# Lecture 12: Kernel SVM & Ensemble Methods

Winter 2018

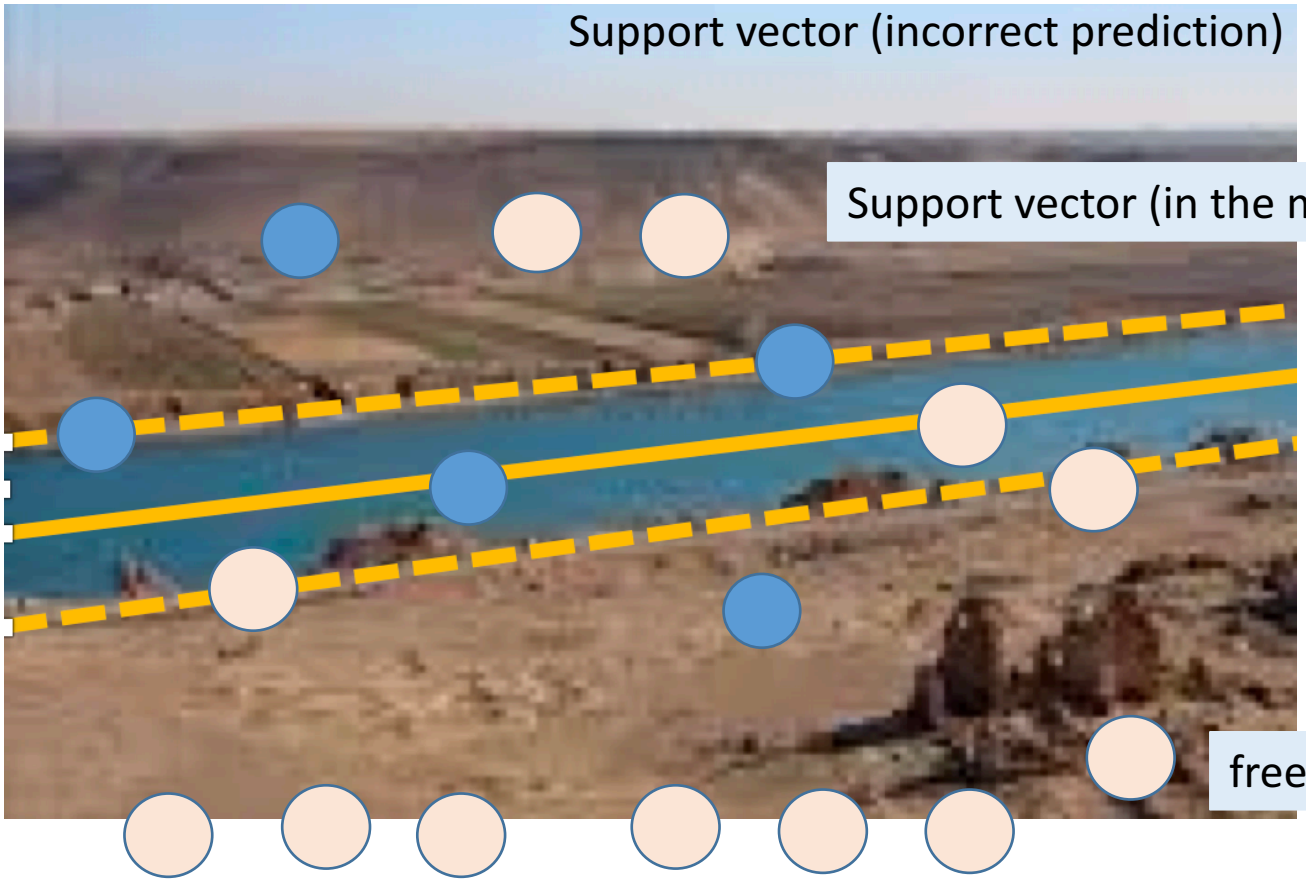
Kai-Wei Chang

CS @ UCLA

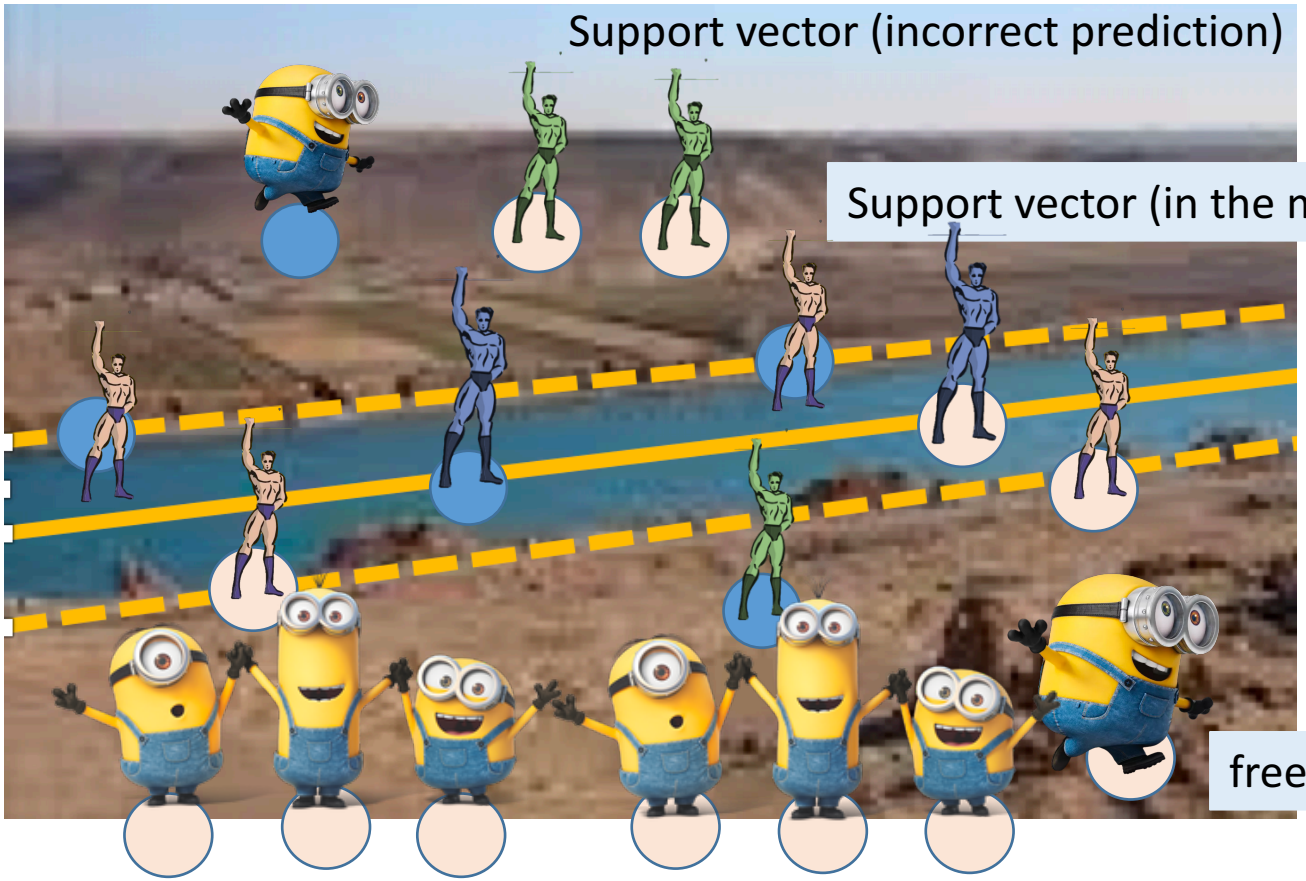
kw+cm146@kwchang.net

The instructor gratefully acknowledges Dan Roth, Vivek Srikuar, Sriram Sankararaman, Fei Sha, Ameet Talwalkar, Eric Eaton, and Jessica Wu whose slides are heavily used, and the many others who made their course material freely available online.

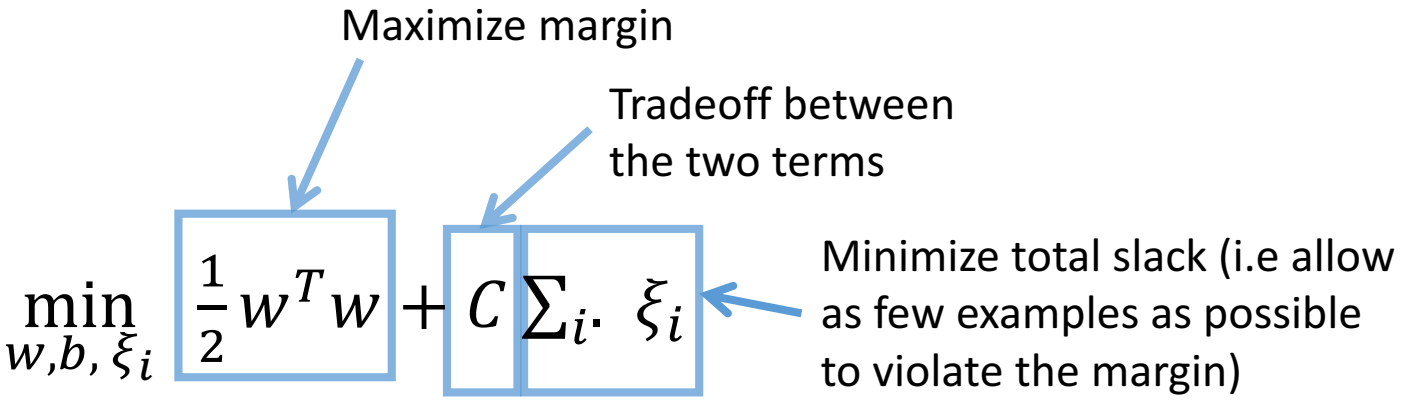
# Recap: support vector machines



# Recap: support vector machines



# Recap Soft SVM




$$\min_{w,b,\xi_i} \frac{1}{2} w^T w + C \sum_i \xi_i$$

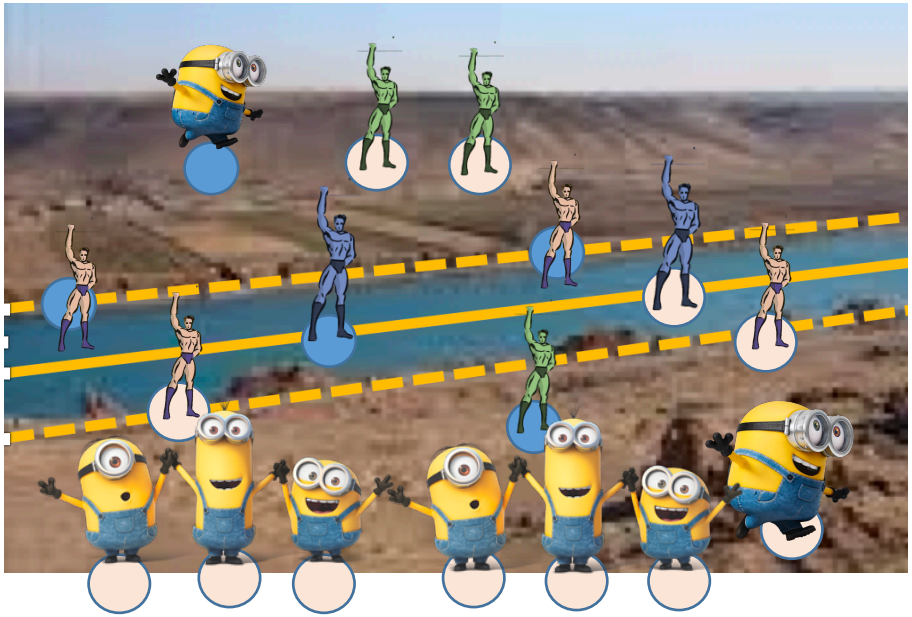
$$s.t. \quad \forall i, \quad y_i (w^T x_i + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$

Equivalently, we can eliminate the slack variables to rewrite this:

$$\min_{w,b,} \frac{1}{2} w^T w + C \sum_i \max(0, 1 - y_i (w^T x_i + b))$$

# Recap: support vector machines

$$\min_{w,b,} \ \frac{1}{2} w^T w + C \sum_i \max(0, 1 - y_i(w^T x_i + b))$$



Match-up quiz

$a)\ y_i(w^T x_i + b) = 1$

$b)\ y_i(w^T x_i + b) < 1$

$c)\ y_i(w^T x_i + b) > 1$

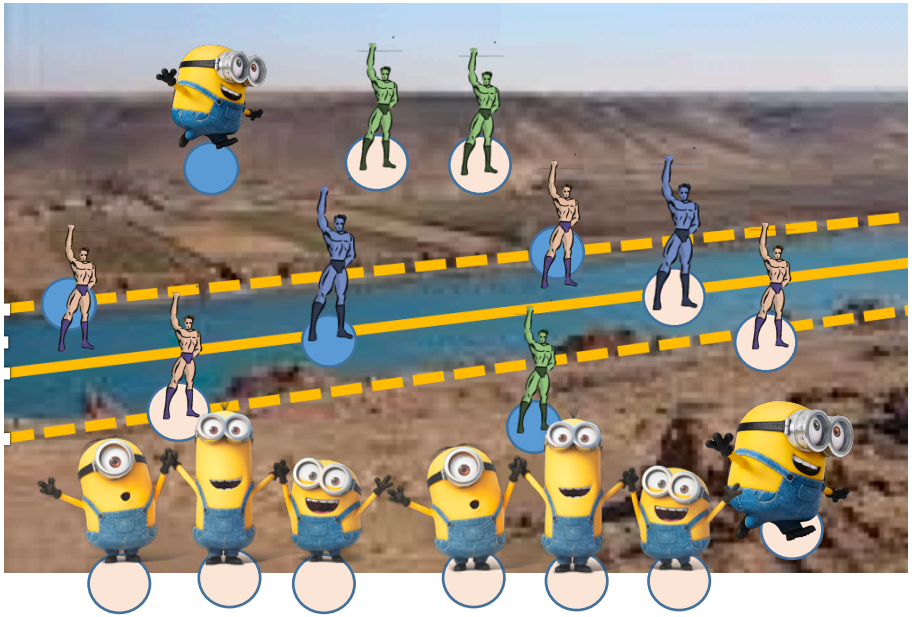
Instances with correct prediction

Instances on the Decision boundary

Instances inside the margin or on the wrong side

# Recap: support vector machines

$$\min_{w,b,} \ \frac{1}{2} w^T w + C \sum_i \max(0, 1 - y_i(w^T x_i + b))$$



## Match-up quiz

$a)\ y_i(w^T x_i + b) = 1$

$b)\ y_i(w^T x_i + b) < 1$

$c)\ y_i(w^T x_i + b) > 1$

$c)\ y_i(w^T x_i + b) > 1$

Instances with correct prediction

$a)\ y_i(w^T x_i + b) = 1$

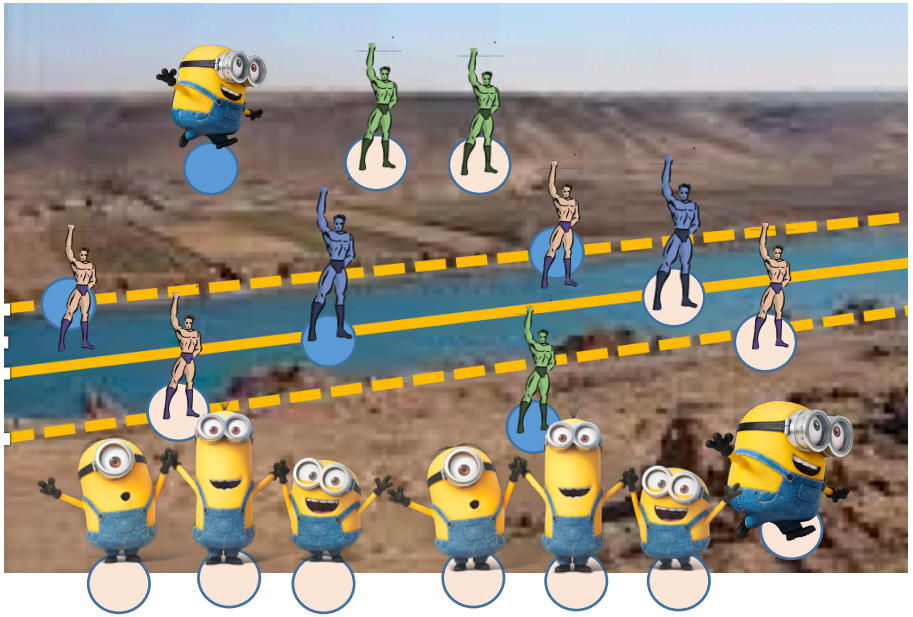
Instances on the Decision boundary

$b)\ y_i(w^T x_i + b) < 1$

Instances inside the margin or on the wrong side

# Recap: support vector machines

$$\min_{w,b,\xi_i} \ \frac{1}{2} w^T w + C \sum_i \xi_i$$



$$s.t. \ \ \forall i, \quad y_i (w^T x_i + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$

