

CSE 417T

Introduction to Machine Learning

Lecture 15

Instructor: Chien-Ju (CJ) Ho

Logistics

- Homework 4 is due November 14 (Monday)
- 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

Logistics: E-Chapters of LFD

- 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
 - Appendix B: Linear Algebra
 - Appendix C: The E-M Algorithm
- How to access e-chapters
 - <http://amlbook.com/eChapters.html>
 - "To access the e-Chapters, please download the PDFs and open them with *the first word of Chapter 4* as the password. "

Exam 1 Discussion

I plan to grade the exam myself, hopefully in around a week but no more than 2 weeks.

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 $\bar{g}(\vec{x}) = \text{sign} \left(\frac{1}{M} \sum_{m=1}^M g_m(\vec{x}) \right)$
 - Regression: Average $\bar{g}(\vec{x}) = \frac{1}{M} \sum_{m=1}^M g_m(\vec{x})$

Note for HW4:
Recommend to transform the labels to
+1/-1 for the convenience of aggregation.

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 $\text{weighted-aggregate}(g_1, \dots, g_T)$
 - Classification: $G(\vec{x}) = \bar{g}(\vec{x}) = \text{sign}\left(\frac{1}{T} \sum_{t=1}^T \alpha_t g_t(\vec{x})\right)$

Questions

How to learn g_t from D_t

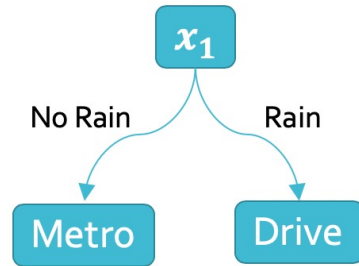
How to reweight the distribution and obtain D_{t+1}

How to perform weighted aggregation

AdaBoost Algorithm

How to learn g_t from D_t

- Decision stump



How to reweight D_{t+1}

- Make $E_{in}^{(D_{t+1})}(g_t) = 0.5$
- So g_t and g_{t+1} are “diverse”

How to weighted aggregation

- More weights on better g_t
- Lower ϵ_t : proxy for better g_t

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

Theoretical Properties of AdaBoost

- See [Freund & Schapire's Tutorial](#) for more discussion
- The training error of AdaBoost converges fast
 - Let $\gamma_t = \frac{1}{2} - \epsilon_t$ (how good each weak learner is better than random guessing)
 - $E_{in} \leq e^{-2 \sum_{t=1}^T \gamma_t^2}$
- Generalization error
 - VC analysis gives us $E_{out} \leq E_{in} + \tilde{O} \left(\sqrt{\frac{T d_{vc}}{m}} \right)$

d_{vc} is the VC dimension of the weak learner
 - It seems as T goes large, overfitting could happen
 - Empirically, AdaBoost is relatively robust to overfitting
 - There are some more delicate analysis using the idea of **margins** to explain why

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.

Brief Discussion on Gradient Boosting

Gradient boosting is **safe to skip** for Exam 2

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



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

- 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 practice
 - See CASI 17.4 and the reference in CASI P.350 for more discussion

[Safe to Skip]

