INF552 Machine Learning

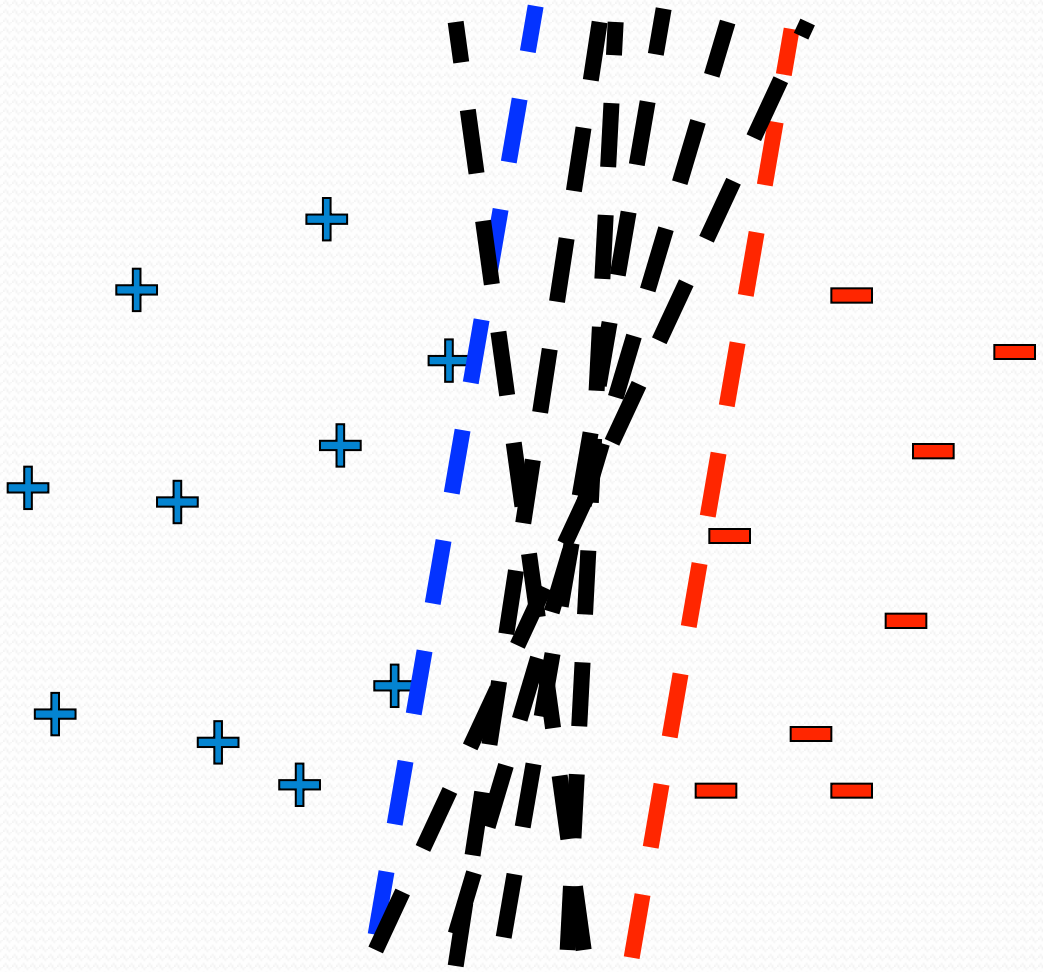
Liyue Fan
liyuefan@usc.edu
Integrated Media Systems Center
University of Southern California

Notes

- HW3 will be out
  - Email ahead if plan to drop by my office
- Project intro and grouping today
- Project team/topic registration on bb due 10/21.
- Email your paper presentations (.ppt or .pptx) to me and Kien.

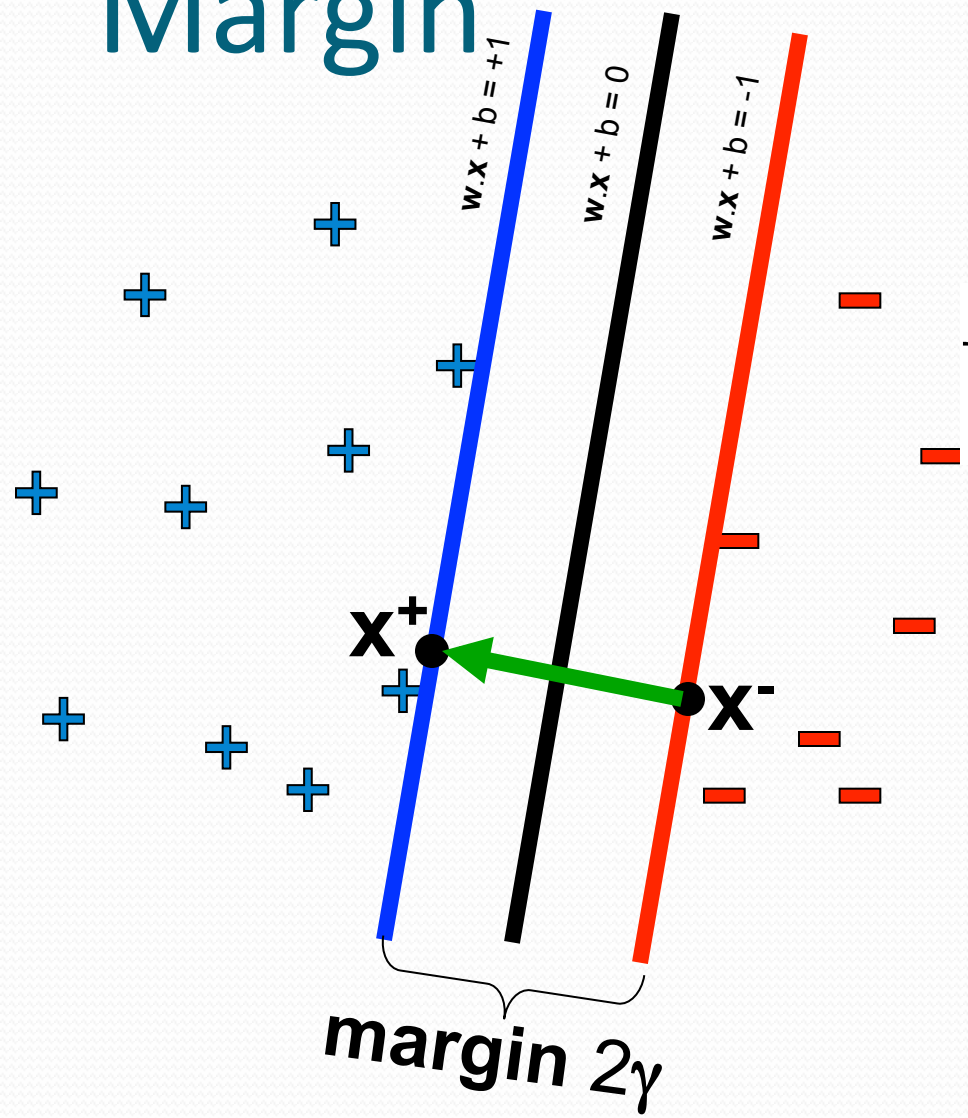
Classification (cont.)

Linear classifiers – Which line is better?



$$\mathbf{w}.\mathbf{x} = \sum_j w^{(j)} x^{(j)}$$

Margin



$$x^+ = x^- + \lambda \frac{W}{\|W\|}$$

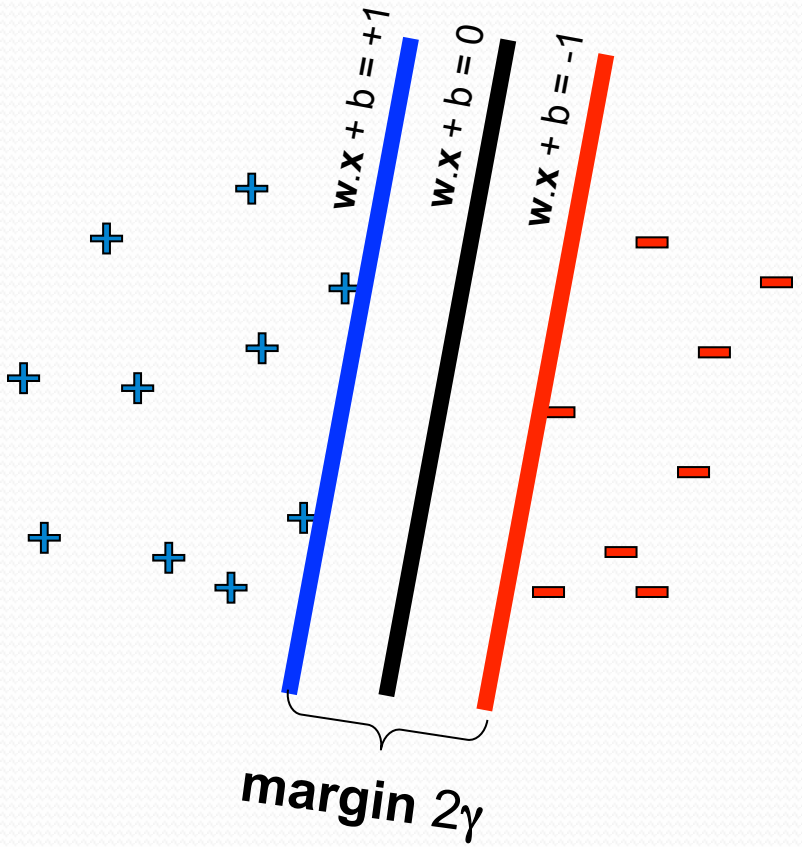
$$\mathbf{w}.\mathbf{x}^+ + b = 1$$

$$\mathbf{w}.(\mathbf{x}^- + \lambda \frac{\mathbf{w}}{||\mathbf{w}||}) + b = 1$$

$$\lambda = \frac{2}{||\mathbf{w}||}$$

$$\gamma = \frac{1}{\sqrt{\mathbf{w}.\mathbf{w}}}$$

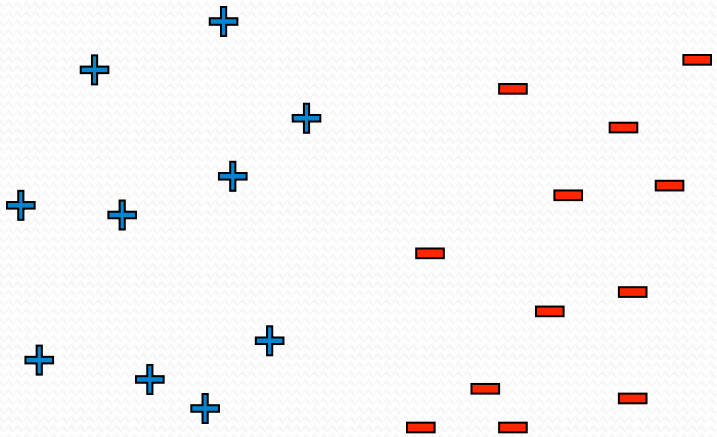
Support vector machines (SVMs)



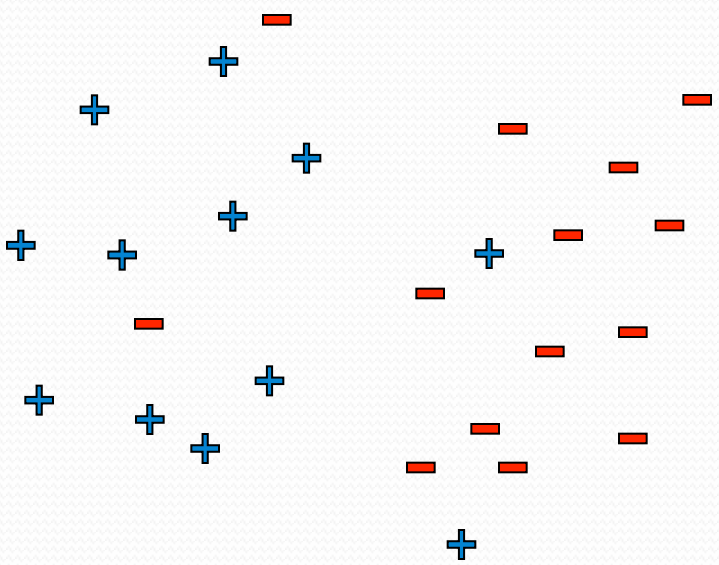
$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w}$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1, \ \forall j$$

- Solve by quadratic programming (QP)
  - Well-studied solution algorithms
  - Complexity depends on *n*
- Hyperplane defined by support vectors

What if the data is not linearly separable?



