COGS 118A, Winter 2020

Supervised Machine Learning Algorithms

Lecture 17: Boosting

Zhuowen Tu

Final project guideline provided.

Ensemble Methods

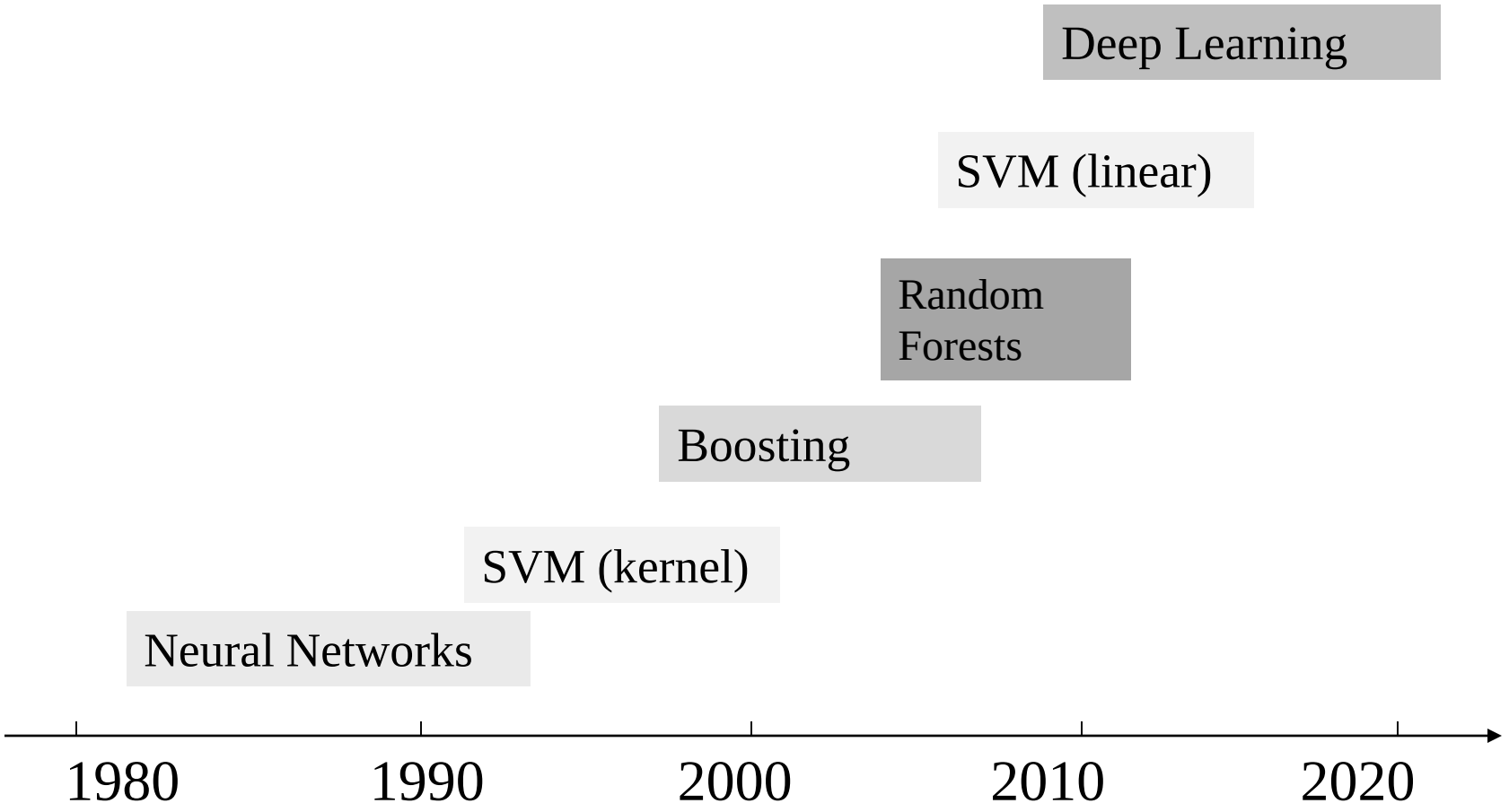
Bagging (Breiman 1994,…)

Random forests (Breiman 2001,…)

Boosting (Freund and Schapire 1995, Friedman et al. 1998,…)

Predict class label for unseen data by aggregating a set of predictions (classifiers learned from the training data).

Trends of classification methods



Empirical Comparisons of Different Algorithms

Caruana and Niculesu-Mizil, ICML 2006

MODEL	1ST	2ND	3RD	4TH	5TH	6TH	7TH	8TH	9TH	10TH
BST-DT	0.580	0.228	0.160	0.023	0.009	0.000	0.000	0.000	0.000	0.000
RF	0.390	0.525	0.084	0.001	0.000	0.000	0.000	0.000	0.000	0.000
BAG-DT	0.030	0.232	0.571	0.150	0.017	0.000	0.000	0.000	0.000	0.000
SVM	0.000	0.008	0.148	0.574	0.240	0.029	0.001	0.000	0.000	0.000
ANN	0.000	0.007	0.035	0.230	0.606	0.122	0.000	0.000	0.000	0.000
KNN	0.000	0.000	0.000	0.009	0.114	0.592	0.245	0.038	0.002	0.000
BST-STMP	0.000	0.000	0.002	0.013	0.014	0.257	0.710	0.004	0.000	0.000
DT	0.000	0.000	0.000	0.000	0.000	0.000	0.004	0.616	0.291	0.089
LOGREG	0.000	0.000	0.000	0.000	0.000	0.000	0.040	0.312	0.423	0.225
NB	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.030	0.284	0.686

Overall rank by mean performance across problems and metrics (based on bootstrap analysis).

BST-DT: boosting with decision tree weak classifier

BAG-DT: bagging with decision tree weak classifier

ANN: neural nets

BST-STMP: boosting with decision stump weak classifier

LOGREG: logistic regression

RF: random forest

SVM: support vector machine

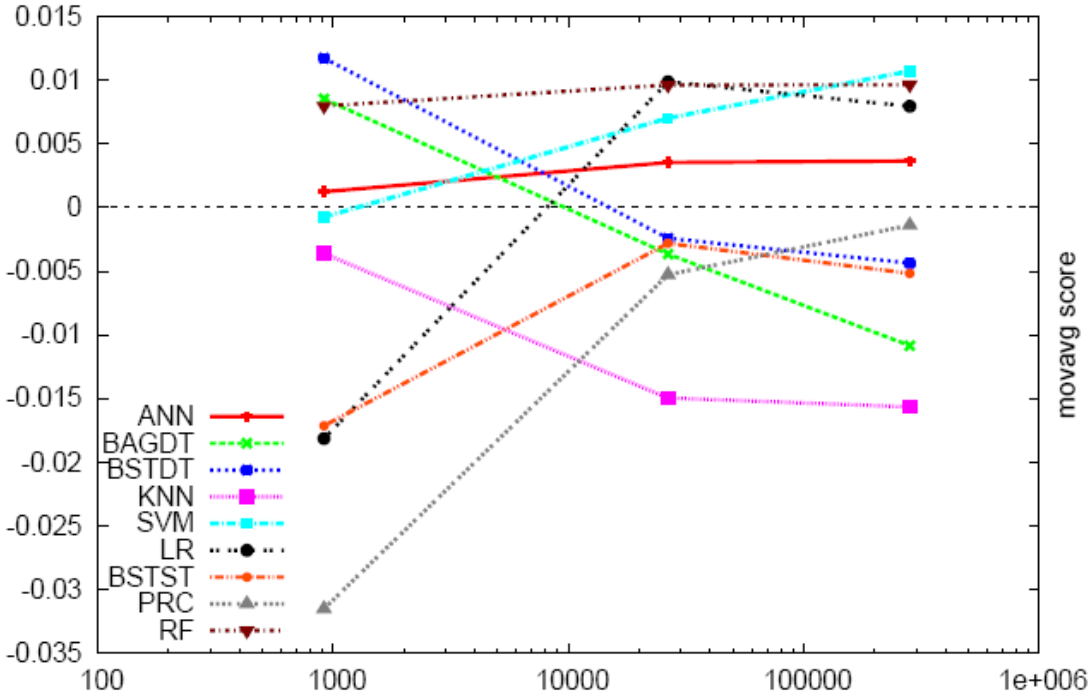
KNN: k nearest neighboorhood

DT: decision tree

NB: naïve Bayesian

It is informative, but by no means final.

Empirical Study on High-dimension



Caruana et al., ICML 2008

Moving average standardized scores of each learning algorithm as a function of the dimension.

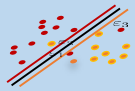
The rank for the algorithms to perform consistently well:

(1) random forest (2) neural nets (3) boosted tree (4) SVMs

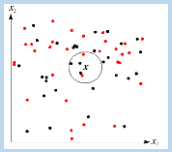
Sampling with replacement

h

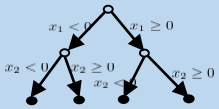
support vector machine



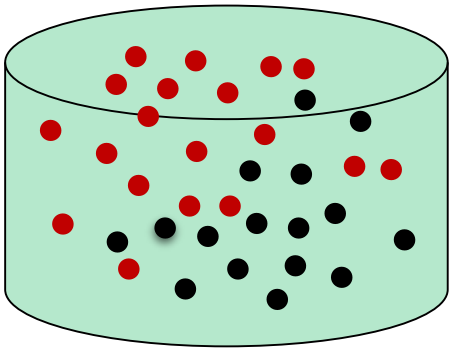
nearest neighborhood



decision tree

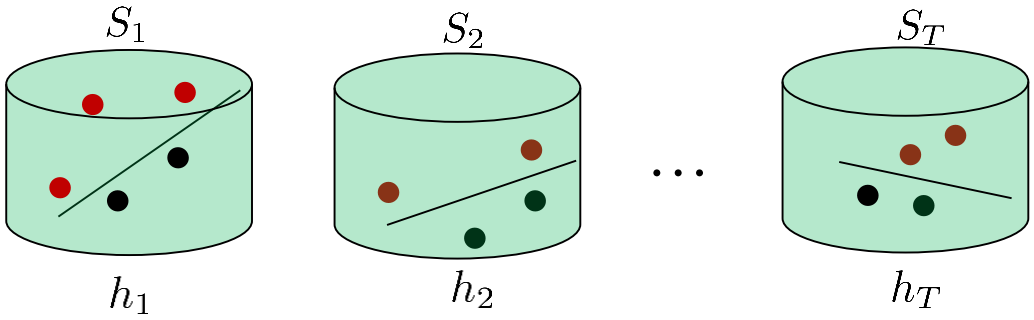


$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$



$y = +1$

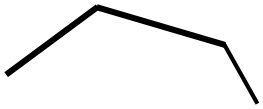
$y = -1$



α_1 α_2 α_T

$$H(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t \times h_t(\mathbf{x})$$

