Recall

- Boosting is another ensemble method (like bagging) that combines the predictions of multiple hypotheses.

- Aims to reduce the bias of a “weak” or highly biased hypothesis set (can also reduce variance).
AdaBoost

- Intuition: iteratively reweight inputs, giving more weight to inputs that are difficult-to-predict correctly

- Analogy:
  - You all have to take the midterm again 😨 ...
  - ... but you’re going to be taking it one at a time.
  - After you finish, you get to tell the next person the questions you struggled with.
  - Hopefully, they can cover for you because...
  - ... if “enough” of you get a question right, you’ll all receive full credit for that problem
• Input: \( \mathcal{D} (\mathcal{Y} = \{-1, +1\}) \), \( T \)

• Initialize input weights: \( \omega_1^{(0)}, \ldots, \omega_n^{(0)} = \frac{1}{n} \)

• For \( t = 1, \ldots, T \)
  1. Train a weak learner (hypothesis), \( h_t \), by minimizing the weighted training error
  2. Compute the weighted training error of \( h_t \):

\[
\epsilon_t = \sum_{i=1}^{n} \omega_i^{(t-1)} [h_t(x_i^+) \neq y_i]
\]

  3. Compute the "importance" of \( h_t \):

\[
\alpha_t = \frac{1}{2} \log \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)
\]

  4. Update the weights:

\[
\omega_i^{(t)} = \frac{\omega_i^{(t-1)} e^{-\alpha_t} \mathbb{I}(h_t(x_i^+) = y_i)}{Z_t} \times \left\{ \begin{array}{ll}
    e^{\alpha_t} & \text{if } h_t(x_i^+) \neq y_i \\
    e^{-\alpha_t} & \text{if } h_t(x_i^+) = y_i
\end{array} \right.
\]

• Output: an aggregated hypothesis

\[
g_T(\vec{x}) = \text{sign}(H_T(\vec{x})) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(\vec{x}) \right)
\]
\[
\frac{1}{n} \sum_{i=1}^{n} \left[ y_i \neq g_T(x_i) \right] \leq \frac{1}{n} \sum_{i=1}^{n} e^{-y_i H_T(x_i)}
\]

\[
= \prod_{t=1}^{T} Z_t
\]

\[
= \prod_{t=1}^{T} 2 \sqrt{\epsilon_t(1 - \epsilon_t)} \to 0 \text{ as } T \to \infty
\]

\[
(\text{as long as } \epsilon_t < \frac{1}{2} \ \forall \ t)
\]
Out-of-sample Error

· For AdaBoost, with high probability:

\[ E_{out}(g) \leq E_{in}(g) + \tilde{O} \left( \sqrt{\frac{d_{vc}(\mathcal{H})T}{n}} \right) \]

where \( d_{vc}(\mathcal{H}) \) is the VC-dimension of the weak learners,
\( T \) is the number of weak learners,
\( n \) is the number of training data points

· Empirical results indicate that increasing \( T \) does not lead to overfitting as this bound would suggest
• After running Adaboost for $T$ rounds, the returned hypothesis is

$$g_T(\vec{x}) = \text{sign}(H_T(\vec{x})) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(\vec{x}) \right)$$

• The margin of training point $(\vec{x}_i, y_i)$ is defined as:

$$m(\vec{x}_i, y_i) = \frac{y_i H_T(\vec{x})}{\sum_{t=1}^{T} \alpha_t}$$

• The margin of $(\vec{x}_i, y_i)$ is positive if $g_T$ is correct at predicting $\vec{x}_i$

• The margin can be interpreted as how confident $g_T$ is in its prediction: the bigger the margin, the more confident
For AdaBoost, with high probability:

\[
E_{out}(g) \leq \frac{1}{n} \sum_{i=1}^{n} [m(x_i, y_i) \leq \theta] + \tilde{O}\left(\sqrt{\frac{d_{vc}(\mathcal{H})}{n\theta^2}}\right)
\]

where \(d_{vc}(\mathcal{H})\) is the VC-dimension of the weak learners, 
\(n\) is the number of training data points, 
\(\theta > 0\) is an arbitrary parameter.
We have learned/will learn about lots of complicated machine learning hypotheses; how about something simple?
The Duck Test

Duck test

From Wikipedia, the free encyclopedia

For the use of "the duck test" within the Wikipedia community, see Wikipedia:DUCK.