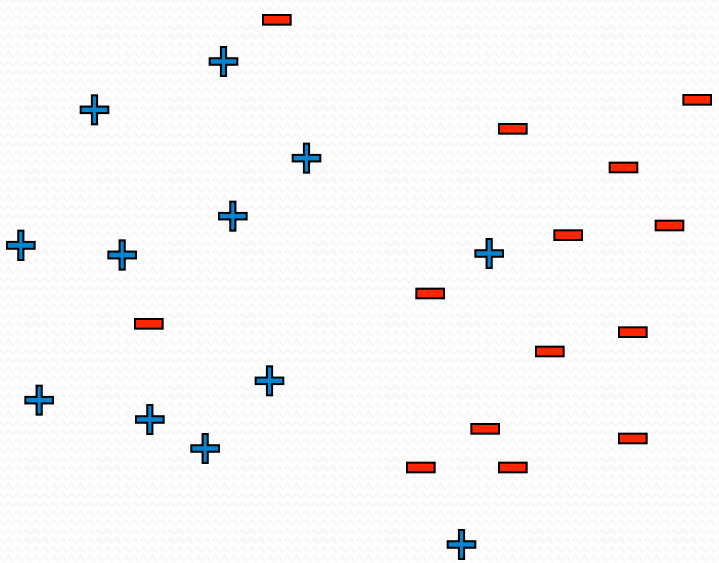
What if the data is not linearly separable?



$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w}$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 \qquad ,\forall j$$

- Minimize **w.w** and number of training mistakes
  - 0/1 loss
  - Slack penalty *C*
  - Not QP anymore
  - Also doesn't distinguish near misses and really bad mistakes

Slack variables – Hinge loss



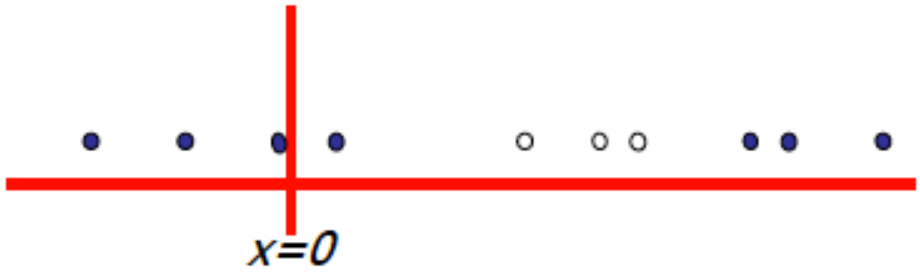
$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w} + C\sum_j \xi_j$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j, \ \forall j$$
$$\xi_j \geq 0, \ \forall j$$

When $0 < \xi_j < 1$, x_j is correctly classified with margin less than 1.

When $\xi_j > 1$, x_j is misclassified.

- C defines a trade-off between margin maximization and error minimization.
  - If too large, high penalty for non-separable cases thus overfits.
  - If too small, too simple solutions thus underfits.
  - Choose from [10^{-6}, 10^{-5},…, 10^{5}, 10^{6}] by accuracy in validation set.

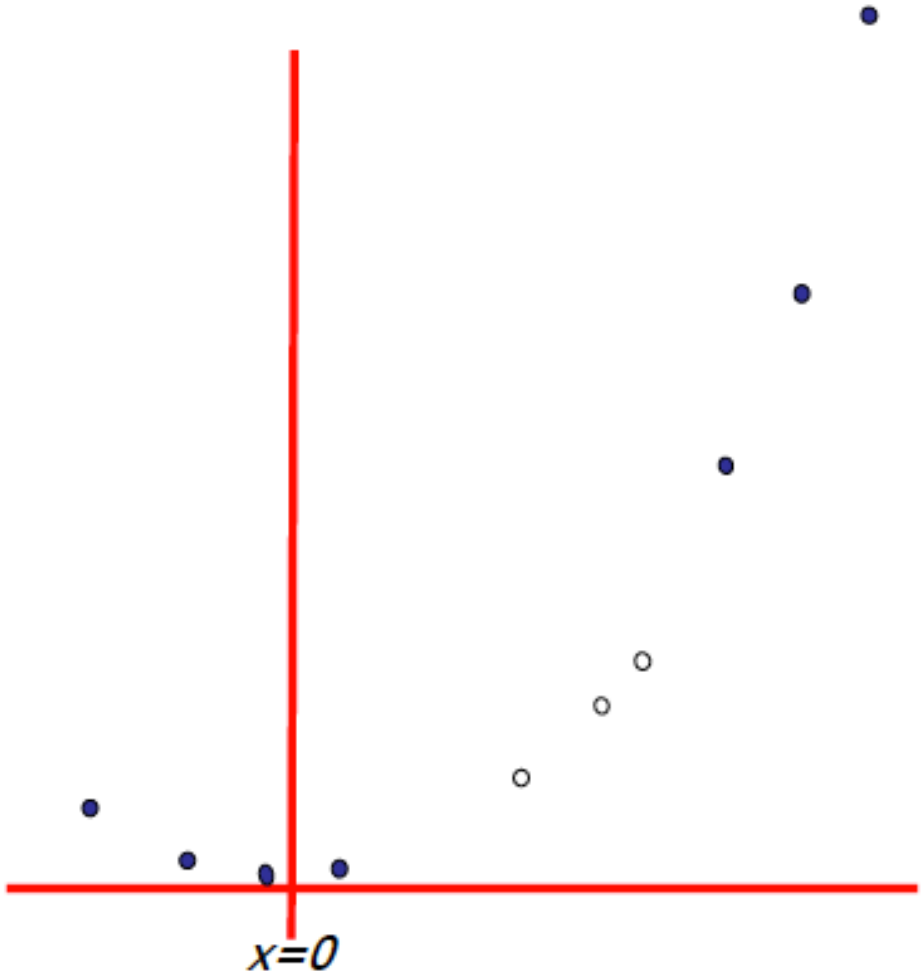
Harder 1-dimensional dataset



That's wiped the smirk off SVM's face.

What can be done about this?

Harder 1-dimensional dataset

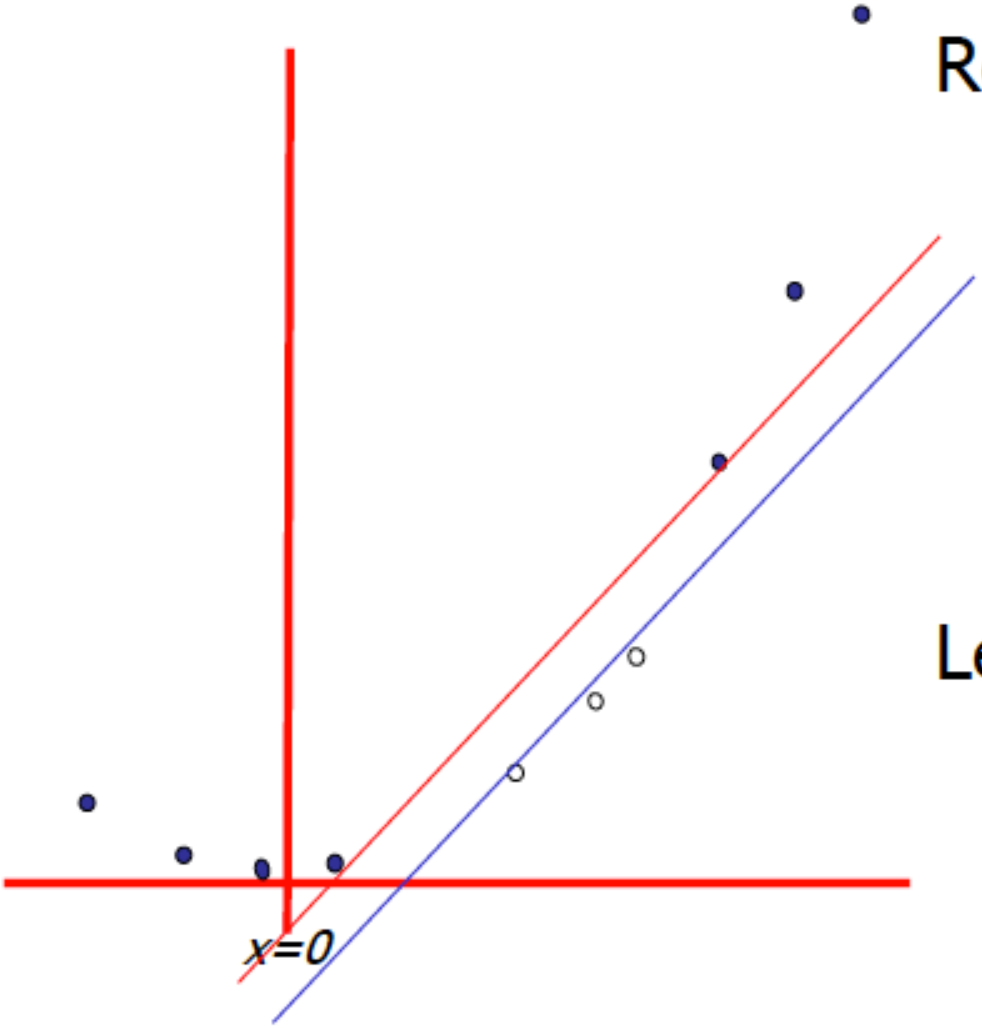


Remember how permitting non-linear basis functions made linear regression so much nicer?

Let's permit them here too

$$\mathbf{z}_k = (x_k, x_k^2)$$

Harder 1-dimensional dataset



Remember how permitting non-linear basis functions made linear regression so much nicer?

Let's permit them here too

$$\mathbf{z}_k = (x_k, x_k^2)$$

Does this always work?