$h_t(\mathbf{x}) \in \{-1, +1\}$



Toy Example

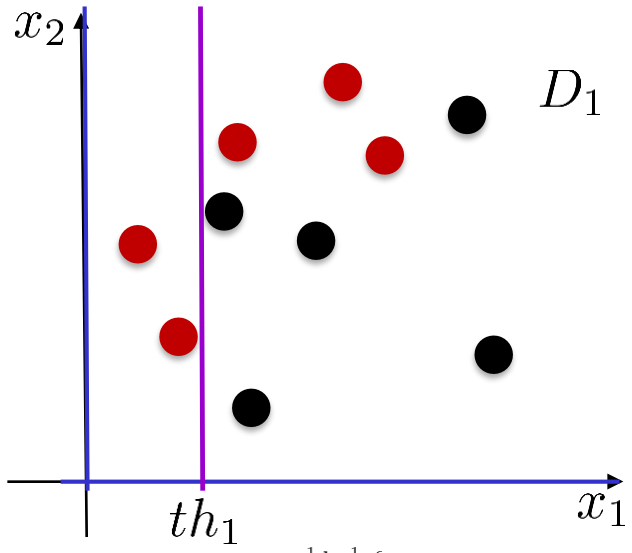
$$S_{training} = \{(\mathbf{x}_i, D_1(i), y_i), i = 1..n\} \qquad \mathbf{x} \in \mathbb{R}^2 \qquad y \in \{-1, +1\}$$

weight (importance) for each sample

$y = +1$

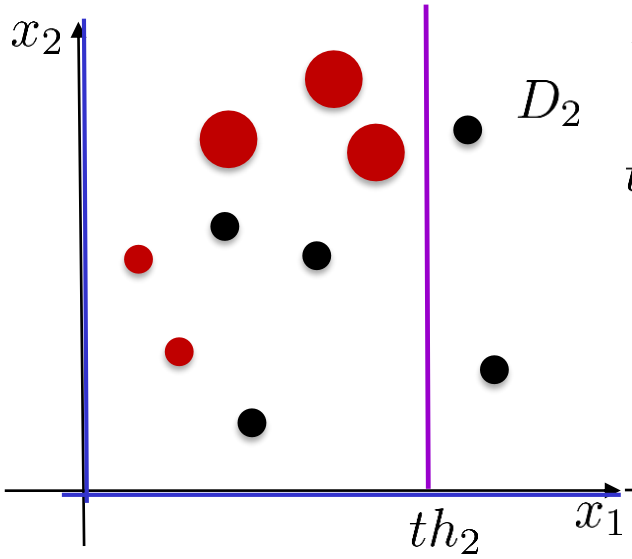
$y = -1$

Minimize: $e_t = \sum_i D_t(i) \times \mathbf{1}(y_i \neq h_t(\mathbf{x}_i))$



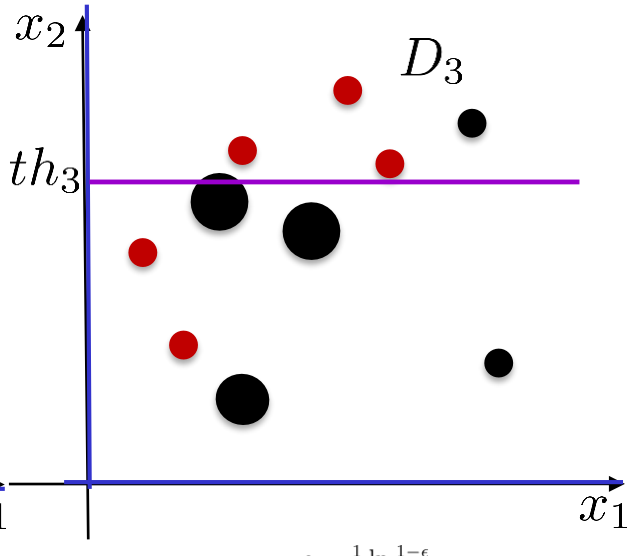
$\epsilon_1 = 3/9 \xrightarrow{\alpha = \frac{1}{2}\ln\frac{1-\epsilon}{\epsilon}} \alpha_1 = 0.42$

$h_1(\mathbf{x}) = sign(x_1 \leq th_1)$



$\epsilon_2 = 0.215 \xrightarrow{\alpha = \frac{1}{2}\ln\frac{1-\epsilon}{\epsilon}} \alpha_2 = 0.63$

$h_2(\mathbf{x}) = sign(x_1 \leq th_2)$



$\epsilon_3 = 0.137 \xrightarrow{\alpha = \frac{1}{2}\ln\frac{1-\epsilon}{\epsilon}} \alpha_3 = 0.92$

$h_3(\mathbf{x}) = sign(x_2 \geq th_3)$

The final classifier: $H(\mathbf{x}) = sign(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}))$

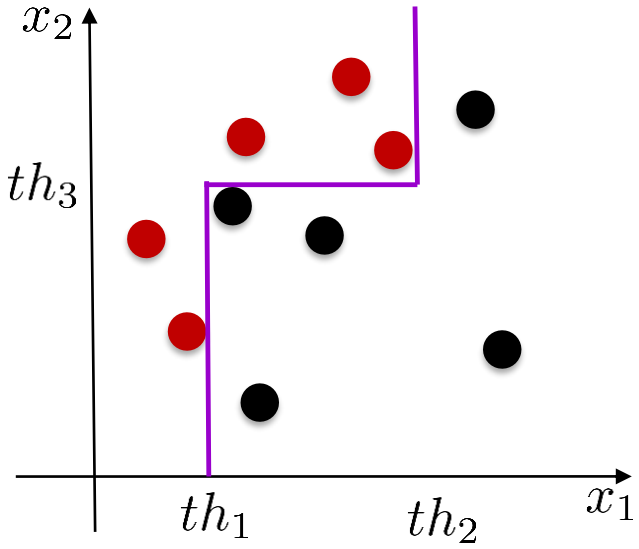
Toy Example

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\} \qquad \mathbf{x} \in \mathbb{R}^2 \qquad y \in \{-1, +1\}$$

$y = +1$

$y = -1$

Weak classifier: h decision stump.



$$H(\mathbf{x}) = 0.42 \times h_1(\mathbf{x}) + 0.65 \times h_2(\mathbf{x}) + 0.92 \times h_3(\mathbf{x})$$

$$= 0.42 \times sign(x_1 \leq th_1) + 0.65 \times sign(x_1 \leq th_2) + 0.92 \times sign(x_2 \geq th_3)$$

A Formal Description of Boosting

- given training set $(x_1, y_1), \ldots, (x_m, y_m)$
- $y_i \in \{-1, +1\}$ correct label of instance $x_i \in X$
- for $t = 1, \ldots, T$:
  - construct distribution D_t on $\{1, \ldots, m\}$
  - find weak classifier ("rule of thumb")

    $h_t : X \to \{-1, +1\}$

    with small error ϵ_t on D_t:

    $\epsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$
- output final classifier H_{final}

AdaBoost (Freund and Schpaire)

- constructing D_t:
  - $D_1(i) = 1/m$ (not necessarily with equal weight)
  - given D_t and h_t:

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

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

where Z_t = normalization constant

$$\alpha_t = \tfrac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) > 0$$

- final classifier:
  - $H_{\text{final}}(x) = \text{sign}\left(\sum_t \alpha_t h_t(x)\right)$

R. Schapire

$\frac{_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$

Toy Example

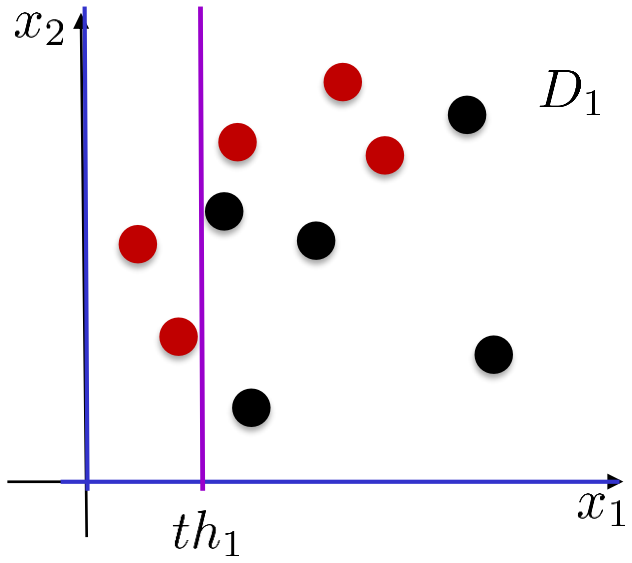
$$S_{training} = \{(\mathbf{x}_i, D_1(i), y_i), i = 1..n\} \qquad \mathbf{x} \in \mathbb{R}^2 \qquad y \in \{-1, +1\}$$

weight) (importance) for each sample

Minimize: $e_t = \sum_i D_t(i) \times \mathbf{1}(y_i \neq h_t(\mathbf{x}_i))$

$y = +1$

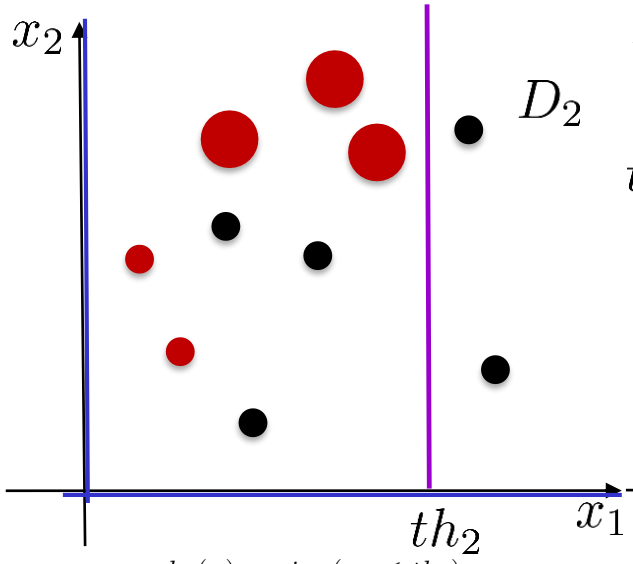
$y = -1$



$h_1(\mathbf{x}) = sign(x_1 \leq th_1)$

$\alpha_1 = \frac{1}{2}\ln(\frac{1-e_1}{e_1}) = 0.42$

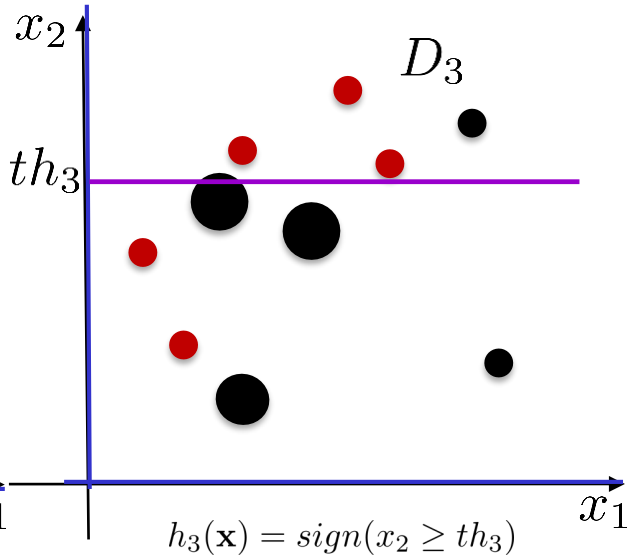
$D_2(i) \propto D_1(i) \times \begin{cases} e^{-\alpha_1} & if\ y_i = h_1(\mathbf{x}_i) \\ e^{\alpha_1} & if\ y_i \neq h_1(\mathbf{x}_i) \end{cases}$



$h_2(\mathbf{x}) = sign(x_1 \leq th_2)$

$\alpha_2 = \frac{1}{2}\ln(\frac{1-e_2}{e_2}) = 0.65$

$D_3(i) \propto D_2(i) \times \begin{cases} e^{-\alpha_2} & if\ y_i = h_2(\mathbf{x}_i) \\ e^{\alpha_2} & if\ y_i \neq h_2(\mathbf{x}_i) \end{cases}$