Gradient Boosting

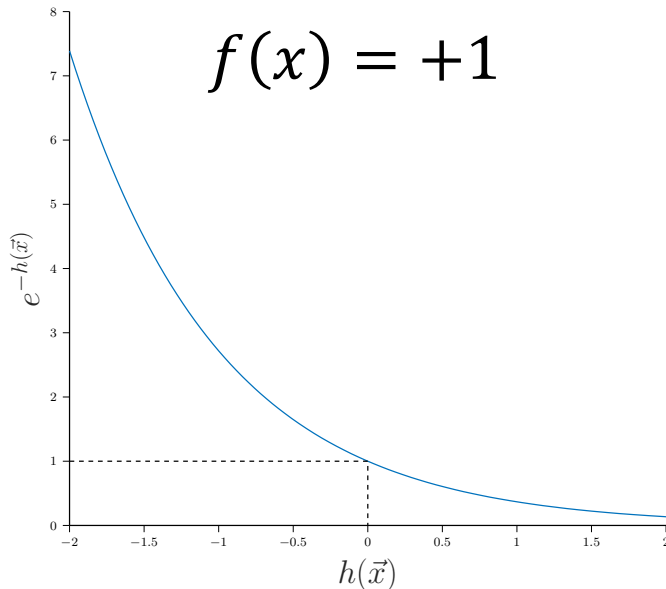
Initialize $G(\vec{x}) = 0$

For $t = 1, \dots, T$

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

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

- AdaBoost is a special case of Gradient Boosting
 - minimizing the exponential **loss** ($e_{\text{exp}}(h(\vec{x}), y) = e^{-yh(\vec{x})}$)
 - 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 movie 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 movie 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 the label of \vec{x} according to its nearest neighbor in D
 - Given $D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_N, y_N)\}; y_n \in \{+1, -1\}$
 - Let $\vec{x}_{[1]}$ be \vec{x} 's nearest neighbor in D , i.e., the closest point to \vec{x} in D
 - Similarly, let $\vec{x}_{[i]}$ be the i^{th} 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

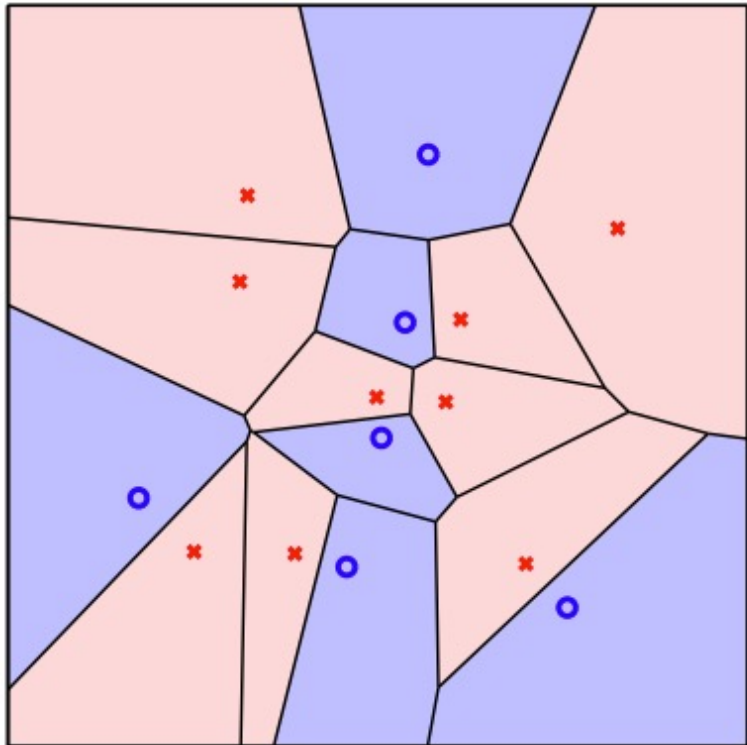
- Machine Learning Whiz Kid (MLWK) proposes the learning algorithm
 - Given D , define the learned hypothesis g as follows

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

- In our discussion earlier, MLWK leads to
 - $E_{in} = 0$, infinite VC dimension, bad generalization
- Why nearest neighbor might be a good model?

Nearest Neighbor

Decision boundary of $g(\vec{x})$



- 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 talked about so far? Why we care?
- What we really care about is E_{out}
 - VC analysis: $E_{out} \leq E_{in} + \text{Generalization error}$
 - 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 $N \rightarrow \infty$, with high probability,

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

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

Proof Sketch of 2-Optimality ($E_{out} \leq 2E_{out}^*$)

- 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 label distributions
 - It is the underlying assumption for nearest neighbor to work
- Let $g^*(\vec{x})$ be the optimal hypothesis (output a binary prediction)
 - $g^*(\vec{x}) = \begin{cases} +1 & \text{if } \pi(\vec{x}) \geq \frac{1}{2} \\ -1 & \text{otherwise} \end{cases}$
 - 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 $N \rightarrow \infty$
 - A new point is “very close” to its nearest neighbor in D
 - $\pi(\vec{x}) \approx \pi(\vec{x}_{[1]})$
 - The error of nearest neighbor hypothesis on a new point is

Proof Sketch of 2-Optimality

- $E_{out}^* = \mathbb{E}_{\vec{x}}[e(g^*(\vec{x}), y)] = \mathbb{E}_{\vec{x}}[\min\{\pi(\vec{x}), 1 - \pi(\vec{x})\}]$
- 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 $N \rightarrow \infty$
 - A new point is “very close” to its nearest neighbor in D
 - $\pi(\vec{x}) \approx \pi(\vec{x}_{[1]})$
 - The error of nearest neighbor hypothesis on a new point is

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

Informal intuitions to summarize the proof:

