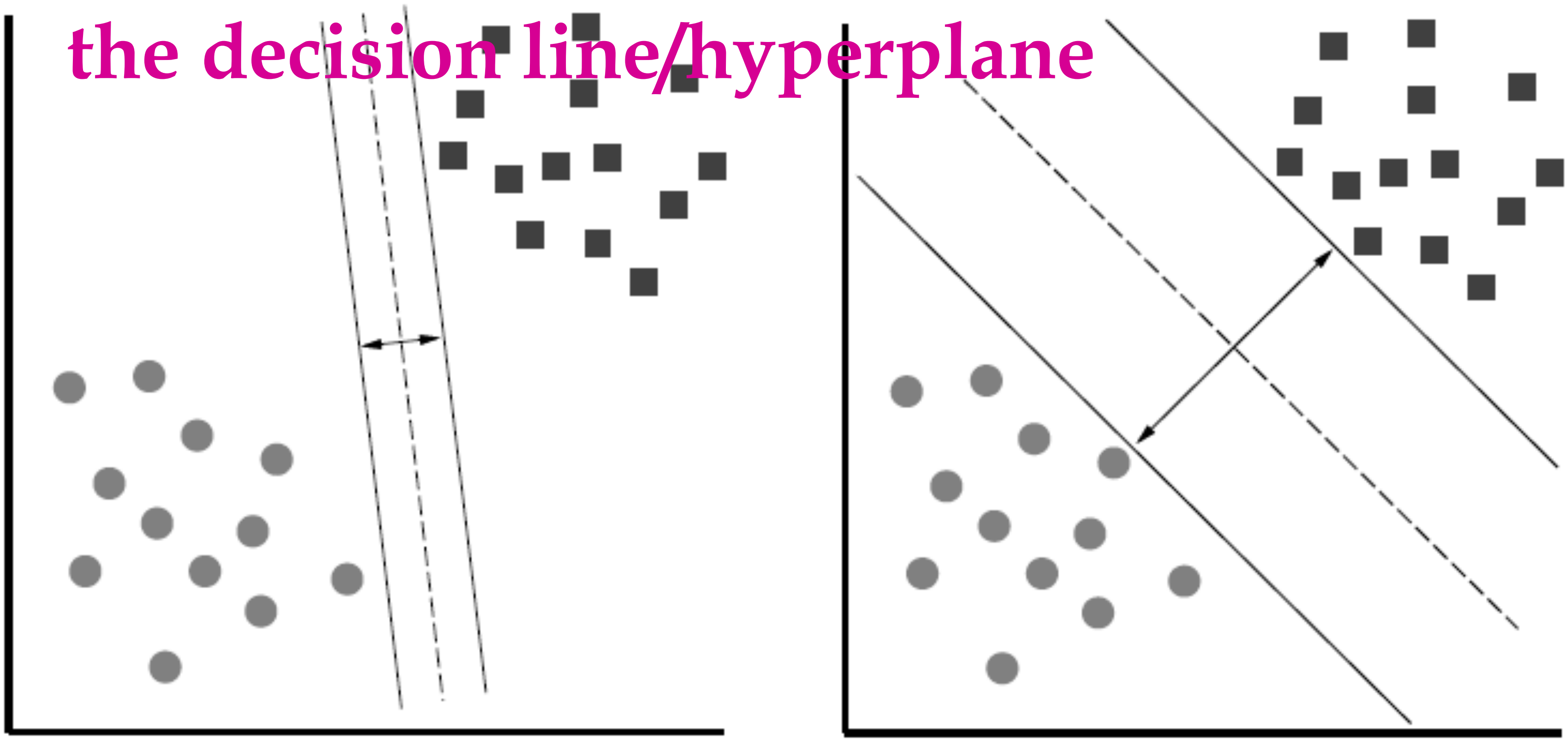
Largest Margin



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

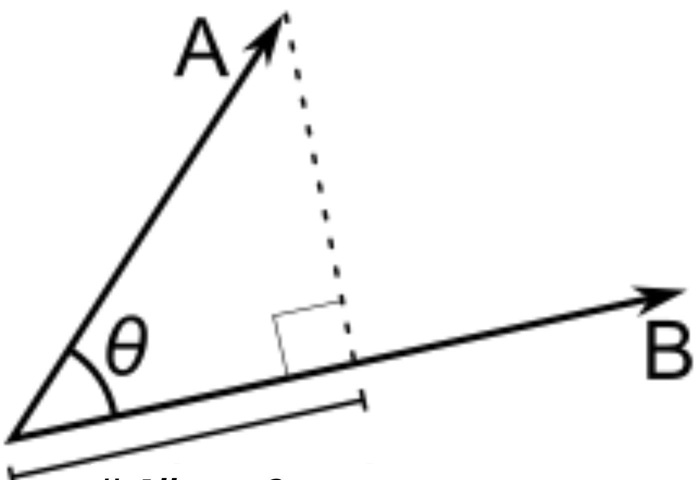


The reason we define margin this way is due to theoretical convenience and existence of generalization error bounds that depend on the value of margin.

