Announcements

- **Mid-semester grades are available on S3.**
  - These grades are based on the score in the mid-term and first 3 HWs.
  - This weights the homework much more than your final grade will.
  - There is still plenty of time to improve!
  - Come talk to us if you are concerned about your grade.

- Homework 5 due date extended to Saturday by 11:59 pm.
- Homework 6 will be released later this week
- Homework 7 will be a mini-project in TensorFlow/Pytorch. More details coming soon!
- There will be recitation this Friday
Outline

1. Recap of Decision Trees
2. Overfitting in Decision Trees
3. Bagging and Random Forests
4. Boosting Methods: AdaBoost
5. Derivation of AdaBoost
Recap of Decision Trees
Suppose the 3 classes are 3 possible treatments for an illness and you recommend treatment 1.

The patient sues you and you need to explain the reasoning behind the decision in court. What would you say?

Saying “I recommended treatment 1 because $\mathbf{w}_{(1)}^\top \mathbf{x} > 0$ and $\mathbf{w}_{(2)}^\top \mathbf{x} < 0$” won’t convince the judge.
Need Interpretable decision boundaries

- Should be able to explain the reasoning in clear terms, for eg. “I always recommend treatment 1 when a patient has fever $\geq 100^\circ F$”
- The rules that you use to make decisions can be easily used by a lay-person without performing complex computations
- Decision trees can provide such simple decision rules
Decision Trees

Medical treatment

- Fever
  - $T > 100$
  - $T < 100$
    - Treatment #1
    - Muscle Pain
      - High
      - Low
      - Treatment #2
      - Treatment #3

- Salary in a company

- Degree
  - High School
  - College
  - Graduate
    - Work Experience
      - < 5yr
      - > 5yr
        - $X_1$
        - $X_2$
        - $X_3$
        - $X_4$
        - $X_5$
        - $X_6$

- Other examples: fault detection in manufacturing systems, student admissions decisions, jail/parole decisions
A tree partitions the feature space along the axes
Learning a tree model

Three things to learn:

1. The structure of the tree – order of features to split
2. The threshold values ($\theta_i$) – these can be categorical
3. The values for the leafs ($A, B, ...$) – labels that we want to assign to that part of the feature space
Learning a tree model

Three things to learn:

1. The structure of the tree – order of features to split

\[ x_1 > \theta_1 \]
\[ x_2 \leq \theta_2 \]
\[ x_1 \leq \theta_4 \]
\[ x_2 > \theta_3 \]
Learning a tree model

Three things to learn:

1. The structure of the tree – order of features to split
2. The threshold values \( \theta_i \) – these can be categorical
Learning a tree model

Three things to learn:

1. The structure of the tree – order of features to split
2. The threshold values ($\theta_i$) – these can be categorical
3. The values for the leafs ($A, B, \ldots$) – labels that we want to assign to that part of the feature space
Example: Choosing whether you want to wait at a restaurant

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Target</th>
<th>Est</th>
<th>Will Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt Bar Fri Hun Pat Price Rain Res Type Est</td>
<td>T F F T Some $$$ F T French 0–10</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>T F F T Full $ F F Thai 30–60</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F T F F Some $ F F Burger 0–10</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T F T T Full $ F F Thai 10–30</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T F T F Full $$$ F T French &gt;60</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F T F T Some $$ T T Italian 0–10</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F T F F None $ T F Burger 0–10</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F F F T Some $$ T T Thai 0–10</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F F F F Full $ T F Burger &gt;60</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T T T T Full $$$ F T Italian 10–30</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F F F T None $ F F Thai 0–10</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T T T T Full $ F F Burger 30–60</td>
<td>T</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Use the attributes to decide whether to wait (T) or not wait (F)
Which attribute to split first?

- Patron is a better choice – gives more information to help distinguish between the labels.
- Intuition: Like playing 20 questions and choosing carefully which question to ask first.
- Idea: use information gain to choose which attribute to split.
Which attribute to split first?

- Patron is a better choice – gives more information to help distinguish between the labels
- Intuition: Like playing 20 questions and choosing carefully which question to ask first
- Idea: use information gain to choose which attribute to split
How to measure information gain $I(X; Y)$?

**Definition (Entropy)**

If a random variable $Y$ takes $K$ possible values, $a_1, a_2...a_K$ then

$$H[Y] = -\sum_{i=1}^{K} \Pr(Y = a_i) \log \Pr(Y = a_i) \quad (0 \log 0 \triangleq 0)$$

**Definition (Conditional Entropy)**

Given two random variables $X$ and $Y$

$$H[Y|X] = \sum_{k} P(X = a_k)H[Y|X = a_k] \quad (1)$$

**Information gain is defined as**

$$I(X; Y) = H[Y] - H[Y|X] \quad (2)$$
Which attribute to split?

• Let us compute the information gain
  \[ I(X; Y) = H[Y] - H[Y|X] \]
  for Patron and Type

• When \( H[Y] \) is fixed, we need only to compare conditional entropy
Information Gain if we split ”Patron”

- \( H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1 \text{ bit} \)
Information Gain if we split "Patron"

- \( H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1 \text{ bit} \)
- \( H(Y|X = \text{none}) = 0 \)
Information Gain if we split "Patron"

- $H(Y) = -\frac{6}{12} \log\frac{6}{12} - \frac{6}{12} \log\frac{6}{12} = 1$ bit
- $H(Y|X = \text{none}) = 0$
- $H(Y|X = \text{some}) = 0$
Information Gain if we split "Patron"

- \( H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1 \) bit
- \( H(Y|X = \text{none}) = 0 \)
- \( H(Y|X = \text{some}) = 0 \)
- \( H(Y|X = \text{full}) = -\left(\frac{2}{2+4} \log \frac{2}{2+4} + \frac{4}{2+4} \log \frac{4}{2+4}\right) \approx 0.9 \) bits
Information Gain if we split "Patron"