The duck test is a form of abductive reasoning. This is its usual expression:

If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck.
The Duck Test for Machine Learning

- Classify a point as the label of the “most similar” training point
• Euclidean distance: \( d(\vec{x}, \vec{x}') = \|\vec{x} - \vec{x}'\| = \sum_{j=1}^{D} (x_j - x'_j)^2 \)

• Alternatives:
  • Cosine similarity: \( d(\vec{x}, \vec{x}') = \frac{\vec{x}^T \vec{x}'}{||\vec{x}|| ||\vec{x}'||} \)
  • Mahalanobis distance: \( d(\vec{x}, \vec{x}') = (\vec{x} - \vec{x}')^T \Sigma^{-1} (\vec{x} - \vec{x}') \)

where \( \Sigma \) is any positive semidefinite matrix
Nearest Neighbor

- Given $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ and a point $\vec{x}$
- Let $\vec{x}_{[1]}(\vec{x})$ be $\vec{x}$’s nearest neighbor i.e. the closest point to $\vec{x}$ in $\mathcal{D}$
- Similarly, let $\vec{x}_{[2]}(\vec{x})$ be the 2nd closest point to $\vec{x}$ in $\mathcal{D}$ and let $\vec{x}_{[i]}(\vec{x})$ be the $i^{th}$ closest point to $\vec{x}$ in $\mathcal{D}$

\[
\begin{align*}
d(\vec{x}, \vec{x}_{[1]}(\vec{x})) & \leq d(\vec{x}, \vec{x}_{[2]}(\vec{x})) \\
& \leq \cdots \\
& \leq d(\vec{x}, \vec{x}_{[n]}(\vec{x}))
\end{align*}
\]
The Nearest Neighbor Hypothesis

\[ g(\vec{x}) = y_{[1]}(\vec{x}) \]
The Nearest Neighbor Hypothesis

\[ g(\vec{x}) = y_{[1]}(\vec{x}) \]
The Nearest Neighbor Hypothesis

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

- Requires no training!

- Always has zero training error!

- Always has zero training error...
Generalization of Nearest Neighbor

- Claim: $E_{out}$ for the nearest neighbor hypothesis is not much worse than the best possible $E_{out}$!

- Formally: under certain conditions, with high probability, $E_{out}(g) \leq 2E_{out}(g^*)$ as $n \to \infty$

- Proof:
  - Assume a binary classification problem: $Y = \{-1, +1\}$
  - Assume labels are noisy: let $\pi(\hat{x}) = P\{y = +1|\hat{x}\}$
  - Assume $\pi(\hat{x})$ is continuous
  - As $n \to \infty$, $\hat{x}_{[1]}(\hat{x}) \to \hat{x} \implies \pi(\hat{x}_{[1]}(\hat{x})) \to \pi(\hat{x})$
Generalization of Nearest Neighbor

• Proof (Continued):

\[ E_{out}(g) = \mathbb{E}_{\tilde{x}}[\mathbb{I}[g(\tilde{x}) \neq y]] = P\{g(\tilde{x}) \neq y\} \]

\[ = P\{g(\tilde{x}) = +1 \cap y = -1\} \]

\[ + P\{g(\tilde{x}) = -1 \cap y = +1\} \]

\[ = \pi(\tilde{x}_{[1]}(\tilde{x})) \left(1 - \pi(\tilde{x})\right) \]

\[ + \left(1 - \pi(\tilde{x}_{[1]}(\tilde{x}))\right) \pi(\tilde{x}) \]

\[ \to \pi(\tilde{x}) \left(1 - \pi(\tilde{x})\right) + \left(1 - \pi(\tilde{x})\right) \pi(\tilde{x}) \]

\[ = 2\pi(\tilde{x}) \left(1 - \pi(\tilde{x})\right) \]

\[ \leq 2 \min\left(\pi(\tilde{x}), (1 - \pi(\tilde{x}))\right) \]
Generalization of Nearest Neighbor

• Claim: $E_{out}$ for the nearest neighbor hypothesis is not much worse than the best possible $E_{out}$!

• Formally: with high probability, $E_{out}(g) \leq 2E_{out}(g^*)$ as $n \to \infty$.