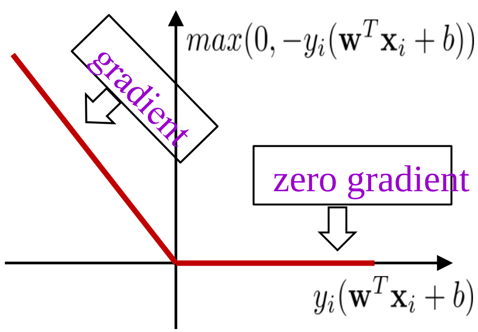
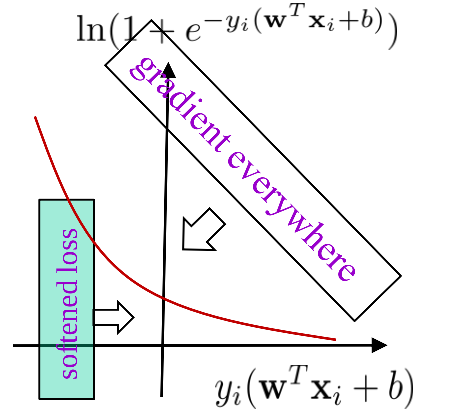
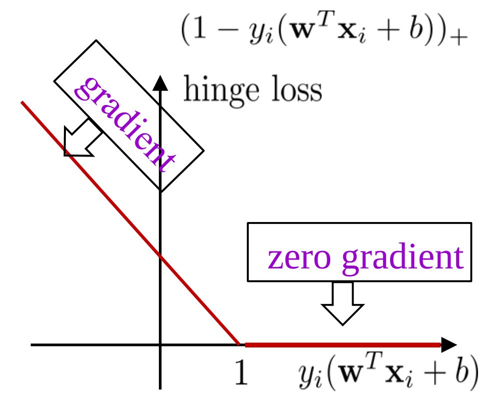
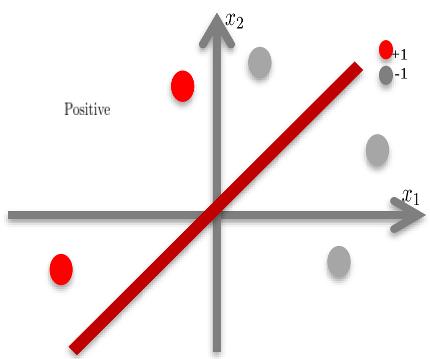
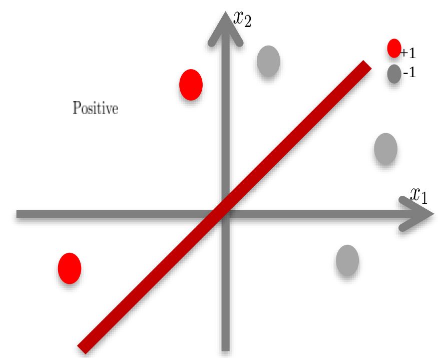
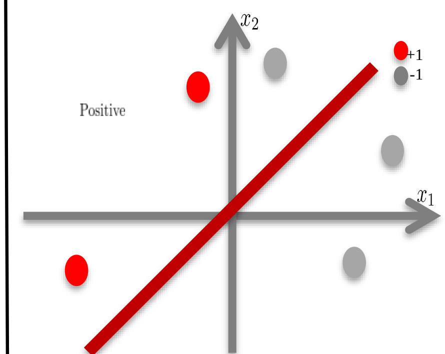
$h_3(\mathbf{x}) = sign(x_2 \geq th_3)$

$\alpha_3 = \frac{1}{2}\ln(\frac{1-e_3}{e_3}) = 0.92$

$D_4(i) \propto D_3(i) \times \begin{cases} e^{-\alpha_3} & if\ y_i = h_3(\mathbf{x}_i) \\ e^{\alpha_3} & if\ y_i \neq h_3(\mathbf{x}_i) \end{cases}$

The final classifier: $H(\mathbf{x}) = sign(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}))$

A Summary

	Perceptron	Logistic Regression	SVM
Training	Training: Minimize $\mathcal{L}(\mathbf{w}, b) = \sum_i max(0, -y_i(\mathbf{w}^T\mathbf{x}_i + b))$  convex optimization	Training: Minimize $\mathcal{L}(\mathbf{w}, b) = \sum_{i=1}^{n} \ln(1 + e^{-y_i(\mathbf{w}^T\mathbf{x}_i + b)})$  convex optimization	Training: Minimize $\mathcal{L}(\mathbf{w}, b) = \frac{1}{2}\|\|\mathbf{w}\|\|^2$ $+C\sum_{i=1}^{n}(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))_+$  convex optimization
Testing	 Test: $f(\mathbf{x}; \mathbf{w}, b) = \begin{cases} +1 & if\ \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & otherwise \end{cases}$	 Test: $f(\mathbf{x}; \mathbf{w}, b) = \begin{cases} +1 & if\ \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & otherwise \end{cases}$	 Test: $f(\mathbf{x}; \mathbf{w}, b) = \begin{cases} +1 & if\ \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & otherwise \end{cases}$

Main motivation

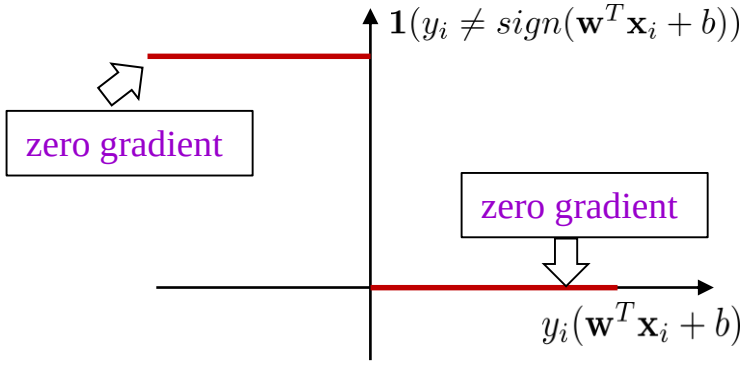
Hard->Half-hard->Soft

Error

Standard loss (error) function

Standard 0/1 loss (gradient 0 nearly everywhere, no gradient feedback):

Training: Minimize $\mathcal{L}(\mathbf{w}, b) = \sum_i \mathbf{1}(y_i \neq sign(\mathbf{w}^T \mathbf{x}_i + b))$



It is the most directly loss, but is also the hardest to minimize.

Zero gradient everywhere!

Motivation for boosting

$$H(\mathbf{x}) = sign(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}))$$

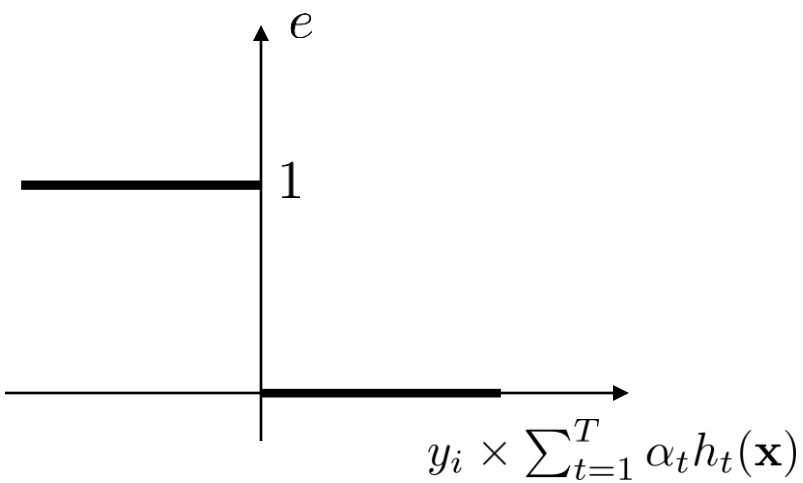
1. Turn direct error measure into an approximation

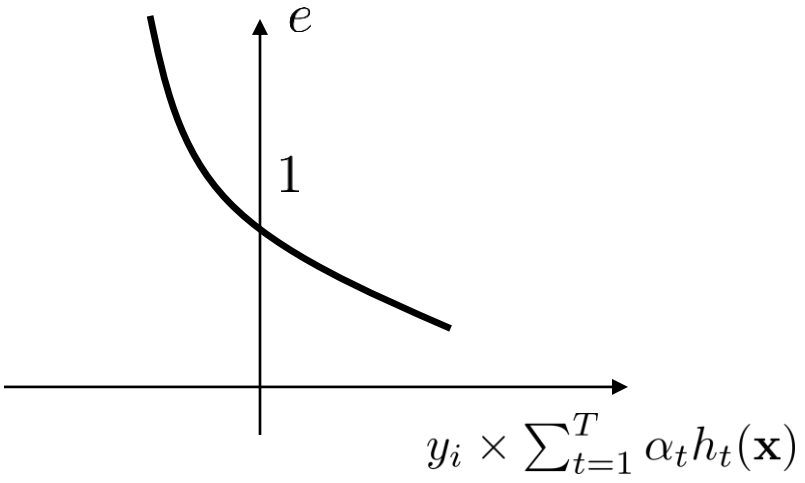
$$e = \sum_i \mathbf{1}(y_i \neq H(\mathbf{x}_i))$$

$$e = \sum_i \mathbf{1}(y_i \times \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) < 0)$$

$$\downarrow$$

$$e = \sum_i \exp\{-y_i \times \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}_i)\}$$





Motivation for boosting

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}))$$

$$e = \sum_i \exp\{-y_i \times \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}_i)\}$$

2. Coordinate descent to minimize the error

$$e = \sum_i \exp\{-y_i \times \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}_i) - y_i \times \alpha_{T+1} h_{T+1}(\mathbf{x}_i)\}$$

$$e = \sum_i D_T \exp\{-y_i \times \alpha_{T+1} h_{T+1}(\mathbf{x}_i)\}$$

Motivation for boosting

$$e = \sum_i D_T^{(i)} \exp\{-y_i \times \alpha_{T+1} h_{T+1}(\mathbf{x}_i)\}$$

2. Coordinate descent to minimize the error

$$h_{T+1}^* = \arg min_{h_{T+1}} \sum_i D_T^{(i)} \exp\{-y_i \times h_{T+1}(\mathbf{x}_i)\}$$

Enumerate all the features and choose the best one to minimize the error.

$$\alpha_{T+1}^* = \arg min_{\alpha_{T+1}} \sum_i D_T^{(i)} \exp\{-y_i \times \alpha_{T+1} \times h_{T+1}^*(\mathbf{x}_i)\}$$

By taking the derivative to set to zero.

Can the training error of a boosting classifier be 0?

A. Yes, and almost always.

B. Yes, but not very often.

C. No, unless under special situations.

D. Not at all.

Training Error

- Theorem:
  - write ϵ_t as $1/2 - \gamma_t$
  - then

$$\begin{aligned} \text{training error}(H_{\text{final}}) &\leq \prod_t \left[2\sqrt{\epsilon_t(1-\epsilon_t)}\right] \\ &= \prod_t \sqrt{1-4\gamma_t^2} \\ &\leq \exp\left(-2\sum_t \gamma_t^2\right) \end{aligned}$$

- so: if $\forall t: \ \gamma_t \geq \gamma > 0$
  then $\text{training error}(H_{\text{final}}) \leq e^{-2\gamma^2 T}$

- AdaBoost is adaptive:
  - does not need to know γ or T a priori
  - can exploit $\gamma_t \gg \gamma$

Training Error

- *Step 2*: training error$(H_{\text{final}}) \leq \prod_t Z_t$
- Proof:

$$\begin{aligned}
\text{training error}(H_{\text{final}}) &= \frac{1}{m}\sum_i \begin{cases} 1 & \text{if } y_i \neq H_{\text{final}}(x_i) \\ 0 & \text{else} \end{cases} \\
&= \frac{1}{m}\sum_i \begin{cases} 1 & \text{if } y_i f(x_i) \leq 0 \\ 0 & \text{else} \end{cases} \\
&\leq \frac{1}{m}\sum_i \exp(-y_i f(x_i)) \\
&= \sum_i D_{\text{final}}(i) \prod_t Z_t \\
&= \prod_t Z_t
\end{aligned}$$

Training Error

- *Step 3*: $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$
- Proof:

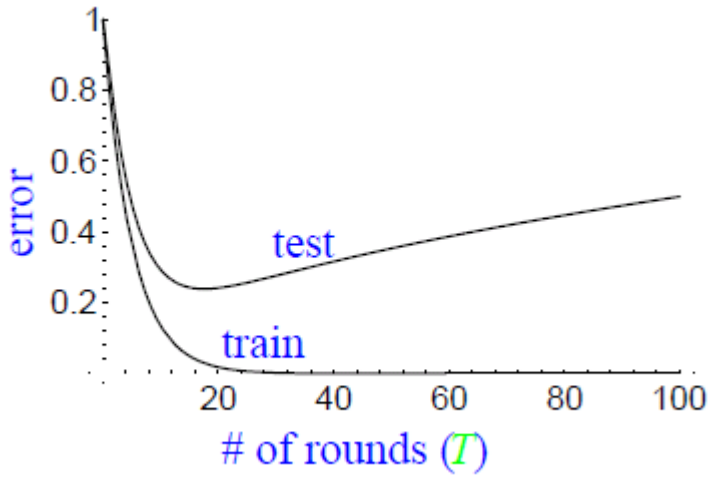
$$
\begin{aligned}
Z_t &= \sum_i D_t(i) \exp(-\alpha_t \, y_i \, h_t(x_i)) \\
&= \sum_{i:y_i \neq h_t(x_i)} D_t(i) e^{\alpha_t} + \sum_{i:y_i = h_t(x_i)} D_t(i) e^{-\alpha_t} \\
&= \epsilon_t \, e^{\alpha_t} + (1-\epsilon_t) \, e^{-\alpha_t} \\
&= 2\sqrt{\epsilon_t(1-\epsilon_t)}
\end{aligned}
$$

$$\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$$

Test Error?