Lemma

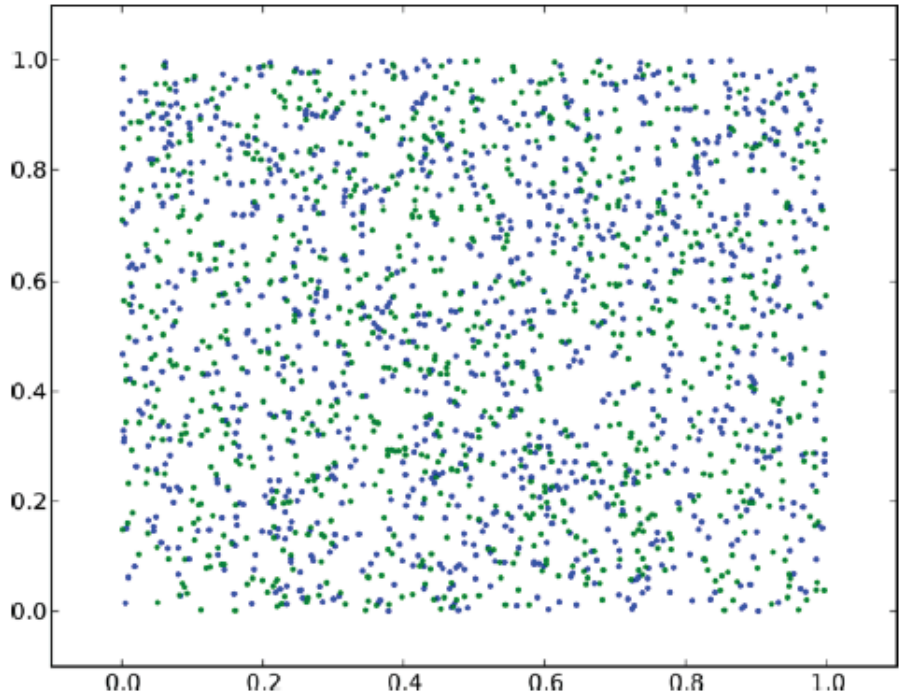
Let $(x_i)_{i=1,\dots,n}$ with $x_i \neq x_j$ for $i \neq j$. Let $\varphi : \mathbb{R}^k \to \mathbb{R}^m$ be a feature map. If the set $\varphi(x_i)_{i=1,\dots,n}$ is linearly independent, then the points $\varphi(x_i)_{i=1,\dots,n}$ are linearly separable.

Lemma

If we choose $m > n$ large enough, we can always find a map φ.

Caveat

Caveat: We can separate *any* set, not just one with "reasonable" y_i:



There is a fixed feature map $\varphi : \mathbb{R}^2 \to \mathbb{R}^{20001}$ such that – no matter how we label them – there is always a hyperplane classifier that has 0 training error.

Kernel Trick

- One of the most interesting and exciting advancement in the last 2 decades of machine learning
  - The “kernel trick”
  - High dimensional feature spaces at no extra cost!

- But first, a detour
  - Constrained optimization!

Dual SVM derivation (1) – the linearly separable case

$$\text{minimize}_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}.\mathbf{w}$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1, \ \ \forall j$$

Dual SVM derivation (1) – the linearly separable case

$$L(\mathbf{w}, \alpha) = \frac{1}{2}\mathbf{w}.\mathbf{w} - \sum_j \alpha_j \left[\left(\mathbf{w}.\mathbf{x}_j + b \right) y_j - 1 \right]$$
$$\alpha_j \geq 0, \ \forall j$$

$$\mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

Dual SVM formulation – the linearly separable case

maximize$_\alpha$ $\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j$

$\sum_i \alpha_i y_i = 0$

$\alpha_i \geq 0$

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$b = y_k - \mathbf{w}.\mathbf{x}_k$$

for any k where $\alpha_k > 0$

Dual SVM formulation – the non-separable case

$$\text{minimize}_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_j \xi_j$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j, \ \ \forall j$$
$$\xi_j \geq 0, \ \ \forall j$$

Dual SVM formulation – the non-separable case

$$\text{maximize}_{\alpha} \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$b = y_k - \mathbf{w}.\mathbf{x}_k$$

for any k where $C > \alpha_k > 0$

Why did we learn about the dual SVM?

- Builds character!
- Exposes structure about the problem
- There are some quadratic programming algorithms that can solve the dual faster than the primal
- The "**kernel trick**"!!!

Dual formulation only depends on dot-products, not on **w**!

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

Feature Vector

$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

Example: $K(x,z) = (x^T z)^2$

$$K(x,z) = \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_i z_i\right)$$
$$= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j$$
$$= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j)$$

$$K(x,z) = \phi(x)^T \phi(z)$$

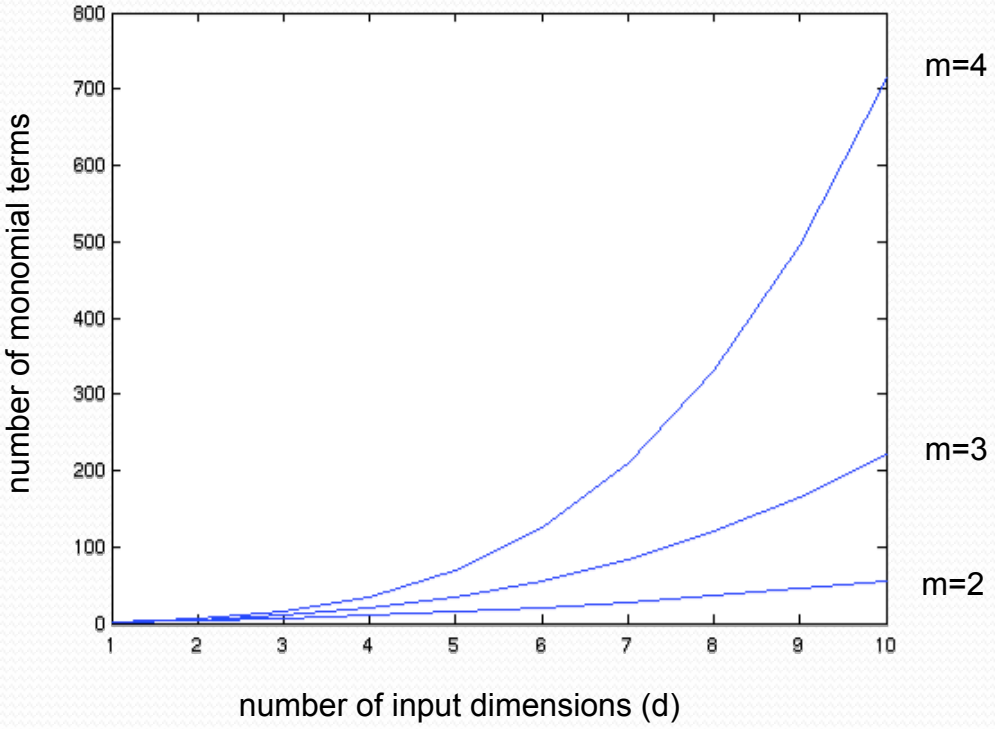
n=3:

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \end{bmatrix}$$

- Computation complexity
  - K(x,z): *O(n)*
  - Φ(x): *O(n^2)*
- Corresponds to high dimensional feature space without explicitly computing the features

Higher order polynomials

$$\#\text{terms} = D = \binom{m+d-1}{m} = \frac{(m+d-1)!}{m!(d-1)!}$$



d – input features

m – degree of polynomial

grows fast!

m = 6, d = 100

D = about 1.6 billion terms

Common kernels

- Polynomials of degree d

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^d$$

- Polynomials of degree up to d

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^d$$

- Gaussian kernel / Radial Basis Function

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{||\mathbf{u} - \mathbf{v}||^2}{2\sigma^2}\right)$$

Corresponds to an infinite dimensional feature mapping Φ

- Sigmoid

$$K(\mathbf{u}, \mathbf{v}) = \tanh(\eta \mathbf{u} \cdot \mathbf{v} + \nu)$$

Kernel Demo

- Demo
  - http://www.eee.metu.edu.tr/~alatan/Courses/Demo/AppletSVM.html

What is a kernel?

- k: X x X → R
- Any measure of “similarity” between two inputs
- Mercer Kernel / Positive Semi-Definite Kernel
  - Often just called “kernel”

How to Check if a Function is a Kernel

Problem:

- Checking if a given $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ fulfills the conditions for a kernel is *difficult*:
- We need to prove or disprove

$$\sum_{i,j=1}^{n} t_i k(x_i, x_j) t_j \geq 0.$$

for *any set* $x_1, \ldots, x_n \in \mathcal{X}$ *and any* $t \in \mathbb{R}^n$ *for any* $n \in \mathbb{N}$.

Workaround:

- It is easy to *construct* functions k that are positive definite kernels.

Constructing Kernels

1) We can *construct kernels from scratch*:

- For any $\varphi : \mathcal{X} \to \mathbb{R}^m$, $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathbb{R}^m}$ is a kernel.

Constructing Kernels

1) We can *construct kernels from scratch*:

- For any $\varphi : \mathcal{X} \to \mathbb{R}^m$, $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathbb{R}^m}$ is a kernel.
- If $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *distance function*, i.e.
  - $d(x, x') \geq 0$ for all $x, x' \in \mathcal{X}$,
  - $d(x, x') = 0$ only for $x = x'$,
  - $d(x, x') = d(x', x)$ for all $x, x' \in \mathcal{X}$,
  - $d(x, x') \leq d(x, x'') + d(x'', x')$ for all $x, x', x'' \in \mathcal{X}$,

  then $k(x, x') := \exp(-d(x, x'))$ is a kernel.

Constructing Kernels

1) We can *construct kernels from scratch*:

- For any $\varphi : \mathcal{X} \to \mathbb{R}^m$, $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathbb{R}^m}$ is a kernel.
- If $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *distance function*, i.e.
  - $d(x, x') \geq 0$ for all $x, x' \in \mathcal{X}$,
  - $d(x, x') = 0$ only for $x = x'$,
  - $d(x, x') = d(x', x)$ for all $x, x' \in \mathcal{X}$,
  - $d(x, x') \leq d(x, x'') + d(x'', x')$ for all $x, x', x'' \in \mathcal{X}$,

  then $k(x, x') := \exp(-d(x, x'))$ is a kernel.

2) We can *construct kernels from other kernels*:

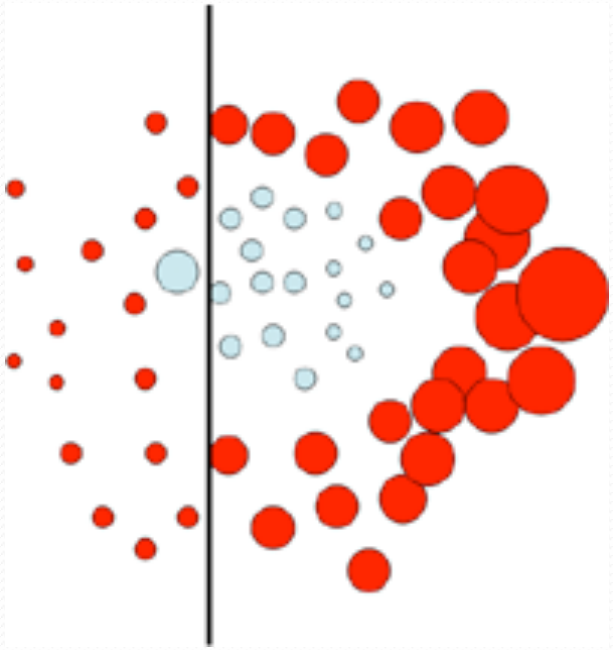
- if k is a kernel and $\alpha > 0$, then αk and $k + \alpha$ are kernels.
- if k_1, k_2 are kernels, then $k_1 + k_2$ and $k_1 \cdot k_2$ are kernels.

