CSE417T – Lecture 18

- Please **mute** yourself and **turn off videos** to save bandwidth.

- If you have questions during the lecture
  - Use chatrooms to post your questions
    - I'll review chatrooms in batches
  - You can also un-mute yourself and ask the questions directly
- The slides are posted on the course website

- RECORD THE LECTURE!
  - Please remind me if I forget to do so.

Logistics: Homework

- Homework 4 will be due April 13 (Monday)
  - Two implementation questions
  - You can work as a group of up to 2 persons
    - Doable and okay for working on the homework yourself
    - Collaboration could be challenging in the current situation
  - Please start it early
    - It was on average the most time consuming assignment for students in the past
  - Keep track of your own late days
    - Gradescope doesn't allow separate deadlines
    - Your submissions won't be graded if you exceed the late-day limit
- HW5 will have a tighter deadline
  - Tentative dates (still subject to change)
    - announce on April 7, due on April 19, 11AM

Logistics: E-Chapters of LFD (AML)

- The textbook offers a set of e-chapters
  - Chap 6: Similarity-Based Methods
  - Chap 7: Neural Networks
  - Chap 8: Support Vector Machines
  - Chap 9: Learning Aides

- How to access
  - http://book.caltech.edu/bookforum/forumdisplay.php?f=148

  - User Name: **bookreaders**
  - Password: *Enter the first word on page 27 of the book.*

Recap

Ensemble Learning

- Goal: Utilize a set of weak learners to obtain a strong learner.

- Format of ensemble learning
  - Construct many diverse weak learners
  - Aggregate the weak learners

Bagging:
- Construct diverse weak learners
  - (Simultaneously) bootstrapping datasets
  - Train weak learners on them
- Aggregate the weak learners
  - Uniform aggregation

Boosting
- Construct diverse weak learners
  - Adaptively generating datasets
  - Train weak learners on them
- Aggregate the weak learners
  - Weighted aggregation

Bagging and Random Forest

- Construct many random trees
  - Bootstrapping datasets (sample with replacement from D)
  - Learn a max-depth tree for each of them
  - Other randomizations (not required in HW4)
    - When choosing split features, choose from a random subset (instead of all features)
    - Randomly project features (similar to non-linear transformation) for each tree

- Aggregate the random trees
  - Classification: Majority vote $\bar{g}(\vec{x}) = sign\left(\frac{1}{M}\sum_{m=1}^{M} g_m(\vec{x})\right)$
  - Regression: Average $\bar{g}(\vec{x}) = \frac{1}{M}\sum_{m=1}^{M} g_m(\vec{x})$

Outline of a Boosting Algorithm

- Initialize D_1 (usually the same as the initial dataset D)
- For $t = 1$ to T
  - Learn g_t from D_t
  - Reweight the distribution and obtain D_{t+1} based on g_t and D_t
- Output weighted-aggregate$(g_1, \dots, g_T)$
  - Classification: $G(\vec{x}) = \bar{g}(\vec{x}) = sign\left(\frac{1}{T}\sum_{t=1}^{T} \alpha_t g_t(\vec{x})\right)$

Questions

How to learn g_t from D_t
How to reweight the distribution and obtain D_{t+1}
How to perform weighted aggregation

AdaBoost Algorithm

- Given $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)\}$
- Initialize $D_1(n) = 1/N$ for all $n = 1, \dots, N$
- For $t = 1, \dots, T$
  - Learn g_t from D_t (using decision stumps)
  - Calculate $\epsilon_t = E_{in}^{(D_t)}(g_t)$
  - Set $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  - Update $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) e^{-\alpha_t y_n g_t(\vec{x}_n)}$
- Output $G(\vec{x}) = sign(\sum_{t=1}^{T} \alpha_t g_t(\vec{x}))$

Lecture Notes Today

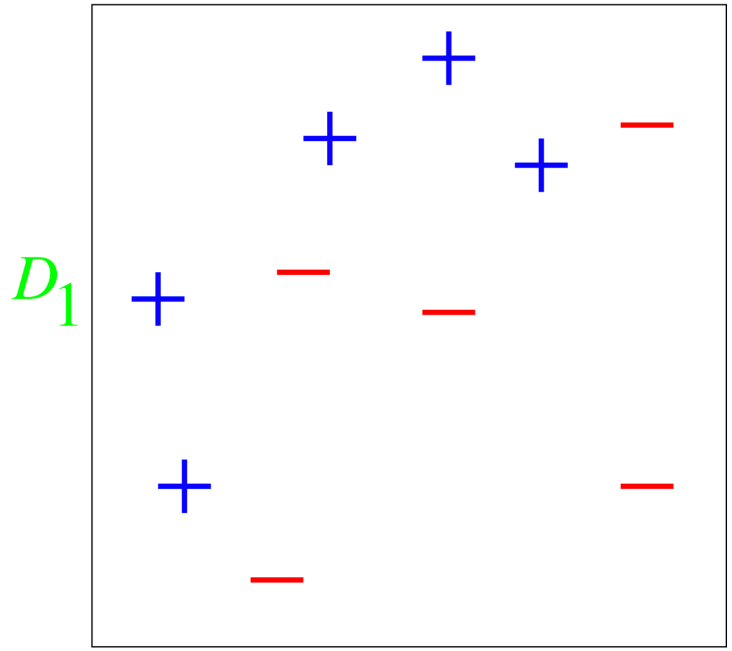
The notes are not intended to be comprehensive. They should be accompanied by lectures and/or textbook.