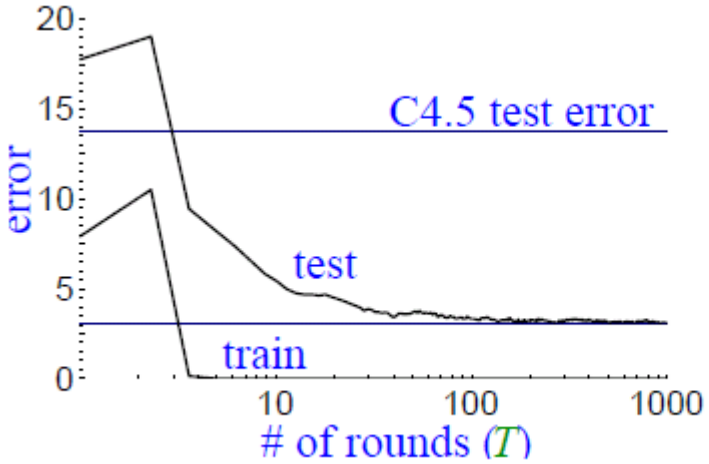
$$e_{testing} \leq e_{training} + bound(generalization(H))$$

expect:



- training error to continue to drop (or reach zero)
- test error to increase when H_{final} becomes "too complex"
  - "Occam's razor"
  - overfitting
    - hard to know when to stop training

Test Error



(boosting C4.5 on "letter" dataset)

- test error does not increase, even after 1000 rounds
  - (total size > 2,000,000 nodes)
- test error continues to drop even after training error is zero!

	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1

- Occam's razor wrongly predicts "simpler" rule is better

Margin Analysis

- Theorem: large margins $\Rightarrow$ better bound on generalization error (independent of number of rounds)
  - proof idea: if all margins are large, then can approximate final classifier by a much smaller classifier (just as polls can predict not-too-close election)

- Theorem: boosting tends to increase margins of training examples (given weak learning assumption)
  - proof idea: similar to training error proof

- so:
  although final classifier is getting larger,
  margins are likely to be increasing,
  so final classifier actually getting close to a simpler classifier,
  driving down the test error

Random Forest compared with Boosting

Pros:

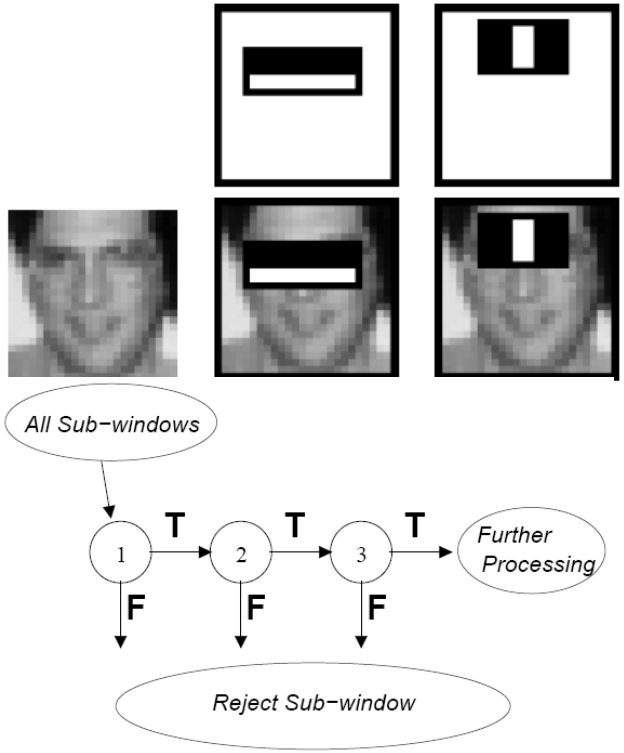
- It is more robust.
- It is faster to train (no reweighting, each split is on a small subset of data and feature).
- Can handle missing/partial data.
- Is easier to extend to online version.

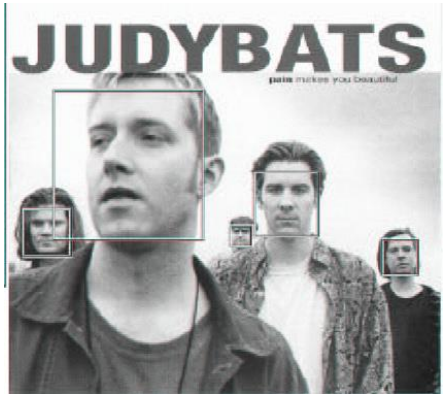
Cons:

- The feature selection process is not explicit.
- Feature fusion is also less obvious.
- Has weaker performance on small size training data.

Face Detection

Viola and Jones 2001





A landmark paper in vision!

1. A large number of Haar features.
2. Use of integral images.
3. Cascade of classifiers.
4. Boosting.

HOG, part-based..

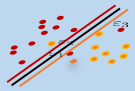
RF, SVM, PBT, NN

All the components can be replaced now.

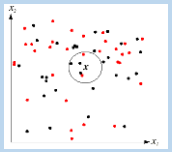
Sampling with replacement

h

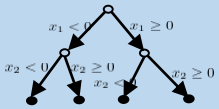
support vector machine



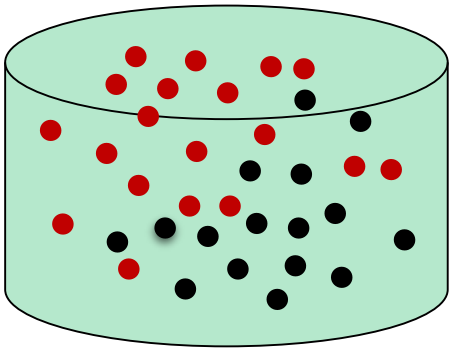
nearest neighborhood



decision tree

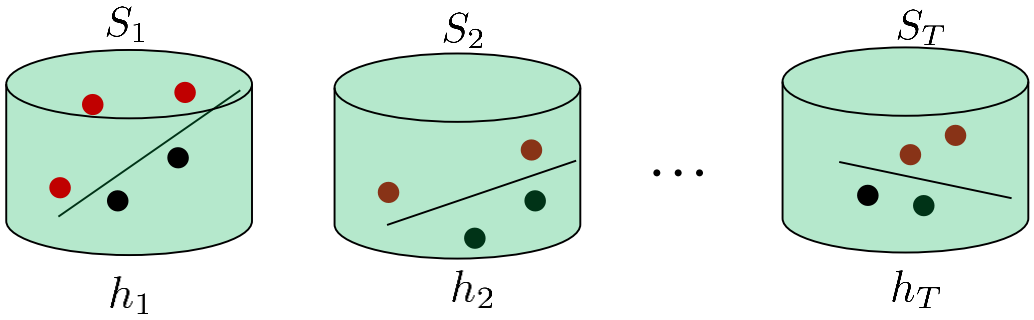


$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$



$y = +1$

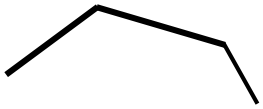
$y = -1$



α_1 α_2 α_T

$$H(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t \times h_t(\mathbf{x})$$

$h_t(\mathbf{x}) \in \{-1, +1\}$



Summary about ensemble learning methods

Bagging and Random Forests classifiers:

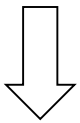
$$H(\mathbf{x}) = sign(\sum_{t=1}^{T} h_t(\mathbf{x})) \qquad h_t(\mathbf{x}) \in \{-1, +1\}$$

Each classifier is equally weighted in bagging and RF.

What if we weigh the individual classifiers differently?

AdaBoost (boosting)

$$e_{training} = \sum_{i=1}^{n} \mathbf{1}(y_i \neq H(\mathbf{x}_i))$$



$$e_{training} = \sum_{i=1}^{n} e^{-y_i \times H(\mathbf{x}_i)}$$

Summary about ensemble learning methods

AdaBoost (boosting)

$$e_{training} = \sum_{i=1}^{n} e^{-y_i \times G(\mathbf{x}_i)}$$

$\Longrightarrow$

$$G(\mathbf{x}_i; \theta) = \sum_{t=1}^{K} \alpha_t h(\mathbf{x}_t))$$

α_t are different and learned sequentially.

Bagging and Random Forests classifiers:

$$H(\mathbf{x}) = sign(\sum_{t=1}^{T} h_t(\mathbf{x}))$$

1. AdaBoost is a greedy algorithm (sequentially learned), which is learned effecitvely without the need to study the joint space of all the features.

2. It performs effective feature selection.

3. We observe a great classification power.

Summary about ensemble learning methods

AdaBoost (boosting)

$$e_{training} = \sum_{i=1}^{n} e^{-y_i \times H(\mathbf{x}_i)}$$

$\Longrightarrow$

$$H(\mathbf{x}_i) = \sum_{t=1}^{T} \alpha_t h(\mathbf{x}_t))$$

α_t are different and learned sequentially.

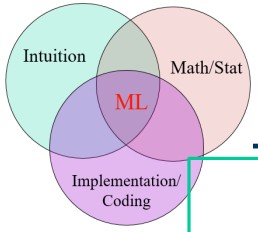
Bagging and Random Forests classifiers:

$$H(\mathbf{x}) = sign(\sum_{t=1}^{T} h_t(\mathbf{x}))$$

1. When the input feature space is high, its greedy approach becomes a bottleneck.

2. It is very hard to make it parallel since the next step $t+1$ depends on the reweighting of the previous step t.

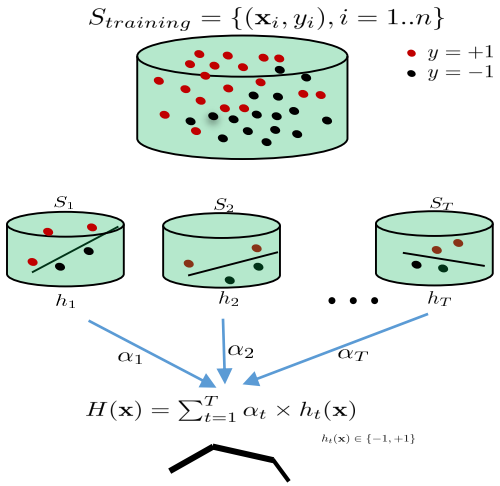
3. When t becomes large (> 20), the reweighting scheme becomes uneffective.