Kernels

- Kernel Logistic Regression
- Kernel Least Squares
- Kernel PCA …

New Topic: Ensemble Methods

Bagging



Boosting

Synonyms

- Ensemble Methods
- Learning Mixture of Experts/Committees

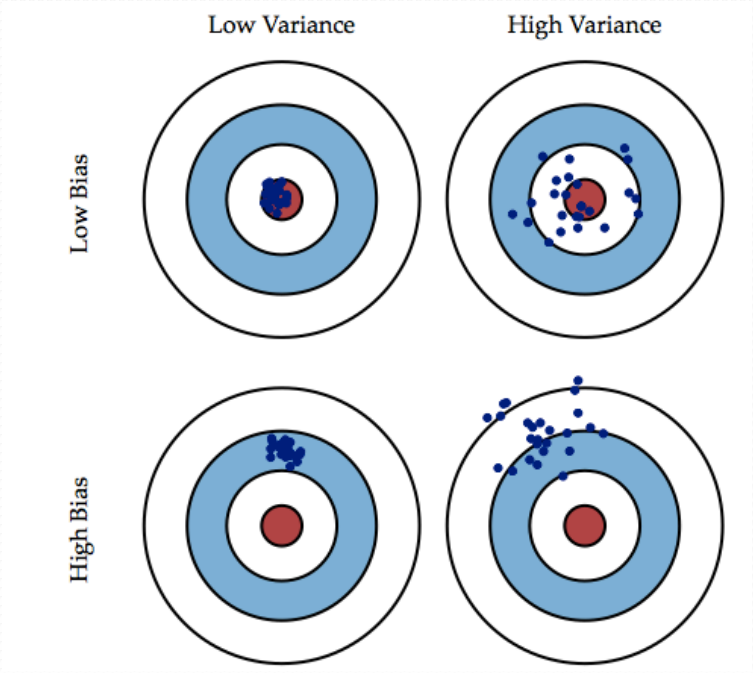
A quick look back

- So far you have learnt
- Regression
  - Least Squares
- Classification
  - Linear
    - Naïve Bayes
    - Logistic Regression
    - SVMs
  - Non-linear
    - Decision Trees
    - Neural Networks
    - K-NNs

Bias-Variance Tradeoff

- Prediction error of model f(x) at x:

$$Err(x) = \underbrace{\left(E[\hat{f}(x)] - f(x)\right)^2}_{\text{bias}^2} + \underbrace{E\left[\hat{f}(x) - E[\hat{f}(x)]\right]^2}_{\text{variance}} + \underbrace{\sigma_e^2}_{\text{irreducible error}}$$



Bias-Variance Tradeoff

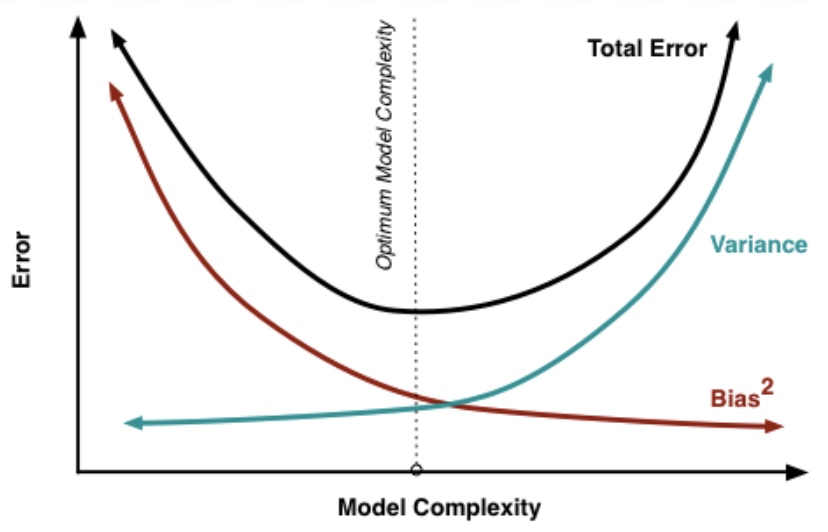
- Simple model: high bias, low variance

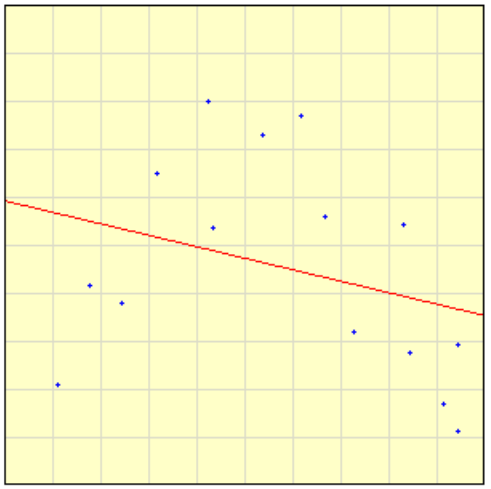
  → underfits

- Complex model: low bias, high variance

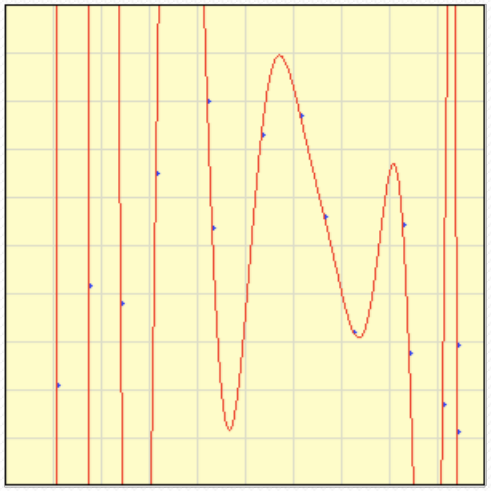
  → overfits

- Finding the optimal model complexity





Polynomial degree = 1



Polynomial degree = 13

Optimal complexity

- Empirically, use cross-validate
- Theoretically, use AIC measure

- Read more on model selection:
- http://www.stat.cmu.edu/~larry/=stat705/Lecture16.pdf

Fighting the bias-variance tradeoff

- **Simple (a.k.a. weak) learners**
  - e.g., naïve Bayes, logistic regression, decision stumps (depth-1 decision trees)
  - **Good:** Low variance, don't usually overfit
  - **Bad:** High bias, can't solve hard learning problems

- **Sophisticated learners**
  - Kernel SVMs, Deep Neural Nets, Deep Decision Trees
  - **Good:** Low bias, have the potential to learn with Big Data
  - **Bad:** High variance, difficult to generalize

- Can we make combine these properties
  - **In general, No!!**
  - **But often yes…**

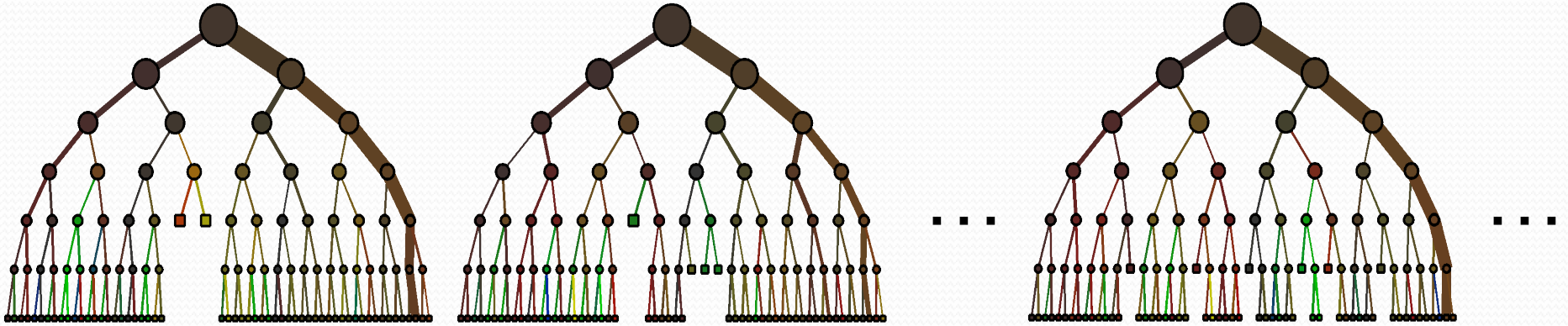
Voting (Ensemble Methods)

- Instead of learning a single classifier, learn **many classifiers**
- **Output class:** (Weighted) vote of each classifier
  - Classifiers that are most "sure" will vote with more conviction
- With sophisticated learners
  - Uncorrelated errors → expected error goes down
  - On average, do better than single classifier!
  - Bagging
- With weak learners
  - each one good at different parts of the input space
  - On average, do better than single classifier!
  - Boosting

Bagging

- Bagging = Bootstrap aggregating
- Let the original training data be L
- Repeat B times:
  - Get a bootstrap sample L_k from L.
  - Train a predictor using L_k.
- Combine B predictors by
  - Voting (for classification problem)
  - Averaging (for regression problem)
- Bootstrap Demo http://wise.cgu.edu/bootstrap/

Decision(Random) Forests



Learn many trees & Average Outputs

Random Forests

1. Select *n*, the number of trees to grow, and *m*, a number no larger than number of variables.

2. For i = 1 to *n*:
3. Draw a bootstrap sample from the data. Call those not in the bootstrap sample the "out-of-bag" data.
4. Grow a "random" tree, where at each node, the best split is chosen among *m* randomly selected variables. The tree is grown to maximum size and not pruned back.
5. Use the tree to predict out-of-bag data.