Let me know if you spot errors.

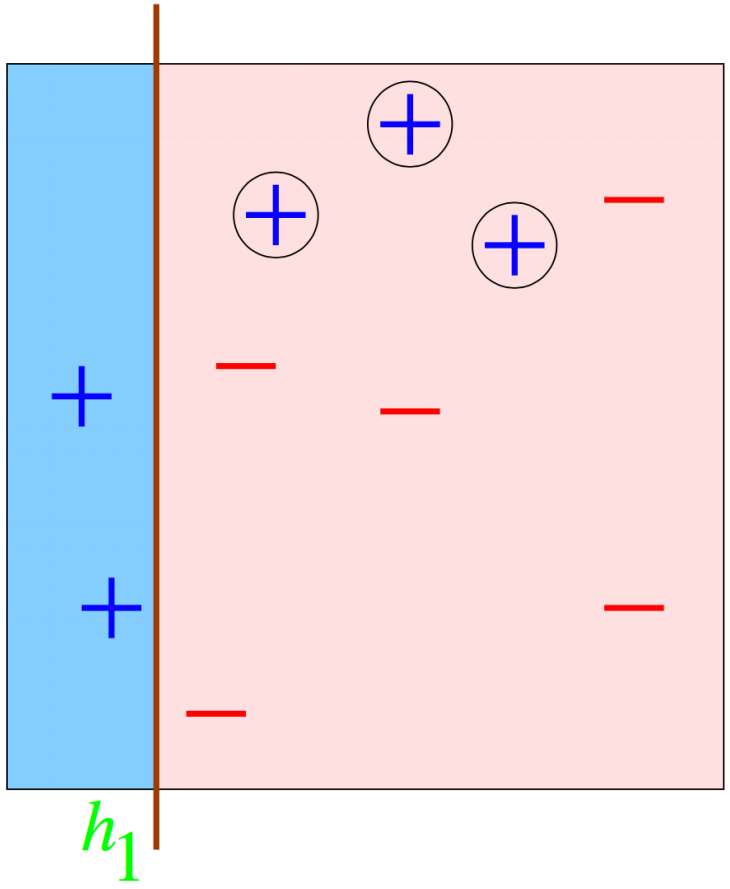
AdaBoost in Action

AdaBoost in Action

- A toy example (by Yoav Freund Rob Schapire)
- Weak learner: decision stump (one-level decision tree)