Why maximizing γ a good idea?

- Remember: Dot product

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

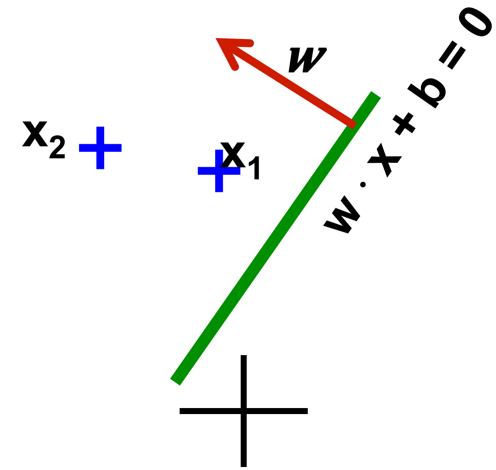


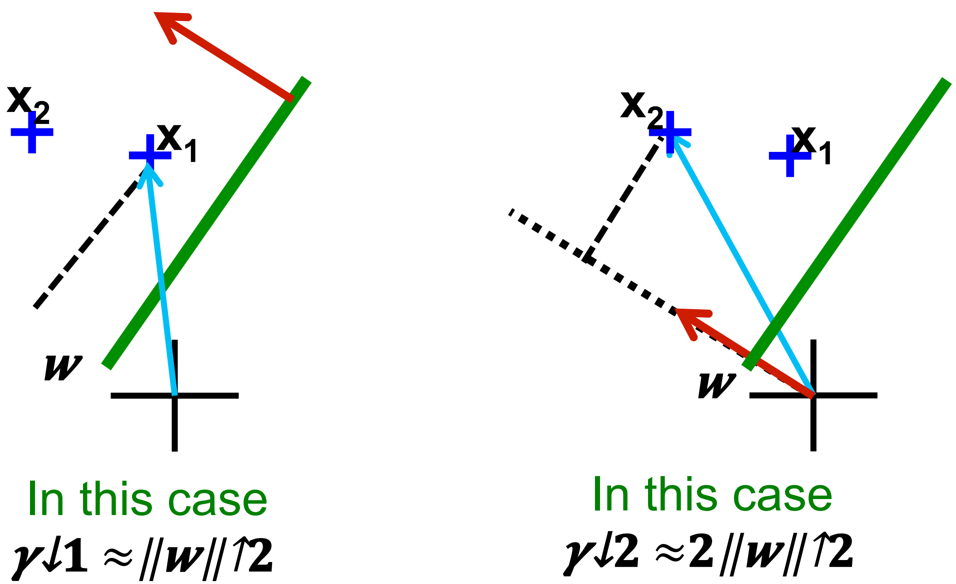
$\|A\| \cos\theta$

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

Why maximizing γ a good idea?

- Dot product
- What is , ?