6. Prediction of test data is done by majority votes from predictions from the ensemble of trees.

Random Forest Classifier

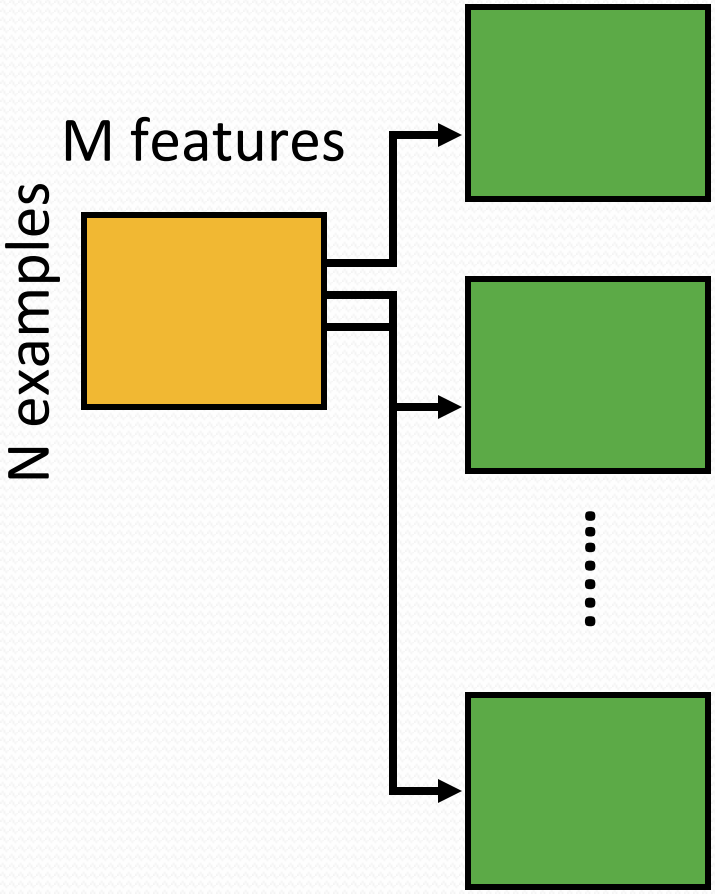
Training Data

M features

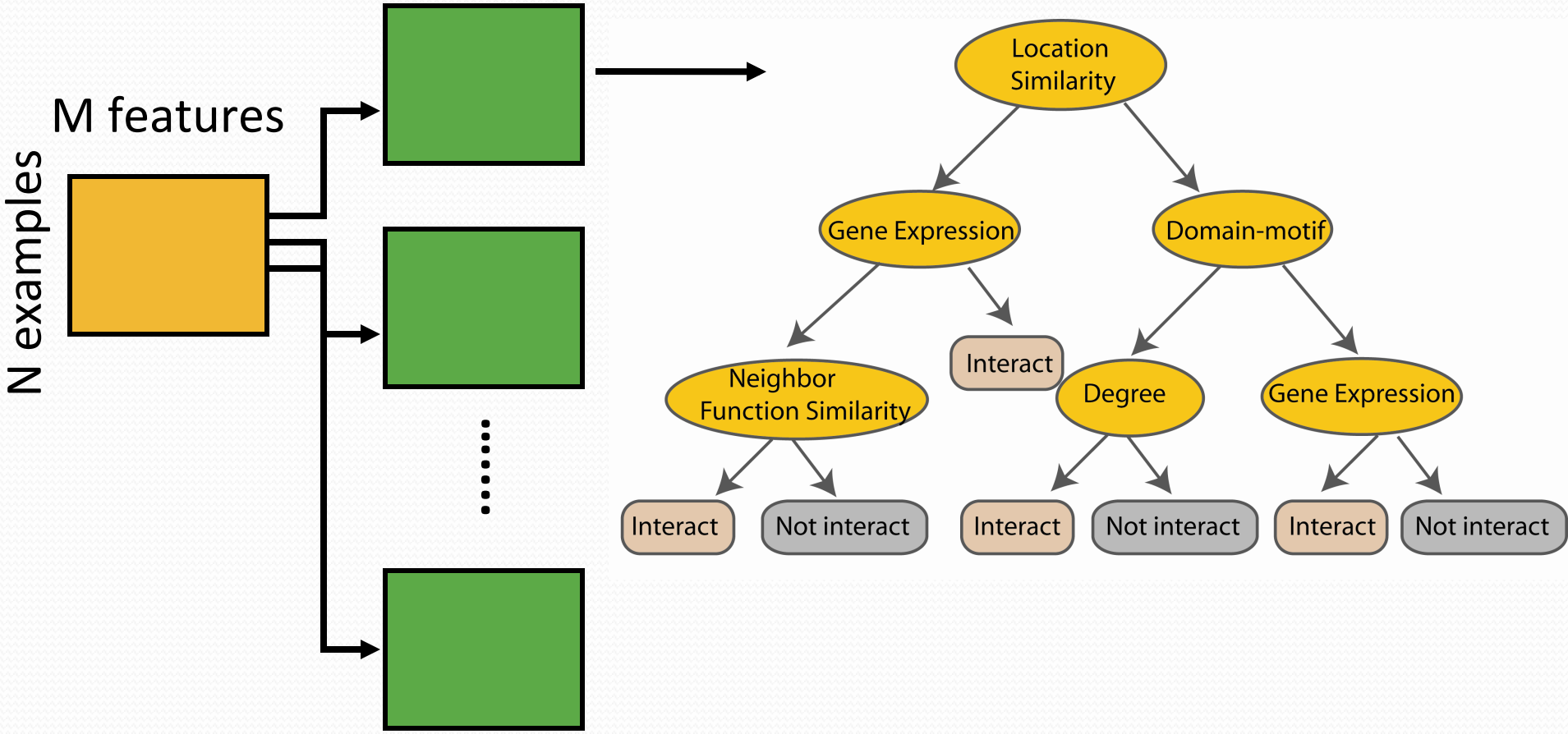
N examples

Create bootstrap samples

from the training data

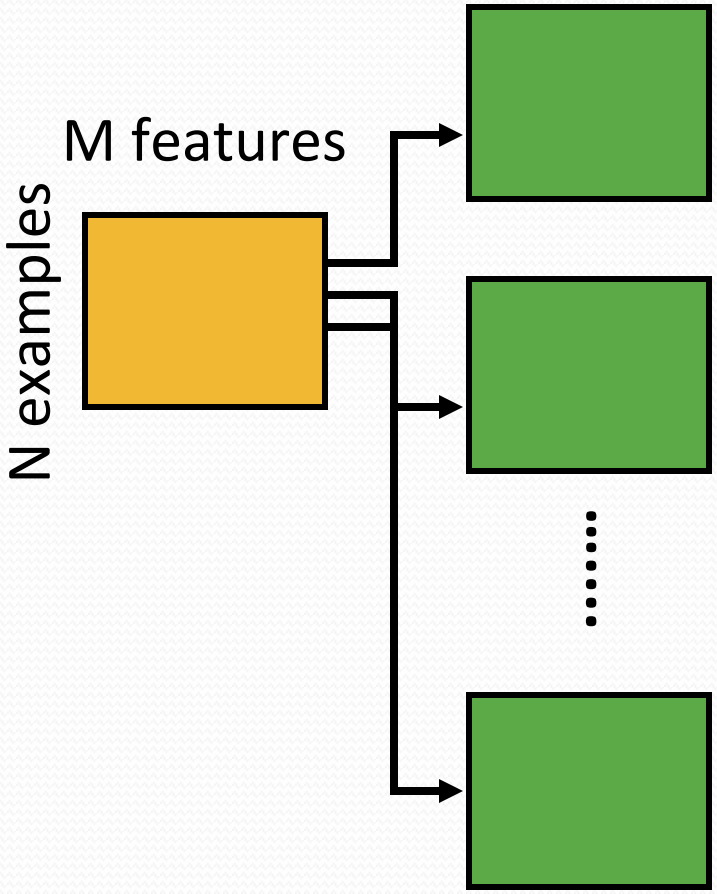


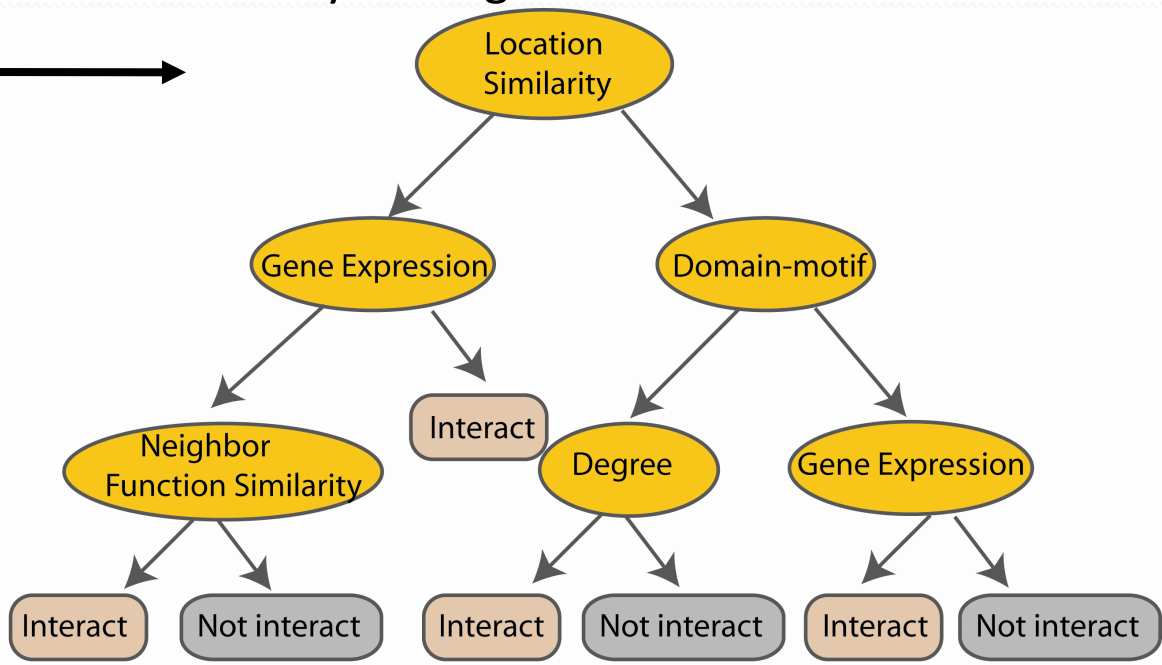
Construct a decision tree



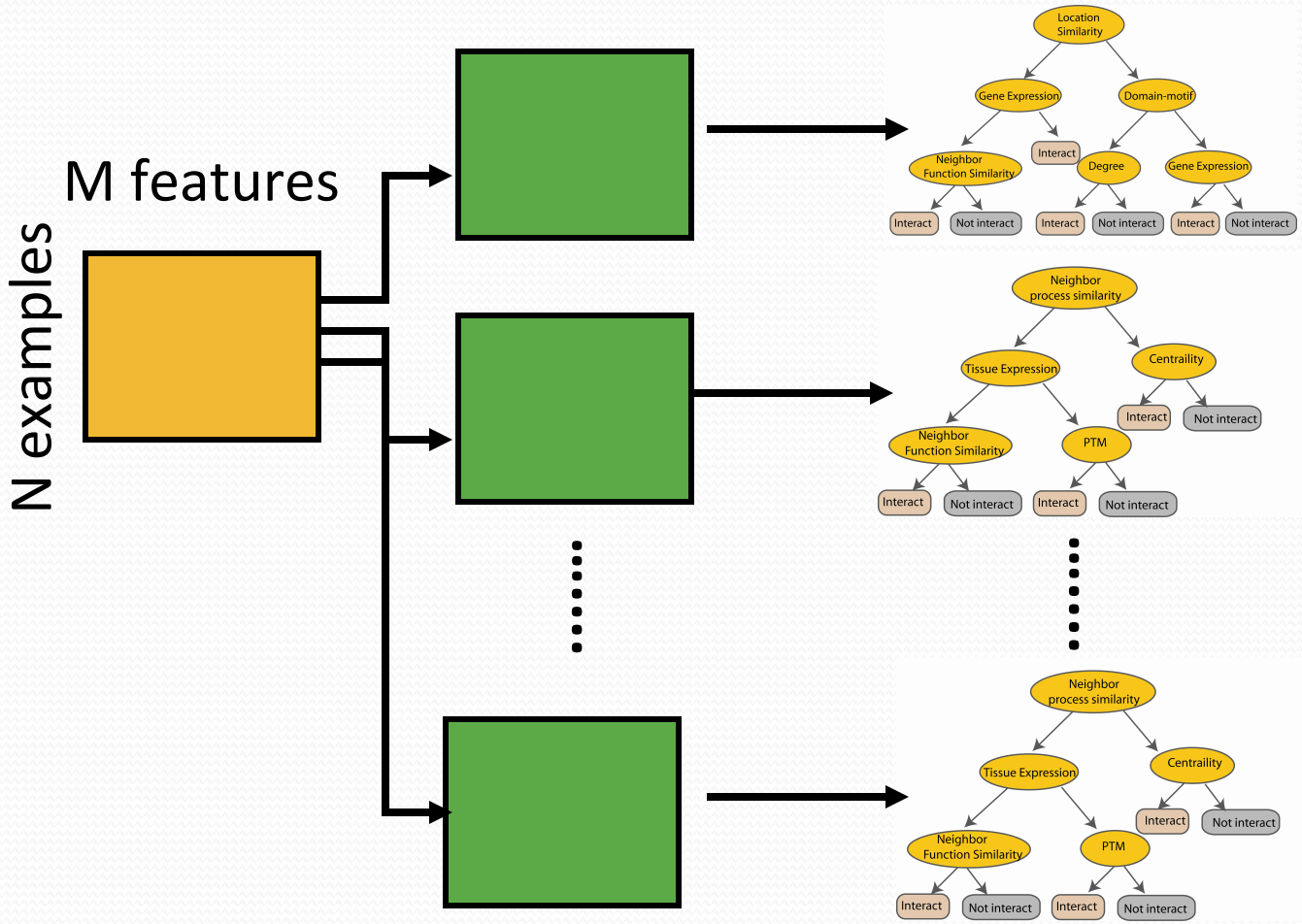
At each node in choosing the split feature