## Match-up quiz

$a) \ 0 < \xi < 1$

b) $\xi = 0$

C) $\xi > 1$

Instances with correct prediction

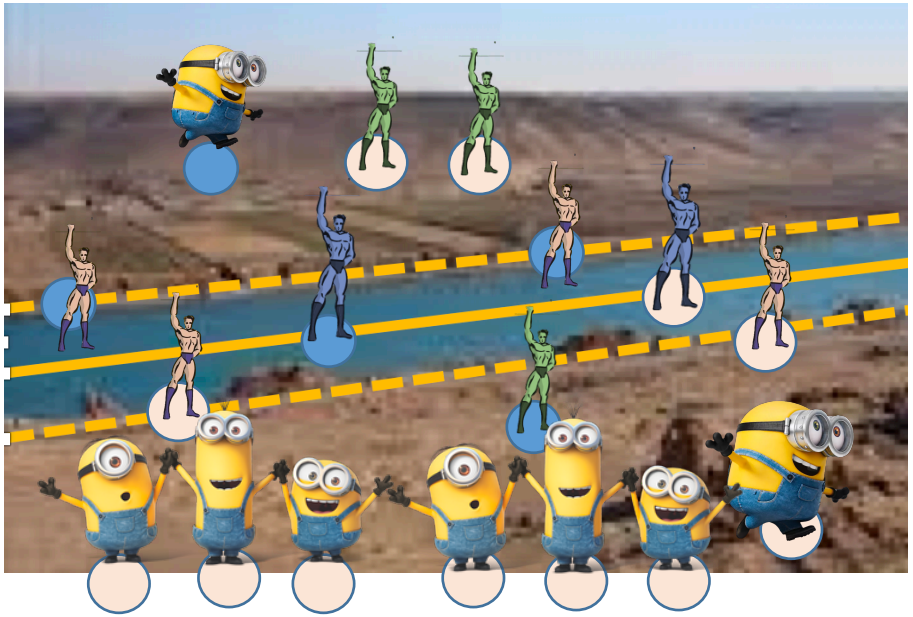
Instances on the Decision boundary

Instances inside the margin or on the wrong side

# Recap: support vector machines

$$\min_{w,b,\xi_i} \ \frac{1}{2} w^T w + C \sum_i \xi_i$$

$$s.t. \ \ \forall i, \quad y_i (w^T x_i + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$



## Match-up quiz

$a) \ 0 < \xi < 1$

b) $\xi = 0$

C) $\xi > 1$

b) $\xi = 0$

Instances with correct prediction

b) $\xi = 0$

Instances on the Decision boundary

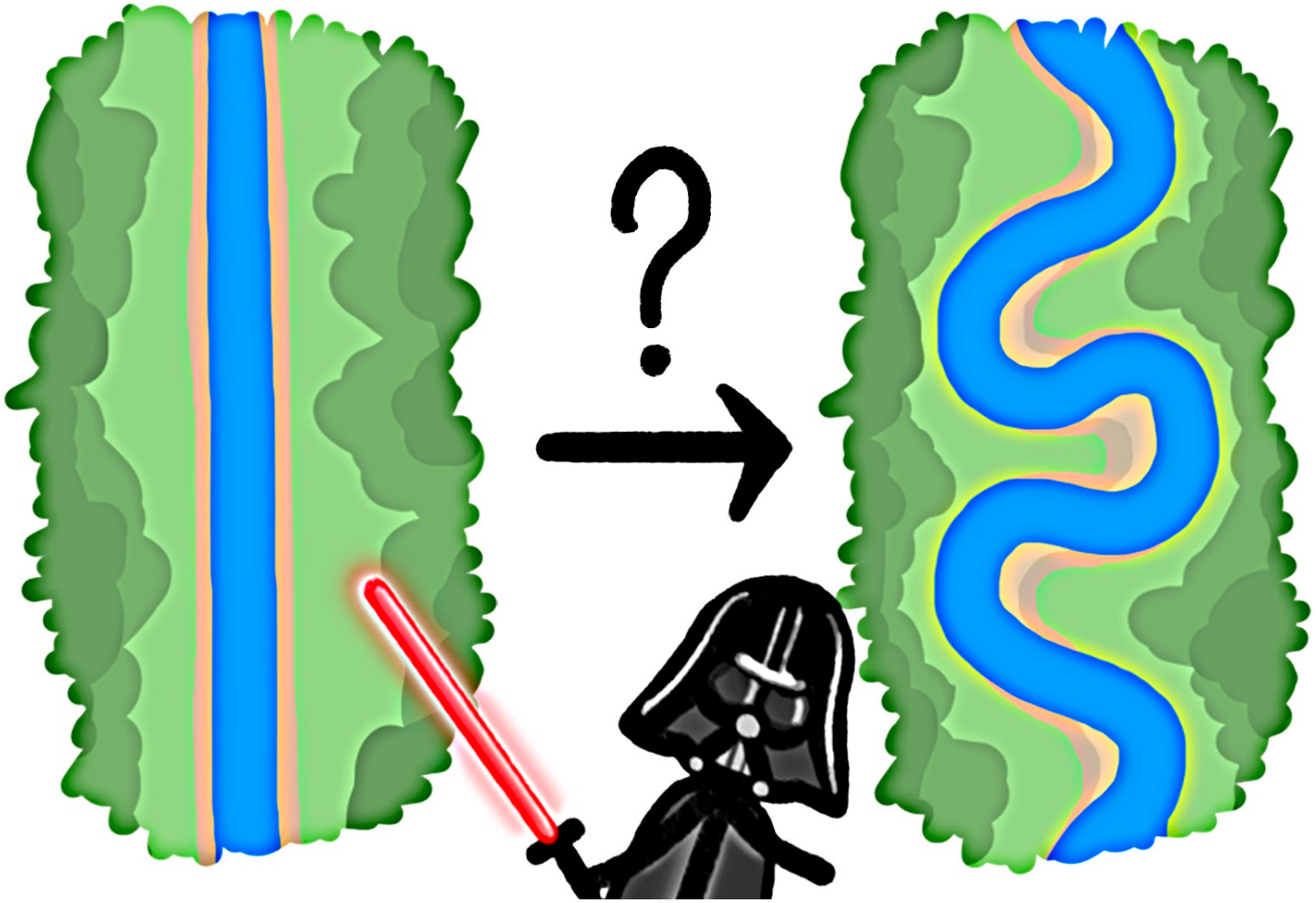
$a) \ 0 < \xi < 1$

Instances inside the margin

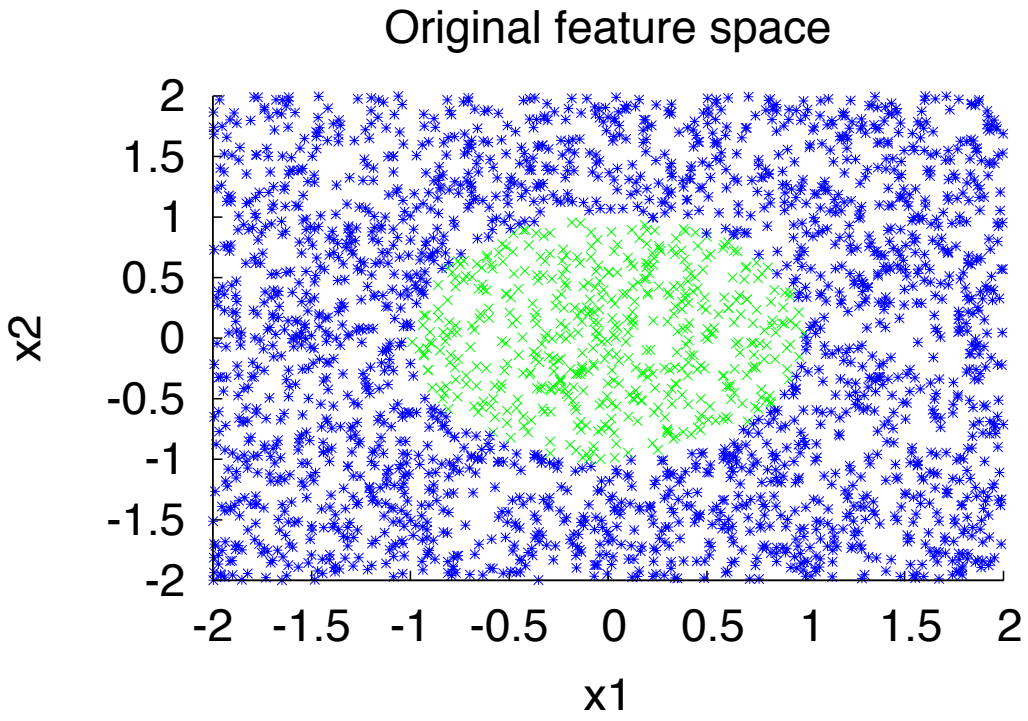
C) $\xi > 1$

Training error

# How about non-linearly separated data?

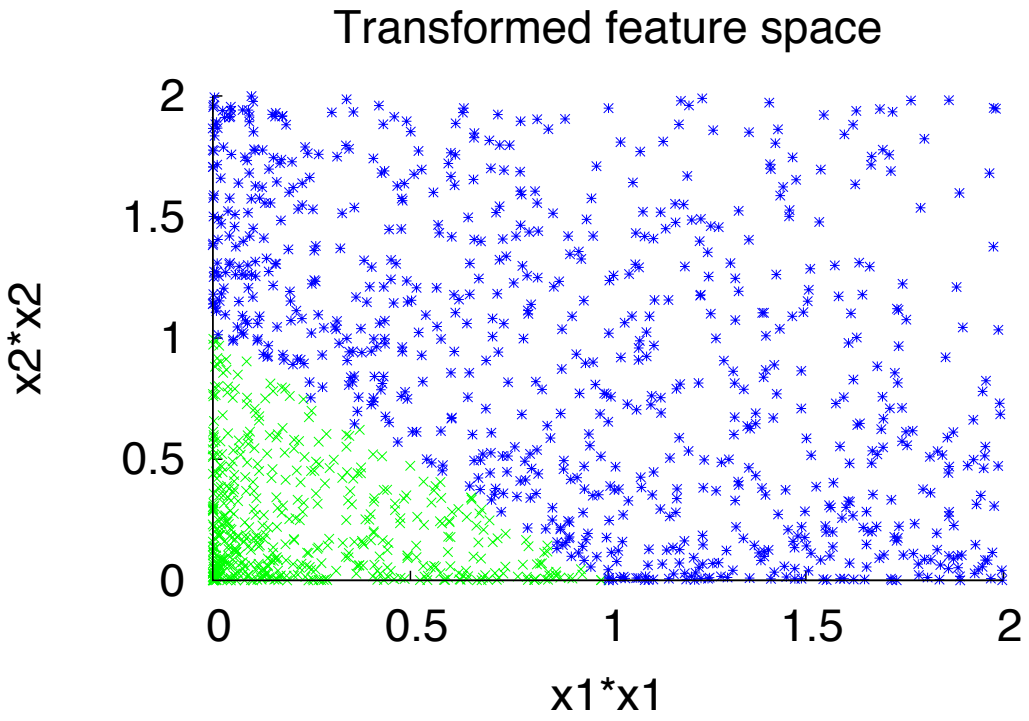


# Recap: Making data linearly separable




$$f(\mathbf{x}) = 1 \text{ iff } x_1^2 + x_2^2 \leq 1$$

# Recap: Making data linearly separable




Transform data: $\mathbf{x} = (x_1, x_2)$ => $\mathbf{x'} = (x_1^2, x_2^2)$

$f(\mathbf{x'}) = 1$ iff $x'_1 + x'_2 \leq 1$

# Can we map data to very high dimensional space?

- Yes – we can map input to an infinite dimensional space
- For example in RBF kernel:

$$\phi(x) = e^{-\gamma x^2}\left[1, \sqrt{\frac{2\gamma}{1!}}x, \sqrt{\frac{(2\gamma)^2}{2!}}x^2, \sqrt{\frac{(2\gamma)^3}{3!}}x^3, \cdots\right]^T$$

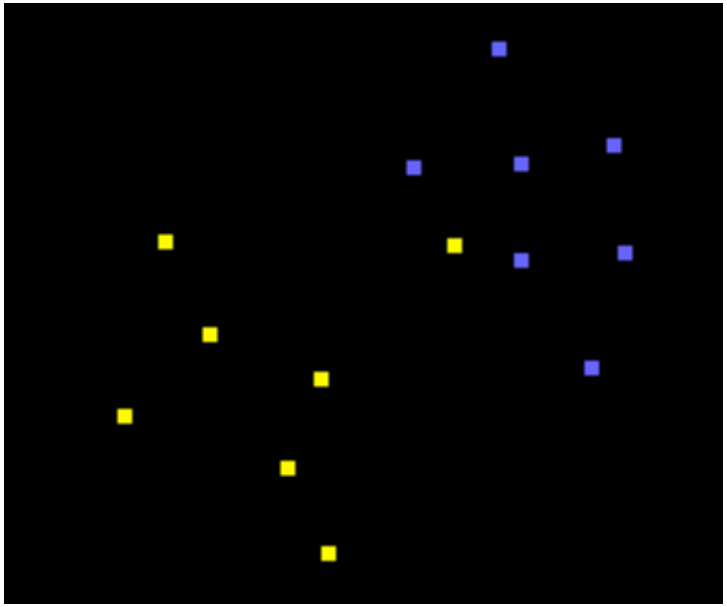
($\gamma$: a hyper-parameter)

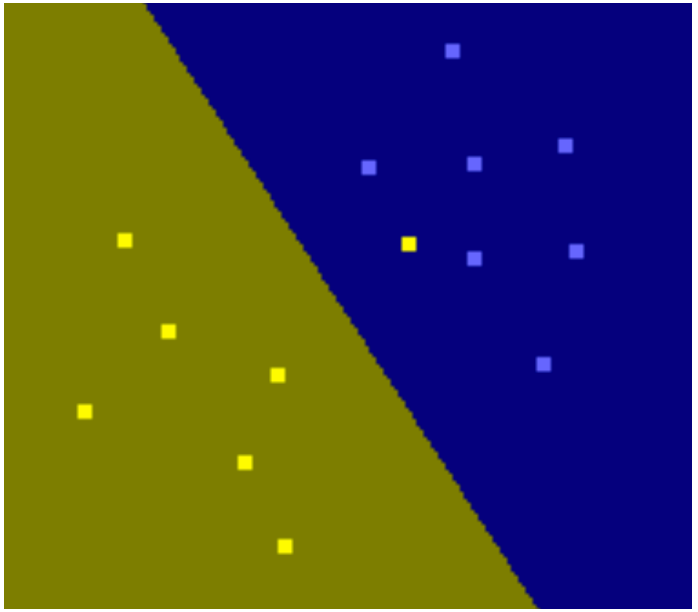
How can we learn a model in an infinite dimensional space?