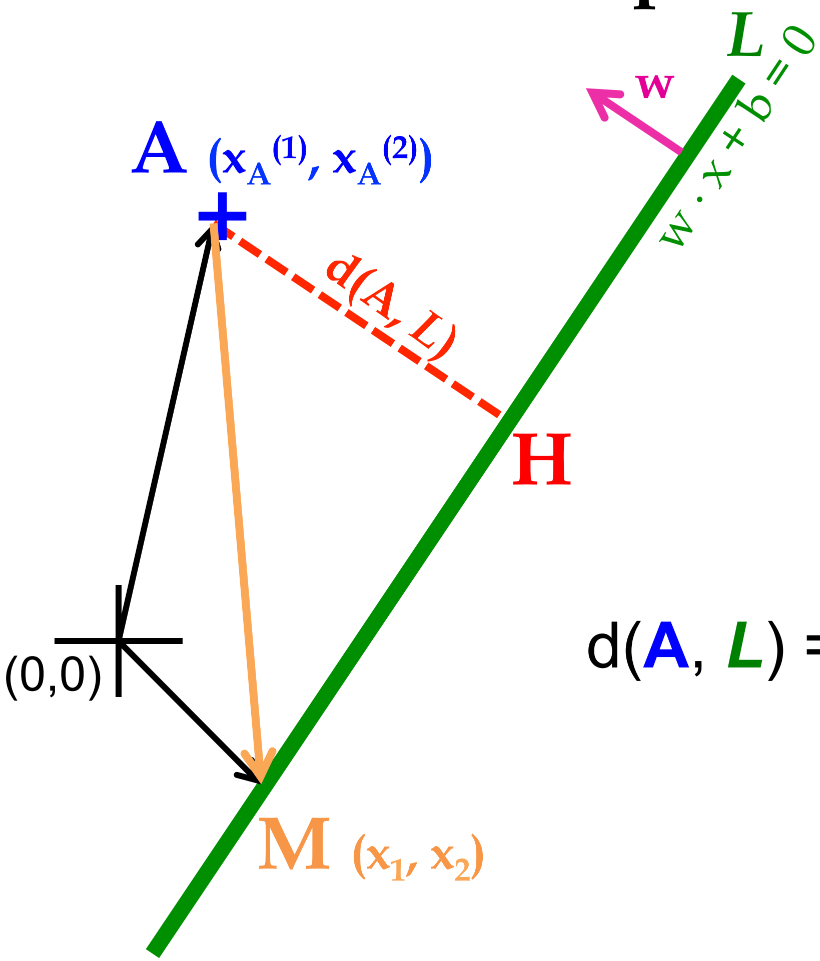




- So, roughly corresponds to the margin
  – Bigger bigger the separation

What is the margin?

Distance from a point to a line



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

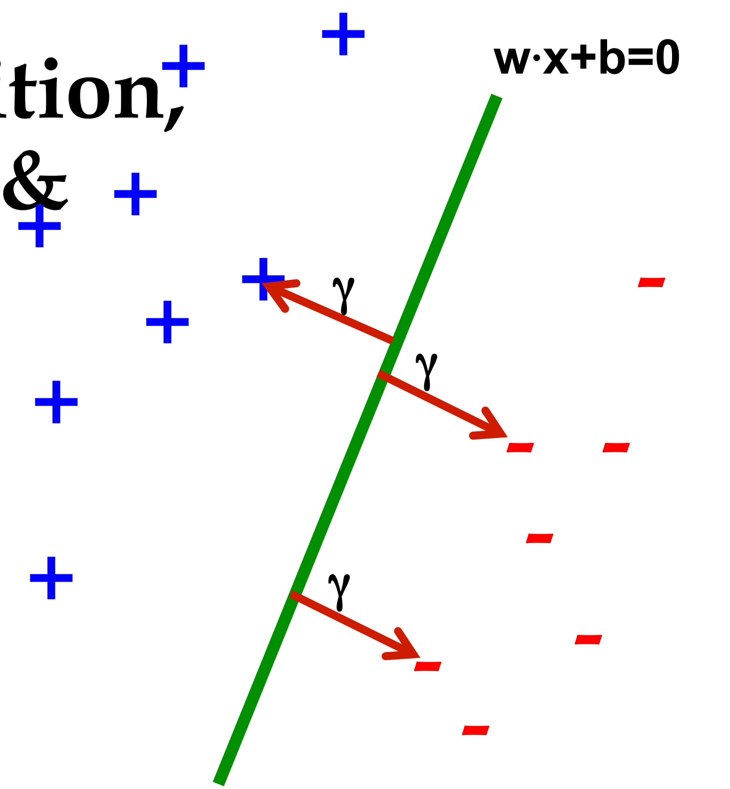
- **Let:**
  - **Line L: w·x+b =** $w^{(1)}x^{(1)}+w^{(2)}x^{(2)}+b=0$
  - **w** = $(w^{(1)}, w^{(2)})$
  - **Point A** = $(x_A^{(1)}, x_A^{(2)})$
  - **Point M** on a line = $(x_M^{(1)}, x_M^{(2)})$