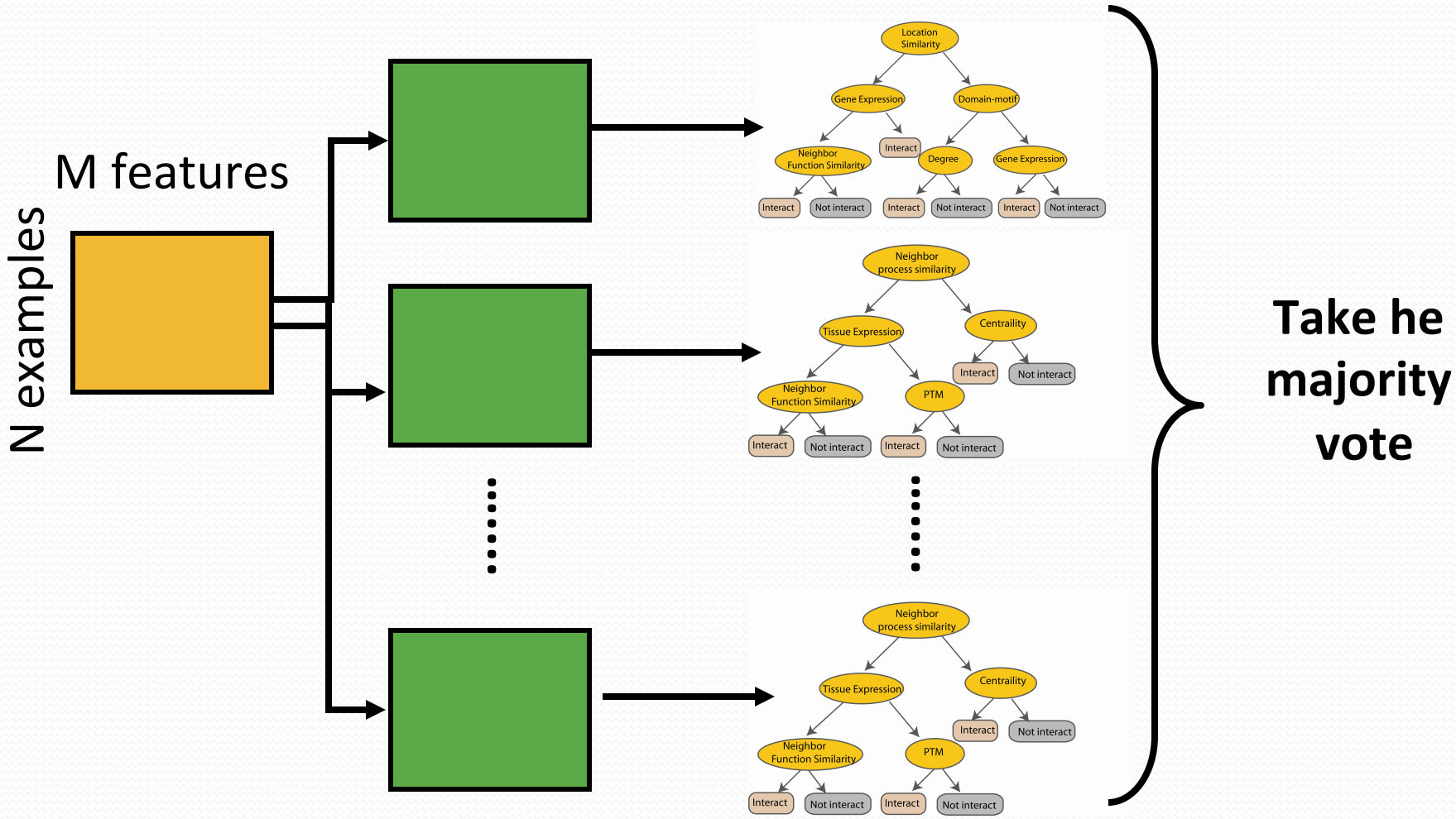
choose only among $m<M$ features

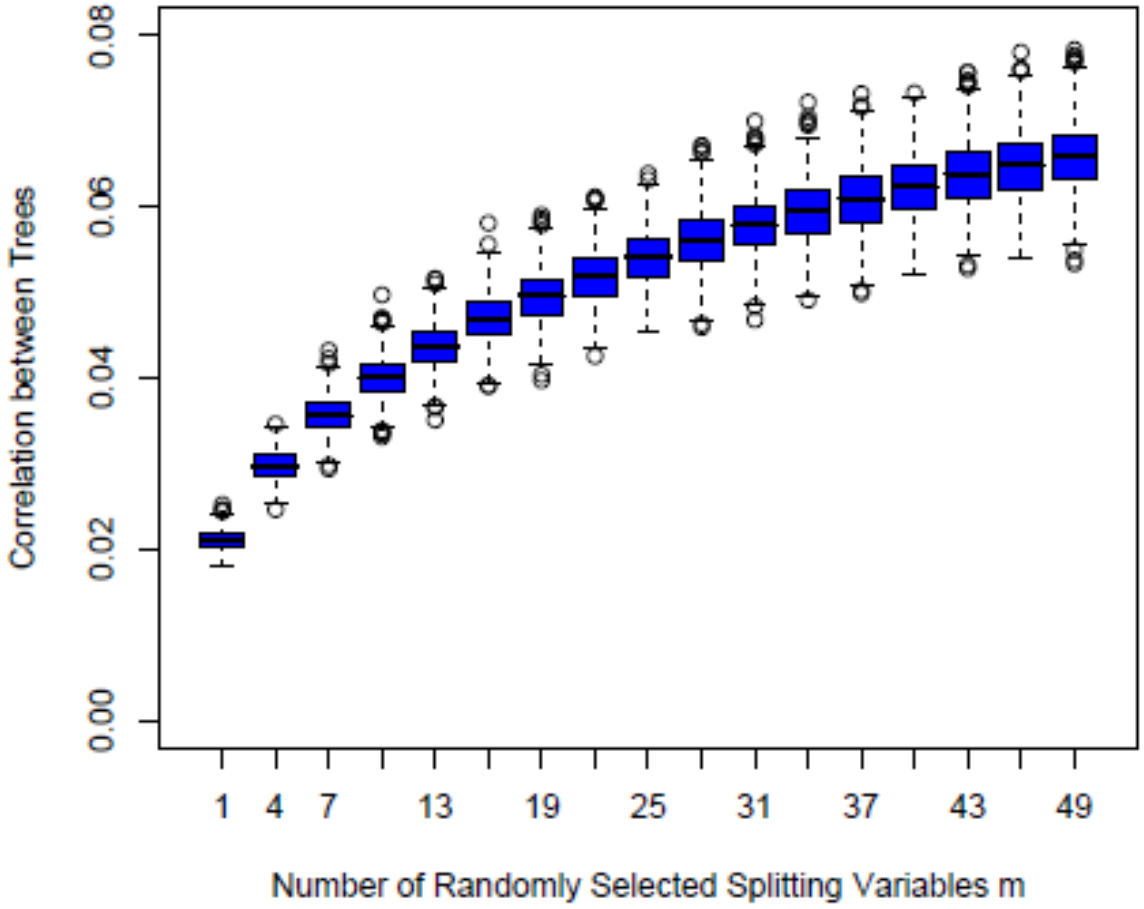




Create decision tree from each bootstrap sample




M features
N examples
Location Similarity
Gene Expression
Domain-motif
Neighbor Function Similarity
Interact
Degree
Gene Expression
Interact
Not interact
Interact
Not interact
Interact
Not interact
Neighbor process similarity
Tissue Expression
Centraility
Interact
Not interact
Neighbor Function Similarity
PTM
Interact
Not interact
Interact
Not interact
Neighbor process similarity
Tissue Expression
Centraility
Interact
Not interact
Neighbor Function Similarity
PTM
Interact
Not interact
Interact
Not interact
Take he majority vote

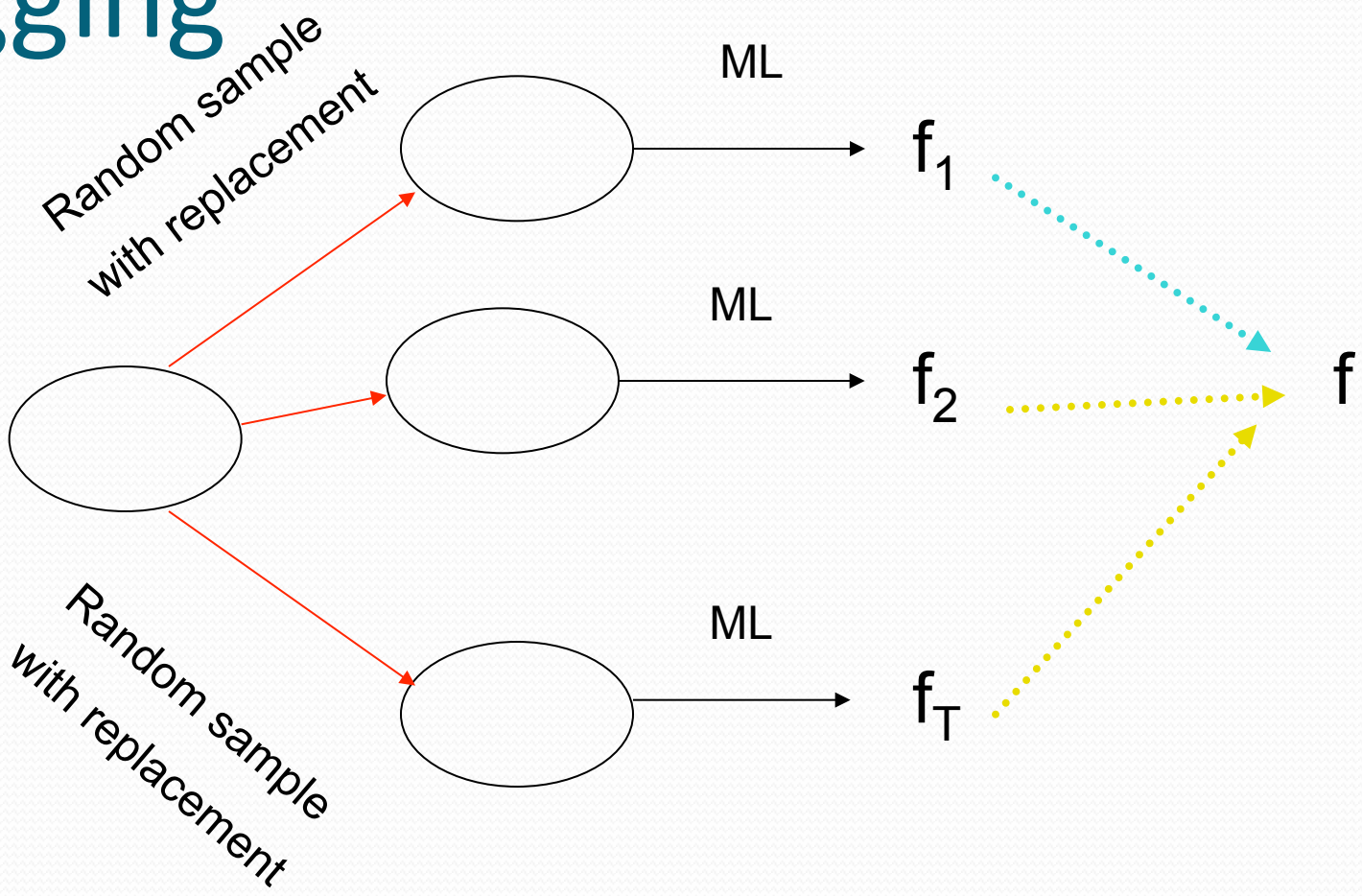

Correlation between Trees
0.00
0.02
0.04
0.06
0.08
1
4
7
13
19
25
31
37
43
49
Number of Randomly Selected Splitting Variables m