# DEMO – SVM

- Linear Kernel (C= 1)

$$\min_{w,b,} \ \frac{1}{2} w^T w + C \sum_i \max(0, 1 - y_i(w^T x_i + b))$$



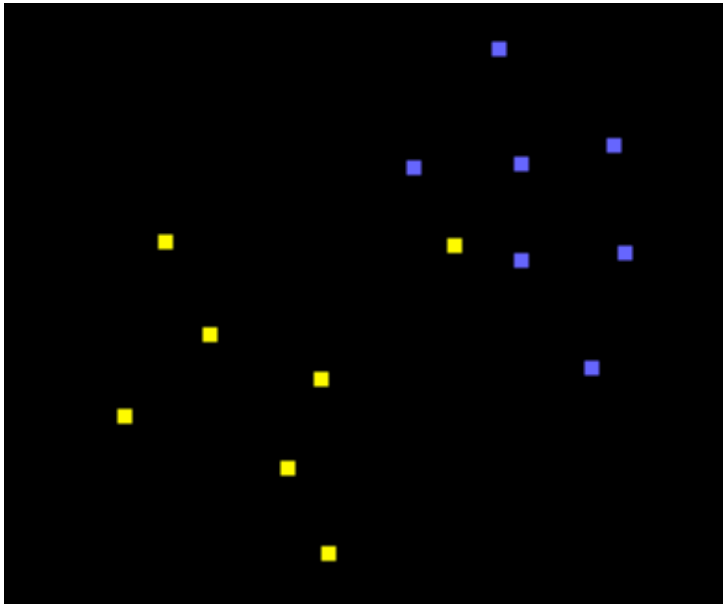


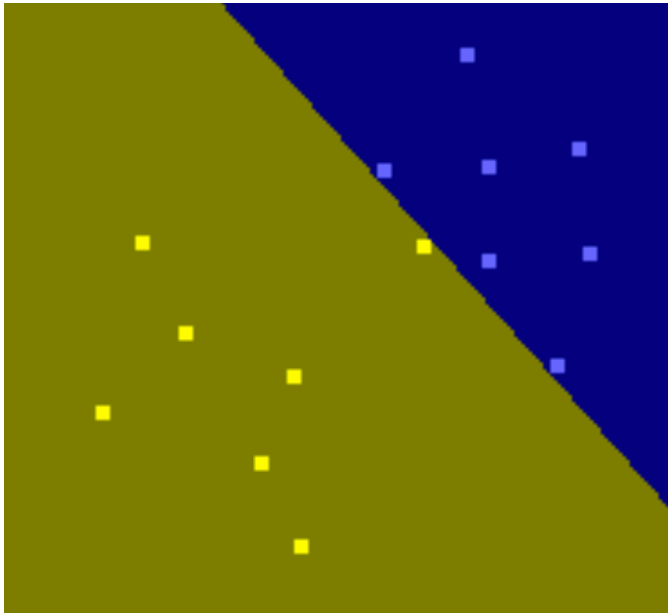
https://www.csie.ntu.edu.tw/~cjlin/libsvm/index.html

# Example – SVM

- Linear Kernel (C= 10000)

$$\min_{w,b,} \ \frac{1}{2} w^T w + C \sum_i \max(0, 1 - y_i(w^T x_i + b))$$

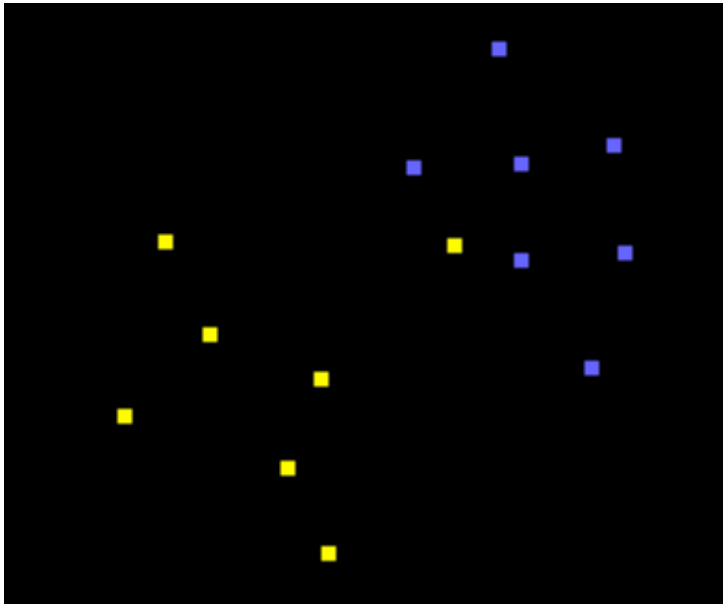


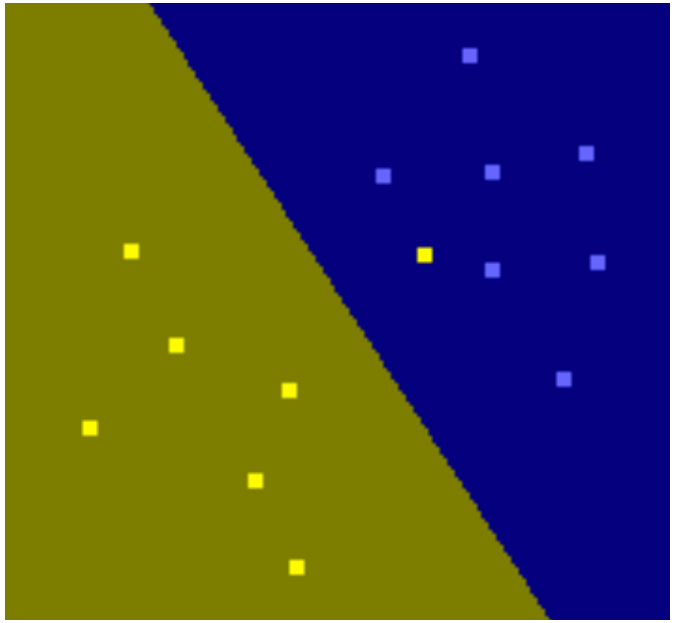


# Example – SVM

- RBF Kernel (C= 1 $\gamma = 0.01$)

$$\min_{w,b,} \ \frac{1}{2} w^T w + C \sum_i \max(0, 1 - y_i(w^T \phi(x_i) + b))$$

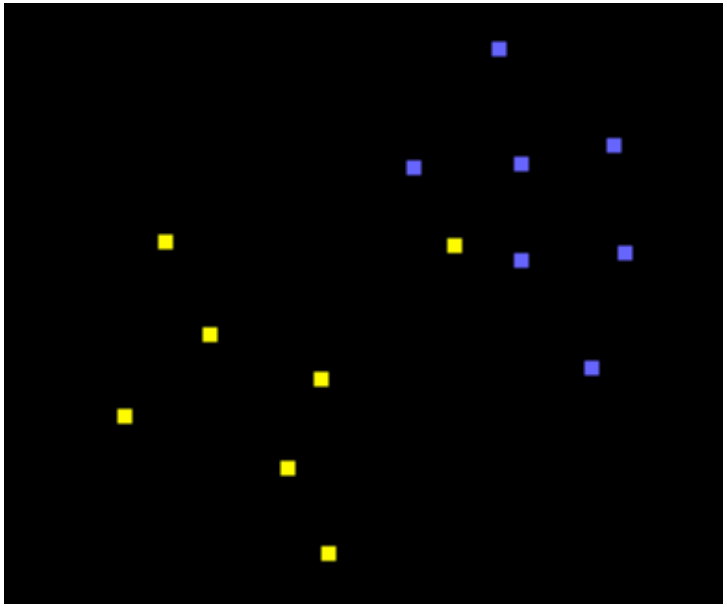


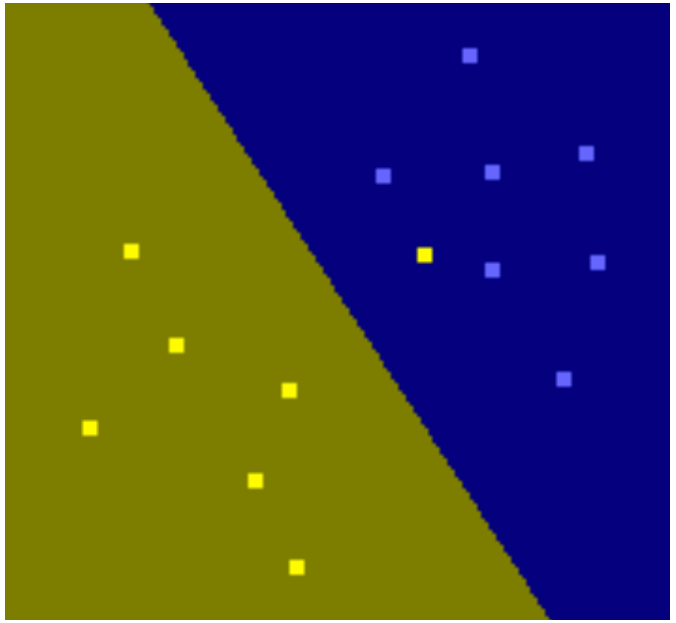


# Example – SVM

- RBF Kernel (C= 1 $\gamma = 0.01$)

$$\min_{w,b,} \ \frac{1}{2} w^T w + C \sum_i \max(0, 1 - y_i(w^T \phi(x_i) + b))$$

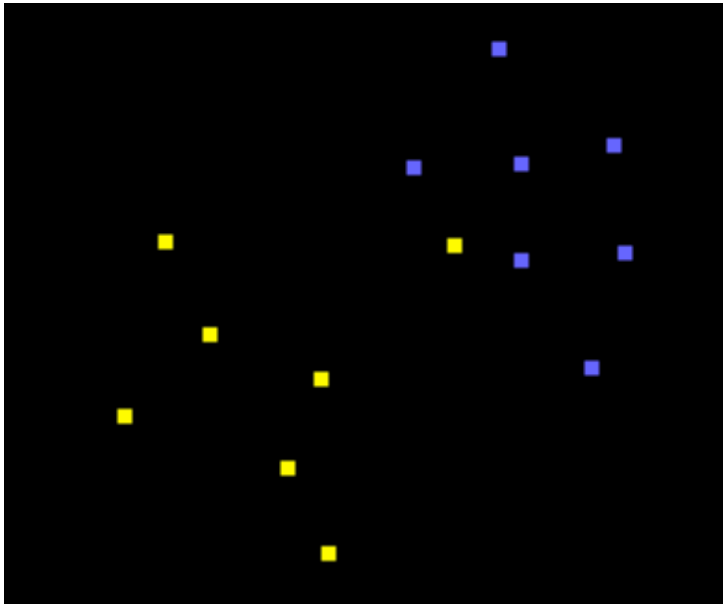


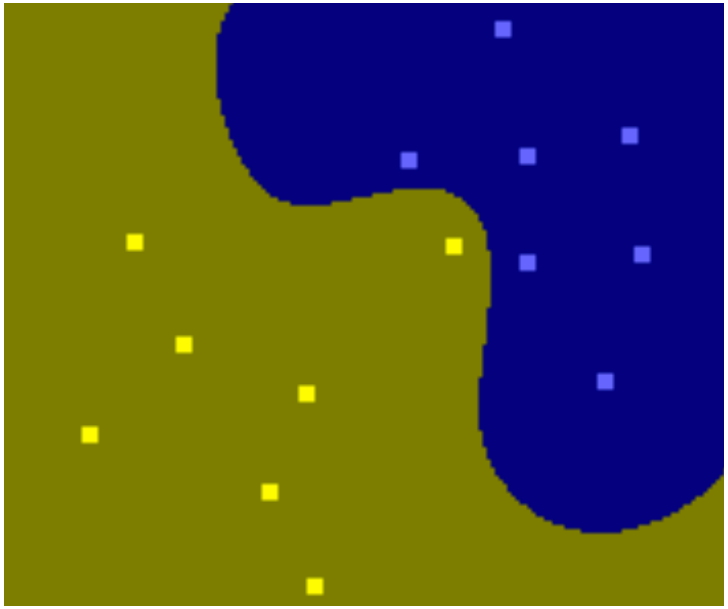


# Example – SVM

- RBF Kernel (C= 10 $\gamma = 100$)

$$\min_{w,b,} \ \frac{1}{2} w^T w + C \sum_i \max(0, 1 - y_i(w^T \phi(x_i) + b))$$

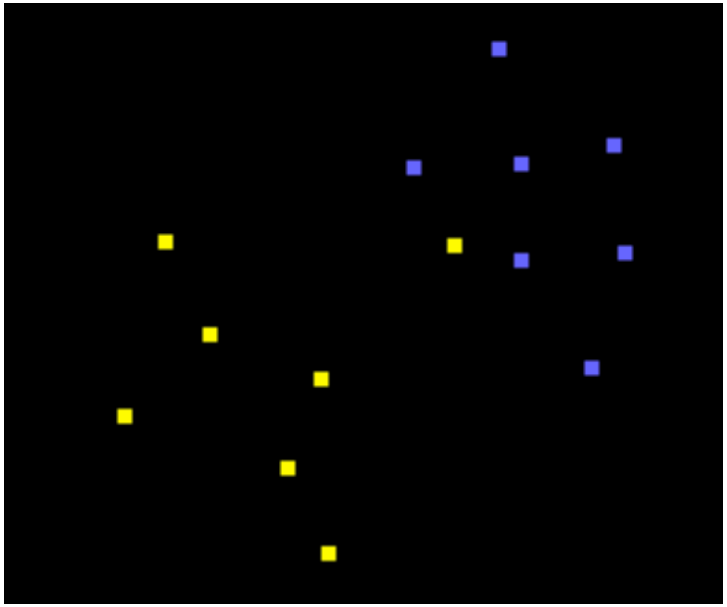


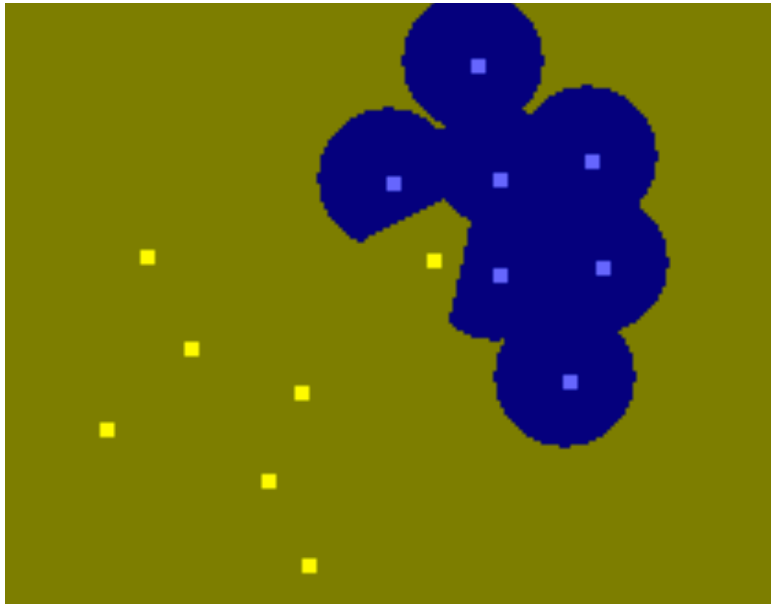


# Example – SVM

- RBF Kernel (C= 10 $\gamma = 10000$)

$$\min_{w,b,} \ \frac{1}{2} w^T w + C \sum_i \max(0, 1 - y_i(w^T \phi(x_i) + b))$$

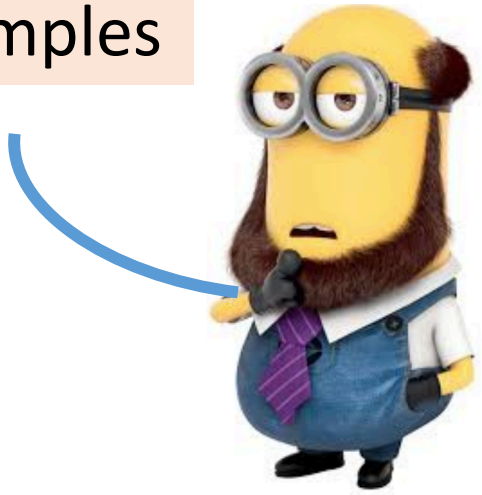




Try it by yourself:
https://www.csie.ntu.edu.tw/~cjlin/libsvm/index.html

How can we learn a model in an infinite dimensional space?