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

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

Support Vector Machine

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



Maximizing the margin

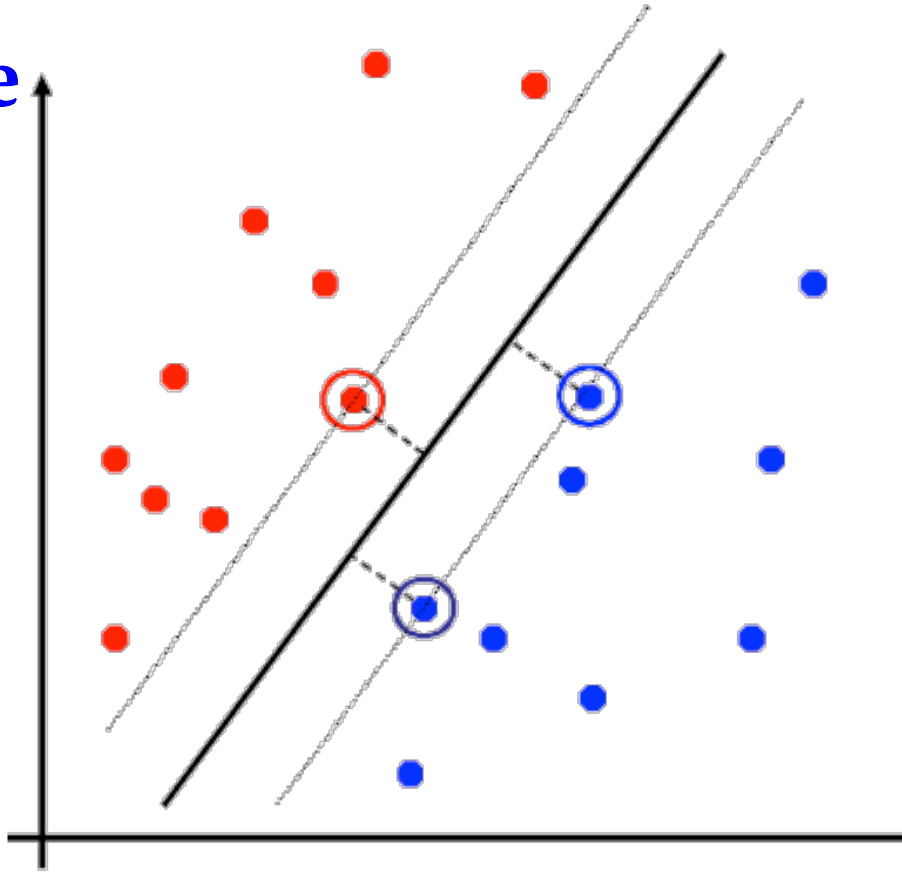
$$\max_{w,\gamma} \gamma$$

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

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

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, ***d*+*1*** support vectors (for ***d*** dim. data)