• Interpretation: half of the data’s predictive power is in the nearest neighbor!
Self-Regularization

The nearest neighbor hypothesis can only be complex when there is more data.
**$k$-Nearest Neighbors ($k$NN)**

- Classify a point as the most common label among the labels of the $k$ nearest training points

- If we have a binary classification problem and $k$ is odd:

$$g(\tilde{x}) = \text{sign} \left( \sum_{i=1}^{k} y_{[i]}(\tilde{x}) \right)$$

- When $k = 1$, $g$ is the nearest neighbor hypothesis

- When $k = n$, $g$ always predicts the most common label in the training dataset
Setting $k$

- When $k = 1$, $g$ is the nearest neighbor hypothesis
  - many, complicated decision boundaries
  - may overfit

- When $k = n$, $g$ always predicts the most common label in the training dataset
  - no decision boundaries
  - may underfit

- $k$ controls the complexity of the hypothesis set $\Rightarrow k$ affects how well the learned hypothesis will generalize
Setting $k$

- Idea: make $k$ a function of $n$, $k(n)$

- 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 $E_{out}(g) \to E_{out}(g^*)$

- Practical rules of thumb:
  - $k = 3$
  - $k = \lfloor \sqrt{n} \rfloor$
  - Cross-validation
$k$NN Pros and Cons

- **Pros:**
  - Intuitive / explainable
  - No training / retraining
  - Provably near-optimal in terms of $E_{out}$

- **Cons:**
  - Computationally expensive
    - Always needs to store all data: $O(nD)$
    - Computing $g(\vec{x})$ requires computing $d(\vec{x}, \vec{x}') \forall \vec{x}' \in \mathcal{D}$ and finding the $k$ closest points: $O(nD + n \log(k))$
  - Suffers from the “curse of dimensionality”
Curse of Dimensionality

- The fundamental assumption of kNN is that “similar” points or points close to one another should have the same label.
- The closer two points are, the more confident we can be that they will have the same label.
- As the number of dimensions the input has grows, the less likely it is that two random points will be close.
- As the number of dimensions the input has grows, it takes more points to “cover” the input space.
Suppose you independently draw two one-dimensional points between 0 and 1 uniformly at random:

\[
\mathbb{E}[d(x, x')] = \mathbb{E}[(x - x')^2] = \mathbb{E}[x^2 - 2xx' + x'^2] = \mathbb{E}[x^2] - 2\mathbb{E}[x]\mathbb{E}[x'] + \mathbb{E}[x'^2] = 2\mathbb{E}[x^2] - 2\mathbb{E}[x]^2 = 2 \left( \frac{1}{3} \right) - 2 \left( \frac{1}{2} \right)^2 = \frac{1}{6}
\]
Suppose you independently draw two two-dimensional points in the unit square uniformly at random:

\[
\begin{align*}
\mathbb{E}[d(x, x')] &= \mathbb{E}[(x_1 - x'_1)^2 + (x_2 - x'_2)^2] \\
&= 2 \mathbb{E}[(x_1 - x'_1)^2] \\
&= 2 \left( \frac{1}{6} \right) = \frac{1}{3}
\end{align*}
\]
Suppose you independently draw two three-dimensional points in the unit cube uniformly at random:

\[
\begin{align*}
\mathbb{E}[d(x, x')] &= \mathbb{E}[(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + (x_3 - x'_3)^2] \\
&= 3\mathbb{E}[(x_1 - x'_1)^2] \\
&= 3 \left( \frac{1}{6} \right) = \frac{1}{2}
\end{align*}
\]
• Assume all dimensions of the input are i.i.d.

\[ P(\vec{x}) = \prod_{j=1}^{D} p(x_j) \]

• Given \( n \) inputs, \( \{\vec{x}_1, \ldots, \vec{x}_n\} \), and a random query point, \( \vec{x}^* \), all drawn i.i.d. from the distribution above, let

\[ d_+ = \max_{\vec{x} \in \{\vec{x}_1, \ldots, \vec{x}_n\}} d(\vec{x}, \vec{x}^*) \quad \text{and} \quad d_- = \min_{\vec{x} \in \{\vec{x}_1, \ldots, \vec{x}_n\}} d(\vec{x}, \vec{x}^*) \]

• Then:

\[
\lim_{D \to \infty} \mathbb{E} \left[ \frac{d_+ - d_-}{d_-} \right] \to 0
\]
Curing the Curse of Dimensionality

- More data
- Fewer dimensions
- Blessing of non-uniformity: data from the real world is rarely uniformly distributed across the input space