Represent the model by training samples



We don't need to represent $w$ explicitly, but only need a way to compute $w^T x$

# Example: The Perceptron Algorithm
[Rosenblatt 1958]

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}$

1. Initialize $\boldsymbol{w} \leftarrow \mathbf{0} \in \mathbb{R}^n$
2. For $(\boldsymbol{x}, y)$ in $\mathcal{D}$:
3. if $y(\boldsymbol{w}^\top \boldsymbol{x}) \leq \mathbf{0}$
4. $\boldsymbol{w} \leftarrow \boldsymbol{w} + y\boldsymbol{x}$
5. 
6. Return $\boldsymbol{w}$

Assume $y \in \{1, -1\}$

Prediction: $y^{\text{test}} \leftarrow \text{sgn}(\boldsymbol{w}^\top \boldsymbol{x}^{\text{test}})$

# Example: Perceptron Algorithm with Mapping

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}$, $\boldsymbol{\phi}(.)$

1. Initialize $\boldsymbol{w} \leftarrow \mathbf{0} \in \mathbb{R}^n$
2. For $(\boldsymbol{x}, y)$ in $\mathcal{D}$:
3. if $y(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})) \leq \mathbf{0}$
4. $\boldsymbol{w} \leftarrow \boldsymbol{w} + y\,\boldsymbol{\phi}(\boldsymbol{x})$
5. 
6. Return $\boldsymbol{w}$

Assume $y \in \{1, -1\}$

Prediction: $y^{\text{test}} \leftarrow \text{sgn}(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})^{\text{test}})$

# Example: Perceptron Algorithm with Mapping

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}$, $\boldsymbol{\phi}(.)$

1. Initialize $\boldsymbol{w} \leftarrow \mathbf{0} \in \mathbb{R}^n$
2. For $(\boldsymbol{x}, y)$ in $\mathcal{D}$:
3. if $y(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})) \leq \mathbf{0}$
4. $\boldsymbol{w} \leftarrow \boldsymbol{w} + y\,\boldsymbol{\phi}(\boldsymbol{x})$
5. 
6. Return $\boldsymbol{w}$

Assume $y \in \{1, -1\}$

How to update w if the dimensionality of w is infinity?

Prediction: $y^{\text{test}} \leftarrow \text{sgn}(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})^{\text{test}})$

# Recap: Dual Representation

if $y(\boldsymbol{w}^\top \boldsymbol{x}) \leq 0$

$\boldsymbol{w} \leftarrow \boldsymbol{w} + y\boldsymbol{x}$

- Let w be an initial weight vector for perceptron. Let $(x_1,+)$, $(x_2,+)$, $(x_3,-)$, $(x_4,-)$ be examples and assume mistakes are made on $x_1$, $x_2$ and $x_4$.
- What is the resulting weight vector?

  w = w + $x_1$ + $x_2$ - $x_4$

- In general, the weight vector w can be written as a linear combination of examples:

$$w = \sum_{1..m} \alpha_i y_i \, x_i$$

- Where $\alpha_i = 1$ if mistake made on $x_i$, otherwise 0.

# Dual Representation with mapping

if $y(\boldsymbol{w}^{\top}\boldsymbol{\phi}(\boldsymbol{x})) \leq \mathbf{0}$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + y\,\boldsymbol{\phi}(\boldsymbol{x})$$

- Let w be an initial weight vector for perceptron. Let $(x_1,+)$, $(x_2,+)$, $(x_3,-)$, $(x_4,-)$ be examples and assume mistakes are made on $x_1$, $x_2$ and $x_4$.
- What is the resulting weight vector?

  w = w + $\phi(x_1)$ + $\phi(x_2)$ - $\phi(x_4)$

- In general, the weight vector w can be written as a linear combination of examples:

  $$w = \sum_{1..m} \alpha_i \, y_i \phi(x_i)$$

- Where $\alpha_i = 1$ if mistake made on $x_i$, otherwise 0.

# Kernel Perceptron Algorithm

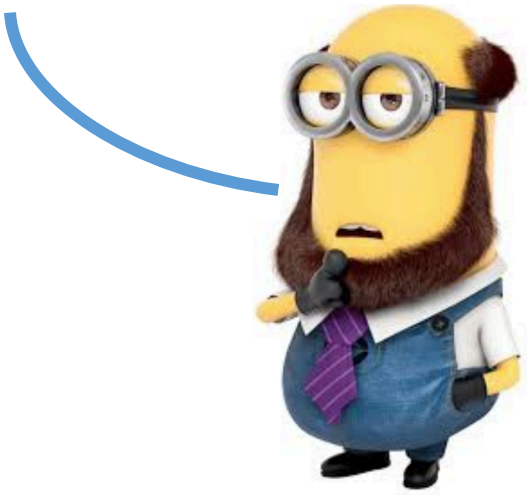
Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}$, $\boldsymbol{\phi}(.)$

1. Initialize $\boldsymbol{w} \leftarrow \mathbf{0} \in \mathbb{R}^n$
2. For $(\boldsymbol{x}, y)$ in $\mathcal{D}$:
3. if $y(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})) \leq 0$
4. $\boldsymbol{w} \leftarrow \boldsymbol{w} + y\,\boldsymbol{\phi}(\boldsymbol{x})$
5. 
6. Return $\boldsymbol{w}$

Prediction: $y^{\text{test}} \leftarrow \text{sgn}(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})^{\text{test}})$

$$w = \sum_{1..l} \alpha_i \, y_i x_i$$

Using $\alpha_i$ instead of w



# Kernel Perceptron Algorithm

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}^{m}{}_{i=1}$, $\boldsymbol{\phi}(.)$

1. Initialize $\boldsymbol{\alpha} \leftarrow \mathbf{0} \in \mathbb{R}^{\boldsymbol{m}}$
2. For $(\boldsymbol{x_j}, y_j)$ in $\mathcal{D}$: (j= $1 \ldots m$)
3. if $y_j \left( (\sum_{1..m} \alpha_i \, y_i \phi(x_i))^T \phi(\boldsymbol{x_j}) \right) \leq \mathbf{0}$
4. $\alpha_j \leftarrow \alpha_j + 1$
5. Return $\boldsymbol{\alpha}$
6.

$$w = \sum_{1..m} \alpha_i \, y_i \phi(x_i)$$

Prediction: $y^{\text{test}} \leftarrow \text{sgn}((\sum_{1..m} \alpha_i \, y_i \phi(x_i))^{\top} \boldsymbol{\phi}(\boldsymbol{x})^{\text{test}})$

# Kernel Perceptron Algorithm

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}_{i=1}^{m}$, $\boldsymbol{\phi(.)}$

1. Initialize $\boldsymbol{\alpha} \leftarrow \mathbf{0} \in \mathbb{R}^{\boldsymbol{m}}$
2. For $(\boldsymbol{x_j}, y_j)$ in $\mathcal{D}$: $(\mathrm{j} = 1 \ldots m)$
3. if $y_j \left( \left( \sum_{1..m} \alpha_i \, y_i \phi(x_i) \right)^T \phi(\boldsymbol{x_j}) \right) \leq \mathbf{0}$
4. $\alpha_j \leftarrow \alpha_j + 1$
5. Return $\boldsymbol{\alpha}$
6.

$$w = \sum_{1..m} \alpha_i \, y_i \phi(x_i)$$

Let $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$

Prediction: $y^{\text{test}} \leftarrow \text{sgn}\left( \left( \sum_{1..m} \alpha_i \, y_i \phi(x_i) \right)^{\mathsf{T}} \boldsymbol{\phi(x)}^{\text{test}} \right)$

# Kernel Perceptron Algorithm

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}^{m}_{i=1}$, $\boldsymbol{\phi(.)}$

1. Initialize $\boldsymbol{\alpha} \leftarrow \mathbf{0} \in \mathbb{R}^{\boldsymbol{m}}$
2. For $(\boldsymbol{x_j}, y_j)$ in $\mathcal{D}$: $(\mathrm{j} = 1 \ldots m)$
3. if $y_j\left(\sum_{1..m} \alpha_i \, y_i K(x_i, x_j)\right) \leq \mathbf{0}$
4. $\alpha_j \leftarrow \alpha_j + 1$
5. Return $\boldsymbol{\alpha}$
6.

$$w = \sum_{1..m} \alpha_i \, y_i \phi(x_i)$$

Let $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$

Prediction: $y^{\text{test}} \leftarrow \text{sgn}\left(\sum_{1..m} \alpha_i \, y_i K(x_i, x^{Test})\right)$

# Kernel Perceptron Algorithm

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}_{i=1}^{m}$, $\boldsymbol{\phi(.)}$

1. Initialize $\boldsymbol{\alpha} \leftarrow \mathbf{0} \in \mathbb{R}^{\boldsymbol{m}}$
2. For $(\boldsymbol{x_j}, y_j)$ in $\mathcal{D}$: $(\mathrm{j} = 1 \ldots m)$
3. if $y_j\left(\sum_{1..m} \alpha_i \, y_i K(x_i, x_j)\right) \leq \mathbf{0}$
4. $\alpha_j \leftarrow \alpha_j + 1$
5. Return $\boldsymbol{\alpha}$
6.

Even if the dimentionality of $\phi(x)$ is infinite,
If we can compute $\boldsymbol{K(x_i, x_j) = \phi(x_i)^T \phi(x_j)}$
we can learn the model and make predictions


Prediction: $y^{\text{test}} \leftarrow \text{sgn}(\sum_{1..m} \alpha_i \, y_i K(x_i, x^{Test}))$

# Kernel SVM

- Can we derive dual representation of SVM?

$$\min_{w,b,\xi_i} \ \frac{1}{2} w^T w + C \sum_i . \ \xi_i$$

$$s.t. \ \ \forall i, \quad y_i (w^T \phi(x_i) + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$

$$w = \sum_{1..m} \alpha_i \, y_i \phi(x_i)$$

# Dual SVM problem

- $w$ may be infinite variables
- We can derive the Lagrangian dual problem

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{e}^T \boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \le \alpha_i \le C, i = 1, \ldots, l \\ & \mathbf{y}^T \boldsymbol{\alpha} = 0, \end{aligned}$$

where $Q_{ij} = y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ and $\mathbf{e} = [1, \ldots, 1]^T$

https://www.csie.ntu.edu.tw/~cjlin/talks/MLSS.pdf

# Decision function in Kernel SVM

- In the optimum, the solutions of the primal and dual problem have the following relation:

$$w^* = \sum_{1..m} \alpha_i^* \, y_i \phi(x_i)$$

- The original decision function: $w^T \phi(x) + b$
- Can be rewrite as:

$$\sum_{i=1..m} \alpha_i \, y_i K(x_i, x^{Test})$$

# Support vectors

$$\sum_{i=1..m} \alpha_i \, y_i K(x_i, x^{Test})$$

- If $\alpha_i = 0 \Rightarrow$ the training sample doesn't affect the prediction

- $\alpha_i > 0 \Rightarrow$ support vectors

# Kernel tricks

Example: $\mathbf{x}_i \in R^3, \phi(\mathbf{x}_i) \in R^{10}$

$$\phi(\mathbf{x}_i) = [1, \sqrt{2}(x_i)_1, \sqrt{2}(x_i)_2, \sqrt{2}(x_i)_3, (x_i)_1^2,$$
$$(x_i)_2^2, (x_i)_3^2, \sqrt{2}(x_i)_1(x_i)_2, \sqrt{2}(x_i)_1(x_i)_3, \sqrt{2}(x_i)_2(x_i)_3]^T$$

Then $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$.

# Common choice of Kernels

$$e^{-\gamma\|\mathbf{x}_i-\mathbf{x}_j\|^2}, \text{ (Radial Basis Function)}$$

$$(\mathbf{x}_i^T\mathbf{x}_j/a+b)^d \text{ (Polynomial kernel)}$$

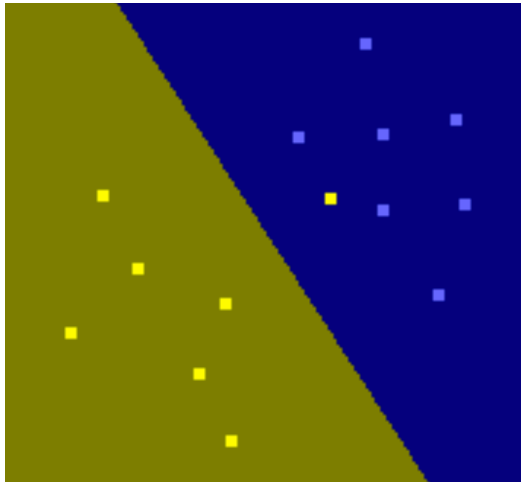
- RBF kernel implicitly map data into a feature space with infinity dimensions (by Taylor explanation)

$$\phi(x) = e^{-\gamma x^2}\left[1, \sqrt{\frac{2\gamma}{1!}}x, \sqrt{\frac{(2\gamma)^2}{2!}}x^2, \sqrt{\frac{(2\gamma)^3}{3!}}x^3, \cdots\right]^T$$

# RBF Kernel

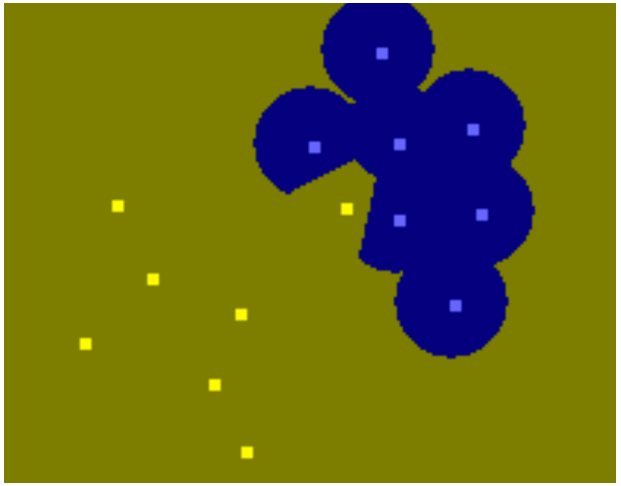
$$e^{-\gamma\|\mathbf{x}_i-\mathbf{x}_j\|^2}$$

Try it out:
https://www.csie.ntu.edu.tw/~cjlin/libsvm/index.html



(C= 1 $\gamma = 0.01$)

(C= 10 $\gamma = 100$)



(C= 10 $\gamma = 10000$)

# What you need to know about SVMs

- What are support vectors
- SVM objectives
  (constraint and unconstraint version)
- Kernel trick, Kernel Perceptron, and Kernel SVM

# Boosting and Ensembles

Yet another way to introduce non-linearity in linear models

# Two heads are better than one

- Is it true in ML?
  - Can we get a stronger learner by combining many weak learners?