- $H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1$ bit
- $H(Y|X = none) = 0$
- $H(Y|X = some) = 0$
- $H(Y|X = full) = -\left(\frac{2}{2+4} \log \frac{2}{2+4} + \frac{4}{2+4} \log \frac{4}{2+4}\right) \approx 0.9$ bits
- Thus the conditional entropy is

$$H(Y|X) = \left(\frac{2}{12} \times 0 + \frac{4}{12} \times 0 + \frac{6}{12} \times 0.9\right) = 0.45 \text{ bits}$$
Information Gain if we split "Patron"

- \( H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1 \) bit
- \( H(Y|X = \text{none}) = 0 \)
- \( H(Y|X = \text{some}) = 0 \)
- \( H(Y|X = \text{full}) = - \left( \frac{2}{2+4} \log \frac{2}{2+4} + \frac{4}{2+4} \log \frac{4}{2+4} \right) \approx 0.9 \) bits
- Thus the conditional entropy is
  \[
  H(Y|X) = \left( \frac{2}{12} \times 0 + \frac{4}{12} \times 0 + \frac{6}{12} \times 0.9 \right) = 0.45 \text{ bits}
  \]
- Information Gain \( I(X; Y) = 1 - 0.45 = 0.55 \text{ bits} \)
Information Gain if we split ”Type”

- $H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1 \text{ bit}$
Information Gain if we split "Type"

\[ H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1 \text{ bit} \]

\[ H(Y|X = \text{french}) = \log 2 = 1 \text{ bit} \]
Information Gain if we split "Type"

- $H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1$ bit
- $H(Y|X = french) = \log 2 = 1$ bit
- $H(Y|X = italian) = \log 2 = 1$ bit

Thus, we should split on "Patron" and not "Type"
Information Gain if we split "Type"

- $H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1$ bit
- $H(Y|X = french) = \log 2 = 1$ bit
- $H(Y|X = italian) = \log 2 = 1$ bit
- $H(Y|X = thai) = \log 2 = 1$ bit

Thus the conditional entropy is $H(Y|X) = \frac{1}{12} \times 1 + \frac{1}{12} \times 1 + \frac{4}{12} \times 1 + \frac{4}{12} \times 1 = 1$ bit

Information Gain $I(X;Y) = 1 - 1 = 0$ bits

Thus, we should split on "Patron" and not "Type"
Information Gain if we split "Type"

- \( H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1 \text{ bit} \)
- \( H(Y|X = \text{french}) = \log 2 = 1 \text{ bit} \)
- \( H(Y|X = \text{italian}) = \log 2 = 1 \text{ bit} \)
- \( H(Y|X = \text{thai}) = \log 2 = 1 \text{ bit} \)
- \( H(Y|X = \text{burger}) = \log 2 = 1 \text{ bit} \)

Thus, we should split on "Patron" and not "Type"
Information Gain if we split "Type"

- \( H(Y) = - \frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1 \text{ bit} \)
- \( H(Y|X = \text{french}) = \log 2 = 1 \text{ bit} \)
- \( H(Y|X = \text{italian}) = \log 2 = 1 \text{ bit} \)
- \( H(Y|X = \text{thai}) = \log 2 = 1 \text{ bit} \)
- \( H(Y|X = \text{burger}) = \log 2 = 1 \text{ bit} \)
- Thus the conditional entropy is
  \[ H(Y|X) = \frac{2}{12} \times 1 + \frac{2}{12} \times 1 + \frac{4}{12} \times 1 + \frac{4}{12} \times 1 = 1 \text{ bit} \]
Information Gain if we split ”Type”

- $H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1 \text{ bit}$
- $H(Y|X = \text{french}) = \log 2 = 1 \text{ bit}$
- $H(Y|X = \text{italian}) = \log 2 = 1 \text{ bit}$
- $H(Y|X = \text{thai}) = \log 2 = 1 \text{ bit}$
- $H(Y|X = \text{burger}) = \log 2 = 1 \text{ bit}$
- Thus the conditional entropy is
  
  $H(Y|X) = \frac{2}{12} \times 1 + \frac{2}{12} \times 1 + \frac{4}{12} \times 1 + \frac{4}{12} \times 1 = 1 \text{ bit}$

- Information Gain $I(X; Y) = 1 - 1 = 0 \text{ bits}$
Information Gain if we split "Type"

- \( H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1 \) bit
- \( H(Y|X = \text{french}) = \log 2 = 1 \) bit
- \( H(Y|X = \text{italian}) = \log 2 = 1 \) bit
- \( H(Y|X = \text{thai}) = \log 2 = 1 \) bit
- \( H(Y|X = \text{burger}) = \log 2 = 1 \) bit
- Thus the conditional entropy is
  \[ H(Y|X) = \frac{2}{12} \times 1 + \frac{2}{12} \times 1 + \frac{4}{12} \times 1 + \frac{4}{12} \times 1 = 1 \] bit
- Information Gain \( I(X; Y) = 1 - 1 = 0 \) bits

Thus, we should split on "Patron" and not "Type"
Information Gain if we split ”Type”

- $H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1$ bit
- $H(Y|X = \text{french}) = \log 2 = 1$ bit
- $H(Y|X = \text{italian}) = \log 2 = 1$ bit
- $H(Y|X = \text{thai}) = \log 2 = 1$ bit
- $H(Y|X = \text{burger}) = \log 2 = 1$ bit
- Thus the conditional entropy is
  \[ H(Y|X) = \frac{2}{12} \times 1 + \frac{2}{12} \times 1 + \frac{4}{12} \times 1 + \frac{4}{12} \times 1 = 1 \text{ bit} \]
- Information Gain $I(X; Y) = 1 - 1 = 0$ bits

Thus, we should split on ”Patron” and not ”Type”
• We do not need to split on "non" and "some" since the decision is deterministic for the training data.
• Thus, we split on "full" next
We will look only at the 6 instances with
Patrons == Full
Greedily we build the tree and get this
1. Recap of Decision Trees