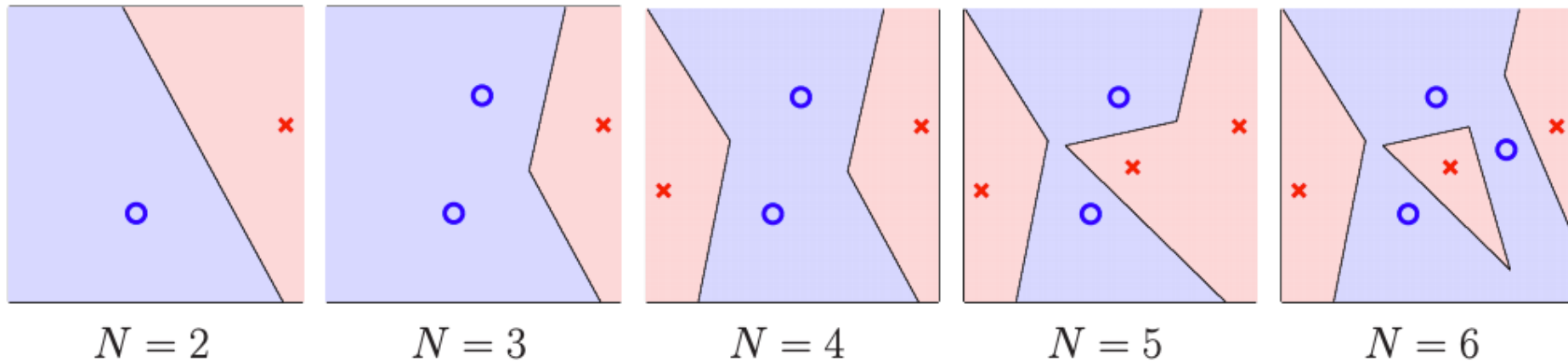
Assumption for nearest neighbor:

- Target function is continuous
- That is, nearby points have similar label distributions

As N goes large, there are “close enough” points for prediction

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

k -Nearest Neighbor

”Stabilize” the Hypothesis

- Instead of a ”single” nearest neighbor
 - Making predictions according to k nearest neighbors
- k -nearest neighbor (K-NN)
 - $g(\vec{x}) = \text{sign}\left(\sum_{i=1}^k y_{[i]}(\vec{x})\right)$
 - (k is often chosen to be an odd number 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 $k(N) \rightarrow \infty$ as $N \rightarrow \infty$ and $\frac{k(N)}{N} \rightarrow 0$ as $N \rightarrow \infty$
 - Then $E_{in}(g) \rightarrow E_{out}(g)$ and $E_{out}(g) \rightarrow E_{out}(g^*)$
 - 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?
 - $d = 1$: $\ell = 0.01$
 - $d = 2$: $\ell = 0.1$
 - $\ell^d = 0.01 \Rightarrow d = 100, \ell \approx 0.95$

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

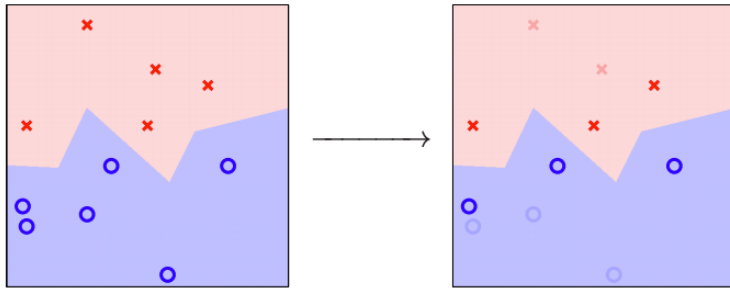
- k -Nearest Neighbor is computationally demanding
 - Need to store all data points: space complexity $O(Nd)$
 - For each prediction for \vec{x}
 - Calculate the distance to every point in D
 - Find the k closest points
 - Time complexity $O(Nd + N \log k)$
- There are still ongoing research to address 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

- Reduce the number of data points
- Store the data in some data structure to speed up searching

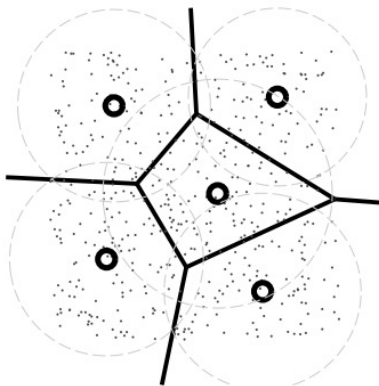
Computational Issues

- 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