# Boosting and Ensembles

- Ensemble methods

- BoostingAdaBoost

# Boosting and Bagging

- A general learning approach for constructing a *strong learner*, given a collection of (possibly infinite) weak learners

- Historically: An answer to a theoretical question in PAC learning

The Strength of Weak Learnability

ROBERT E. SCHAPIRE

1989-90

# Why it may work?

- The predictors make different types of mistakes
- Combine them may make a stronger predictor

# Practical story: the Netflix prize

- Goal: predict how a user will rate a movie
  - Based on other user's ratings of the movie.
  - Based on this user's ratings of other movies.
  - Application called collaborative filtering.
  - Netflix prize: 1M prize for the first team to do 10% better than the Netflix system.
- Winner: an ensemble of more than 800 rating systems.

# Bagging

Short for ***B**ootstrap **agg**regating* [Breiman, 1994]

- Given a training set with m examples
- Repeat t = 1, 2, ⋯, m:
  - Draw $m'$ (< m) samples with replacement from the training set
  - Train a classifier (any classifier you like) $h_i$
- Construct final classifier by taking votes from each $h_i$
- Any other ideas beyond simple voting?

# Bagging

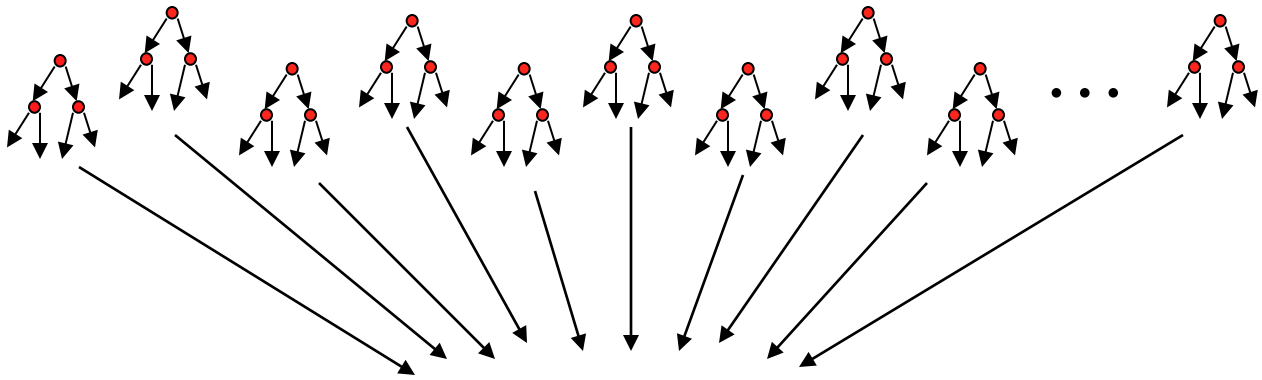
Short for ***B**ootstrap **agg**regat**ing***

- A method for generating multiple versions of a predictor and using these to get an aggregated predictor.
  - Averages over the versions when predicting a numerical outcome (regression)
  - Does a plurality vote when predicting a class (classification)

- Instability of the prediction method: If perturbing the training set can cause significant changes in the learned classifier *then* bagging can improve accuracy

# Example: Bagged Decision Trees

- Draw T bootstrap samples of data
- Train trees on each sample ! T trees
- Average prediction of trees on out-of-bag samples



Average prediction

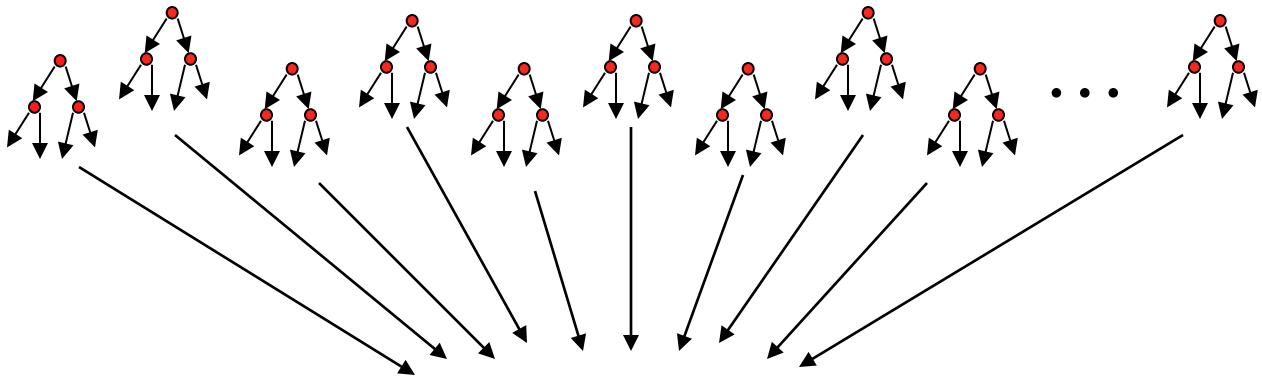
(0.23 + 0.19 + 0.34 + 0.22 + 0.26 + … + 0.31) / # Trees = 0.24

Vote prediction

Major(T, F, T, T, F, T, T, T) = True

# Random Forests (Bagged Trees++)

- Draw T (possibly ***1000s***) bootstrap samples of data
- ***Draw sample of available attributes at each split***
- Train trees on each sample/attribute set ! T trees
- Average prediction of trees on out-of-bag samples



Average prediction

(0.23 + 0.19 + 0.34 + 0.22 + 0.26 + … + 0.31) / # Trees = 0.24

Vote prediction

Major(T, F, T, T, F, T, T, T) = True

# Boosting and Ensembles

- Ensemble methods
- Boosting AdaBoost

# Practical Example: How may I help you?

**Goal**: Automatically categorize type of phone call requested by a phone customer

- "yes I'd like to place a collect call please" ! `Collect`
- "Operator I need to make a call but need to bill it to my office" ! `ThirdNumber`
- "I'd like to place my call on my master card please" ! `CallingCard`

**Important observation**

"Rules of thumb" are *often* correct

Eg: If *card* occurs in the utterance, then predit `CallingCard`

But hard to find a single prediction rule

# One boosting approach

- Select a set of examples
- Derive a rough rule of thumb
- Example a second set of examples
- Derive a second rule of thumb
- Repeat T times…
- Combine rules of thumb into a single prediction rule

# One boosting approach

- Select a small subset of examples
- Derive a rough rule of thumb
- Example a second set of examples
- Derive a second rule of thumb
- Repeat T times…
- Combine rules of thumb into a single prediction rule

Need to specify:
How to select these subsets?

Need to specify:
How to combine these rules of thumb?

rule of thumb – weak classifier

# One boosting approach

- Select a small subset of examples
- Derive a rough rule of thumb
- Example a second set of examples
- Derive a second rule of thumb
- Repeat T times…
- Combine rules of thumb into a single prediction rule

Boosting: A general method for converting rough rules of thumb into accurate prediction classifiers

# Advanced topic: (not covered in the exam) Boosting: Theoretical Motivation

- Strong PAC algorithm
  - For any distribution over examples,
  - For any $\delta > 0$, $\epsilon > 0$,
  - Given a polynomially many random examples
  - Finds a hypothesis with error $\epsilon$ with probability, $1 - \delta$

- Weak PAC algorithm
  - Same, but only for $\epsilon > \frac{1}{2}$
- Question [Kearns and Valiant '88]:
  - Does weak learnability imply strong learnability?

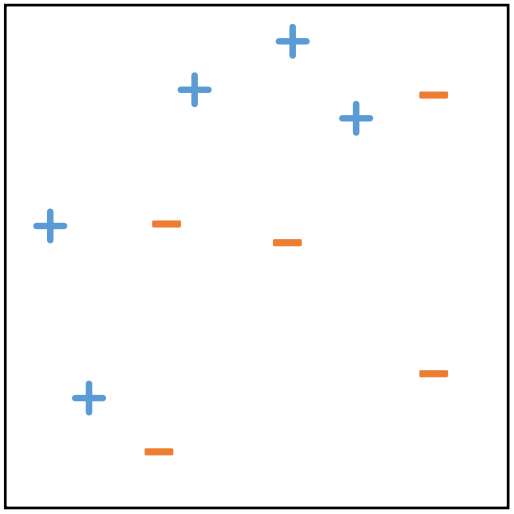
# History: Early boosting algorithms

- [Schapire '89]
  - First provable boosting algorithm
  - Call weak learner three times on three modified distributions
  - Get slight boost in accuracy
  - Apply recursively
- [Freund '90]
  - "Optimal" algorithm that "boosts by majority"
- [Drucker, Schapire & Simard '92]
  - First experiments using boosting
  - Limited by practical drawbacks
- [Freund & Schapire '95]
  - Introduced *AdaBoost* algorithm
  - Strong practical advantages over previous boosting algorithms
- AdaBoost was followed by a huge number of papers and practical applications
  - And a Gödel prize for Freund and Schapire

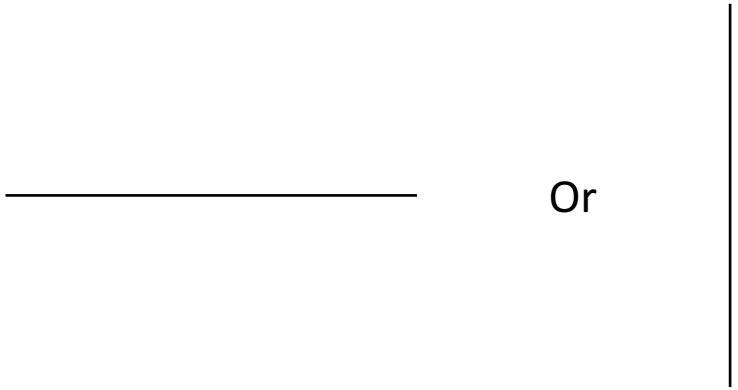
# Boosting and Ensembles

- Ensemble methods
- AdaBoost
  - Intuition
  - The algorithm
  - Why does it work

# A toy example

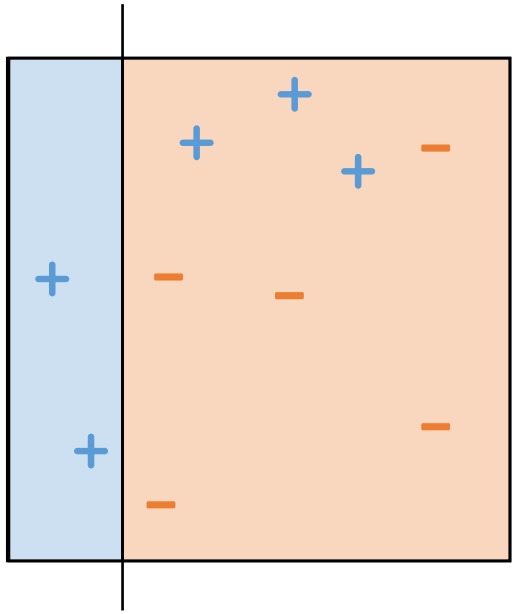


Our weak learner: An axis parallel line




Initially all examples are equally important

# A toy example



Our weak learner: An axis parallel line



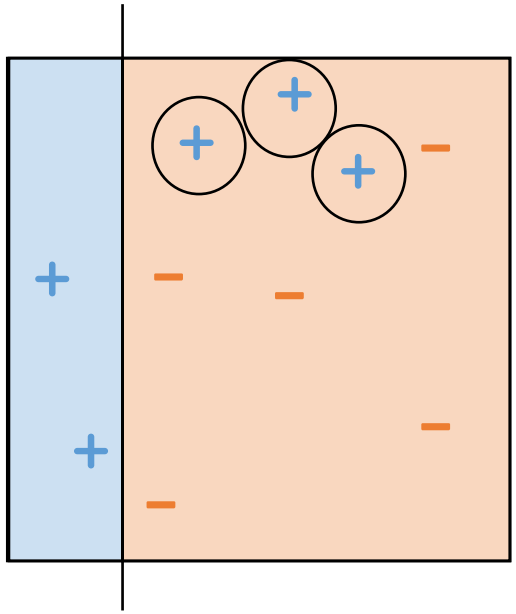
Or



Initially all examples are equally important

$h_1$ = The best classifier on this data

# A toy example



Our weak learner: An axis parallel line



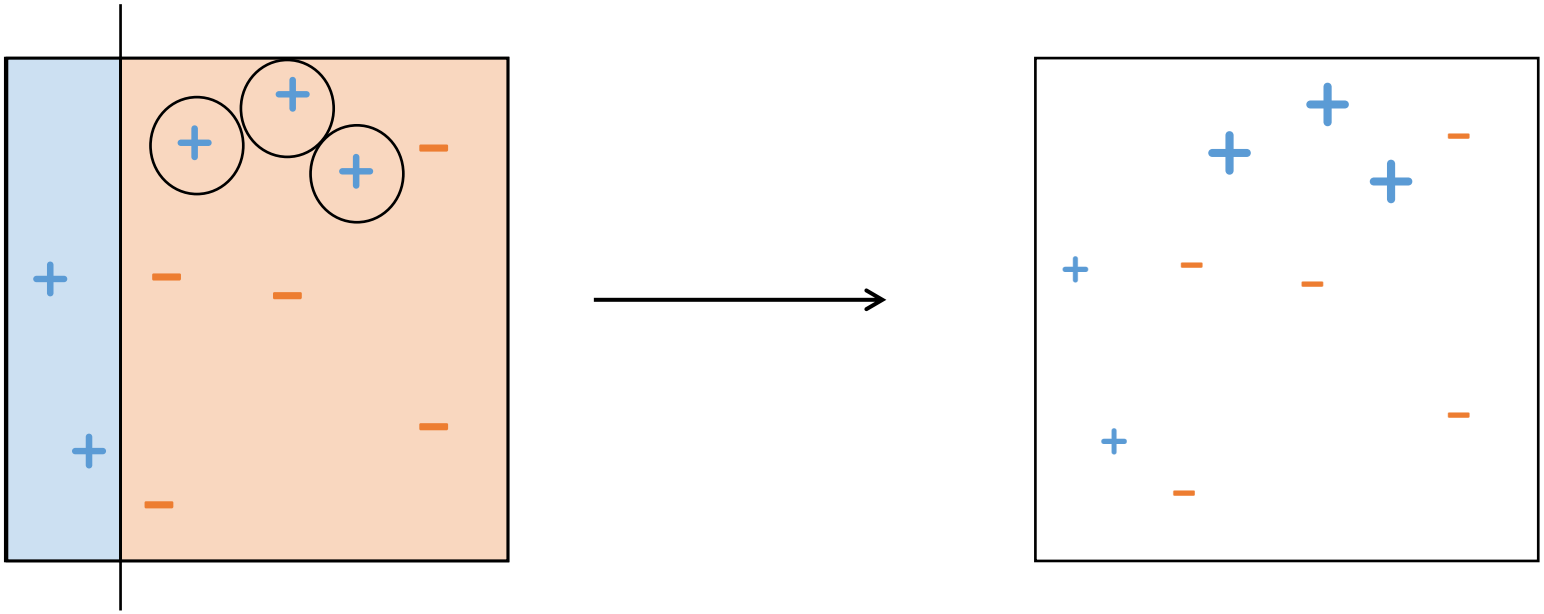
Or



Initially all examples are equally important

$h_1$ = The best classifier on this data
Clearly there are mistakes. Error $\epsilon_1$ = 0.3