2. Overfitting in Decision Trees

3. Bagging and Random Forests

4. Boosting Methods: AdaBoost

5. Derivation of AdaBoost
Overfitting in Decision Trees
What is the optimal Tree Depth?

- We need to be careful to pick an appropriate tree depth
  - If the tree is too deep, we can overfit
  - If the tree is too shallow, we underfit

Two approaches to avoid overfitting:

- **Fixing Max Depth:** Stop splitting the tree after a few levels. The max depth $D$ is a hyper-parameter that can be tuned using cross-validation

- **Pruning Deep Decision Trees:** Create a deep tree to fit the training data perfectly and then prune it. Here, the hyper-parameter to be tuned is the number of terminal/leaf nodes $T$. 
Here the depth is set to $D = 2$

Tune $D$ using cross-validation
Pruning Deep Decision Trees

- Prune the leaf nodes that cause minimize increase in the classification error for the training dataset
- Tune the number of terminal/leaf nodes $T$ using cross-validation
How to classify with a shallow/pruned decision tree?

- If we stop here, not all training samples would be classified correctly.
- More importantly, how do we classify a new instance?
- We label the leaves of this smaller tree with the majority of training sample’s labels.
Advantages of using trees

• Can be interpreted by humans (as long as the tree is not too big)
• Handles both numerical and categorical data
• Compact representation: unlike Nearest Neighbors we don’t need training data at test time
• But, like NN, decision trees are non-parametric because the number of parameters depends on the data
• Building block for ensemble methods (e.g., boosting)
Advantages of using trees

- Can be interpreted by humans (as long as the tree is not too big)
- Handles both numerical and categorical data
- Compact representation: unlike Nearest Neighbors we don’t need training data at test time
- But, like NN, decision trees are *non-parametric* because the number of parameters depends on the data
- Building block for ensemble methods (e.g., boosting)
Summary of Decision Trees

Advantages of using trees

• Can be interpreted by humans (as long as the tree is not too big)
• Handles both numerical and categorical data
• Compact representation: unlike Nearest Neighbors we don’t need training data at test time
• But, like NN, decision trees are non-parametric because the number of parameters depends on the data
• Building block for ensemble methods (eg. boosting)

Disadvantages
Summary of Decision Trees

Advantages of using trees

- Can be interpreted by humans (as long as the tree is not too big)
- Handles both numerical and categorical data
- Compact representation: unlike Nearest Neighbors we don’t need training data at test time
- But, like NN, decision trees are non-parametric because the number of parameters depends on the data
- Building block for ensemble methods (eg. boosting)

Disadvantages

- Heuristic training techniques
Summary of Decision Trees

Advantages of using trees

- Can be interpreted by humans (as long as the tree is not too big)
- Handles both numerical and categorical data
- Compact representation: unlike Nearest Neighbors we don’t need training data at test time
- But, like NN, decision trees are non-parametric because the number of parameters depends on the data
- Building block for ensemble methods (eg. boosting)

Disadvantages

- Heuristic training techniques
  - Finding partition of space that minimizes empirical error is NP-hard
  - We resort to greedy approaches with limited theoretical underpinnings
- Tend to be unstable – small changes in training data can drastically impact the structure
Outline

1. Recap of Decision Trees
2. Overfitting in Decision Trees
3. Bagging and Random Forests
4. Boosting Methods: AdaBoost
5. Derivation of AdaBoost
Ensemble Methods: Combining the decision of several weak classifiers can make one strong classifier.
Motivation: Fighting the Bias-Variance Tradeoff

- Simple (a.k.a weak) learners such as Naive Bayes, logistic regression, decision stumps (shallow decision trees)

- **Pros:** Low variance, don’t overfit
- **Cons:** High Bias, can’t fit complex boundaries

Can we improve weak learners?

**Idea:** Train several weak learners and take an average or majority vote of their outputs
• Instead of learning a single (weak) classifier, learn many weak classifiers, preferably those that are good at different parts of the input spaces

• **Predicted Class:** (Weighted) Average or Majority of output of the classifiers

• Strength in Diversity!

\[
H: X \rightarrow Y (-1,1) \\
H(X) = h_1(X) + h_2(X) \\
H(X) = \text{sign} \left( \sum_{t} \alpha_t h_t(X) \right)
\]
Next, we will learn the following ensemble methods:

- Bagging or Bootstrap Aggregation
- Random Forests
- AdaBoost

\[ H(X) \rightarrow Y \; (-1, 1) \]
\[ H(X) = h_1(X) + h_2(X) \]
\[ H(X) = \text{sign} \left( \sum_t \alpha_t h_t(X) \right) \]
Bagging and Random Forests
To avoid overfitting a decision tree to a given dataset we can average an ensemble of trees learnt on random subsets of the training data.

**Bagging Trees (Training Phase):**

- For $b = 1, 3, \ldots, B$
  - Choose $n$ training samples $(x_i, y_i)$ from $\mathcal{D}$ uniformly at random
  - Learn a decision tree $h_b$ on these $n$ samples
- Store the $B$ decision trees $h_1, h_2, \ldots, h_B$
- Optimal $B$ (typically in 1000s) chosen using cross-validation

**Bagging Trees (Test Phase):**

- For a test unlabeled example $x$
Bagging or Bootstrap Aggregating

To avoid overfitting a decision tree to a given dataset we can average an ensemble of trees learnt on random subsets of the training data.

**Bagging Trees (Training Phase):**

- For $b = 1, 3, \cdots, B$
  - Choose $n$ training samples $(x_i, y_i)$ from $D$ uniformly at random
  - Learn a decision tree $h_b$ on these $n$ samples
- Store the $B$ decision trees $h_1, h_2, \ldots h_B$
- Optimal $B$ (typically in 1000s) chosen using cross-validation

**Bagging Trees (Test Phase):**

- For a test unlabeled example $x$
- Find the decision from each of the $B$ trees
To avoid overfitting a decision tree to a given dataset we can average an ensemble of trees learnt on random subsets of the training data.