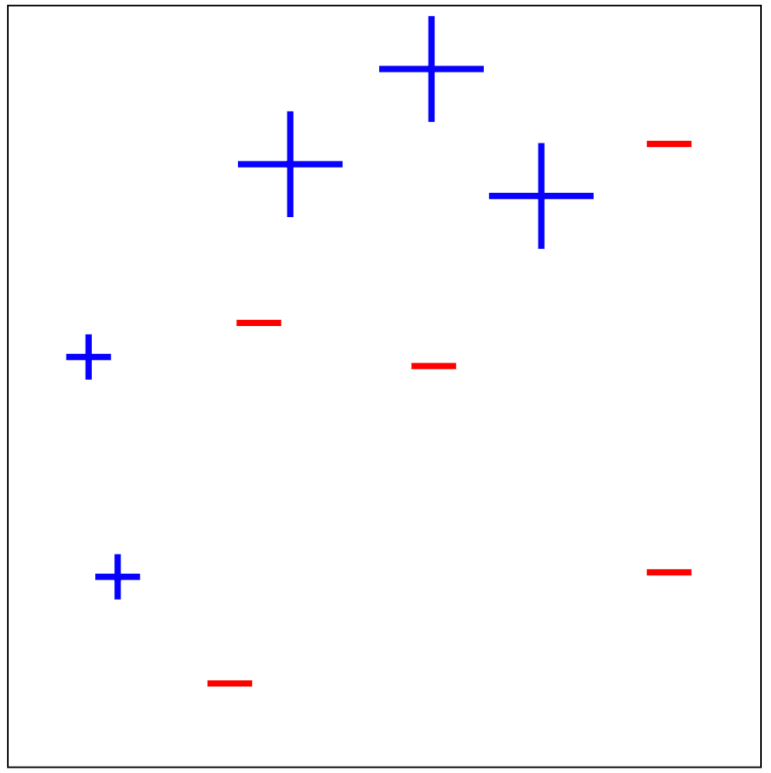
Round 1



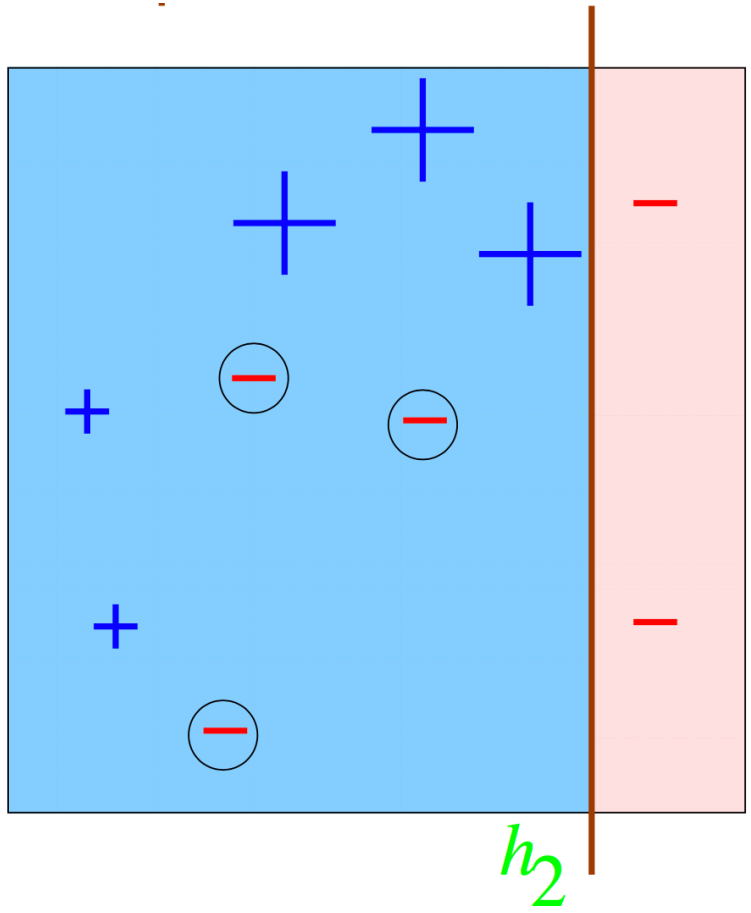
$\varepsilon_1 = 0.30$

$\alpha_1 = 0.42$

D_2

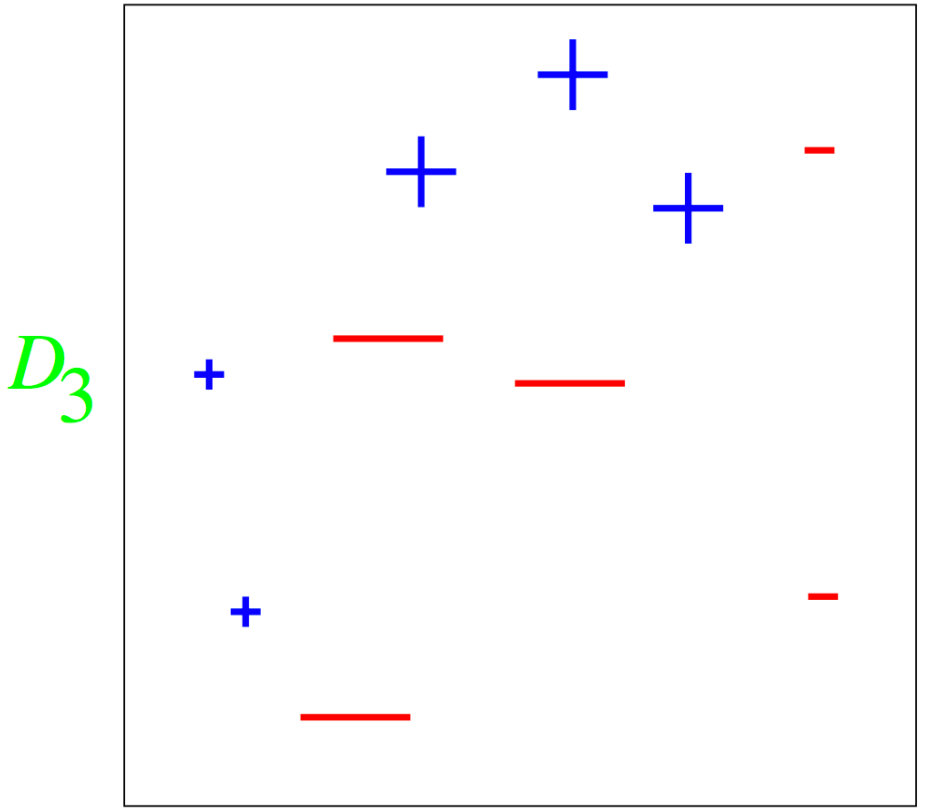


Round 2

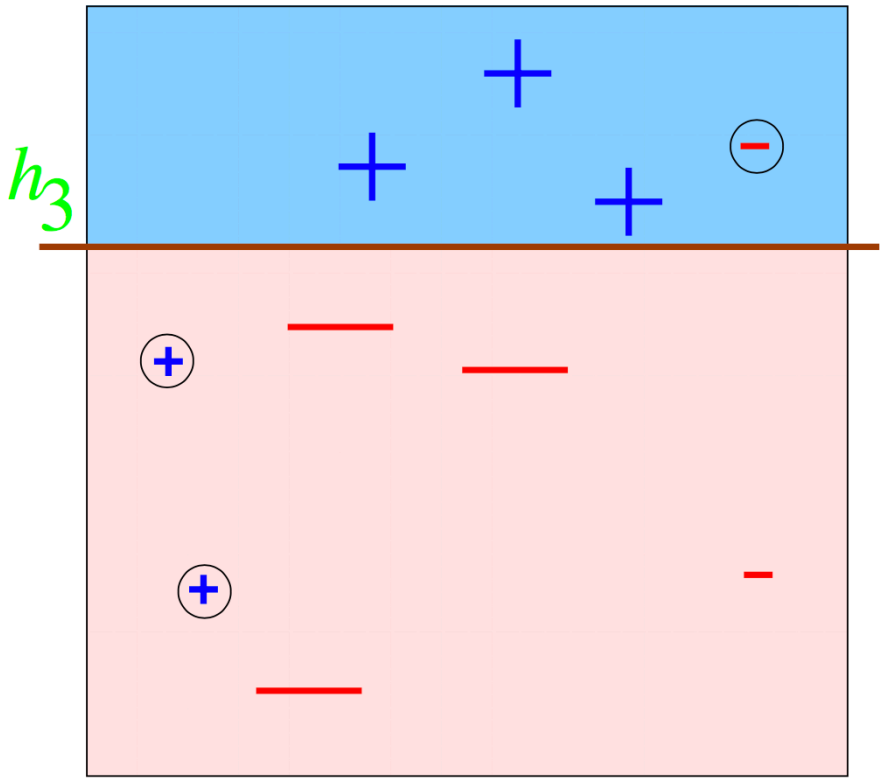