# A toy example

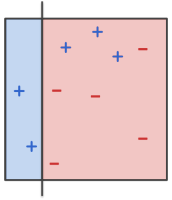


Initially all examples are equally important

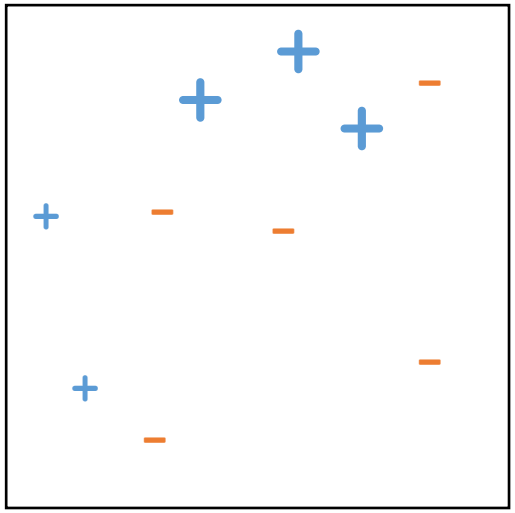
$h_1$ = The best classifier on this data
Clearly there are mistakes. Error $\epsilon_2$ = 0.3

For the next round, increase the importance of the examples with mistakes and down-weight the examples that $h_1$ got correctly

# A toy example

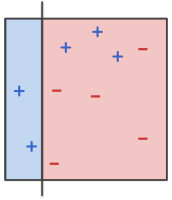


$h_1$

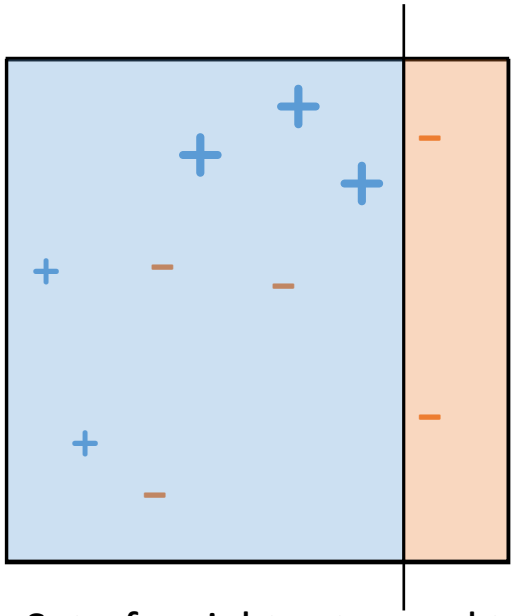


$D_t$ = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

# A toy example

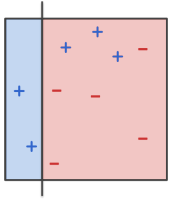


$h_1$

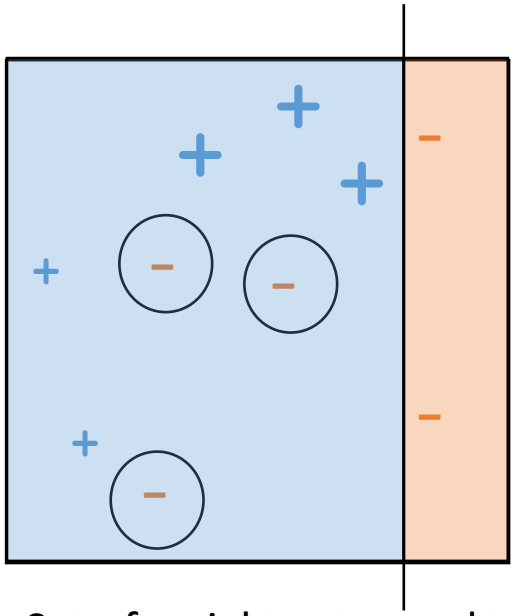


$D_t$ = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

# A toy example



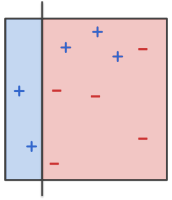
$h_1$



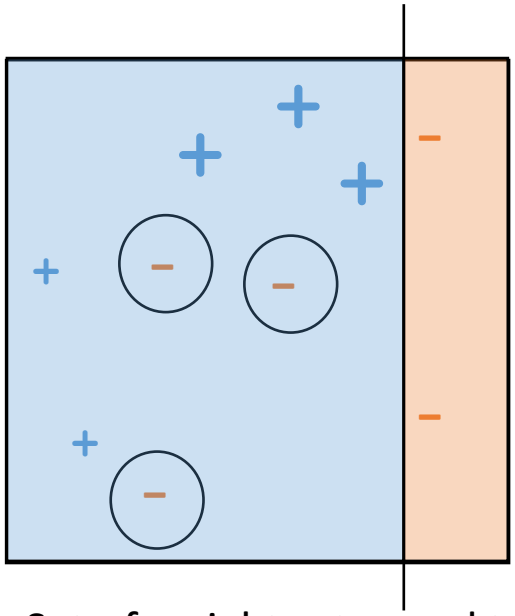
$D_t$ = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

$h_2$ = A classifier learned on this data. *Has an error* $\epsilon_2 = 0.21$

# A toy example



$h_1$

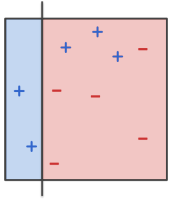


$D_t$ = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

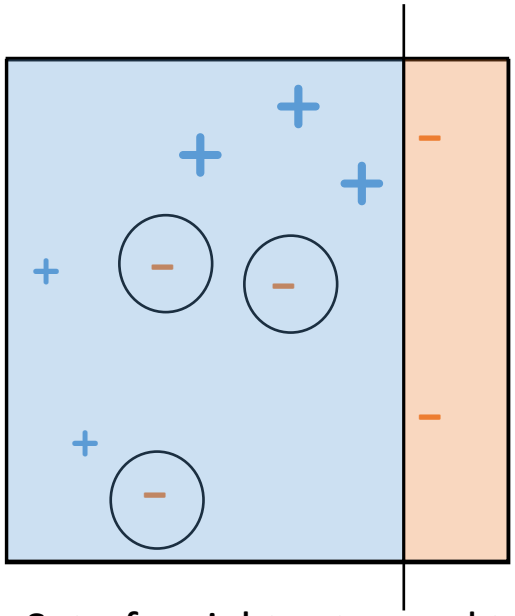
$h_2$ = A classifier learned on this data. *Has an error* $\epsilon_2 = 0.21$

Why not 0.3? Because while computing error, we will weight each example $x_i$ by its $D_t(i)$

# A toy example



$h_1$



$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{m} D_t(i)\ y_i h(x_i)\right)$$

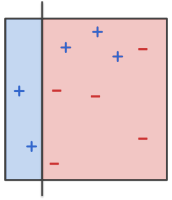
Why is this a reasonable definition?

$D_t$ = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

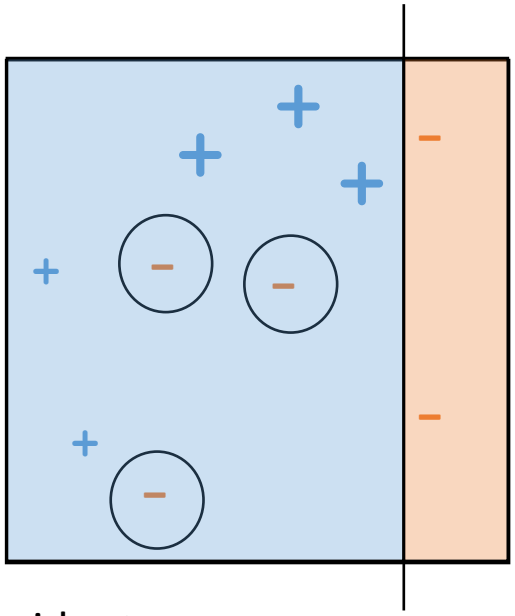
$h_2$ = A classifier learned on this data. *Has an error* $\epsilon_2 = 0.21$

Why not 0.3? Because while computing error, we will weight each example $x_i$ by its $D_t(i)$

# A toy example



$h_1$



$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{m} D_t(i)\ y_i h(x_i)\right)$$

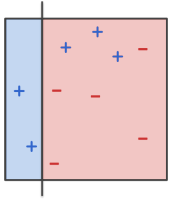
Why is this a reasonable definition?

Consider two cases

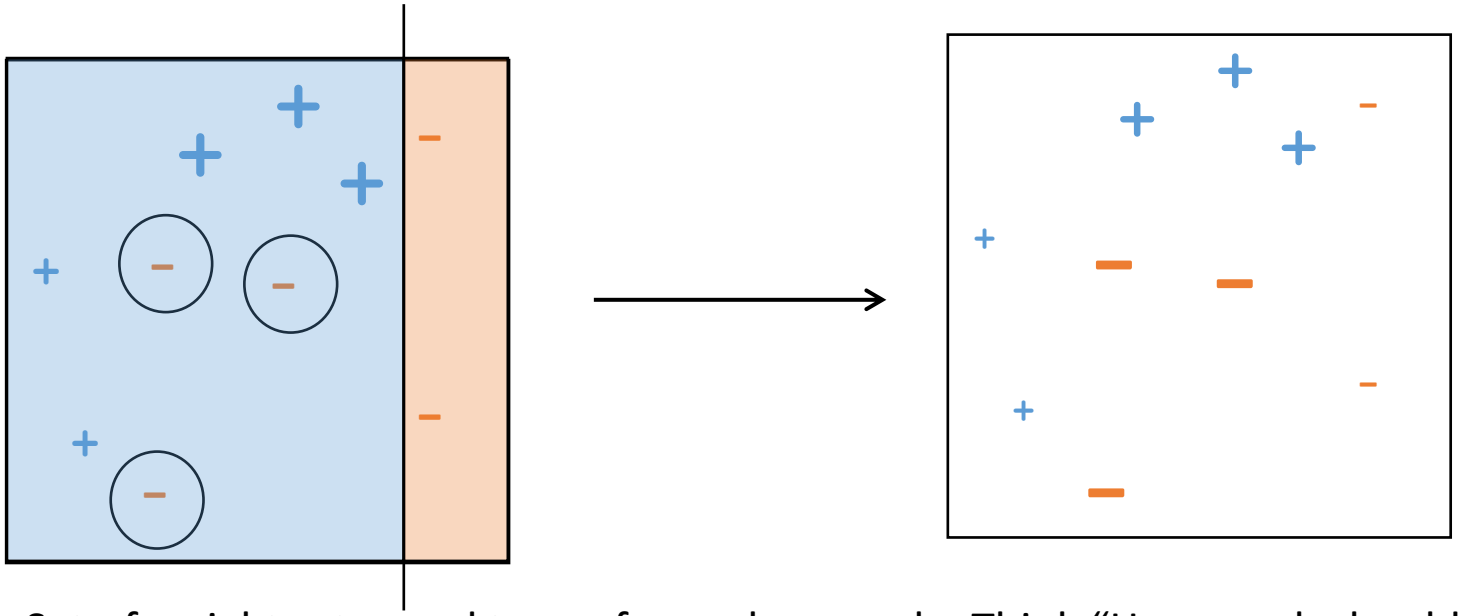
Case 1: When $y = +1$, $h(x) = -1$ OR $y = -1$, $h(x) = +1$

Case 2: When $y = h(x)$

# A toy example



$h_1$

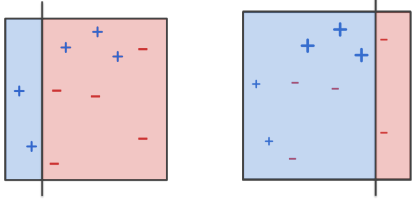


$D_t$ = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

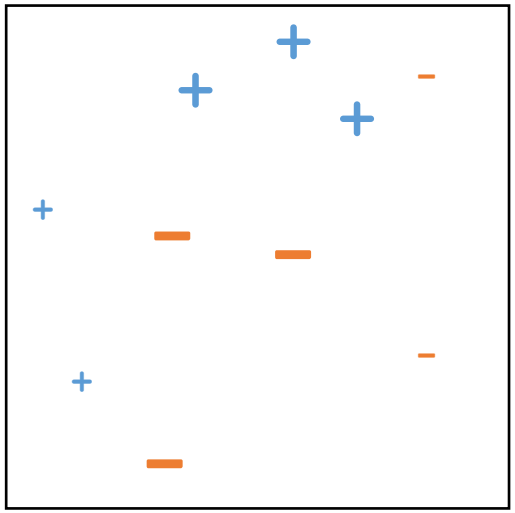
$h_2$ = A classifier learned on this data. *Has an error* $\epsilon_2 = 0.21$

For the next round, increase the importance of the mistakes and down-weight the examples that $h_2$ got correctly

# A toy example

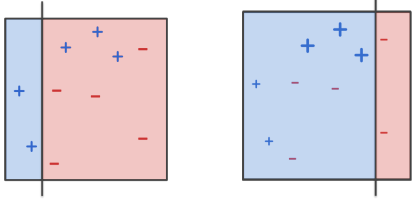


$h_1$ $h_2$

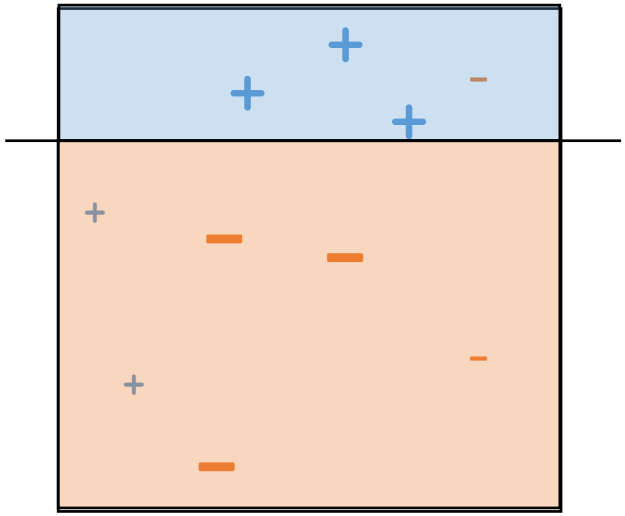


$D_t$ = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

# A toy example



$h_1$ $h_2$

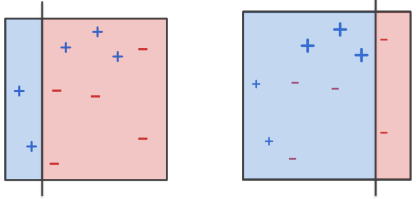


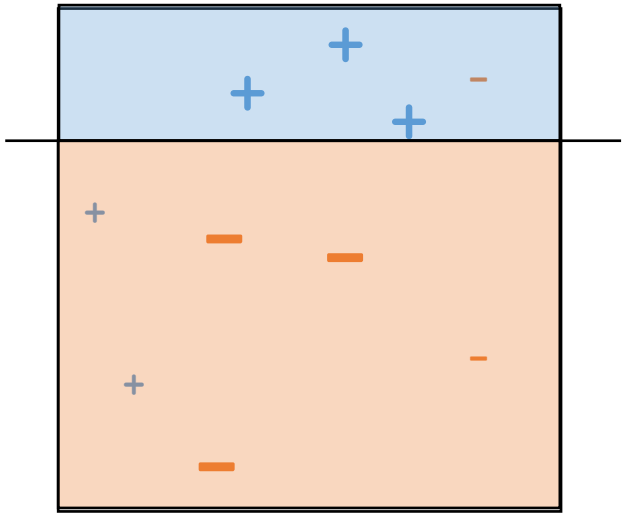
$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{m} D_t(i)\ y_i h(x_i)\right)$$

$D_t$ = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

$h_3$ = A classifier learned on this data. *Has an error* $\epsilon_3 = 0.14$

# A toy example



$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{m} D_t(i)\ y_i h(x_i)\right)$$