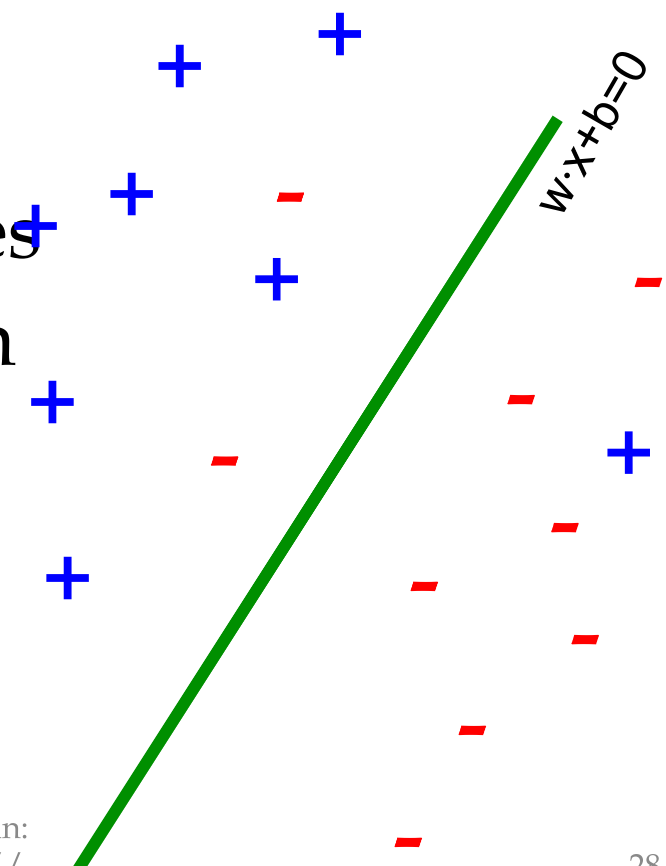
Non-linearly Separable Data

- **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!**

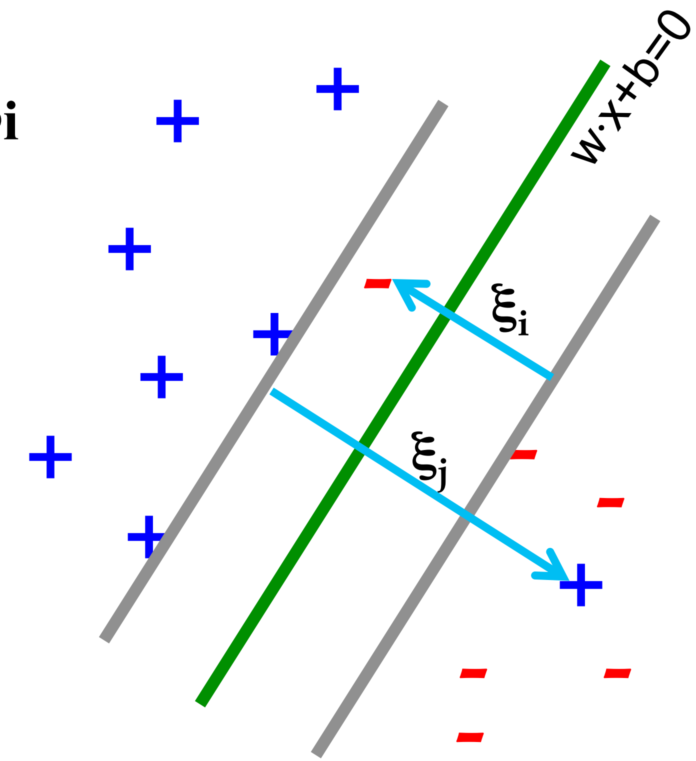
Support Vector Machines

- **Introduce** **slack variables** ξ_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 ξ_i