Recap: Boosting

Intuition: Boosting (ensemble) learning works almost all the time by combining a few weak classifiers.

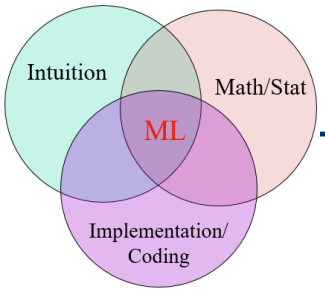
- Any standard classifiers such as SVM, logistic regression, decision tree, can be combined in boosting.
- Typically, decision stump or decision tree is adopted as weak classifier in boosting.
- Combing 50-200 weak classifiers works the best in boosting.
- The training can be hindered by the sequential process.



Empirical Observatations

- Boosting-decision tree (C4.5) often works very well.
- 2~3 level decision tree has a good balance between effectiveness and efficiency.
- Random Forests requires less training time.
- They both can be used in regression.
- One-vs-all works well in most cases in multi-class classification.
- They both are implicit and not so compact.

Recap for the classifiers we have learned.



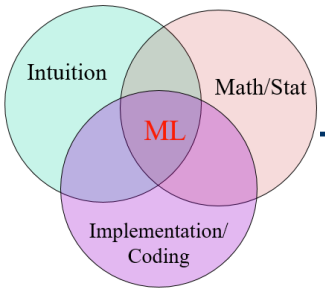
Recap: Supervised Learning

Intuition: A prediction task with a clear objective (e.g. a yes or no decision, which school to go to, a price to estimate, etc.) in which some history data for training can be acquired with the known prediction results already.

Math:

Training: $S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$

Testing: $S_{testing} = \{(\mathbf{x}_i), i = 1..u\}, what\ is\ y_i?$

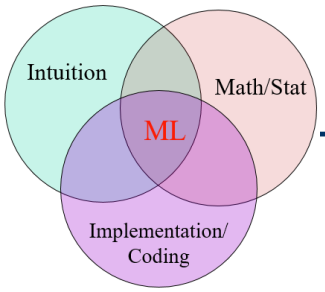


Recap: One-hot Encoding

Intuition: Turn categorical features (e.g. city, school, name, etc.) to which arithmetic operations cannot be applied into real numbers for direct mathematical manipulations. After the transformation, the input features originally described as categories have no difference to those in real numbers. They will be treated canonically in the later mathematical and statistical functions.

Math: $x_{raw} \in \{Los\ Angeles, San\ Diego, Irvine\}$

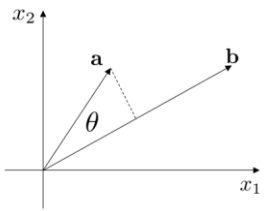
category	$\mathbf{x}_{one-hot}$
Los Angeles	1, 0, 0
San Diego	0, 1, 0
Irvine	0, 0, 1



Recap: Vector Calculus

Intuition: Both the input data and the model parameters are represented as vectors in high dimensional spaces. Using vector calculus allows us to apply mathematical operations on the vectors to train a model, make an estimation, and make a prediction using principled and sound mathematical/statistical formulations that can scale.

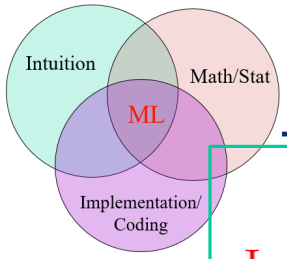
Math:



$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

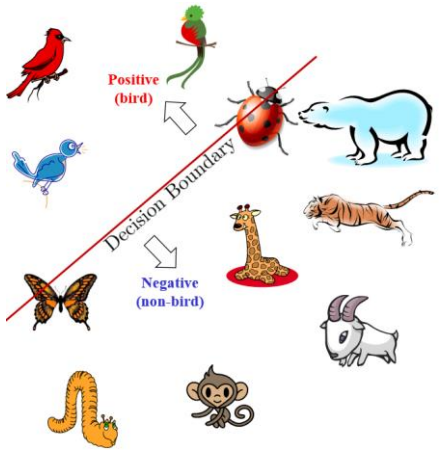
$$< \mathbf{a}, \mathbf{b} > \equiv \mathbf{a} \cdot \mathbf{b} \equiv \mathbf{a}^T \mathbf{b} \equiv a_1 b_1 + a_2 b_2 + a_3 b_3$$

$$cos(\theta) = \frac{<\mathbf{a},\mathbf{b}>}{||\mathbf{a}||_2 \times ||\mathbf{b}||_2}$$



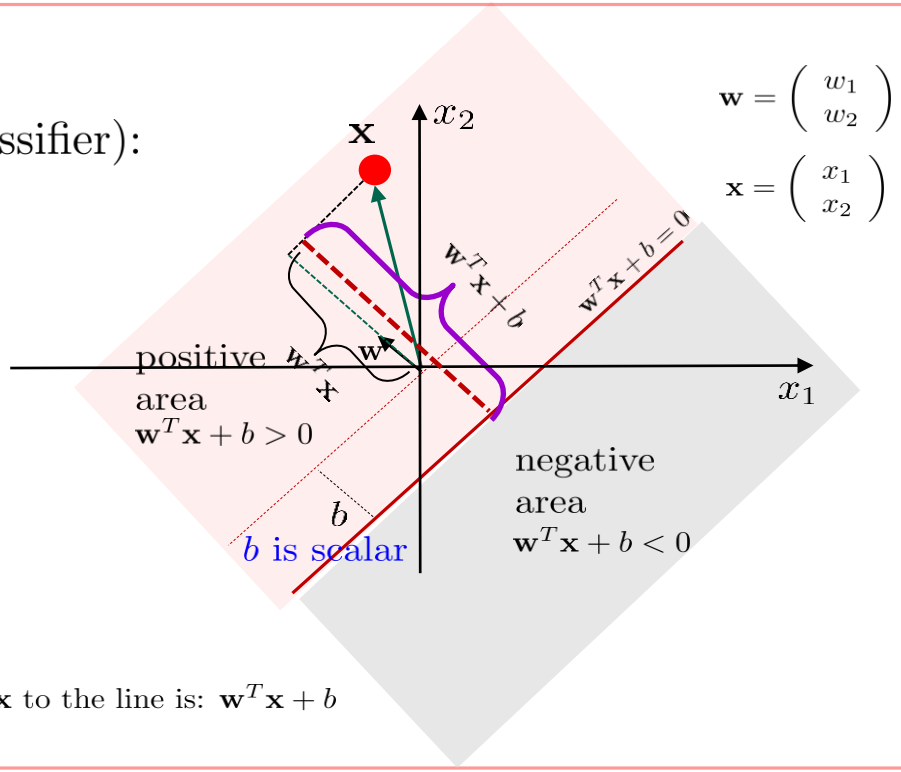
Recap: Decision Boundary

Intuition: Decision boundary of a discriminative classifier is a set that consists of possible samples that are on the border (typically $50\% - 50\%$) of the separation between the positive and negative areas (for two-class classification).

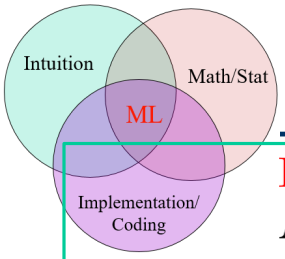


Math:

Decision boudary (for a linear classifier): $\{\mathbf{x}; \forall \mathbf{x}\ \textit{such that}\ \mathbf{w}^T\mathbf{x} + b = 0\}$



The distance (signed) of any point $\mathbf{x}$ to the line is: $\mathbf{w}^T\mathbf{x} + b$



Recap: Decision Stump Classifier

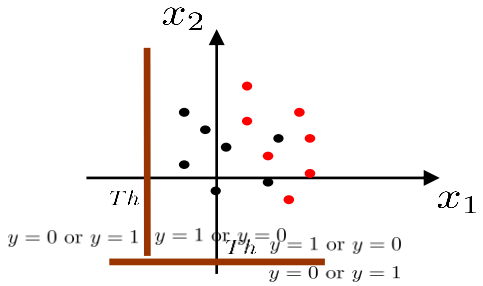
Intuition: Find the best feature at the best threshold to separate the two classes. After training, only one feature from the input is needed for testing/prediction. The model complexity (only the feature index and one threshold are stored) is extremely low; its computational complexity is also very low) since only one operation (comparison) is needed. So even the decision stump classifier might be restricted in its classification power, it is still practically very useful in many situations, e.g. to be adopted in the AdaBoost classifier as weak classifiers and KD-tree for each tree node.

Math:

$$f(\mathbf{x}, j, Th) = \begin{cases} 1 & if\ \mathbf{x}(j) \geq Th \\ 0 & otherwise \end{cases}$$

$$(j^*, Th^*) = \arg\min_{j,Th} e_{training}(j, Th) = \arg\min_{j,Th} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i, j, Th))$$

Implementation:

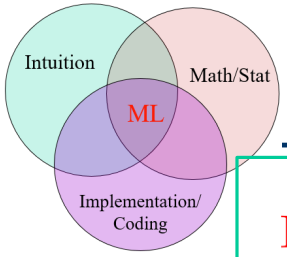


for j=1 to m

for $Th = min\_val\_j$ to max_val_j

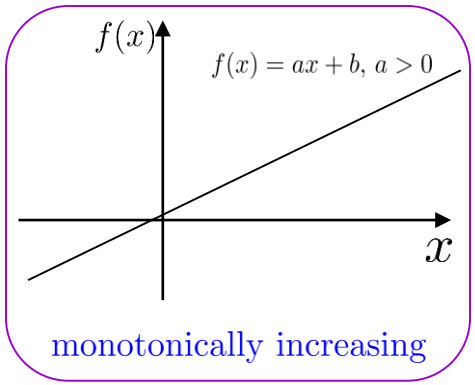
Compute $e_{training}(j, Th) = min(\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(y_i \neq f_{>}(\mathbf{x}_i, j, Th)), \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(y_i \neq f_{<}(\mathbf{x}_i, j, Th)))$

Report the (j^*, Th^*) with the lowest $e_{training}(j, Th)$

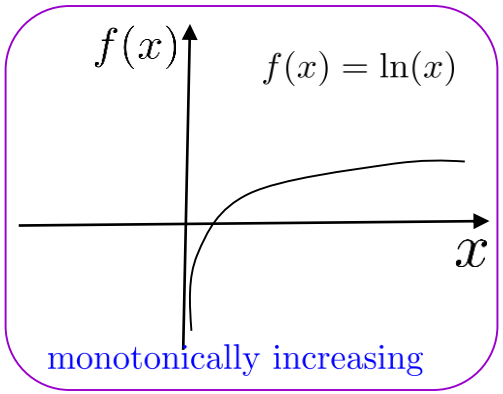


Recap: Monotonicity

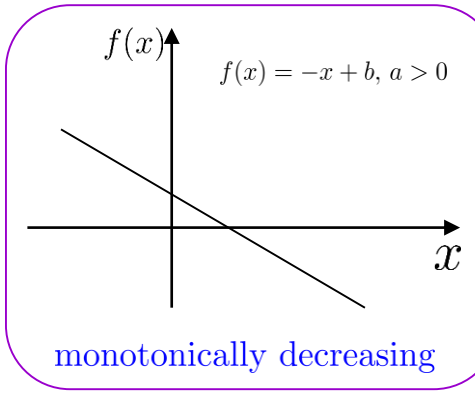
Intuition: In machine learning, we use the monotonicity of functions to help significantly reduce the difficulty/complexity of an estimation/learning problem.