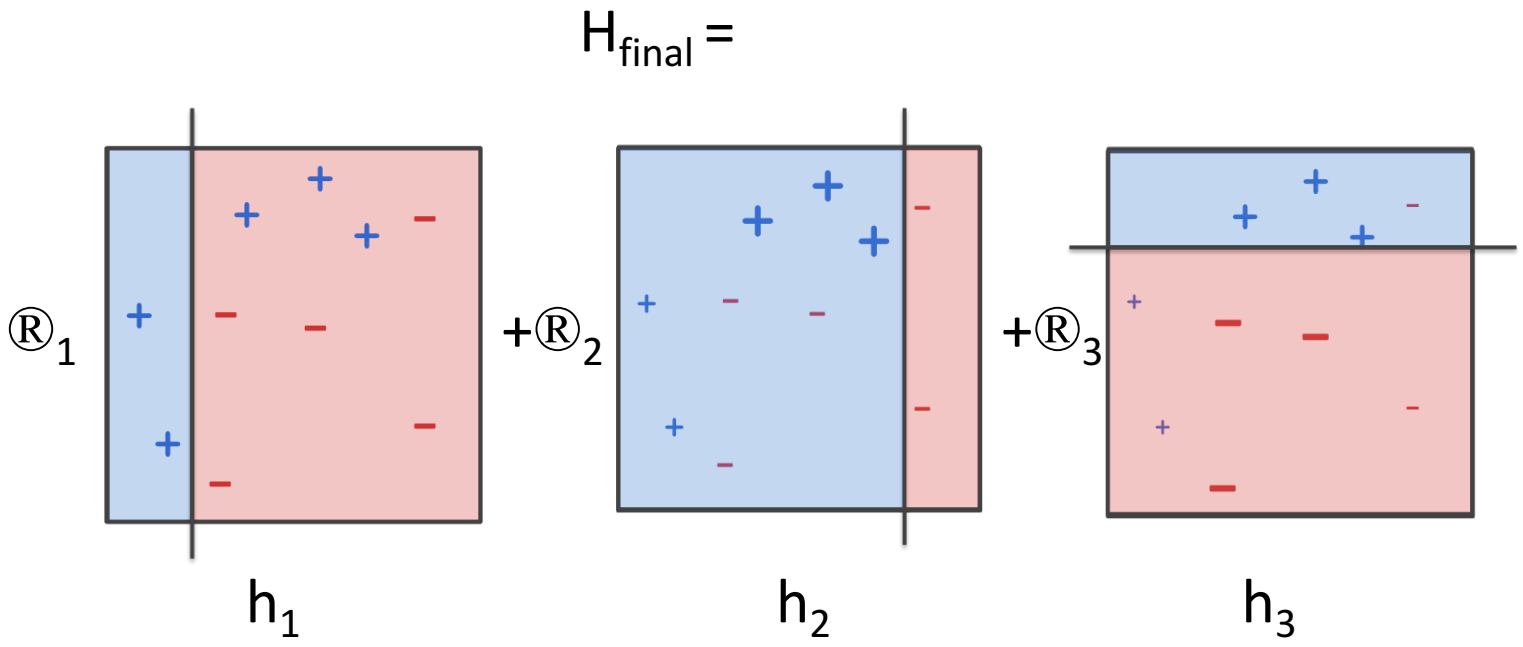
$D_t$ = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

$h_3$ = A classifier learned on this data. *Has an error* $\epsilon_3 = 0.14$

Why not 0.3? Because while computing error, we will weight each example $x_i$ by its $D_t(i)$

# A toy example

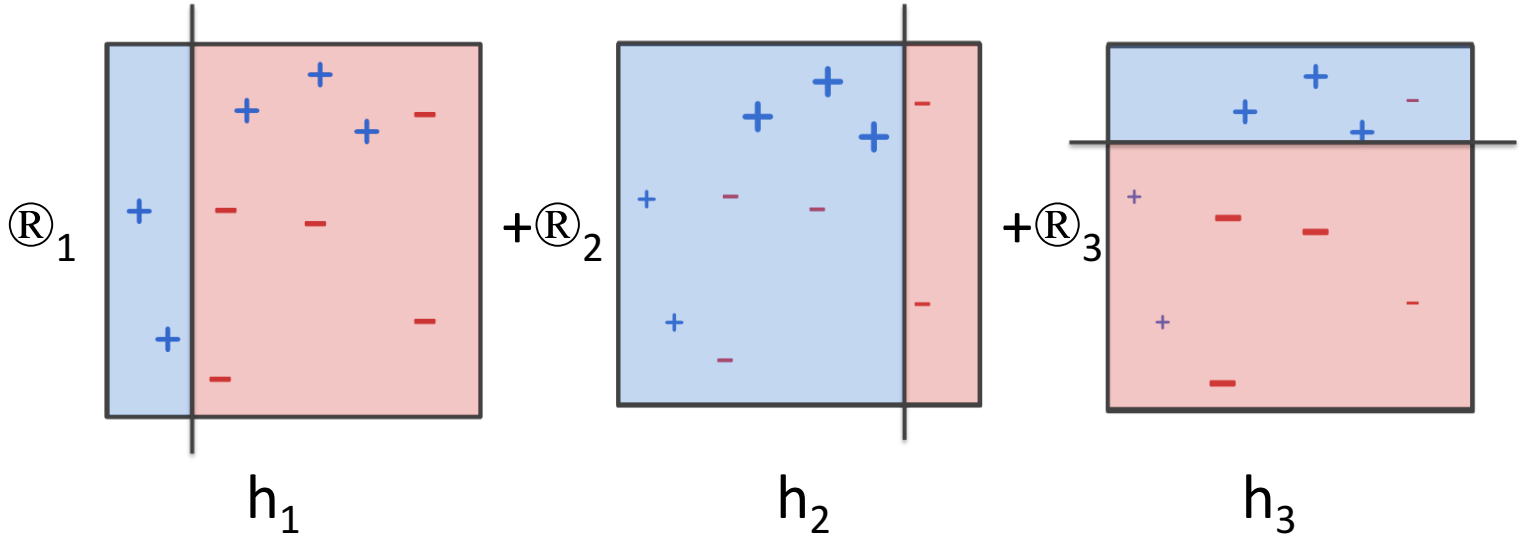
The final hypothesis is a combination of all the $h_i$'s we have seen so far



# A toy example

The final hypothesis is a combination of all the $h_i$'s we have seen so far

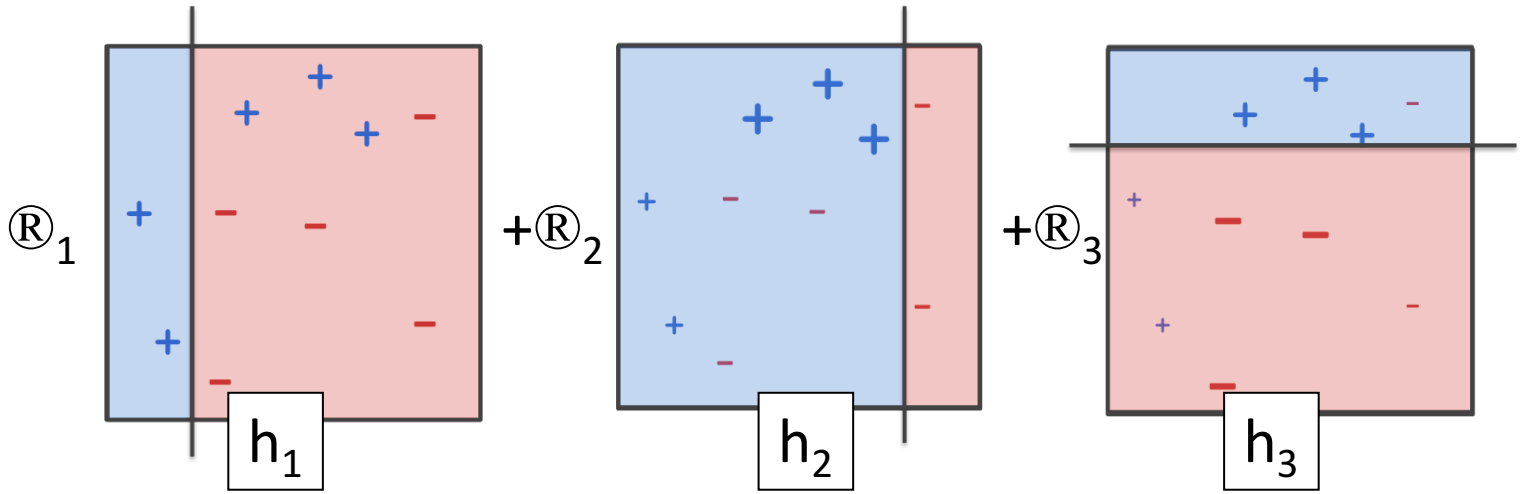
$H_{final} =$

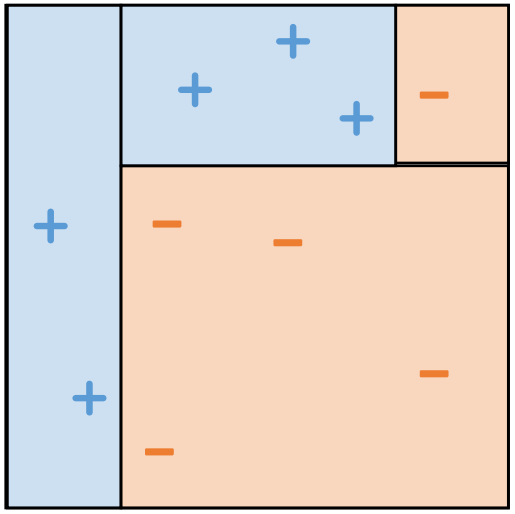



Think of the ® values as the vote for each weak classifier and the boosting algorithm has to somehow specify them

# A toy example: final classifier:

$H_{final} =$



# An outline of Boosting

How to?

Given a training set $(x_1, y_1), \cdots, (x_m, y_m)$

- Instances $x_i \in X$ labeled with $y_i \in \{-1, +1\}$

- For t = 1, 2, ⋯, T:
  - Construct a distribution $D_t$ on {1, 2, ⋯, m}
  - Find a weak hypothesis (rule of thumb) $h_t$ such that it has a small *weighted* error $\epsilon_t$ on $D_t$

How to?

How to?

- Construct a final output $H_{final}$

How to?

# AdaBoost: Constructing $D_t$

We have m examples

$D_t$ is a set of weights over the examples at round t

$$D_t(1), D_t(2), \cdots D_t(m)$$

At every round, the weak learner looks for hypotheses $h_t$ that emphasizes examples that have a higher $D_t$

# AdaBoost: Constructing $D_t$

Initially (t = 1), use the uniform distribution over all examples

$$D_1(i) = \frac{1}{m}$$

# AdaBoost: Constructing $D_t$

Initially (t = 1), use the uniform distribution over all examples

$$D_1(i) = \frac{1}{m}$$

After t rounds

- What we have
  - $D_t$ and the hypothesis $h_t$ that was learned
  - The error $\epsilon_t$ of that hypothesis on the training data

- (Intuition) What we want from the $(t+1)^{th}$ round
  - Find a hypothesis so that examples that were incorrect in the previous round are correctly predicted by the new one
  - That is, increase the importance of misclassified examples and decrease the importance of correctly predicted ones

# AdaBoost: Constructing $D_t$

Initially (t = 1), use the uniform distribution over all examples

$$D_1(i) = \frac{1}{m}$$

After t rounds, we have some $D_t$ and a hypothesis $h_t$ that the weak learner produced (choose some $\alpha_t > 0$)

Create $D_{t+1}$ as follows:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

$$= \frac{D_t(i)}{Z_t} \cdot \exp\left(-\alpha_t \cdot y_i h_t(x_i)\right)$$

Demote correctly predicted examples

Promote incorrectly predicted examples

# AdaBoost: Constructing $D_t$

After t rounds, we have some $D_t$ and a hypothesis $h_t$ that the weak learner produced

Create $D_{t+1}$ as follows:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

$$= \frac{D_t(i)}{Z_t} \cdot \exp\left(-\alpha_t \cdot y_i h_t(x_i)\right)$$

$Z_t$: A normalization constant. Ensures that the weights $D_{t+1}$ add up to 1

# AdaBoost: Constructing $D_t$

After t rounds, we have some $D_t$ and a hypothesis $h_t$ that the weak learner produced

Create $D_{t+1}$ as follows:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

$$= \frac{D_t(i)}{Z_t} \cdot \exp\left(-\alpha_t \cdot y_i h_t(x_i)\right)$$

Our magical choice of $\alpha_t$

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

Show that $\epsilon_t > 0.5 \Rightarrow \alpha_t > 0$

# An outline of Boosting

How to?

Given a training set $(x_1, y_1), \cdots, (x_m, y_m)$

- Instances $x_i \in X$ labeled with $y_i \in \{-1, +1\}$

- For t = 1, 2, ⋯, T:
  - Construct a distribution $D_t$ on {1, 2, ⋯, m}
  - Find a weak hypothesis (rule of thumb) $h_t$ such that it has a small *weighted* error $\epsilon_t$ on $D_t$

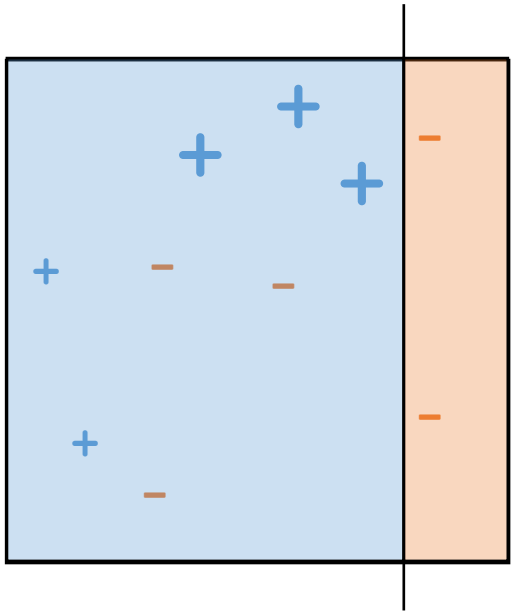
✓

If the classifier is simple, this is often easy

- Construct a final output $H_{final}$

How to?

# Example (Decision Stumps




We can simply enumerate all possible ways of putting vertical and horizontal lines to separate the data points into two classes and nd the one with the smallest weighted classfication error!

This efficency of finding weak classifier is an attractive quality of boosting

# An outline of Boosting

How to?

Given a training set $(x_1, y_1), \cdots, (x_m, y_m)$

- Instances $x_i \in X$ labeled with $y_i \in \{-1, +1\}$

- For t = 1, 2, ⋯, T:
  - Construct a distribution $D_t$ on {1, 2, ⋯, m}
  - Find a weak hypothesis (rule of thumb) $h_t$ such that it has a small *weighted* error $\epsilon_t$ on $D_t$

✓

✓

- Construct a final output $H_{final}$

How to?