$\varepsilon_2 = 0.21$

$\alpha_2 = 0.65$

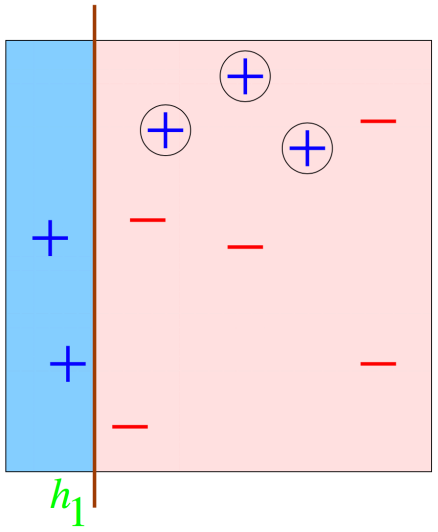


Round 3



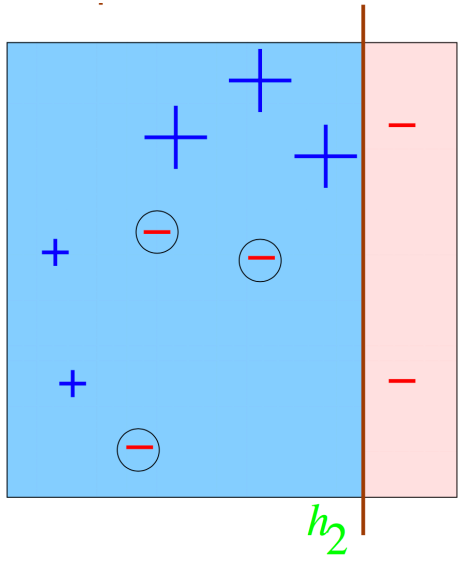
$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$