monotonically increasing



monotonically increasing



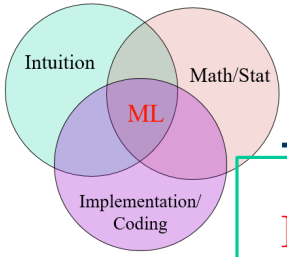
monotonically decreasing

Math:

$$w^* = \arg\max_w \prod_{i=1}^n p(y_i|x_i; w)$$

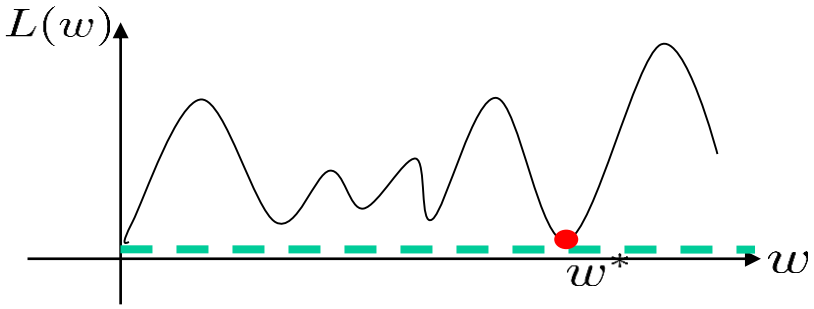
$$= \arg\max_w \ln[\prod_{i=1}^n p(y_i|x_i; w)]$$

$$= \arg\min_w -\sum_{i=1}^n \ln[p(y_i|x_i; w)]$$



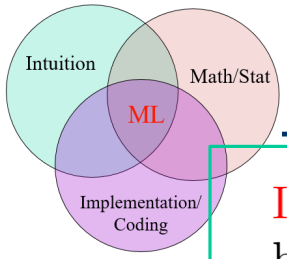
Recap: Estimation

Intuition: Typically, optimization in machine learning refers the process in which an objective function is minimized/maximized. A major task in machine learning is to perform training to attain the optimal parameter that minimizes/maximizes the corresponding objective function. Note that the optimal model parameter is not the minimal/maximal value of the function itself.



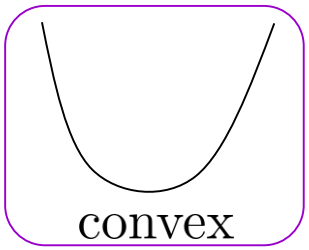
Math:

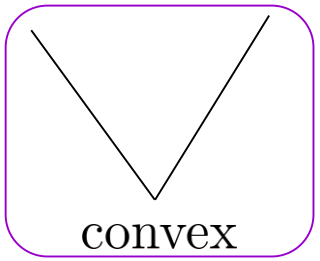
$$w^* = \arg\min_w L(w)$$

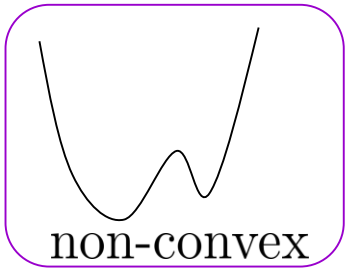


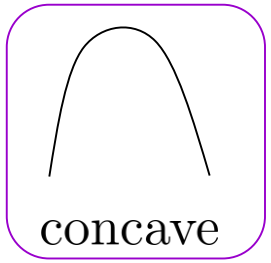
Recap: Convexity

Intuition: Understanding the convexity of the estimation functions allows us to better design the learning algorithms and allows us to judge the quality (global vs. local optimal) of the learned models.









Math:

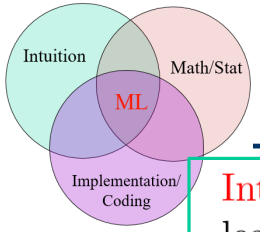
$$\forall w_0, w_1, a \in [0, 1]$$

$$aL(w_0) + (1 - a)L(w_1) \geq L(aw_0 + (1 - a)w_1)$$

or

Alternatively (for differentiable function):

$$L(w_1) \geq L(w_0) + < \nabla L(w_0), w_1 - w_0 >$$



Recap: Linear Regression

Intuition: Linear (polynomial) regression is one of the most widely used machine learning models. Typically, a squared loss is adopted in training to minimize the averaged difference between the ground-truth values and model predictions for all the input data samples. In this case, the loss/objective function is in a quadratic (convex) form w.r.t. the model parameters, hence a convex function with a unique closed-form (analytical) solution by setting the gradient of the loss function to be zero. The learned model predicts a real number (e.g. length, price, weight) for a given input.

Math:

$$\underset{Y}{\begin{pmatrix} 1 \\ 1.9 \\ 1.05 \\ 4.1 \\ 2.1 \end{pmatrix}} = \underset{X}{\begin{pmatrix} 1, 1, 0.5 \\ 1, 3, 0.9 \\ 1, 2, 1.0 \\ 1, 5, 6.7 \\ 1, 4, 2.5 \end{pmatrix}} \underset{W}{\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}}$$

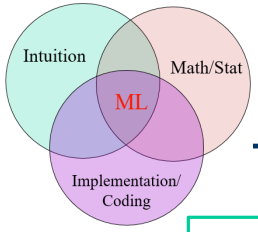
$$W^* = \arg min_W = \arg min_W L(W) = (XW - Y)^T (XW - Y)$$
$$L(W) = W^T X^T XW - W^T X^T Y - Y^T XW + Y^T Y$$
$$\frac{dL(W)}{dW} = 2X^T XW - 2X^T Y = 0$$
$$W^* = (X^T X)^{-1} X^T Y$$

Implementation:

In [ ]:

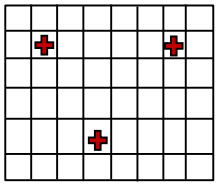
For simplification $A = (X^T X)$

```
import numpy as np
from numpy.linalg import inv
# Comput X^T X and denote it as A
A = np.dot(np.transpose(X), X)
# Obtain optimal W
W = np.dot(inv(A),np.dot(np.transpose(X),Y))
```

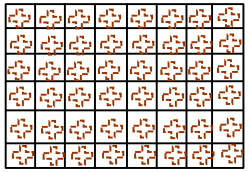


Recap: Error Metrics

Intuition: The overall effectiveness of a discriminative classifier can be evaluated using error metrics. Typically, the averaged error (or $accuracy = 1 - error$) of all the training samples is used for evaluation. However, the error can be misleading, especially for unbalanced dataset where e.g. the number of positive samples is very small w.r.t. the number of negative samples (e.g. cancerous vs. non-cancerous). Therefore, a pair of numbers are often computed precision vs. recall or specificity vs. sensitivity to give a more objective measure. A good classifier often has balanced precision and recall values although ideally we hope to have 1 for both. In machine learning, even the classifier model has been trained and fixed, there often exists a threshold that a user can customize to prefer a higher precision or a higher recall. To have a single number for judging the quality of a classifier, we use the F-score which is a combination of precision and recall within range $[0, 1]$. An extreme/trivial classifier that predicts all the samples as positives (or all as negatives) leads to $F - score = 0$ and the perfect classifier with 0 error attains $F - score = 1$.



3 targets to hit

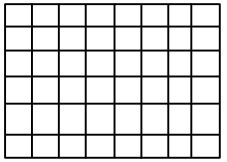


all hit

precision=$\frac{3}{48}$

recall=$\frac{3}{3}$

F-value=$\frac{0.125}{1.0625}$



zero hit

precision=$\frac{0}{0}$

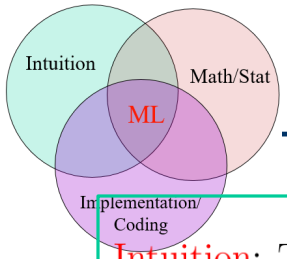
recall=$\frac{0}{3}$

F-value=$\frac{0}{1}$

Math: error: $e = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i; W))$ $\quad \mathbf{1}(z) = \begin{cases} 1 & if\ z = TRUE \\ 0 & otherwise \end{cases}$

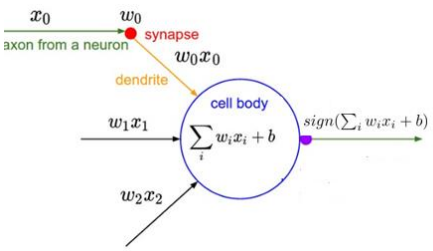
$$\text{precision} = \frac{P(sick+\ and\ test+)}{P(test+)} \quad \text{recall} = \frac{P(sick+\ \ and\ \ test+)}{P(sick+)}$$

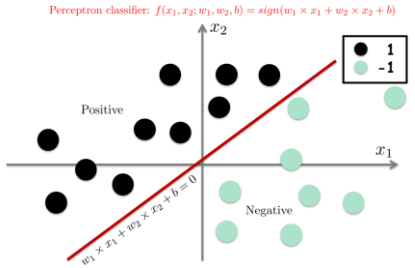
$$\text{F-value} = \frac{2 \times precision \times recall}{precision + recall}$$



Recap: Perceptron

Intuition: The perceptron classifier itself is still a linear classifier using the sign (positive or negative) of the output as the prediction. Training a perceptron is done through an iterative procedure by visiting each individual sample to update the modal until convergence. This can be viewed as an extreme case of stochastic gradient descent with the batch size being 1. The perceptron classifier itself has limited classification power and cannot well classify non-separable samples (e.g. sample distribution as XOR). Therefore, it has been ignored for many years in the computing field. Although perceptron itself is limited, adding perceptrons with an activation function (e.g. sigmoid or ReLU) and building layers of them have led to significantly enhanced power in the modern deep learning era.

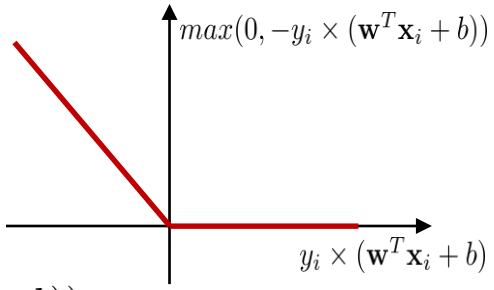




Math:

$$f(\mathbf{x}; \mathbf{w}, b) = \begin{cases} +1 & if\ \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & otherwise \end{cases}$$

$$S = \{(\mathbf{x}_i, y_i), i = 1..n\}$$



Training: Minimize $\mathcal{L}(\mathbf{w}, b) = \sum_i max(0, -y_i \times (\mathbf{w}^T\mathbf{x}_i + b))$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + (target_i - output_i)\mathbf{x}_i \qquad target_i = y_i$$
$$b_{t+1} \leftarrow b_t + (target_i - output_i) \qquad output_i = sign(\mathbf{w}^T\mathbf{x}_i + b)$$

Recap: Perceptron

Implementation:

Initialize the weights $\mathbf{w} \in \mathbb{R}$ and $b \in \mathbb{R}$ for

$$f(\mathbf{x}|\mathbf{w}; b) = \begin{cases} +1 & if\ \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & otherwise \end{cases}$$

- Step 1: Choose a data point $\mathbf{x}_i$.
- Step 2: Compute the model output for the data point.

  $output_i = f(\mathbf{x}_i; \mathbf{w}; b)$
- Step 3: Compare model output to the target output.
- If correct classification, go to Step 5; if not, go to Step 4.
- Step 4: Update weights using perceptron learning rule.

  $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + (target_i - output_i)\mathbf{x}_i$

  $b_{t+1} \leftarrow b_t + (target_i - output_i)$
- Step 5: If you have visited all the data points and they are all corrected classifier, then exit; otherwise visit next data point and go back to step 2.