Random forest

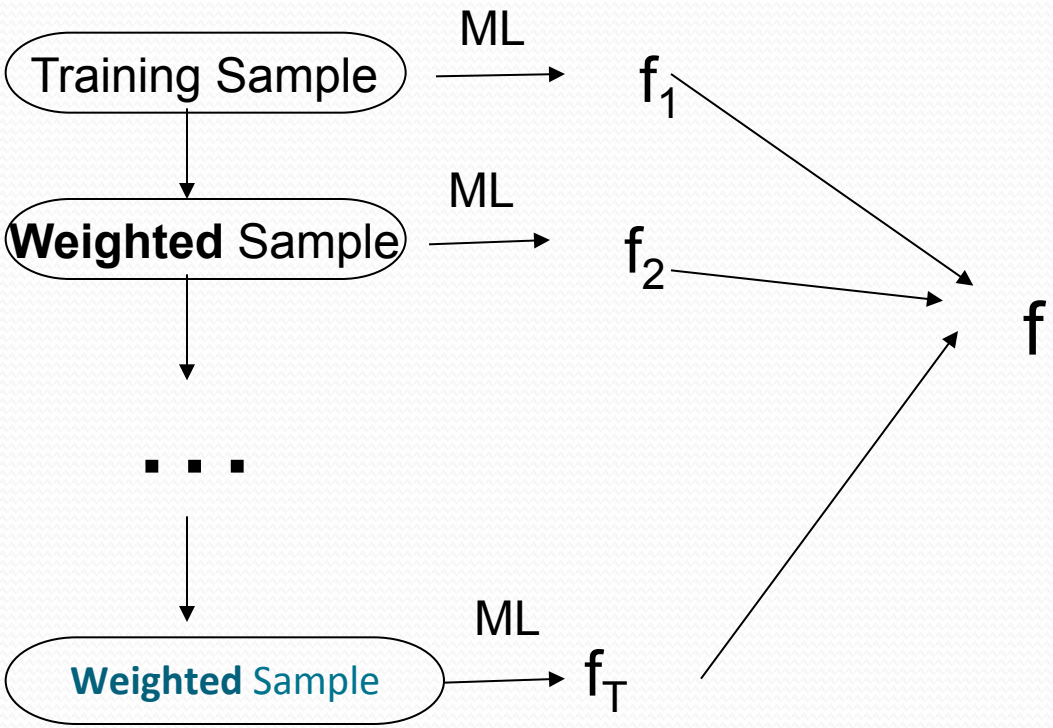
- Available package:
- http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

- To read more:
- http://www-stat.stanford.edu/~hastie/Papers/ESLII.pdf

Bagging


Random sample
with replacement
ML
ML
ML
Random sample
with replacement
f_1
f_2
f_T
f

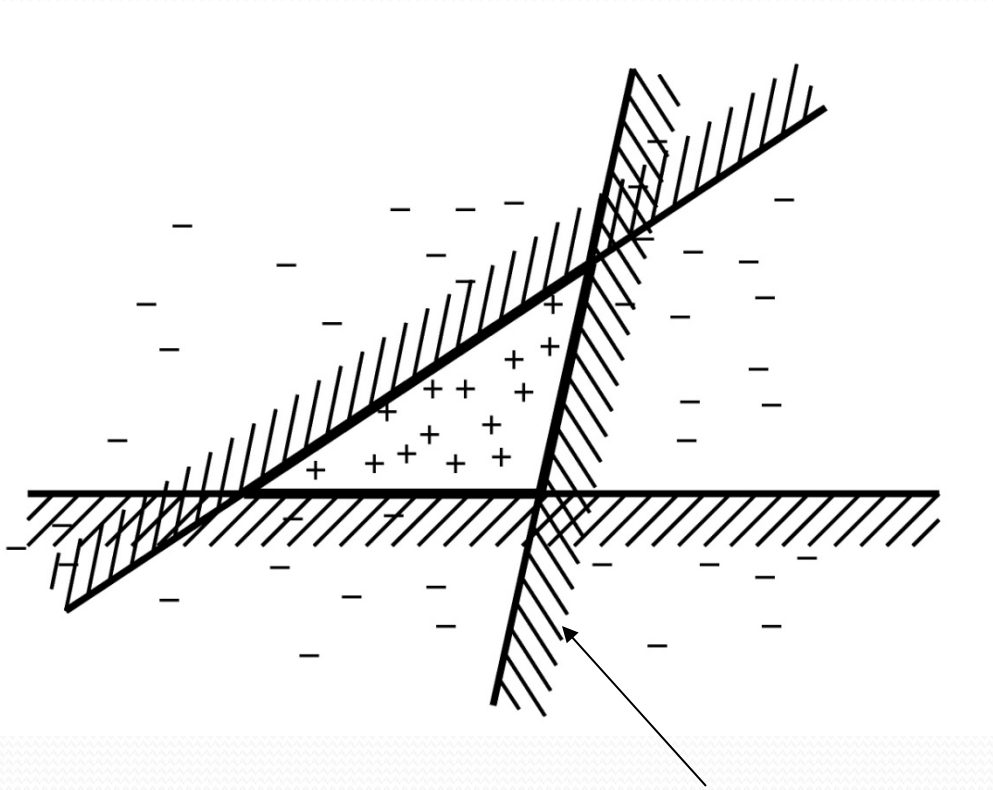
Boosting


Training Sample
ML
f_1
Weighted Sample
ML
f_2
f
Weighted Sample
ML
f_T

Boosting

- Main idea:
  - train classifiers (e.g. decision trees) in a sequence.
  - a new classifier should focus on those cases which were incorrectly classified in the last round.
  - combine the classifiers by letting them vote on the final prediction (like bagging).
  - each classifier could be (should be) very “weak”, e.g. a decision stump.

Example



this line is one simple classifier saying that

everything to the left + and everything to the right is -

[Schapire, 1989]

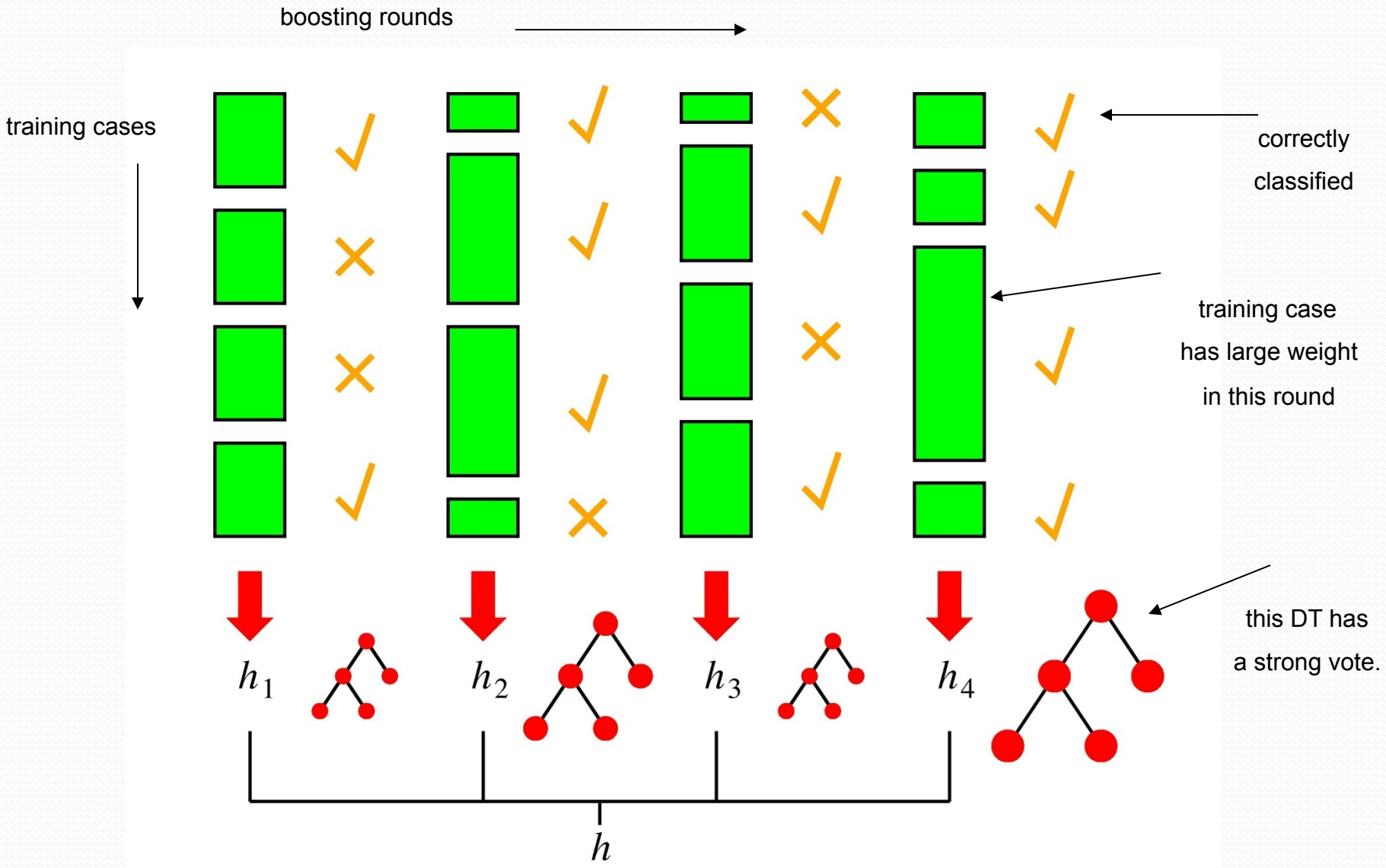
AdaBoost (Adaptive Boosting)

- Idea: given a weak learner, run it multiple times on (reweighted) training data, then let the learned classifiers vote

- On each iteration *t*:
  - weight each training example by how incorrectly it was classified
  - Learn a hypothesis – h_t
  - A strength for this hypothesis – α_t

- Final classifier:
  - A linear combination of the votes of the different classifiers weighted by their strength

Boosting in a Picture



Learning from Weighted Data

- **Sometimes not all data points are equal**
  - Some data points are more equal than others
- **Consider a weighted dataset**
  - D(i) – weight of *i* th training example ($\mathbf{x}^i$,y^i)
  - Interpretations:
    - *i* th training example counts as D(i) examples
    - If I were to "resample" data, I would get more samples of "heavier" data points

- **Now, in all calculations, whenever used, *i* th training example counts as D(i) "examples"**
  - e.g., MLE for Naïve Bayes, redefine *Count(Y=y)* to be weighted count

AdaBoost Algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = 1/m$.