**Bagging Trees (Training Phase):**

- For $b = 1, 3, \ldots, B$
  - Choose $n$ training samples $(x_i, y_i)$ from $\mathcal{D}$ uniformly at random
  - Learn a decision tree $h_b$ on these $n$ samples
- Store the $B$ decision trees $h_1, h_2, \ldots h_B$
- Optimal $B$ (typically in 1000s) chosen using cross-validation

**Bagging Trees (Test Phase):**

- For a test unlabeled example $x$
- Find the decision from each of the $B$ trees
- Assign the majority label as the label for $x$
• We get different splits and thresholds for different $b$
• Predict the label assigned by majority of the $B$ trees
• Reduces variance without increasing bias, thus avoiding overfitting
Random Forests

- **Limitation of Bagging**: If one or more features are very informative, they will be selected by almost every tree in the bag, reducing the diversity.

- **Key Idea on Random Forests**: Reduces correlation between trees in the bag without increasing variance too much.

- Same as bagging in terms of sampling training data.

- Before each split, select $m \leq d$ features at random as candidates for splitting $m \sim \sqrt{d}$.

- Take majority vote of $B$ such trees.
Random Forests

Increasing \( m \), the number of splitting candidates chosen increases the correlation among the trees in the bag.
Increasing $m$ decreases the bias but increases variance in the ensemble classifier.
1. Recap of Decision Trees

2. Overfitting in Decision Trees

3. Bagging and Random Forests

4. Boosting Methods: AdaBoost

5. Derivation of AdaBoost
Boosting Methods: AdaBoost
Limitations of Bagging and Random Forests

- Bagging: Significant correlation between trees that are learnt on different training datasets
- Random Forests try to resolve this by doing "feature bagging" but some correlation still remains
- All $B$ trees are given the same weight when taking the average
Limitations of Bagging and Random Forests

- Bagging: Significant correlation between trees that are learnt on different training datasets
- Random Forests try to resolve this by doing "feature bagging" but some correlation still remains
- All $B$ trees are given the same weight when taking the average

**Boosting methods:** Force classifiers to learn on different parts of the feature space, and take their weighted average
Boosting

**High-level idea**: combine a lot of classifiers

- Sequentially construct / identify these classifiers, $h_t(\cdot)$, one at a time
- Use *weak* classifiers to arrive at a complex decision boundary (*strong* classifier), where $\beta_t$ is the contribution of each weak classifier
**High-level idea:** combine a lot of classifiers

- Sequentially construct / identify these classifiers, $h_t(\cdot)$, one at a time
- Use *weak* classifiers to arrive at a complex decision boundary (*strong* classifier), where $\beta_t$ is the contribution of each weak classifier

$$h[x] = \text{sign} \left[ \sum_{t=1}^{T} \beta_t h_t(x) \right]$$
**High-level idea:** combine a lot of classifiers

- Sequentially construct / identify these classifiers, $h_t(\cdot)$, one at a time
- Use *weak* classifiers to arrive at a complex decision boundary (*strong* classifier), where $\beta_t$ is the contribution of each weak classifier

$$h[x] = \text{sign} \left[ \sum_{t=1}^{T} \beta_t h_t(x) \right]$$

**Adaboost Algorithm: Assumptions**

- Black-box routine for constructing a sequence of weak classifiers $h(\cdot)$
- The classifier needs to know how to optimize on *weighted* samples
The Adaboost Algorithm

- Given: $N$ samples $\{x_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers.

  1. Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample.
  2. For $t = 1$ to $T$
     1. Train a weak classifier $h_t(x)$ using current weights $w_t(n)$, by minimizing $\epsilon_t = \sum_n w_t(n) I[y_n \neq h_t(x_n)]$ (the weighted classification error).
     2. Compute contribution for this classifier: $\beta_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$.
     3. Update weights on training points $w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_n h_t(x_n)}$ and normalize them such that $\sum_n w_{t+1}(n) = 1$.
  3. Output the final classifier $h[x] = \text{sign} \left[ \sum_{t=1}^T \beta_t h_t(x) \right]$. 
The Adaboost Algorithm

- Given: $N$ samples $\{x_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
The Adaboost Algorithm

- Given: \( N \) samples \( \{x_n, y_n\} \), where \( y_n \in \{+1, -1\} \), and some way of constructing weak (or base) classifiers
- Initialize weights \( w_1(n) = \frac{1}{N} \) for every training sample
- For \( t = 1 \) to \( T \)
  1. Train a weak classifier \( h_t(x) \) using current weights \( w_t(n) \), by minimizing

\[
\epsilon_t = \sum_n w_t(n) [y_n \neq h_t(x_n)]
\]
The Adaboost Algorithm

- Given: \( N \) samples \( \{x_n, y_n\} \), where \( y_n \in \{+1, -1\} \), and some way of constructing weak (or base) classifiers
- Initialize weights \( w_1(n) = \frac{1}{N} \) for every training sample
- For \( t = 1 \) to \( T \)
  1. Train a weak classifier \( h_t(x) \) using current weights \( w_t(n) \), by minimizing

\[
\epsilon_t = \sum_n w_t(n) \cdot [y_n \neq h_t(x_n)] \quad \text{(the weighted classification error)}
\]

- Output the final classifier

\[ h[x] = \text{sign} \left[ \sum_{t=1}^{T} \beta_t h_t(x) \right] \]
The Adaboost Algorithm

- Given: $N$ samples $\{x_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to $T$
  1. Train a weak classifier $h_t(x)$ using current weights $w_t(n)$, by minimizing
     $$
     \epsilon_t = \sum_n w_t(n) [y_n \neq h_t(x_n)] \quad \text{(the weighted classification error)}
     $$
  2. Compute contribution for this classifier
The Adaboost Algorithm

- Given: \( N \) samples \( \{x_n, y_n\} \), where \( y_n \in \{+1, -1\} \), and some way of constructing weak (or base) classifiers
- Initialize weights \( w_1(n) = \frac{1}{N} \) for every training sample
- For \( t = 1 \) to \( T \)
  1. Train a weak classifier \( h_t(x) \) using current weights \( w_t(n) \), by minimizing
     \[
     \epsilon_t = \sum_{n} w_t(n) \mathbb{I}[y_n \neq h_t(x_n)] \quad \text{(the weighted classification error)}
     \]
  2. Compute contribution for this classifier: \( \beta_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t} \)
The Adaboost Algorithm