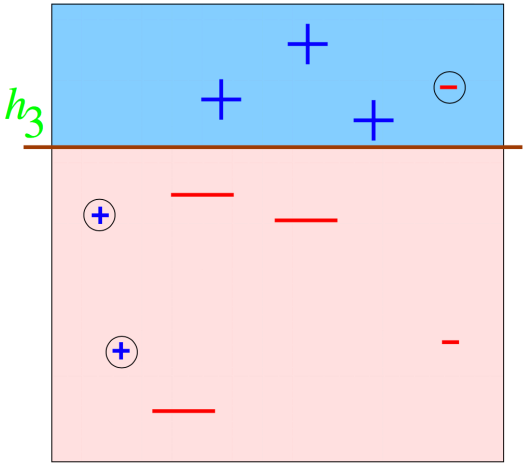
$\varepsilon_1 = 0.30$

$\alpha_1 = 0.42$



$\varepsilon_2 = 0.21$

$\alpha_2 = 0.65$

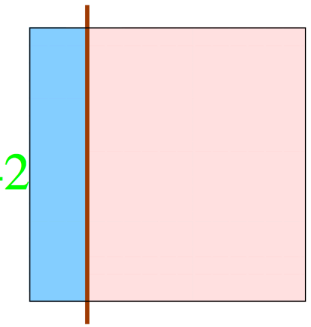


$\varepsilon_3 = 0.14$

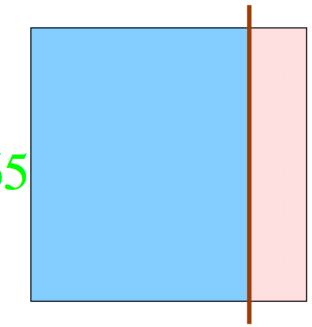
$\alpha_3 = 0.92$

H_{final}

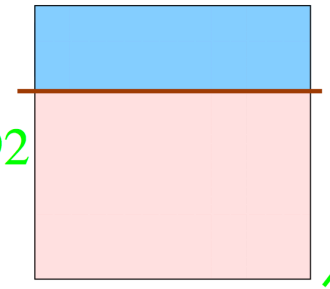
$= \text{sign}\left(0.42 \right.$



$+ 0.65$

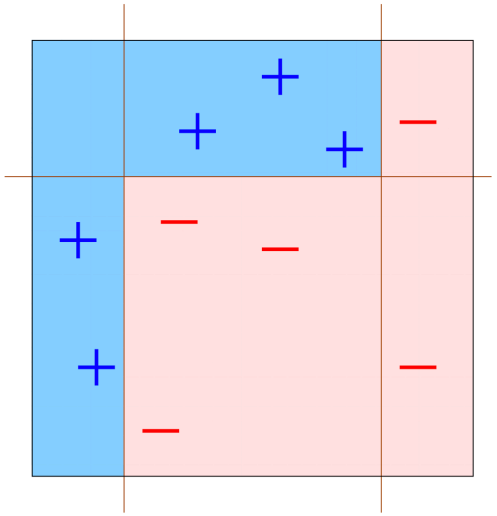


$+ 0.92$



$\left. \right)$

$=$



Brief Discussion on Gradient Boosting

Gradient boosting is **safe to skip**

Look at the AdaBoost Algorithm Again

Given $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)\}$
Initialize $D_1(n) = 1/N$ for all $n = 1, \dots, N$
For $t = 1, \dots, T$
- Learn g_t from D_t (using decision stumps)
- Calculate $\epsilon_t = E_{in}^{(D_t)}(g_t)$
- Set $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
- Update $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) e^{-\alpha_t y_n g_t(\vec{x}_n)}$

Output $G(\vec{x}) = sign(\sum_{t=1}^{T} \alpha_t g_t(\vec{x}))$

Initialize $G(\vec{x}) = 0$

For $t = 1, \dots, T$

$G(\vec{x}) \leftarrow G(\vec{x}) + \alpha_t g_T(\vec{x})$

Output $sign(G(\vec{x}))$

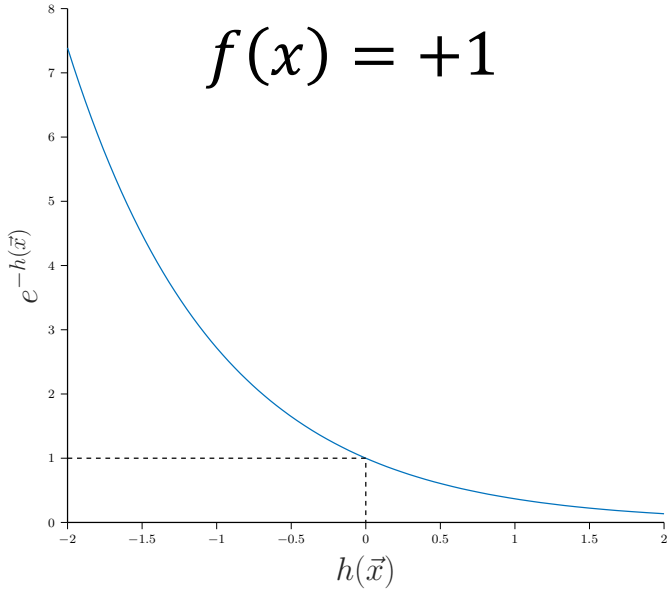
- The format is similar to gradient descent!
  - If we consider the space of the weak learners (i.e., $g_t(\vec{x})$) as the space of "weights"
  - This observation leads to a general class of boosting algorithms: gradient boosting
  - XGBoost is one implementation of gradient boosting that is popular in competitions
  - See CASI 17.4 and the reference in CASI P.350 for more discussion

[Safe to Skip]

Gradient Boosting

Initialize $G(\vec{x}) = 0$
For $t = 1, \dots, T$
$G(\vec{x}) \leftarrow G(\vec{x}) + \alpha_t g_T(\vec{x})$
Output $sign(G(\vec{x}))$

- AdaBoost is a special case of Gradient Boosting
  - minimizing the exponential loss ($e_{\exp}(h(\vec{x}), y) = e^{-yh(\vec{x})}$)
  - using decision stump as the weak learners



- e_{exp} is a surrogate loss function of the binary classification error we care about
  - Minimizing an alternative error (loss function) is a common trick in ML, especially when the target loss function is hard to optimize.
  - There are some theoretical discussions on when doing this makes sense ("calibration": whether minimizing the surrogate is consistent with minimizing the original loss).