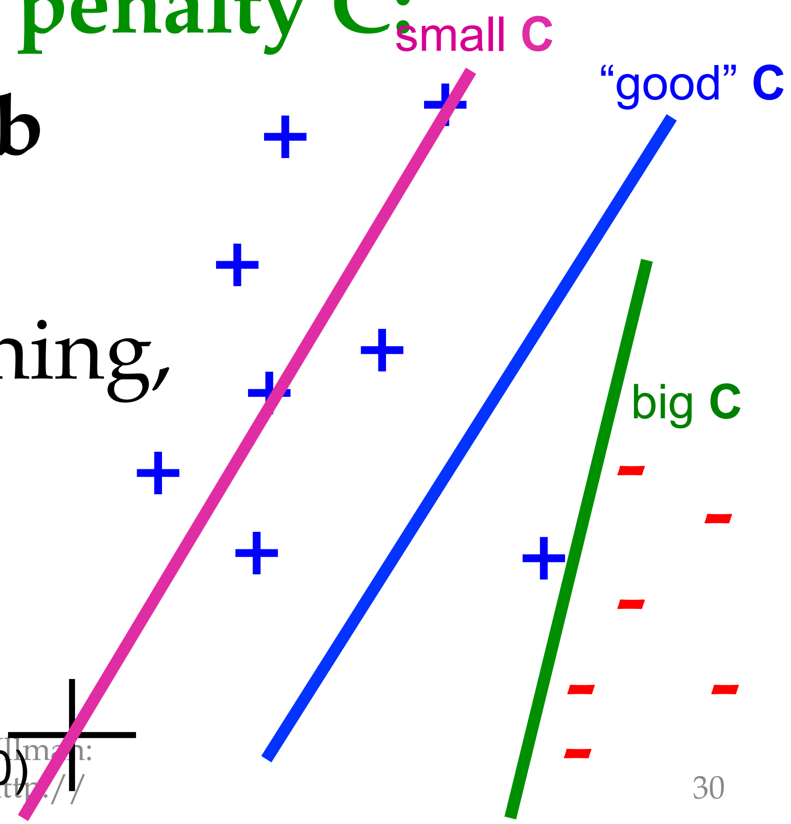


For each data point:
If margin ≥ 1, don't care
If margin < 1, pay linear penalty

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 ξ_i to anything, then **w=0** (basically ignores the data)



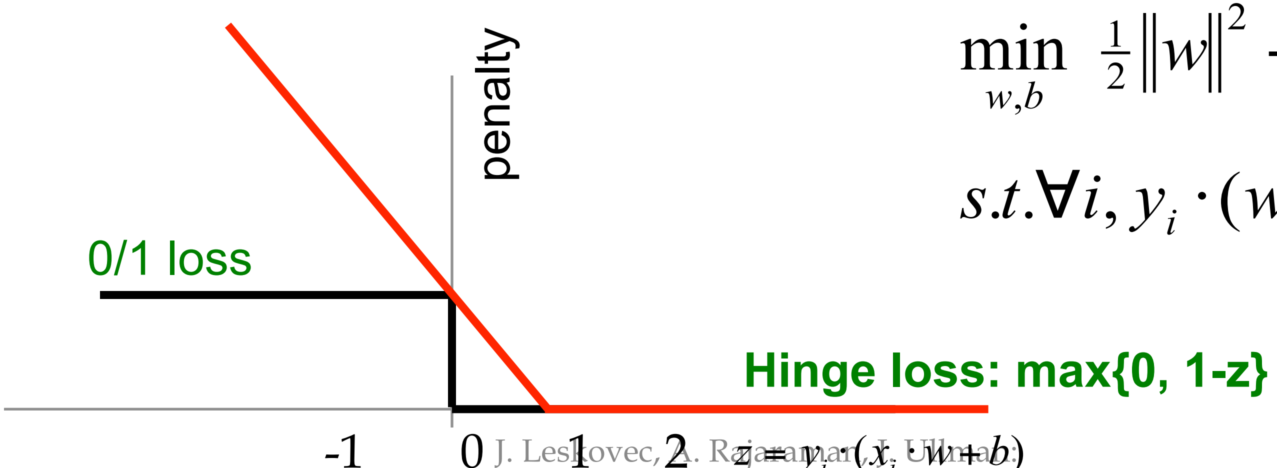
Support Vector Machines

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

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

Regularization parameter

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



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

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

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 and !**
  - **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*?

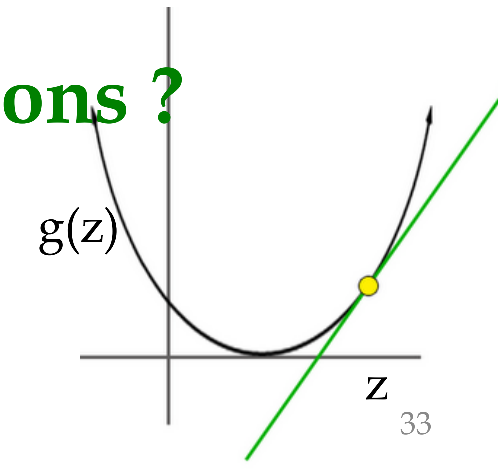
- **Want to estimate w, b!**
- **Alternative approach:**
  - **Want to minimize *f(w,b)*:**

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

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

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

- **Side note:**
  - **How to minimize convex functions ?**
  - Use gradient descent: $\mathbf{min}_z\ \mathbf{g(z)}$
  - 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:**
    $w^{(j)} \leftarrow w^{(j)} - \eta \nabla f^{(j)}$

η…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: $w^{(j)} \leftarrow w^{(j)} - \eta \nabla f^{(j)}(x_i)$**

An observation

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

Key computational point:

- If $x^i = 0$ then the gradient of w^j is zero
- so when processing an example you only need to update weights for the **non-zero** features of an 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