- Given: $N$ samples $\{x_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to $T$
  1. Train a weak classifier $h_t(x)$ using current weights $w_t(n)$, by minimizing
     \[ \epsilon_t = \sum_n w_t(n) \cdot [y_n \neq h_t(x_n)] \quad \text{(the weighted classification error)} \]
     \[ \beta_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t} \]
  2. Compute contribution for this classifier
  3. Update weights on training points
The Adaboost Algorithm

• Given: \( N \) samples \( \{x_n, y_n\} \), where \( y_n \in \{+1, -1\} \), and some way of constructing weak (or base) classifiers

• Initialize weights \( w_1(n) = \frac{1}{N} \) for every training sample

• For \( t = 1 \) to \( T \)
  1. Train a weak classifier \( h_t(x) \) using current weights \( w_t(n) \), by minimizing

\[
\epsilon_t = \sum_n w_t(n) \cdot [y_n \neq h_t(x_n)] \quad \text{(the weighted classification error)}
\]

  2. Compute contribution for this classifier: \( \beta_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t} \)

  3. Update weights on training points

\[
w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_n h_t(x_n)}
\]
The Adaboost Algorithm

- Given: \( N \) samples \( \{x_n, y_n\} \), where \( y_n \in \{+1, -1\} \), and some way of constructing weak (or base) classifiers
- Initialize weights \( w_1(n) = \frac{1}{N} \) for every training sample
- For \( t = 1 \) to \( T \)
  1. Train a weak classifier \( h_t(x) \) using current weights \( w_t(n) \), by minimizing
     \[
     \epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(x_n)] \quad \text{(the weighted classification error)}
     \]
  2. Compute contribution for this classifier: \( \beta_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t} \)
  3. Update weights on training points
     \[
     w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_n h_t(x_n)}
     \]
     and normalize them such that \( \sum_n w_{t+1}(n) = 1 \)
The Adaboost Algorithm

- Given: $N$ samples $\{x_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to $T$
  1. Train a weak classifier $h_t(x)$ using current weights $w_t(n)$, by minimizing
     $$\epsilon_t = \sum_n w_t(n) \cdot [y_n \neq h_t(x_n)]$$ (the weighted classification error)
  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$
  3. Update weights on training points
     $$w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_n h_t(x_n)}$$
     and normalize them such that $\sum_n w_{t+1}(n) = 1$
- Output the final classifier
  $$h[x] = \text{sign} \left[ \sum_{t=1}^{T} \beta_t h_t(x) \right]$$
Example

10 data points and 2 features

- The data points are clearly not linearly separable
- In the beginning, all data points have equal weights (the size of the data markers “+” or “-”)

\[ D_1 \]

\[ + \quad + \quad - \]
\[ + \quad - \quad - \]
\[ + \quad - \quad - \]
Example

10 data points and 2 features

- The data points are clearly not linearly separable
- In the beginning, all data points have equal weights (the size of the data markers “+” or “-”)
- Base classifier $h(\cdot)$: horizontal or vertical lines ('decision stumps')
  - Depth-1 decision trees, i.e., classify data based on a single attribute.
Round 1: $t = 1$

\[ h_1 \]

3 misclassified (with circles): $\epsilon = 0.30$, $\beta = 0.42$.

Weights recomputed; the 3 misclassified data points receive larger weights.
Round 1: \( t = 1 \)

- 3 misclassified (with circles): \( \epsilon_1 = 0.3 \rightarrow \beta_1 = 0.42 \).
- Weights recomputed; the 3 misclassified data points receive larger weights.
Round 2: $t = 2$

$2 = 0.21$

$2 = 0.65$

Note that $\epsilon^2 \neq 0$.

Data points classified correctly in both rounds have small weights.

• 3 misclassified data points get larger weights.
Round 2: \( t = 2 \)

- 3 misclassified (with circles): \( \epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65 \).
  Note that \( \epsilon_2 \neq 0.3 \) as those 3 data points have weights less than \( 1/10 \)
- 3 misclassified data points get larger weights
- Data points classified correctly in both rounds have small weights
Round 3: $t = 3$

- 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.

- Previously correctly classified data points are now misclassified, hence our error is low; what's the intuition?

- Since they have been consistently classified correctly, this round's mistake will hopefully not have a huge impact on the overall prediction.
Round 3: $t = 3$

- 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.
- Previously correctly classified data points are now misclassified, hence our error is low; what’s the intuition?
Round 3: $t = 3$

- 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.
- Previously correctly classified data points are now misclassified, hence our error is low; what’s the intuition?
  - Since they have been consistently classified correctly, this round’s mistake will hopefully not have a huge impact on the overall prediction.
Final classifier: combining 3 classifiers

\[ H_{\text{final}} = \text{sign}(0.42 + 0.65 + 0.92) \]

- All data points are now classified correctly!
• Given: $N$ samples $\{x_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
• Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
• For $t = 1$ to $T$
  1. Train a weak classifier $h_t(x)$ using current weights $w_t(n)$, by minimizing
     \[ \epsilon_t = \sum_n w_t(n)[y_n \neq h_t(x_n)] \]
Given: $N$ samples $\{x_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers.

Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample.