For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

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

where Z_t is a normalization factor

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

Output the final classifier:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

How to choose α_t for h_t?

- For boolean target function [Freund & Schapire'97]:

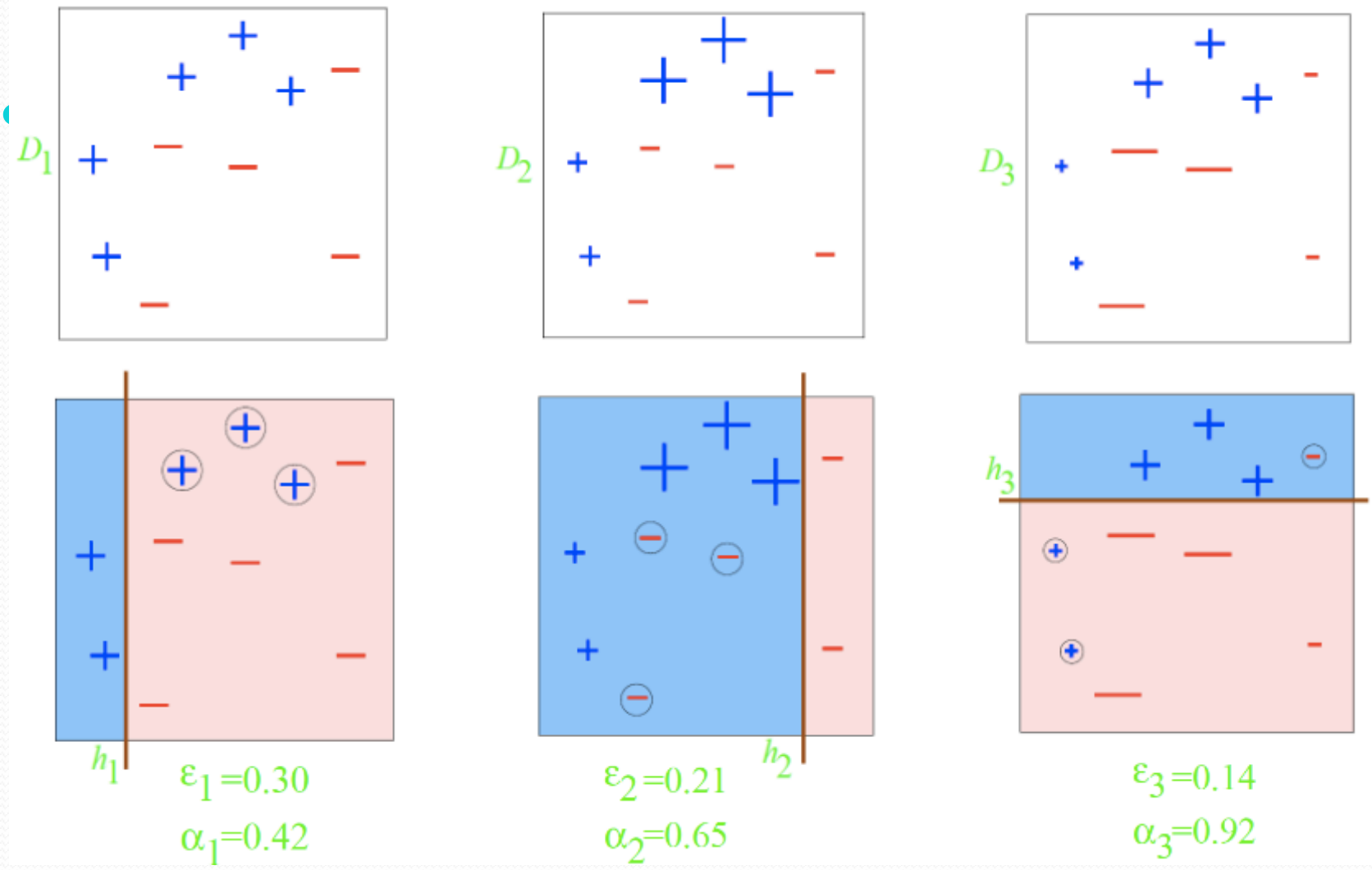
$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

Where:

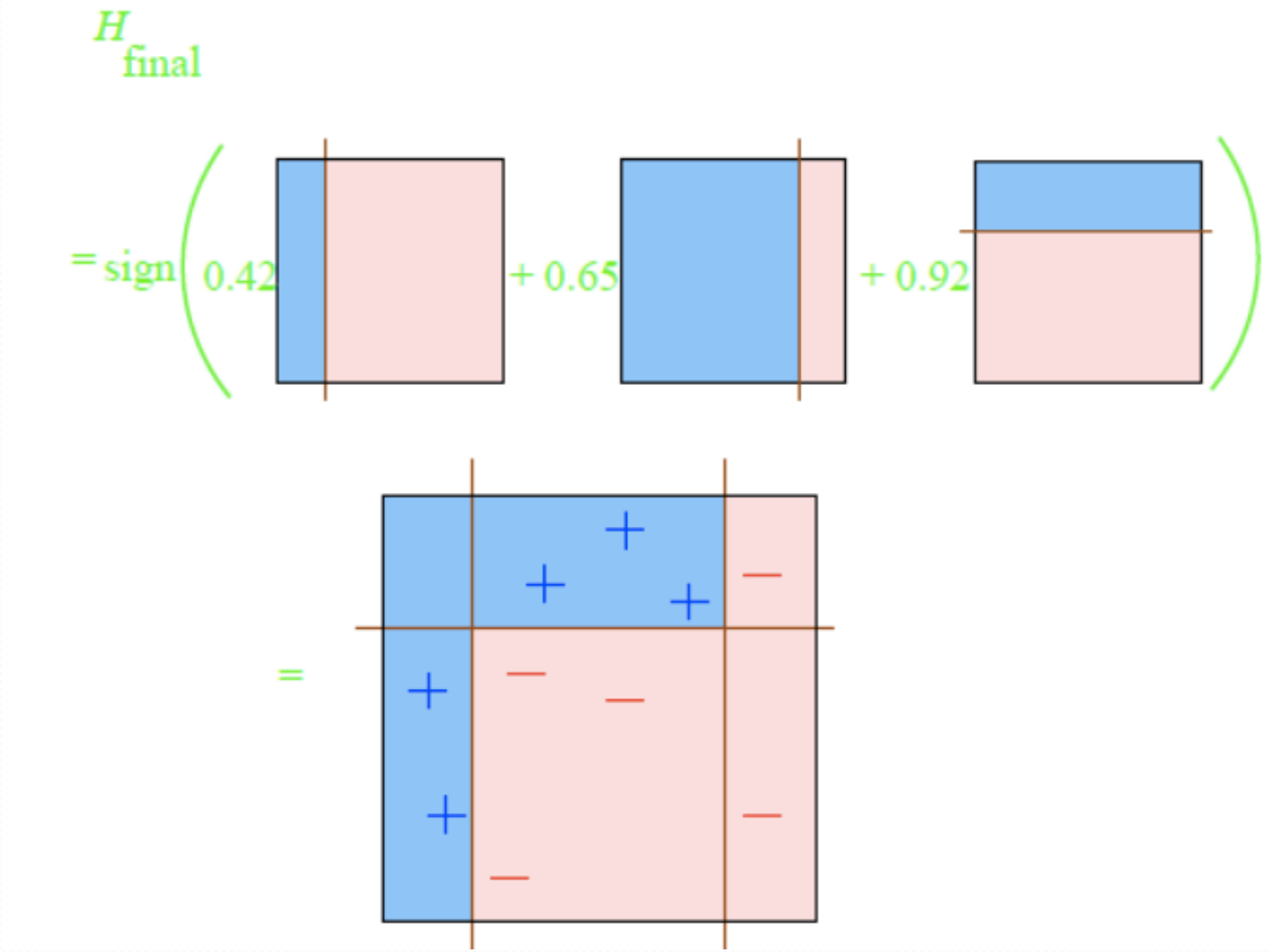
$$\epsilon_t = \sum_{i=1}^{m} D_t(i) \delta(h_t(x_i) \neq y_i)$$

to minimize training error!

Boosting Example (decision stumps)



Boosting Example



Boosting

- Demo
  - http://people.csail.mit.edu/torralba/shortCourseRLOC/boosting/boosting.html

Types of Boosting

Loss Name	Loss Formula	Boosting Name
Regression: Squared Loss	$(y - f(x))^2$	L2Boosting
Regression: Absolute Loss	$\lvert y - f(x) \rvert$	Gradient Boosting
Classification: Exponential Loss	$e^{-yf(x)}$	AdaBoost
Classification: Log/Logistic Loss	$\log\left(1 + e^{-yf(x)}\right)$	LogitBoost

What you should know

- Voting/Ensemble methods
- Bagging
  - How to sample
- Boosting
  - General outline
  - How to adjust weight:
    - Adaboost

Learning Theory

Probably Approximately Correct (PAC) Learning

What does it formally mean to learn?

Learning Theory

- We have explored **many** ways of learning from data

- But…
  - How good is our classifier, really?
  - How much data do I need to make it “good enough”?

A simple setting…

- Classification
  - N data points
  - **Finite** space H of possible hypothesis
    - e.g. dec. trees of depth d

- A learner finds a hypothesis *h* that is **consistent** with training data
  - Gets zero error in training – $error_{train}(h) = 0$

- What is the probability that *h* has more than ε true error?
  - $error_{true}(h) \geq \varepsilon$

Generalization error in finite hypothesis spaces [Haussler '88]

- ***Theorem***:
  - Hypothesis space *H* finite
  - dataset *D* with *N* i.i.d. samples
  - $0 < \varepsilon < 1$

For any learned hypothesis *h* that is consistent on the training data:

$$P(\text{error}_{true}(h) > \epsilon) \leq |H|e^{-N\epsilon}$$

Even if *h* makes zero errors in training data, may make errors in test

Using a PAC bound

$$P(\text{error}_{true}(h) > \epsilon) \leq |H|e^{-N\epsilon}$$

- Typically, 2 use cases:
  - 1: Pick ε and δ, give you *N*
  - 2: Pick N and δ, give you ε

Haussler '88 bound

$$P(\text{error}_{true}(h) > \epsilon) \leq |H| e^{-N\epsilon}$$

- Strengths:
  - Holds for all (finite) H
  - Holds for all data distributions
- Weaknesses
  - Consistent classifier
  - Finite hypothesis space

Generalization bound for |H| hypothesis

- ***Theorem***:
  - Hypothesis space *H* finite
  - dataset *D* with *N* i.i.d. samples
  - $0 < \varepsilon < 1$

For any learned hypothesis *h*:

$$P\left(\text{error}_{true}(h) - \text{error}_{train}(h) > \epsilon\right) \leq |H| e^{-2N\epsilon^2}$$

PAC bound and Bias-Variance tradeoff

$$P\left(\text{error}_{true}(h) - \text{error}_{train}(h) > \epsilon\right) \leq |H| e^{-2N\epsilon^2}$$

or, after moving some terms around, with probability at least 1-δ:

$$\text{error}_{true}(h) \leq \text{error}_{train}(h) + \sqrt{\frac{\ln |H| + \ln \frac{1}{\delta}}{2N}}$$

Important: PAC bound holds for all *h*,

but doesn't guarantee that algorithm finds best *h*!!!