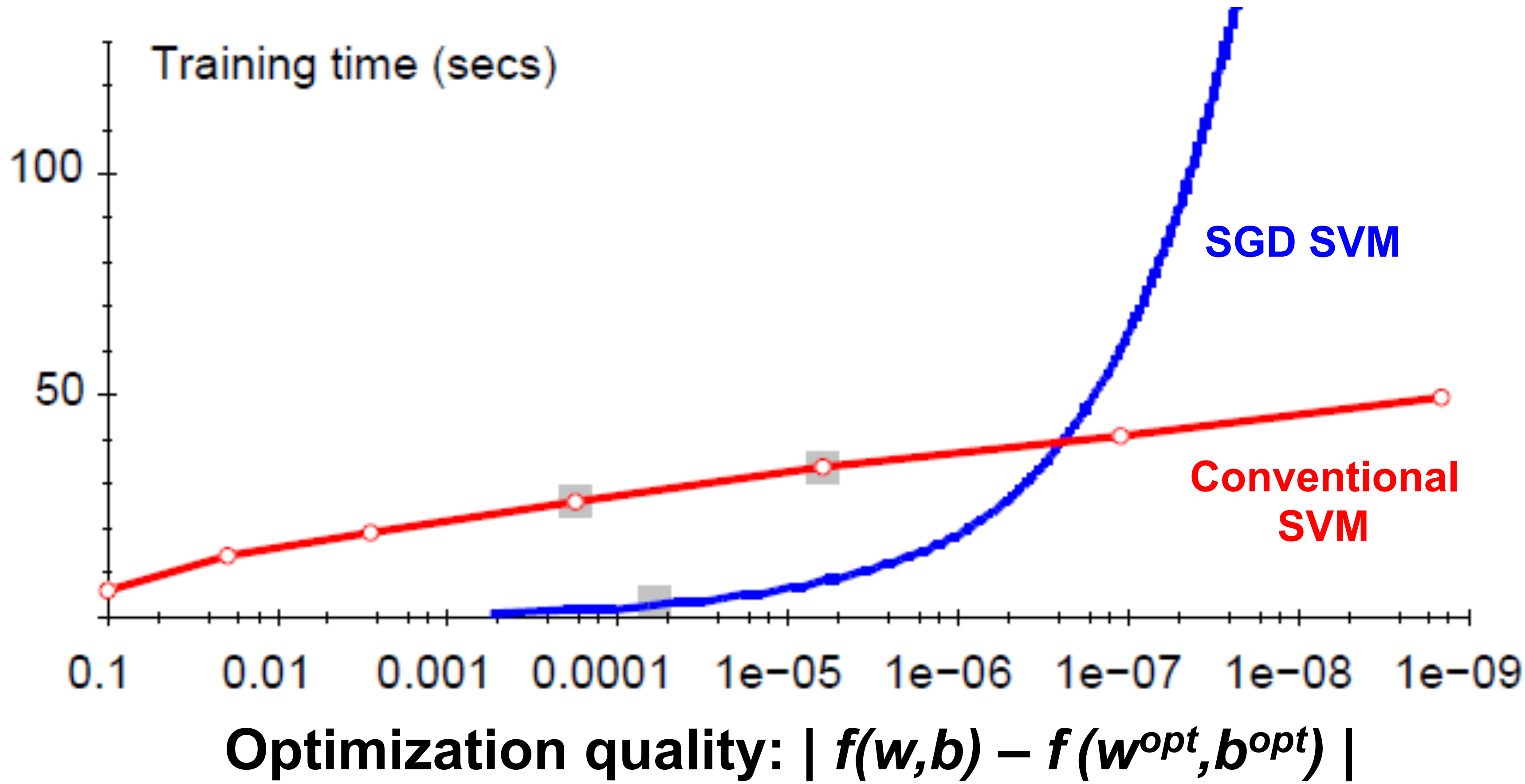
Example: Text categorization

- **Questions:**
  - **(1)** Is **SGD** successful at minimizing $f(w,b)$?
  - **(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