For $t = 1$ to $T$

1. Train a weak classifier $h_t(x)$ using current weights $w_t(n)$, by minimizing

$$
\epsilon_t = \sum_n w_t(n) [y_n \neq h_t(x_n)] \quad \text{(the weighted classification error)}
$$

2. Compute contribution for this classifier.
Another Look at the Adaboost Algorithm

- Given: $N$ samples $\{x_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to $T$
  1. Train a weak classifier $h_t(x)$ using current weights $w_t(n)$, by minimizing

$$
\epsilon_t = \sum_n w_t(n)[y_n \neq h_t(x_n)] \quad \text{(the weighted classification error)}
$$

  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$
  3. Update weights on training points

$$
w_{t+1}(n) \propto w_t(n)e^{-\beta_ty_nh_t(x_n)}
$$

and normalize them such that $\sum_n w_{t+1}(n) = 1$
- Output the final classifier

$$
h[x] = \text{sign} \left[ \sum_{t=1}^{T} \beta_t h_t(x) \right]
$$
1. Recap of Decision Trees

2. Overfitting in Decision Trees

3. Bagging and Random Forests

4. Boosting Methods: AdaBoost

5. Derivation of AdaBoost
Derivation of AdaBoost
It minimizes a loss function related to classification error.

**Classification loss**

- Suppose we want to have a classifier

\[
h(x) = \text{sign}[f(x)] = \begin{cases} 
1 & \text{if } f(x) > 0 \\
-1 & \text{if } f(x) < 0 
\end{cases}
\]

- One seemingly natural loss function is 0-1 loss:

\[
\ell(h(x), y) = \begin{cases} 
0 & \text{if } yf(x) > 0 \\
1 & \text{if } yf(x) < 0 
\end{cases}
\]

Namely, the function \( f(x) \) and the target label \( y \) should have the same sign to avoid a loss of 1.
Surrogate loss

0 – 1 loss function $\ell(h(x), y)$ is non-convex and difficult to optimize.
We can instead use a surrogate loss – what are examples?
Surrogate loss

0 – 1 loss function $\ell(h(x), y)$ is non-convex and difficult to optimize. We can instead use a surrogate loss – what are examples?

**Exponential Loss**

\[ \ell^{\text{EXP}}(h(x), y) = e^{-yf(x)} \]
Choosing the $t$-th classifier

Suppose a classifier $f_{t-1}(x)$, and want to add a weak learner $h_t(x)$

$$f(x) = f_{t-1}(x) + \beta_t h_t(x)$$

note: $h_t(\cdot)$ outputs $-1$ or $1$, as does sign $[f_{t-1}(\cdot)]$
Choosing the $t$-th classifier

Suppose a classifier $f_{t-1}(x)$, and want to add a weak learner $h_t(x)$

$$f(x) = f_{t-1}(x) + \beta_t h_t(x)$$

Note: $h_t(\cdot)$ outputs $-1$ or $1$, as does sign $[f_{t-1}(\cdot)]$

How can we ‘optimally’ choose $h_t(x)$ and combination coefficient $\beta_t$?

Adaboost greedily **minimizes the exponential loss function**!

$$(h^*_t(x), \beta^*_t) = \arg\min_{(h_t(x), \beta_t)} \sum_n e^{-y_n f(x_n)}$$
Choosing the $t$-th classifier

Suppose a classifier $f_{t-1}(x)$, and want to add a weak learner $h_t(x)$

$$f(x) = f_{t-1}(x) + \beta_t h_t(x)$$

note: $h_t(\cdot)$ outputs $-1$ or $1$, as does $\text{sign}[f_{t-1}(\cdot)]$

How can we ‘optimally’ choose $h_t(x)$ and combination coefficient $\beta_t$?

Adaboost greedily minimizes the exponential loss function!

$$(h^*_t(x), \beta^*_t) = \arg\min_{(h_t(x), \beta_t)} \sum_n e^{-y_nf(x_n)}$$

$$= \arg\min_{(h_t(x), \beta_t)} \sum_n e^{-y_n[f_{t-1}(x_n) + \beta_t h_t(x_n)]}$$
Choosing the $t$-th classifier

Suppose a classifier $f_{t-1}(x)$, and want to add a weak learner $h_t(x)$

$$f(x) = f_{t-1}(x) + \beta_t h_t(x)$$

note: $h_t(\cdot)$ outputs $-1$ or $1$, as does sign $[f_{t-1}(\cdot)]$

How can we ‘optimally’ choose $h_t(x)$ and combination coefficient $\beta_t$?

Adaboost greedily minimizes the exponential loss function!

$$(h^*_t(x), \beta^*_t) = \underset{(h_t(x), \beta_t)}{\text{argmin}} \sum_n e^{-y_nf(x_n)}$$

$$= \underset{(h_t(x), \beta_t)}{\text{argmin}} \sum_n e^{-y_n[f_{t-1}(x_n)+\beta_t h_t(x_n)]}$$

$$= \underset{(h_t(x), \beta_t)}{\text{argmin}} \sum_n w_t(n)e^{-y_n\beta_t h_t(x_n)}$$

where we have used $w_t(n)$ as a shorthand for $e^{-y_n f_{t-1}(x_n)}$
The new classifier

We can decompose the *weighted* loss function into two parts

\[
\sum_n w_t(n) e^{-y_n \beta_t h_t(x_n)}
\]

\[
= \sum_n w_t(n) e^{\beta_t \mathbb{I}[y_n \neq h_t(x_n)]} + \sum_n w_t(n) e^{-\beta_t \mathbb{I}[y_n = h_t(x_n)]}
\]

We have used the following properties to derive the above

- \(y_n h_t(x_n)\) is either 1 or -1 as \(h_t(x_n)\) is the output of a binary classifier
- The indicator function \(\mathbb{I}[y_n = h_t(x_n)]\) is either 0 or 1, so it equals \(1 - \mathbb{I}[y_n \neq h_t(x_n)]\)
The new classifier

We can decompose the *weighted* loss function into two parts

\[
\sum_n w_t(n) e^{-y_n \beta_t h_t(x_n)}
\]

\[
= \sum_n w_t(n) e^{\beta_t} \mathbb{1}[y_n \neq h_t(x_n)] + \sum_n w_t(n) e^{-\beta_t} \mathbb{1}[y_n = h_t(x_n)]
\]

\[
= \sum_n w_t(n) e^{\beta_t} \mathbb{1}[y_n \neq h_t(x_n)] + \sum_n w_t(n) e^{-\beta_t} (1 - \mathbb{1}[y_n \neq h_t(x_n)])
\]
The new classifier