[Safe to Skip]

Similarity-Based Method Nearest Neighbor

Movie Rating Prediction

- Below is the historical move ratings from users (5 is the highest)

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6
Alice	5	4	3	1	5	2
Bob	4	5	3	2	5	
Charlie	1	2	4	4	2	3
David	2	3	2	4	4	4
…						

- What do you think Bob's rating will be for Movie 6?
  - Maybe 2, since Bob's taste seems to be similar with Alice's

Movie Recommendation

- Below is the historical move ratings from users (5 is the highest)

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6
Alice	5	4		1		
Bob	4			2	5	
Charlie	1		4		2	
David		3	2			4
…						

- Which movie will you recommend to Alice, why?
  - Maybe Movie 5, since Bob's taste seems to be similar with Alice's

Nearest Neighbor

- Predict $\vec{x}$ according to its nearest neighbor
  - Given $D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_N, y_N)\}$
  - Let $\vec{x}_{[1]}$ be $\vec{x}$'s nearest neighbor, i.e., the closest point to $\vec{x}$ in D
  - Similarly, let $\vec{x}_{[i]}$ be the ith closest point to $\vec{x}$ in D
    - With some distance measure $d(\vec{x}, \vec{x}')$
      - $d(\vec{x}, \vec{x}_{[1]}) \leq d(\vec{x}, \vec{x}_{[2]}) \leq \dots \leq d(\vec{x}, \vec{x}_{[N]})$
  - Let $y_{[i]}(\vec{x})$ or $y_{[i]}$ be the label of $\vec{x}_{[i]}$

- Nearest neighbor hypothesis

$$g(\vec{x}) = y_{[1]}(\vec{x})$$

Common distance measures:

- Euclidean distance: $d(\vec{x}, \vec{x}') = \|\vec{x} - \vec{x}'\|$
- Cosine similarity: $d(\vec{x}, \vec{x}') = \frac{\vec{x} \cdot \vec{x}'}{\|\vec{x}\| \|\vec{x}'\|}$
- And others…

Wait....

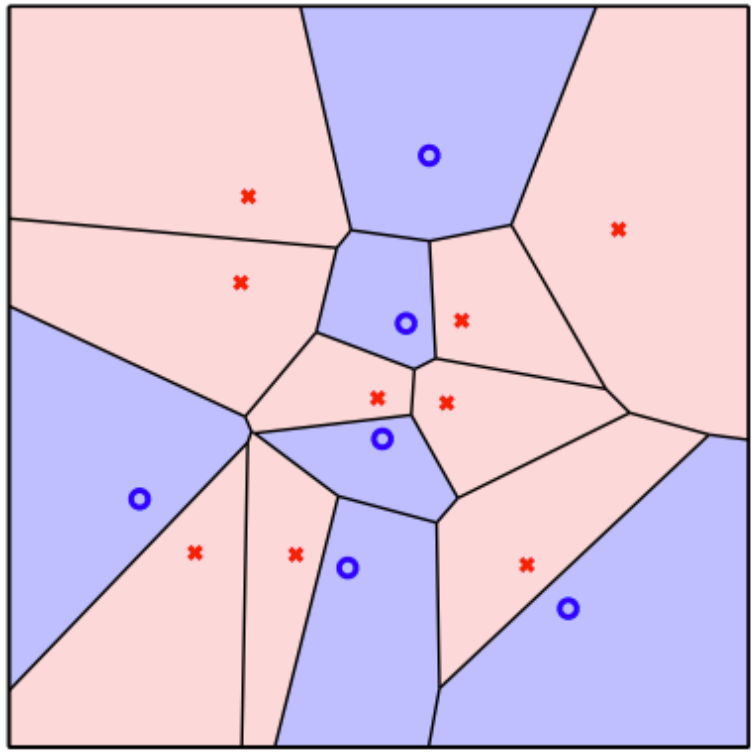
- In the practice question of exam 1:
  - Machine Leaning Whiz Kid (MLWK) proposes the following learning algorithm
    - Given D, define the learned hypothesis g as follows

$$g(\vec{x}) = \begin{cases} y_n & \text{if } x \text{ is equal to some } \vec{x}_n \in D \\ 1 & \text{otherwise} \end{cases}$$

    - In our discussion earlier, MLWK leads to
      - $E_{in} = 0$, infinite VC dimension, bad generalization

Nearest Neighbor

$g(\vec{x})$ looks like a Voronoi diagram



- Properties of Nearest Neighbor (NN)
  - No training is needed
  - Good interpretability
  - In-sample error $E_{in} = 0$
  - VC dimension is ∞
- This seems to imply bad learning models from what we talk about so far? Why we care?
- What we really care about is E_{out}
  - VC analysis: $E_{out} \leq E_{in}$ + Generalization error
    - We can infer E_{out} through E_{in} and model complexity
  - NN has nice guarantees outside of VC analysis

Nearest Neighbor is 2-Optimal

- Given mild conditions, for nearest neighbor, when $N \to \infty$, with high probability,

$$E_{out} \leq 2E_{out}^{*}$$

- That is, we can not infer E_{out} from E_{in}, but we know it cannot be much worse than the best anyone can do.