Main motivation

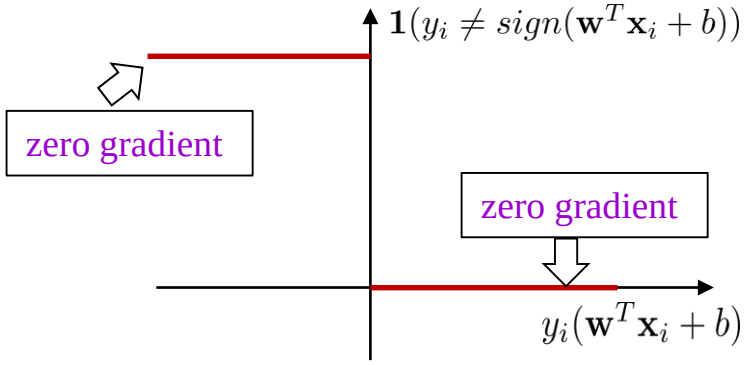
Hard->Half-hard->Soft

Error

Standard loss (error) function

Standard 0/1 loss (gradient 0 nearly everywhere, no gradient feedback):

Training: Minimize $\mathcal{L}(\mathbf{w}, b) = \sum_i \mathbf{1}(y_i \neq sign(\mathbf{w}^T \mathbf{x}_i + b))$



It is the most directly loss, but is also the hardest to minimize.

Zero gradient everywhere!

Main motivation

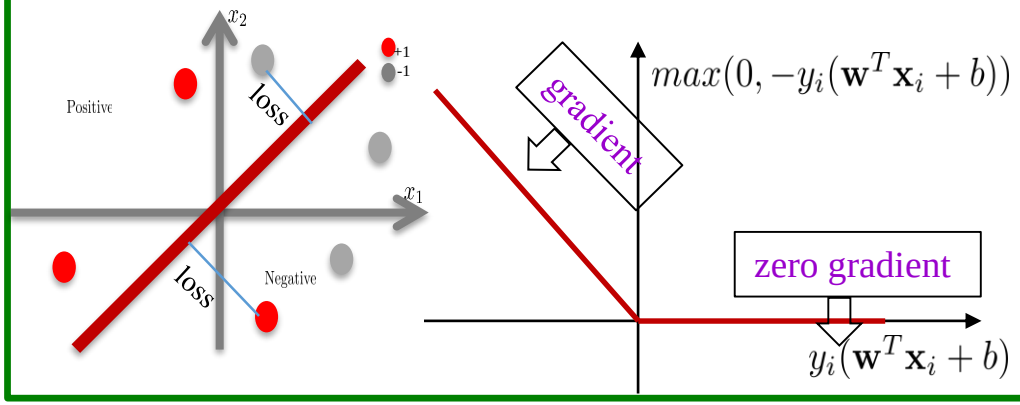
Hard->Half-hard->Soft

Error

Half-hard loss (error) function

Loss implicitly used in the perceptron algorithm: with gradient feedback when the target (ground-truth label) and the output (classification) are different).

Training: Minimize $\mathcal{L}(\mathbf{w}, b) = \sum_i max(0, -y_i(\mathbf{w}^T\mathbf{x}_i + b))$



Zero loss for correct classification (no gradient).

A loss based on the distance to the decision boundary for misclassification (with gradient).

Used in the perceptron training.

Soft loss (error) function

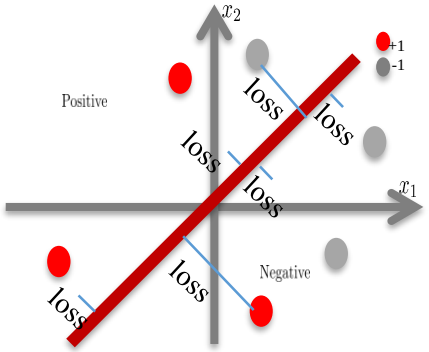
Main motivation

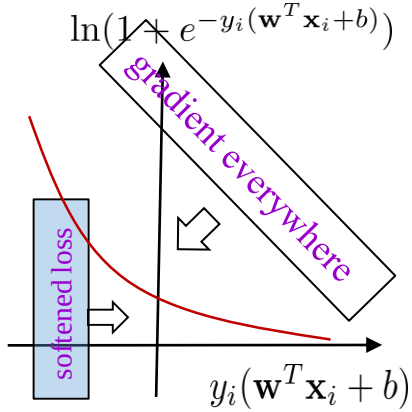
Hard->Half-hard->Soft

Error

Loss used in logistic regression.

Training: minimize $\mathcal{L}(\mathbf{w}, b) = \sum_{i=1}^{n} \ln(1 + e^{-y_i(\mathbf{w}^T \mathbf{x}_i + b)})$





Every data point receives a loss (gradient everywhere).

A loss based on the distance to the decision boundary for wrong classification (has a gradient).

Used in logistic regression classifier.

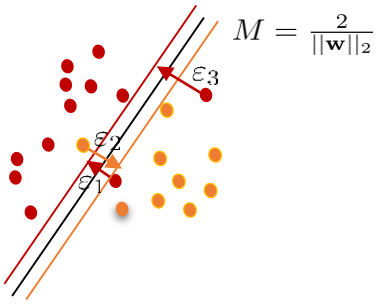
Loss in SVM

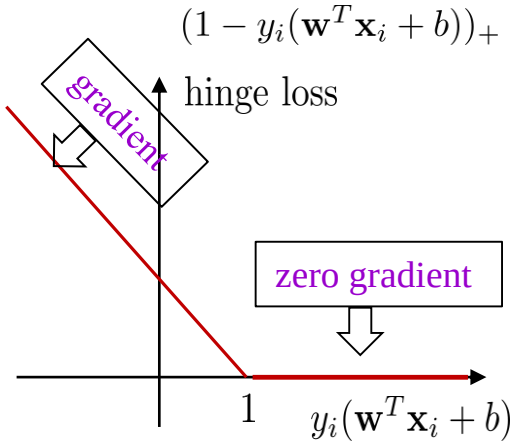
Main motivation

Hard->Hinge

Error

Minimize $\mathcal{L}(\mathbf{w}, b) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))_+$

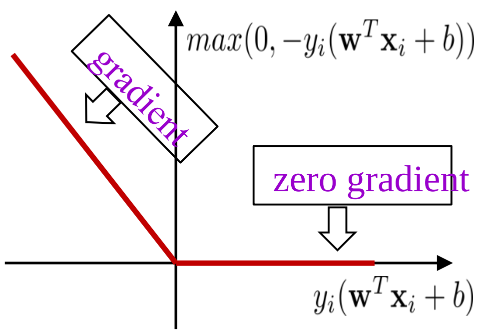
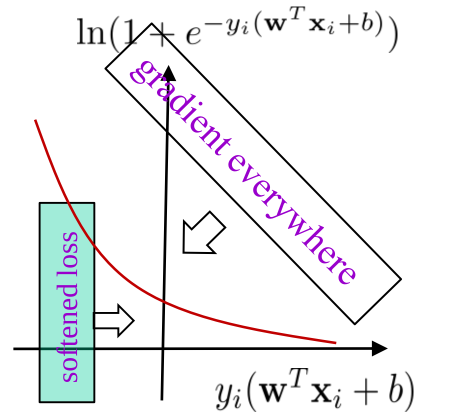
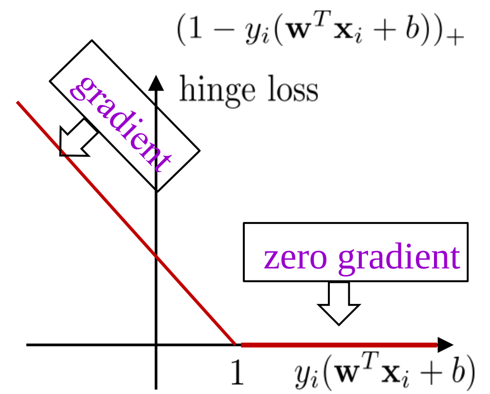
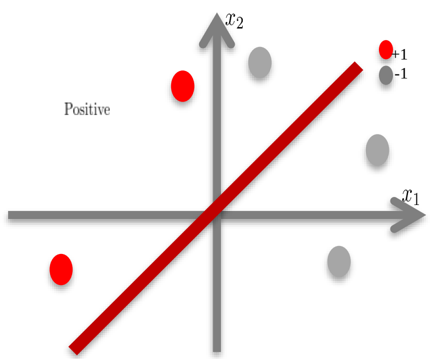
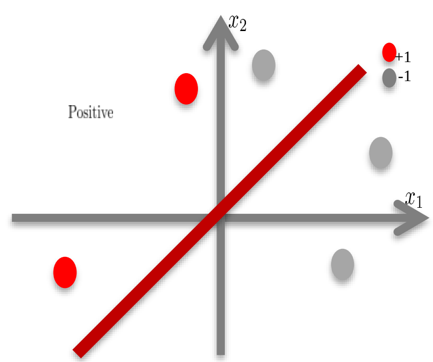
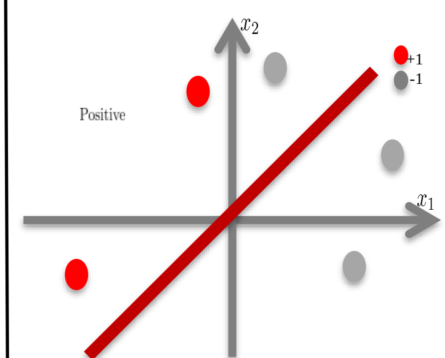


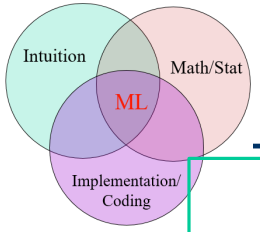


Zero loss for correct classification beyond the margin (no gradient).

A loss based on the distance to the decision boundary for misclassification or within the margin (with gradient).

A Summary

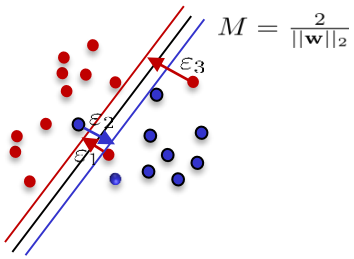
	Perceptron	Logistic Regression	SVM
Training	Training: Minimize $\mathcal{L}(\mathbf{w},b)=\sum_i max(0,-y_i(\mathbf{w}^T\mathbf{x}_i+b))$  convex optimization	Training: Minimize $\mathcal{L}(\mathbf{w},b)=\sum_{i=1}^n \ln(1+e^{-y_i(\mathbf{w}^T\mathbf{x}_i+b)})$  convex optimization	Training: Minimize $\mathcal{L}(\mathbf{w},b)=\frac{1}{2}\lVert\mathbf{w}\rVert^2$ $+C\sum_{i=1}^n(1-y_i(\mathbf{w}^T\mathbf{x}_i+b))_+$  convex optimization
Testing	 Test: $f(\mathbf{x};\mathbf{w},b)=\begin{cases}+1 & if\ \mathbf{w}^T\mathbf{x}+b\geq 0\\ -1 & otherwise\end{cases}$	 Test: $f(\mathbf{x};\mathbf{w},b)=\begin{cases}+1 & if\ \mathbf{w}^T\mathbf{x}+b\geq 0\\ -1 & otherwise\end{cases}$	 Test: $f(\mathbf{x};\mathbf{w},b)=\begin{cases}+1 & if\ \mathbf{w}^T\mathbf{x}+b\geq 0\\ -1 & otherwise\end{cases}$

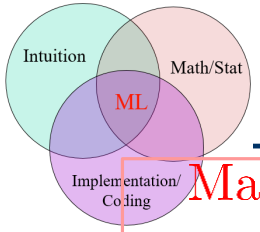


Recap: Support Vector Machine

Intuition: It explicitly introduces a "regularization" (margin) into the objective function to combine with a classification error (restricted using a hinge loss) term.