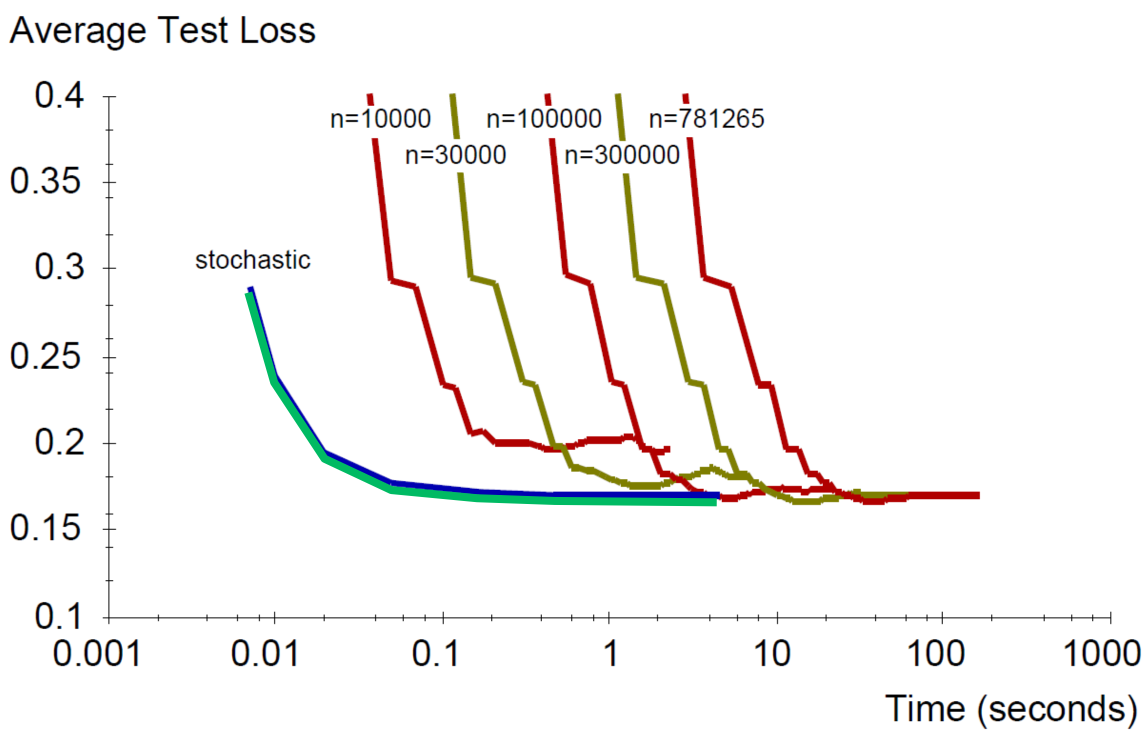
Optimization "Accuracy"



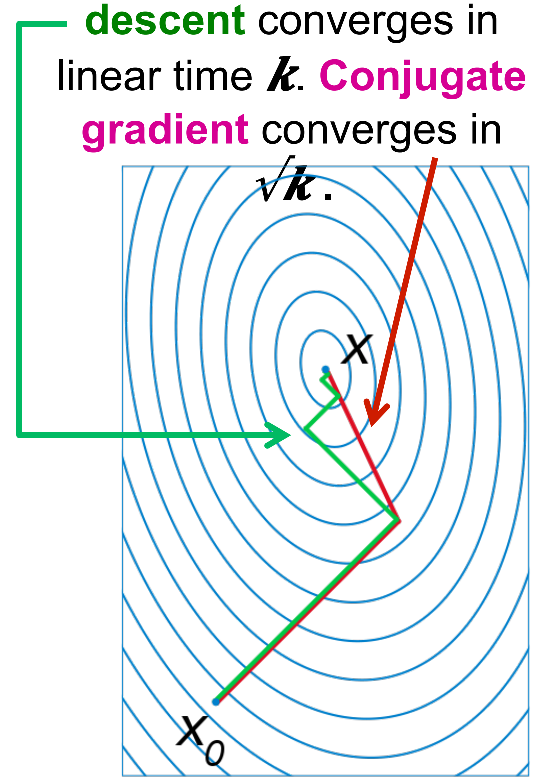
For optimizing *f(w,b) within reasonable* quality
SGD-SVM is super fast

SGD vs. Batch Conjugate Gradient

- **SGD** on full dataset vs. **Conjugate Gradient** on a sample of $\boldsymbol{n}$ 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) CG 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