We can decompose the \textit{weighted} loss function into two parts

\[
\sum_n w_t(n) e^{-y_n \beta_t h_t(x_n)}
\]

\[
= \sum_n w_t(n) e^{\beta_t \mathbb{I}[y_n \neq h_t(x_n)]} + \sum_n w_t(n) e^{-\beta_t \mathbb{I}[y_n = h_t(x_n)]}
\]

\[
= \sum_n w_t(n) e^{\beta_t \mathbb{I}[y_n \neq h_t(x_n)]} + \sum_n w_t(n) e^{-\beta_t (1 - \mathbb{I}[y_n \neq h_t(x_n)])}
\]

\[
= (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(x_n)] + e^{-\beta_t} \sum_n w_t(n)
\]

We have used the following properties to derive the above

- $y_n h_t(x_n)$ is either 1 or -1 as $h_t(x_n)$ is the output of a binary classifier
- The indicator function $\mathbb{I}[y_n = h_t(x_n)]$ is either 0 or 1, so it equals $1 - \mathbb{I}[y_n \neq h_t(x_n)]$
Summary

\[(h_t^*(x), \beta_t^*) = \arg\min_{h_t(x), \beta_t} \sum_n w_t(n)e^{-y_n \beta_t h_t(x_n)}\]

\[= \arg\min_{h_t(x), \beta_t} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(x_n)]\]

\[+ e^{-\beta_t} \sum_n w_t(n)\]

What term(s) must we optimize to choose \(h_t(x_n)\)?
Finding the optimal weak learner

Summary

\[(h_t^*(x), \beta_t^*) = \arg\min_{(h_t(x), \beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(x_n)}\]

\[= \arg\min_{(h_t(x), \beta_t)} \left(e^{\beta_t} - e^{-\beta_t}\right) \sum_n w_t(n) [y_n \neq h_t(x_n)]\]

\[+ e^{-\beta_t} \sum_n w_t(n)\]

What term(s) must we optimize to choose \(h_t(x_n)\)?

\[h_t^*(x) = \arg\min_{h_t(x)} \epsilon_t = \sum_n w_t(n) [y_n \neq h_t(x_n)]\]
Finding the optimal weak learner

Summary

$$(h^*_t(x), \beta^*_t) = \arg\min_{h_t(x), \beta_t} \sum_n w_t(n)e^{-y_n \beta_t h_t(x_n)}$$

$$= \arg\min_{h_t(x), \beta_t} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{1}[y_n \neq h_t(x_n)]$$

$$+ e^{-\beta_t} \sum_n w_t(n)$$

What term(s) must we optimize to choose $h_t(x_n)$?

$$h^*_t(x) = \arg\min_{h_t(x)} \epsilon_t = \sum_n w_t(n) \mathbb{1}[y_n \neq h_t(x_n)]$$

Minimize weighted classification error as noted in step 1 of Adaboost!
How to choose $\beta_t$?

Summary

$$(h_t^*(x), \beta_t^*) = \underset{(h_t(x), \beta_t)}{\text{argmin}} \sum_{n} w_t(n)e^{-y_n \beta_t h_t(x_n)}$$

$$= \underset{(h_t(x), \beta_t)}{\text{argmin}} (e^{\beta_t} - e^{-\beta_t}) \sum_{n} w_t(n)[y_n \neq h_t(x_n)]$$

$$+ e^{-\beta_t} \sum_{n} w_t(n)$$

What term(s) must we optimize?
How to choose $\beta_t$?

**Summary**

$$(h^*_t(x), \beta^*_t) = \text{argmin}_{(h_t(x), \beta_t)} \sum_n w_t(n)e^{-y_n \beta_t h_t(x_n)}$$

$$= \text{argmin}_{(h_t(x), \beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) [y_n \neq h_t(x_n)]$$

$$+ e^{-\beta_t} \sum_n w_t(n)$$

**What term(s) must we optimize?**

We need to minimize the entire objective function with respect to $\beta_t$!
How to choose $\beta_t$?

Summary

$$(h^*_t(x), \beta^*_t) = \arg\min_{(h_t(x), \beta_t)} \sum_n w_t(n)e^{-y_n\beta_t h_t(x_n)}$$

$$= \arg\min_{(h_t(x), \beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(x_n)] + e^{-\beta_t} \sum_n w_t(n)$$

What term(s) must we optimize?

We need to minimize the entire objective function with respect to $\beta_t$!

We can do this by taking derivative with respect to $\beta_t$, setting to zero, and solving for $\beta_t$. After some calculation and using $\sum_n w_t(n) = 1$, we find:

$$\beta^*_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

which is precisely step 2 of Adaboost! (Exercise – verify the solution)
Once we find the optimal weak learner we can update our classifier:

\[ f(x) = f_{t-1}(x) + \beta_t h_t(x) \]
Updating the weights

Once we find the optimal weak learner we can update our classifier:

\[ f(x) = f_{t-1}(x) + \beta_t^* h_t^*(x) \]

We then need to compute the weights for the above classifier as:

\[ w_{t+1}(n) = e^{-y_nf(x_n)} = e^{-y_n[f_{t-1}(x) + \beta_t^* h_t^*(x_n)]} \]
Updating the weights

Once we find the optimal weak learner we can update our classifier:

\[ f(x) = f_{t-1}(x) + \beta_t^* h_t^*(x) \]

We then need to compute the weights for the above classifier as:

\[ w_{t+1}(n) = e^{-y_n f(x_n)} = e^{-y_n [f_{t-1}(x) + \beta_t^* h_t^*(x_n)]} \]

\[ = w_t(n) e^{-y_n \beta_t^* h_t^*(x_n)} \]
Once we find the optimal weak learner we can update our classifier:

\[ f(x) = f_{t-1}(x) + \beta_t^* h_t^*(x) \]