# The final hypothesis

- After T rounds, we have
  - T weak classifiers $h_1$, $h_2$, $\cdots$ $h_T$
  - T values of $®_t$

- Recall that each weak classifier is takes an example $\mathbf{x}$ and produces a -1 or a +1
- Define the final hypothesis $\mathrm{H}_{\mathrm{final}}$ as

$$H_{final}(x) = \mathrm{sgn}\left(\sum_t \alpha_t h_t(x)\right)$$

# AdaBoost: The full algorithm

Given a training set $(x_1, y_1), \cdots, (x_m, y_m)$

Instances $x_i \in X$ labeled with $y_i \in \{-1, +1\}$

T: a parameter to the learner

1. Initialize $D_1(i) = 1/m$ for all $i = 1, 2, \cdots, m$
2. For $t = 1, 2, \cdots T$:
   1. Find a classifier $h_t$ whose *weighted classification error* is better than chance
   2. Compute its vote

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$$

   3. Update the values of the weights for the training examples

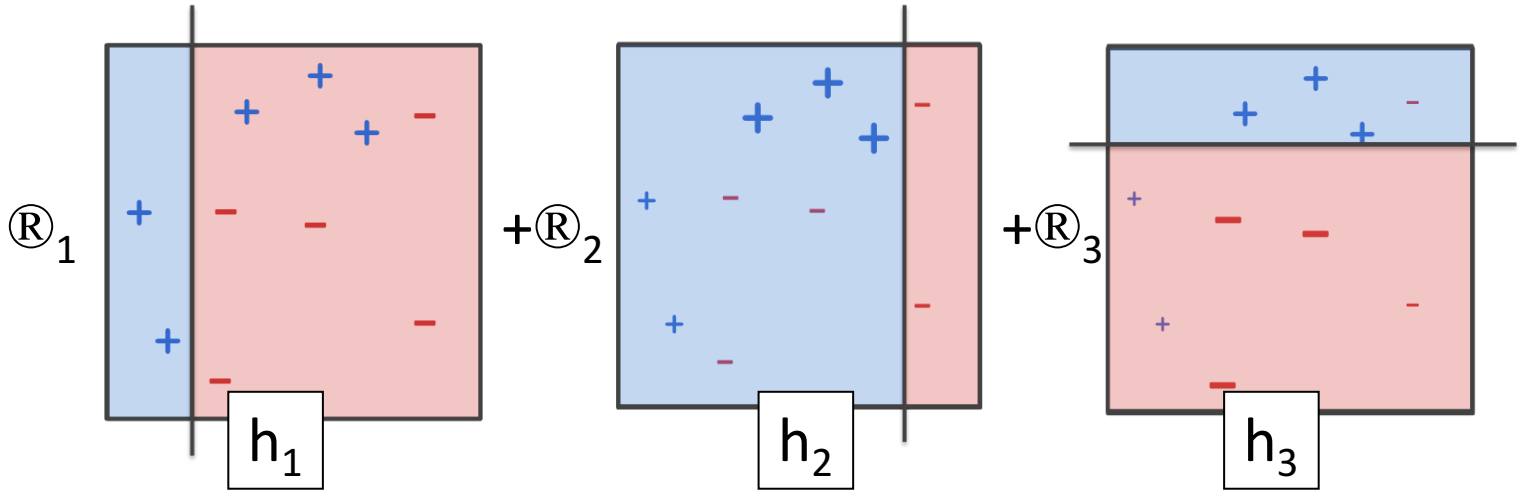
$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \exp\left(-\alpha_t \cdot y_i h_t(x_i)\right)$$

3. Return the final hypothesis

$$H_{final}(x) = \operatorname{sgn}\left( \sum_t \alpha_t h_t(x) \right)$$

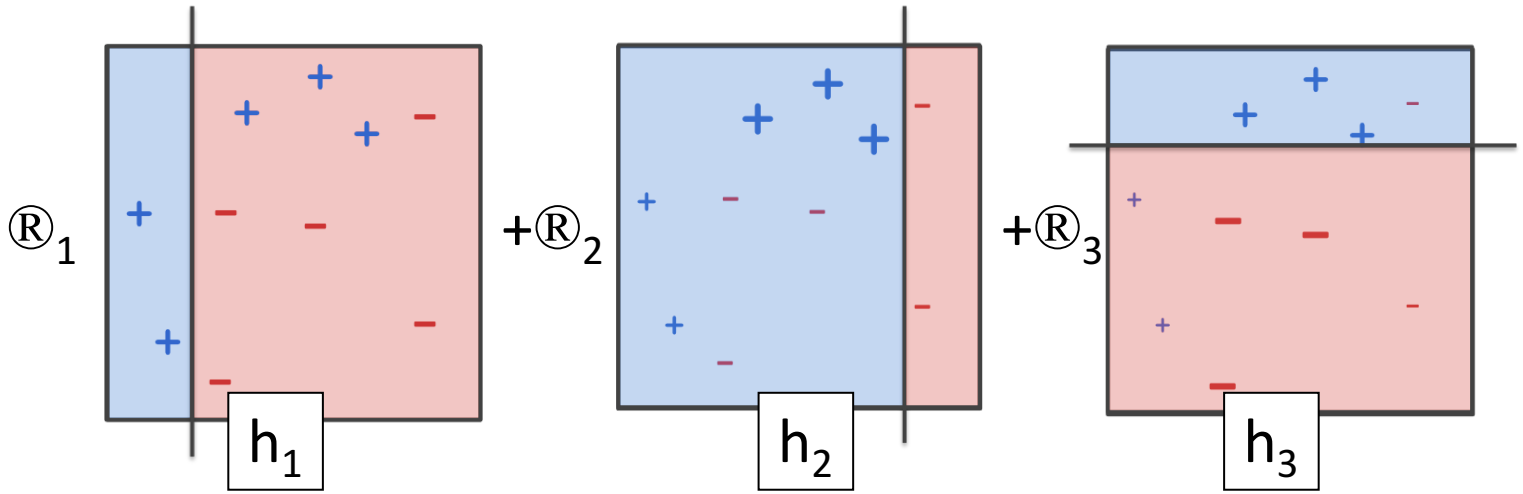
# Back to the toy example

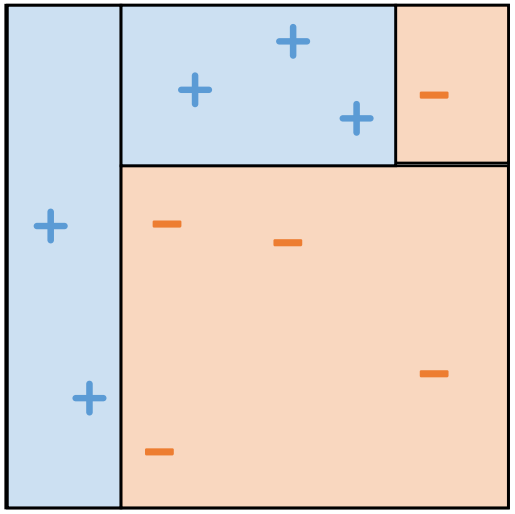
$H_{final} =$



# Back to the toy example

$H_{final} =$



# Analyzing the training error

**Theorem**:

- Run AdaBoost for T rounds
- Let $\epsilon_t$ = ½ - $\gamma_t$
- Let 0 < $\gamma \leq \gamma_t$ for all t
- Then,

$$\text{Training error}(H_{\text{final}}) \leq e^{-2\gamma^2 T}$$

# Analyzing the training error

**Theorem**:

- Run AdaBoost for T rounds
- Let ²t = ½ - °t
- Let 0 < ° · °t for all t
- Then,

We have a weak learner

$$\text{Training error}(H_{\text{final}}) \leq e^{-2\gamma^2 T}$$

# Analyzing the training error

**Theorem**:

- Run AdaBoost for T rounds
- Let ² t = ½ - ° t
- Let 0 < ° · ° t for all t
- Then,

We have a weak learner

As T increases, the training error drops exponentially

$$\text{Training error}(H_{\text{final}}) \leq e^{-2\gamma^2 T}$$

# Analyzing the training error

**Theorem**:

- Run AdaBoost for T rounds
- Let $^2_t$ = ½ - $°_t$
- Let 0 < ° · $°_t$ for all t
- Then,

We have a weak learner

As T increases, the training error drops exponentially

$$\text{Training error}(H_{\text{final}}) \leq e^{-2\gamma^2 T}$$

Proof is simple, see pointer on website

# Analyzing the training error

**Theorem**:

- Run AdaBoost for T rounds
- Let $^{2}_{t}$ = ½ - $^{\circ}_{t}$
- Let 0 < $^{\circ} \cdot ^{\circ}_{t}$ for all t
- Then,

We have a weak learner

As T increases, the training error drops exponentially

$$\text{Training error}(H_{\text{fin}}) \leq e^{-2\gamma^2 T}$$

Is it sufficient to upper bound the training error?

Proof is simple, see pointer on website

# Analyzing the training error

**Theorem**:

- Run AdaBoost for T rounds
- Let $\epsilon_t = ½ - \gamma_t$
- Let $0 < \gamma \leq \gamma_t$ for all t
- Then,

We have a weak learner

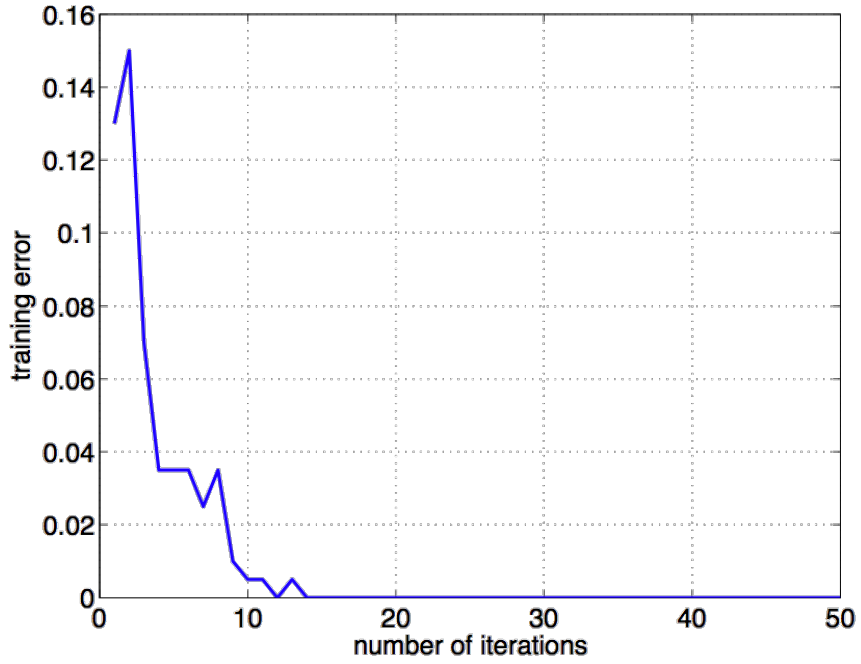
As T increases, the training error drops exponentially

$$\text{Training error}(H_{\text{final}}) \leq e^{-2\gamma^2 T}$$

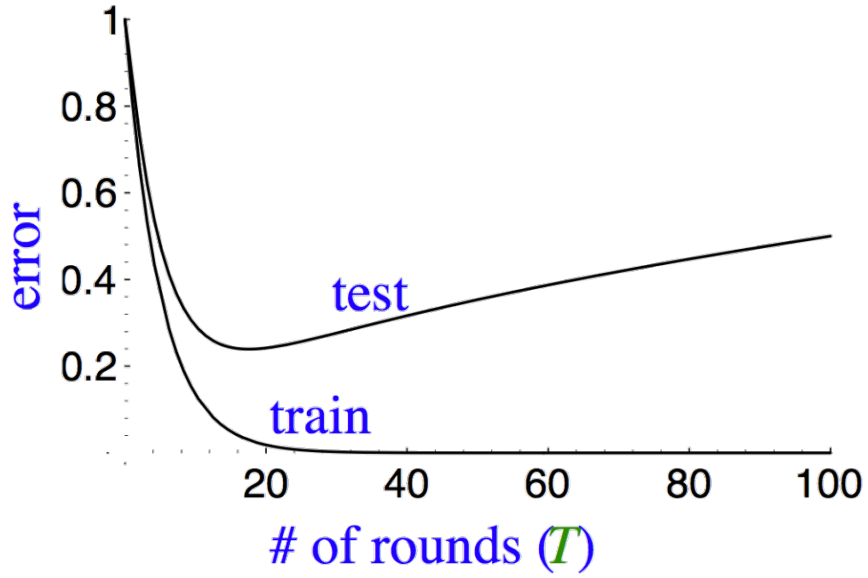
See the note at: https://svivek.com/teaching/machine-learning/fall2017/readings/boosting.pdf

# Adaboost: Training error

The training error of the combined classifier decreases exponentially fast if the errors of the weak classifiers (the $\epsilon_t$) are strictly better than chance



# What about the test error?



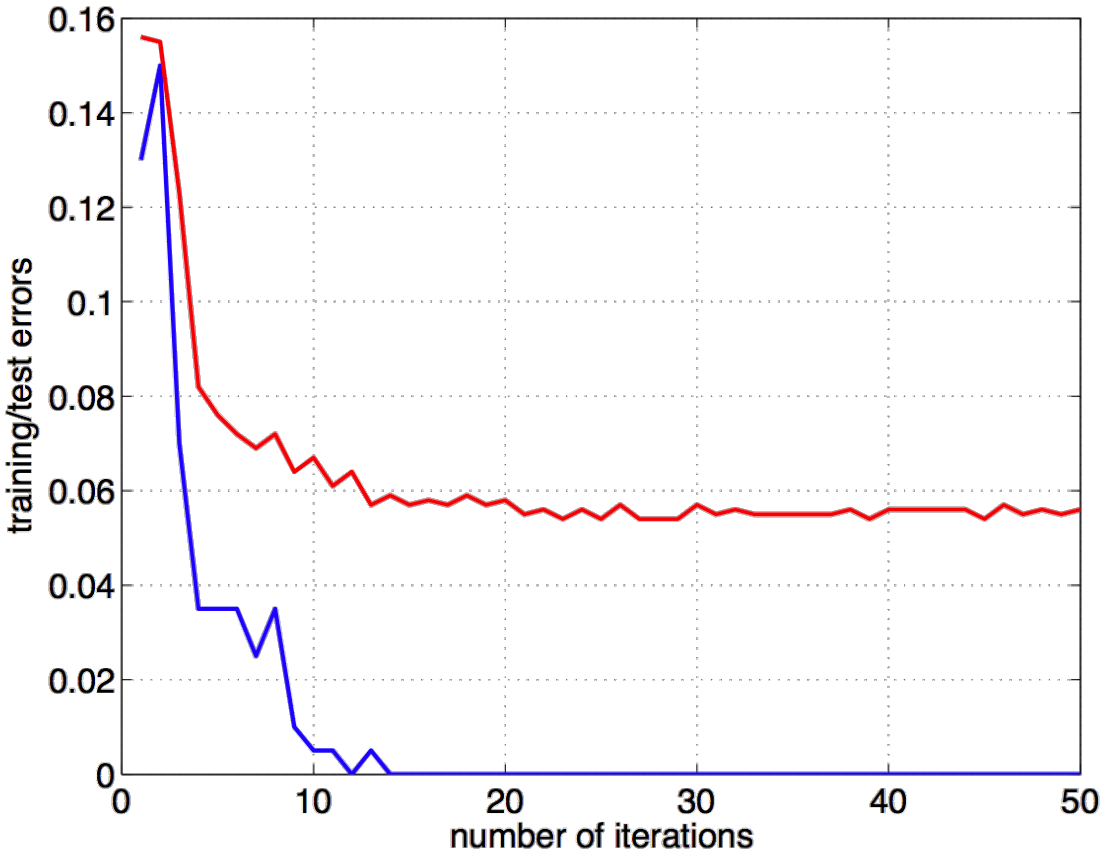

What the theory tells us:

Training error will keep decreasing or reach zero (the AdaBoost theorem)

Test error will increase after the $H_{final}$ becomes too "complex"

- Overfitting

# In practice


0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
training/test errors
0
10
20
30
40
50
number of iterations

# Boosting

- Initialization:
  - Weigh all training samples equally
- Iteration Step:
  - Train model on weighted train set
  - Compute weighted error of model on train set
  - Increase weights on training cases model gets wrong
- Typically requires 100's to 1000's of iterations
- Return final model:
  - Carefully weighted prediction of each model

# What you have learned

- Bagging: A simple but effective approach to combine multiple classifiers
- Boosting: Algorithm and key intuition

- More practically used algorithms:
  - Bagging+ Decision tree ⇒ Random forest
  - Gradient boosting
    https://en.wikipedia.org/wiki/Grdient_boosting