Proof Sketch of 2-Optimality

- Setup
  - The target function is noisy: $\pi(\vec{x}) = \Pr[y = +1|\vec{x}]$
  - The noisy target π is continuous in $\vec{x}$
    - Similar $\vec{x}$ should have similar labels
    - The underlying assumption for nearest neighbor to work

- Let $g^*(\vec{x})$ be the optimal hypothesis
  - $g^*(\vec{x}) = \begin{cases} +1 & \text{if } \pi(\vec{x}) \geq \frac{1}{2} \\ -1 & \text{otherwise} \end{cases}$
  - Pointwise-error $e(g^*(\vec{x}), y) = \min\{\pi(\vec{x}), 1 - \pi(x)\}$

- $E^*_{out} = \mathbb{E}_{\vec{x}}[e(g^*(\vec{x}), y)] = \mathbb{E}_{\vec{x}}[\min\{\pi(\vec{x}), 1 - \pi(x)\}]$

Proof Sketch of 2-Optimality

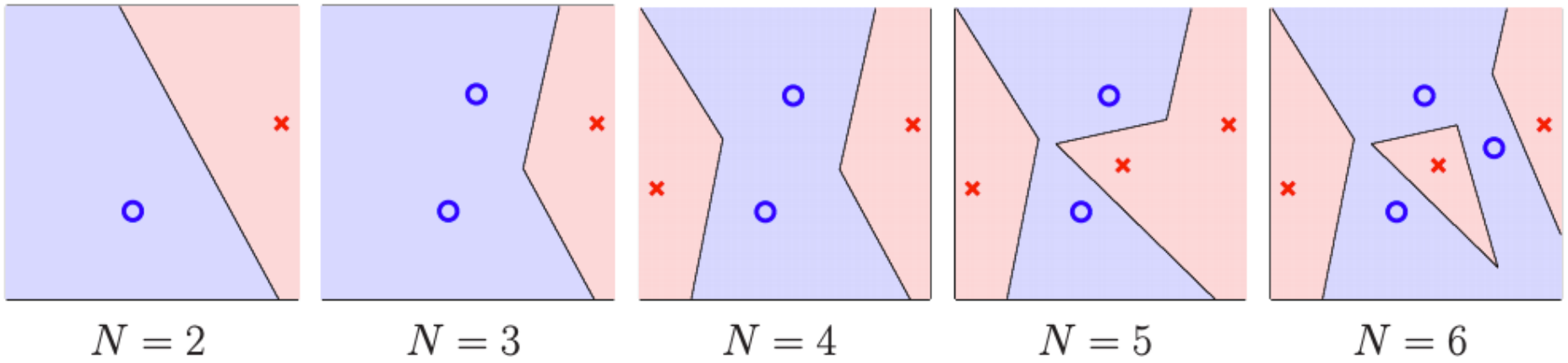
- $E^*_{out} = \mathbb{E}_{\vec{x}}[e(g^*(\vec{x}), y)] = \mathbb{E}_{\vec{x}}[\min\{\pi(\vec{x}), 1 - \pi(x)\}]$
- Proof sketch:
  - For a new point $(\vec{x}, y)$, let $(\vec{x}_{[1]}, y_{[1]})$ be its nearest neighbor in D
  - Consider the case when $N \to \infty$
    - A new point is "very close" to its nearest neighbor in D
    - $\pi(\vec{x}) \approx \pi(\vec{x}_{[1]})$
  - Error of nearest neighbor hypothesis on a new point is

$$\begin{aligned}\Pr[y \neq y_{[1]}] &= \Pr[y = +1, y_{[1]} = -1] + \Pr[y = -1, y_{[1]} = +1] \\ &= \pi(\vec{x})\left(1 - \pi(\vec{x}_{[1]})\right) + \left(1 - \pi(\vec{x})\right)\pi(\vec{x}_{[1]}) \\ &\approx 2\,\pi(\vec{x})\left(1 - \pi(\vec{x})\right) \\ &\leq 2\min\{\pi(\vec{x}), 1 - \pi(\vec{x})\}\end{aligned}$$

Nearest Neighbor is Self-Regularizing

- Intuition of regularization:
  - Use simpler hypothesis if we don't have enough data