- It achieves unprecedented robustness when training a linear classifier due to the use of margin term in training.
- The learned model is based on a balance between classification error and margin. The balancing term C is typically attained using cross-validation.
- Kernel based SVM makes non-separable samples feasible to classify by projecting the data onto higher dimensional spaces.
- The features defined under kernels don't need to be computed explicitly.
- The learned weights $\mathbf{w}$ is carried in the weights for the samples and those samples with non-zero weights are called support vectors.





Recap: Support Vector Machine

Math:

Training :

$$\text{Minimize } \mathcal{L}(\mathbf{w}, b) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))_+$$

$$\frac{\mathcal{L}(\mathbf{w},b)}{\partial \mathbf{w}} = \mathbf{w} + C\sum_{i=1}^{n}\begin{cases}0 & if\ \ y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1\\ -y_i\mathbf{x}_i & otherwise\end{cases}$$

$$\frac{\mathcal{L}(\mathbf{w},b)}{\partial b} = C\sum_{i=1}^{n}\begin{cases}0 & if\ \ y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1\\ -y_i & otherwise\end{cases}$$

Testing :

$$f(\mathbf{x}; \mathbf{w}, b) = \begin{cases}+1 & if\ \mathbf{w}^T\mathbf{x} + b \geq 0\\ -1 & otherwise\end{cases}$$

Implementation:

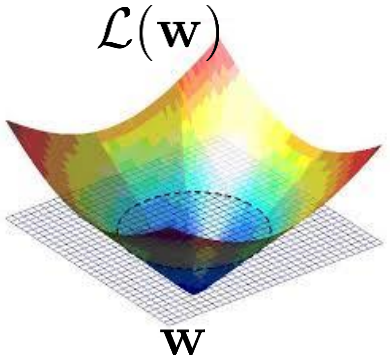
Gradient Descent Direction

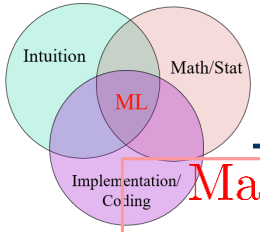
(a) Pick a direction $\nabla\mathcal{L}(\mathbf{w}_t, b_t)$

(b) Pick a step size λ_t

(c) $\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda_t \times \nabla\mathcal{L}_{\mathbf{w}_t}(\mathbf{w}_t, b_t)$ such that function decreases;
$b_{t+1} = b_t - \lambda_t \times \nabla\mathcal{L}_{b_t}(\mathbf{w}_t, b_t)$

(d) Repeat





Recap: Classifier Complexity and VC Dimension

Math:

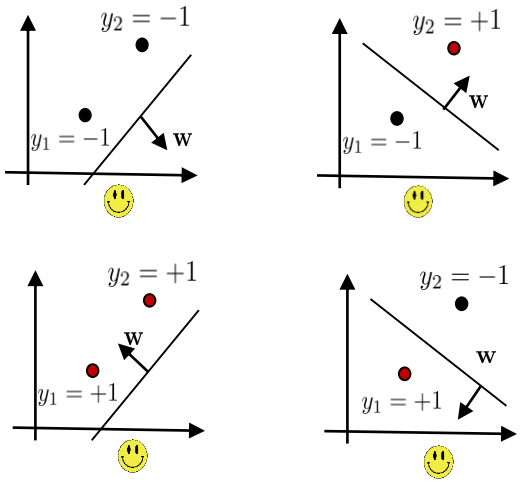
$$e_{testing} \leq e_{training} + \sqrt{\frac{h(\log(2n/h+1)) - \log(\eta/4)}{n}}$$

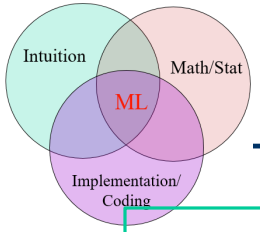
h : the complexity (VC dimension) of a classifier
n: the number of training samples

η: confidence level, can be ignored for the moment

Intituion:

- The concept and theory of VC dimension, named after Vapnik and Chervonenkis, defines the maximum capability of a classifier f.
- VC dimension (h) reports the maximum number of points a classifier f can shatter.
- It is done by checking the number of shattering sequentially from 1,2,3,… until it fails.





Recap: Structural risk minimization and cross-validation

1. Given a set of training data: $S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$

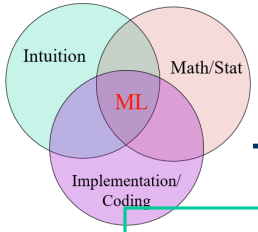
$$e_{testing} \leq e_{training} + \sqrt{\frac{h(\log(2n/h+1)) - \log(\eta/4)}{n}}$$

where $e_{testing}$ is unobserved but can be estimated.

2. To achieve the minimal testing error for a given classifier $f(x;\theta)$, we want to: **(a)** attain a small training error $e_{training}$, and **(b)** adopt a large training set of large n **(c)** while making $f(\mathbf{x};\mathbf{w})$ as simple as possible (characterized by the power/VC dimension of $f(\mathbf{x};\mathbf{w}) - - h$).

3. The optimal choise for $f(\mathbf{x};\mathbf{w})$ can be guided by the structural risk minimization principle (in theory). Typically, there will additional hyper-parameters γ.

4. In practice, we use e.g. cross-validation to do hyper-parameter tuning on $\gamma$to choose $f(\mathbf{x};\mathbf{w})$. γ can be e.g. the parameter C in SVM, the choice between L2 vs. L1, the type of classifier, etc.

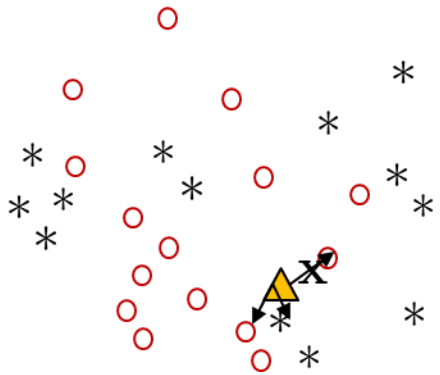


Recap: Nearest neighborhood classifier

1. Very easy to implement.
2. Works very well in practice.
3. Non-parametric model.
4. Model complexity is too high when the training set is large.
5. Computational complexity is high.

$\circ \quad y = +1$

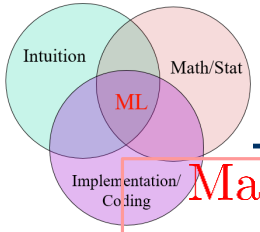
$* \quad y = -1$



$k = 3$

2 positives
1 negative

$y = +1 \rightarrow \mathbf{x}$



Recap: Kernel-based Support Vector Machine

Math:

Training : Minimize $\mathcal{L}(\mathbf{w}, b) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))_+$

$\Longrightarrow$ Find $\arg\max_{\alpha_1,\ldots\alpha_n} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n}\alpha_j\alpha_i Q_{ij}$

$\mathbf{w}^* = \sum_{i=1}^{n}\alpha_i\mathbf{x}_i$

$b^* = y_k(1 - \varepsilon_k) - \mathbf{w}\cdot\mathbf{x}_k$ where $k = \arg\max_i \alpha_i$

Ridge regression: $\alpha = (C \times (XX^T) + I)^{-1}Y$

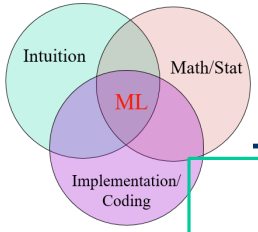
Testing : Learned classifier: $sign(\sum_i \alpha_i K(\mathbf{x}_i, \mathbf{x}))$

Pros:

- It is very robust.
- Works very well in practice.
- Mathematically well-defined and can be extended to many places.

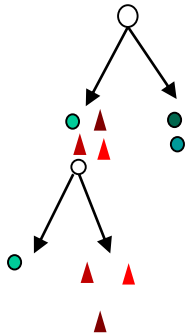
Cons:

- No intrinsic feature selection stage.
- May not be able to deal with large amount training data with high dimension due to its kernel.



Recap: Decision Tree

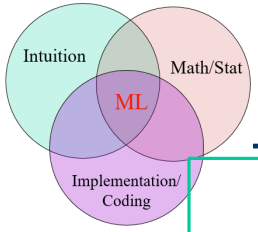
- Decision tree classifier is one of the most widely used classifiers in machine learning.
- It is a non-parametric model that can grow deep.
- Its key spirit is about divide-and-conquer.
- It has a nice balance between model complexity and classification power.
- It is often combined with other methods such as Boosting and Bagging to achieve enhanced performances.



Math:

$$f^* = \arg max_f \quad gain(S^{(f)}_{left}) + gain(S^{(f)}_{right}) - gain(S)$$

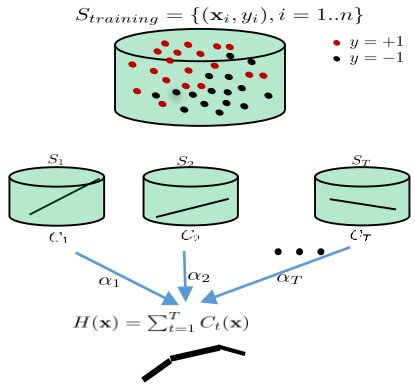
$$gain(S) = -|S| \times Entropy(Y_S)$$

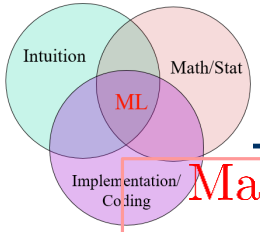


Recap: Bagging

Intuition: Bagging (ensemble) learning works almost all the time by combining a few weak classifiers.

- Any standard classifiers such as SVM, logistic regression, decision tree, can be combined in bagging.
- The more variant the base weak classifiers are, the more apparent the effect will be for bagging.
- Typically, combing 50-200 weak classifiers works the best in bagging.
- The training process can be highly efficient by having the weak classifiers trained in parallel.





Recap: Bagging

Math:

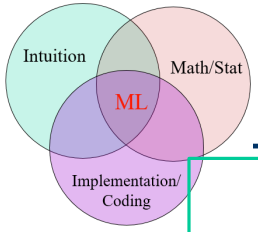
$$H(\mathbf{x}) = \sum_{t=1}^{T} h_t(\mathbf{x})$$

Implementation:

Given a training set S
For $t = 1$ to T do:
    Build subset S_t by sampling with replacement from S
    Learn classifier h_t from S_t
Make predictions according to majority vote of the set of T classifiers.

Features of Random Forests

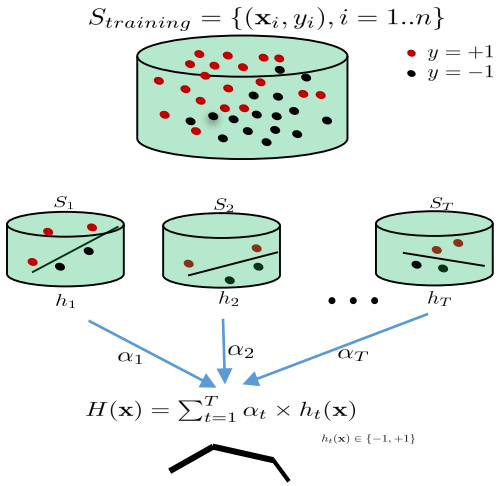
- It is unexcelled in accuracy among current algorithms.
- It runs efficiently on large data bases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what featuress are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing error in class population unbalanced data sets.



Recap: Boosting

Intuition: Boosting (ensemble) learning works almost all the time by combining a few weak classifiers.

- Any standard classifiers such as SVM, logistic regression, decision tree, can be combined in boosting.
- Typically, decision stump or decision tree is adopted as weak classifier in boosting.
- Combing 50-200 weak classifiers works the best in boosting.
- The training can be hindered by the sequential process.



Thanks everyone for your hard work!

Provide your feedback at www.cape.ucsd.

Good luck and stay healthy!