We then need to compute the weights for the above classifier as:

\[ w_{t+1}(n) = e^{-y_nf(x_n)} = e^{-y_n[f_{t-1}(x)+\beta_t^* h_t^*(x_n)]} \]

\[ = w_t(n)e^{-y_n\beta_t^* h_t^*(x_n)} = \begin{cases} w_t(n)e^{\beta_t^*} & \text{if } y_n \neq h_t^*(x_n) \\ w_t(n)e^{-\beta_t^*} & \text{if } y_n = h_t^*(x_n) \end{cases} \]

**Intuition** Misclassified data points will get their weights increased, while correctly classified data points will get their weight decreased.
Note that the AdaBoost algorithm itself never specifies how we would get $h^*_t(x)$ as long as it minimizes the weighted classification error

$$\epsilon_t = \sum_n w_t(n) [y_n \neq h^*_t(x_n)]$$

In this aspect, the AdaBoost algorithm is a meta-algorithm and can be used with any type of classifier
E.g., Decision Stumps

How do we choose the decision stump classifier given the weights at the second round of the following distribution?

We can simply enumerate all possible ways of putting vertical and horizontal lines to separate the data points into two classes and find the one with the smallest weighted classification error! Runtime?

- Presort data by each feature in $O(dN \log N)$ time
- Evaluate $N + 1$ thresholds for each feature at each round in $O(dN)$ time
- In total $O(dN \log N + dNT)$ time – this efficiency is an attractive quality of boosting!
E.g., Decision Stumps

How do we choose the decision stump classifier given the weights at the second round of the following distribution?

We can simply enumerate all possible ways of putting vertical and horizontal lines to separate the data points into two classes and find the one with the smallest weighted classification error! Runtime?
E.g., Decision Stumps

How do we choose the decision stump classifier given the weights at the second round of the following distribution?

We can simply enumerate all possible ways of putting vertical and horizontal lines to separate the data points into two classes and find the one with the smallest weighted classification error! Runtime?

- Presort data by each feature in $O(dN \log N)$ time
- Evaluate $N + 1$ thresholds for each feature at each round in $O(dN)$ time
- In total $O(dN \log N + dNT)$ time – this efficiency is an attractive quality of boosting!
• Decision Trees have high variance and thus can change drastically with small changes in training data

• Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness

• Two common ensemble methods for decision trees
  - Bagging – Train $B$ trees on random subsets of training data. Can lead to highly correlated trees
  - Random Forests – In addition to bagging, randomly sample $m$ candidate features at each split. De-correlates the trees better

• Boosting Methods: AdaBoost
  - Sequentially add weak classifiers
  - Increase the weight hard training points
  - Take weighted average of the outputs of the resulting classifiers

• Decision trees are commonly used as base classifiers in AdaBoost.
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data.
- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness.
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data
- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness
- Two common ensemble methods for decision trees
  - Bagging – Train $B$ trees on random subsets of training data. Can lead to highly correlated trees
  - Random Forests – In addition to bagging, randomly sample $m$ candidate features at each split. De-correlates the trees better
  - Boosting Methods: AdaBoost
    - Sequentially add weak classifiers
    - Increase the weight hard training points
    - Take weighted average of the outputs of the resulting classifiers

Decision trees are commonly used as base classifiers in AdaBoost.
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data.
- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness.
- Two common ensemble methods for decision trees:
  - Bagging – Train $B$ trees on random subsets of training data. Can lead to highly correlated trees.
  - Random Forests – In addition to bagging, randomly sample $m$ candidate features at each split. De-correlates the trees better.
- Boosting Methods: AdaBoost
  - Sequentially add weak classifiers.
  - Increase the weight of hard training points.
  - Take a weighted average of the outputs of the resulting classifiers. Decision trees are commonly used as base classifiers in AdaBoost.
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data.
- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness.
- Two common ensemble methods for decision trees:
  - Bagging – Train $B$ trees on random subsets of training data. Can lead to highly correlated trees.
  - Random Forests – In addition to bagging, randomly sample $m$ candidate features at each split. De-correlates the trees better.
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data.
- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness.
- Two common ensemble methods for decision trees:
  - Bagging – Train \( B \) trees on random subsets of training data. Can lead to highly correlated trees.
  - Random Forests – In addition to bagging, randomly sample \( m \) candidate features at each split. De-correlates the trees better.
- Boosting Methods: AdaBoost.
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data
- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness
- Two common ensemble methods for decision trees
  - Bagging – Train $B$ trees on random subsets of training data. Can lead to highly correlated trees
  - Random Forests – In addition to bagging, randomly sample $m$ candidate features at each split. De-correlates the trees better
- Boosting Methods: AdaBoost
  - Sequentially add weak classifiers
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data.
- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness.
- Two common ensemble methods for decision trees:
  - Bagging – Train \( B \) trees on random subsets of training data. Can lead to highly correlated trees.
  - Random Forests – In addition to bagging, randomly sample \( m \) candidate features at each split. De-correlates the trees better.
- Boosting Methods: AdaBoost
  - Sequentially add weak classifiers.
  - Increase the weight hard training points.
• Decision Trees have high variance and thus can change drastically with small changes in training data

• Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness

• Two common ensemble methods for decision trees
  • Bagging – Train $B$ trees on random subsets of training data. Can lead to highly correlated trees
  • Random Forests – In addition to bagging, randomly sample $m$ candidate features at each split. De-correlates the trees better

• Boosting Methods: AdaBoost
  • Sequentially add weak classifiers
  • Increase the weight hard training points
  • Take weighted average of the outputs of the resulting classifiers
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data.
- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness.
- Two common ensemble methods for decision trees:
  - Bagging – Train \( B \) trees on random subsets of training data. Can lead to highly correlated trees.
  - Random Forests – In addition to bagging, randomly sample \( m \) candidate features at each split. De-correlates the trees better.
- Boosting Methods: AdaBoost
  - Sequentially add weak classifiers.
  - Increase the weight hard training points.
  - Take weighted average of the outputs of the resulting classifiers.
- Decision trees are commonly used as base classifiers in AdaBoost.