- Nearest neighbor hypothesis



The complexity of hypothesis grows with the number of data points

Short Break and Questions

k-Nearest Neighbor

”Stabilize” the Hypothesis

- Instead of ”single” nearest neighbor
  - Making predictions according to k nearest neighbors

- k-nearest neighbor (K-NN)
  - $g(\vec{x}) = sign\left(\sum_{i=1}^{k} y_{[i]}(\vec{x})\right)$
  - (k is often odd for binary classification)

Impacts of k

- $k = 1$: the nearest neighbor hypothesis
  - many, complicated decision boundaries
  - may overfit
- $k = N$, g predicts the most common label in the training dataset
  - no decision boundaries
  - may underfit
- k controls the complexity of the hypothesis set
  - k affects how well the learned hypothesis will generalize

How to Choose k

- Making the choice of k a function of N, denoted by $k(N)$
  - Theorem:
    - If $k(N) \to \infty$ as $N \to \infty$ and $\frac{k(N)}{N} \to 0$ as $N \to \infty$
    - Then $E_{in}(g) \to E_{out}(g)$ and $E_{out}(g) \to E_{out}(g^*)$
  - Example: $k(N) = \sqrt{N}$ satisfies the condition

- Practical rule of thumb:
  - $k = 3$ is often a good enough choice
  - Using validation to choose k

Summary of k-NN So Far

- Pros
  - Simple algorithm
  - Good interpretations
  - Nice theoretical guarantee
  - Easy to adapt to regression (average of nearest neighbors) and multi-class classification (majority voting)

- Cons
  - Computational issue
    - each prediction requires $O(N)$ computation
  - Curse of dimensionality

Curse of Dimensionality

- Generally, higher dimensions implies harder learning (think VC)
- Things are worse with similarity-based methods
  - that rely on assumptions that points close to one another have similar labels
- As the dimension grows, most of the points will not be close to each other…

Illustration of Curse of Dimensionality

- Think about Euclidean distance: $d(\vec{x}, \vec{x}') = ||\vec{x} - \vec{x}'||$

- Illustration
  - Consider the space $[0,1]^d$ (a hypercube with length of each side = 1)
  - What's the side length ℓ of a hypercube that takes up 1% of the space?
    - $d = 1: \ \ell = 0.01$
    - $d = 2: \ \ell = 0.1$
    - $\ell^d = 0.01$ => $d = 100, \ \ell \approx 0.95$

Illustration of Curse of Dimensionality

- Consider the distance to the origin when $d = 100$
  - Consider the case that the value of each dimension is uniformly drawn
  - Only 1% of the points will be in the hypercube $[0,0.95]^{100}$
  - Most of the points will be far away from the origin
  - Most of the points will be far away from each other

- No simple solutions….
  - Dimension reduction techniques are often adopted (see LFD 9.2)

Computational Issues [Safe to Skip]

- k-Nearest Neighbor is computationally demanding
  - Need to store all data points: space complexity $O(Nd)$
  - For each prediction for $\vec{x}$
    - Calculate the distance to every point in D
    - Find the k closest points
    - Time complexity $O(Nd + Nlogk)$
- There are still ongoing research to address this issue
- Two general approaches:
  - Reduce the number of data points
  - Store the data in some data structure to speed up searching
  - See LFD 6.2.3 for more discussion

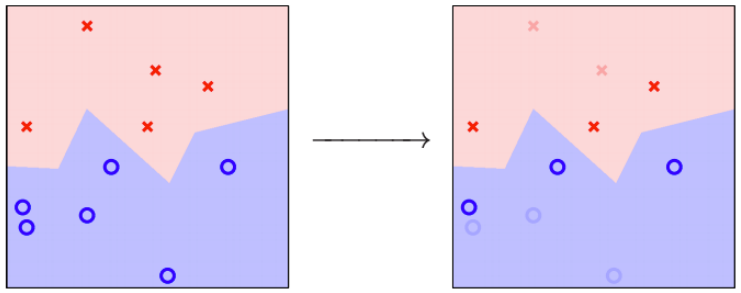
Computational Issues [Safe to Skip]

- Reduce the number of data points

- Store the data in some data structure to speed up searching

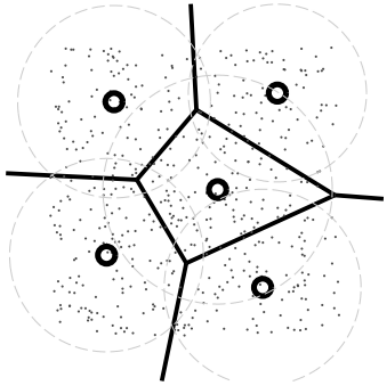
Computational Issues [Safe to Skip]

- Reduce the number of data points



- Intuition: remove points that will not impact the decision boundary.
- Generally a hard problem. But there are some heuristic approaches.

- Store the data in some data structure to speed up searching



- Intuition: Clustering data points
- For a new data point, first find a nearest cluster. Then find the nearest points within that cluster