DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 18 | Part 1

Boosting

Today

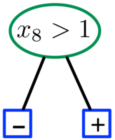
- Can we **combine** very **simple** models and get good results?
- **Yes**: **boosting**.

Weak Learners

- A **weak classifier** is one which performs only a little better than chance.
- A learning algorithm capable of consistently producing weak classifiers is called a **weak learner**.
- Usually very simple, fast.

Example

- A **decision stump** is a **weak classifier**.



- **Weak learner**: the strategy discussed last time for picking question.

Example

- The full decision tree learning algorithm is a **strong learner**.

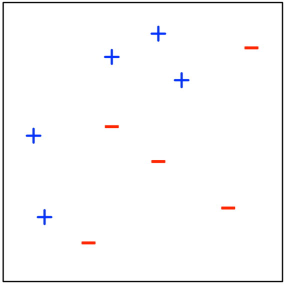
The Question

- Can we “boost” the quality of a weak learner?

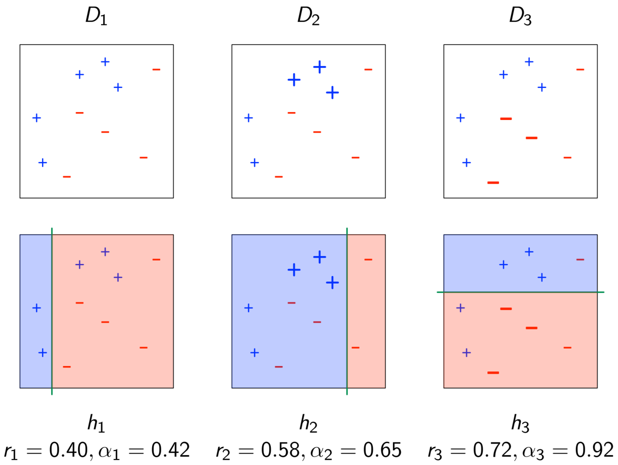
Boosting: The Idea

- Train a weak classifier, $H_1 : \mathcal{X} \to [-1, 1]$.
- Increase weight (importance) of misclassified points, train another classifier H_2.
- Repeat, creating more classifiers, updating weights.
- Final classifier: a linear combination of $H_1, \ldots, H_k$.

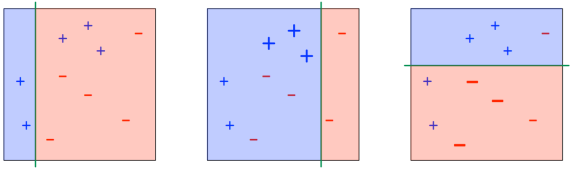
Example



Example


D_1
D_2
D_3
h_1
$r_1 = 0.40, \alpha_1 = 0.42$
h_2
$r_2 = 0.58, \alpha_2 = 0.65$
h_3
$r_3 = 0.72, \alpha_3 = 0.92$

Example



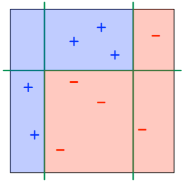
h_1 $\alpha_1 = 0.42$

h_2 $\alpha_2 = 0.65$

h_3 $\alpha_3 = 0.92$

Final classifier:

$\text{sign}\left(0.42h_1(x) + 0.65h_2(x) + 0.92h_3(x)\right)$



The Details

- **Q1**: How do we measure the performance of a classifier on a weighted data set?
- **Q2**: How do we update the point weights?
- **Q3**: How do we combine the classifiers?

AdaBoost

- Yoav Freund (UCSD) and Robert Schapire.
- A theoretically-sound answer to these questions.

Q1: Measuring Performance

- Suppose weights at step t are in $\vec{\omega}^{(t)}$.
  - Assume normalized s.t. weights add to one.

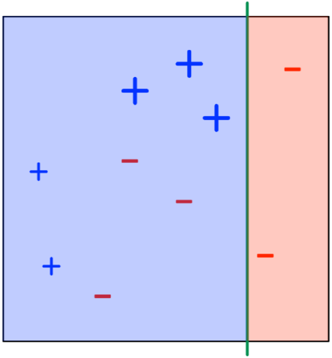
- We use weights to learn a classifier $H_t : \mathcal{X} \to [-1, 1]$.

- The "margin":

$$r_t = \sum_{i=1}^{n} \omega_i^{(t)} y_i H_t(\vec{x}^{(i)}) \in [-1, 1]$$

The Margin

$$r_t = \sum_{i=1}^{n} \omega_i^{(t)} y_i H_t(\vec{x}^{(i)}) \in [-1, 1]$$



The Margin

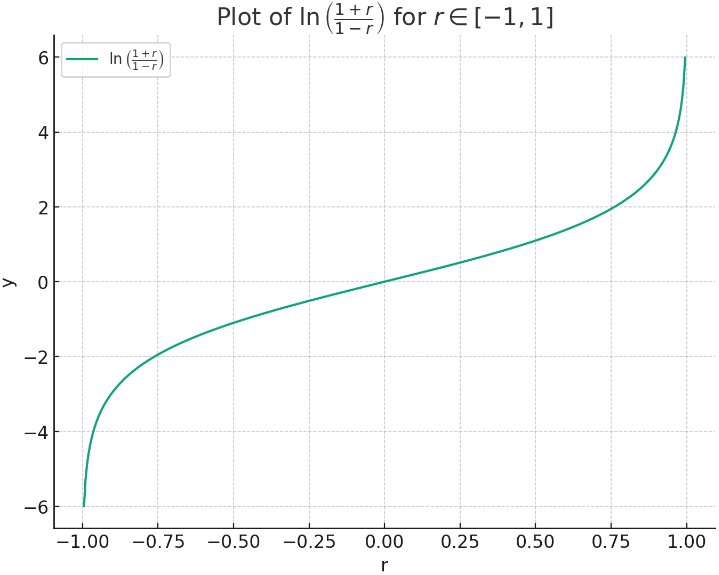
$$r_t = \sum_{i=1}^{n} \omega_i^{(t)} y_i H_t(\vec{x}^{(i)}) \in [-1, 1]$$

- The larger r_t, the better H_t is doing on the "important" points.

Q1: Measuring Performance

- The **performance** of H_t:

$$\alpha_t = \frac{1}{2} \ln \frac{1 + r_t}{1 - r_t}$$



Plot of $\ln\left(\frac{1+r}{1-r}\right)$ for $r \in [-1, 1]$
$\ln\left(\frac{1+r}{1-r}\right)$
y
6
4
2
0
−2
−4
−6
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
r

Q2: Updating Weights

- We use weights to learn a classifier $H_t : \mathcal{X} \to [-1, 1]$.
- Weigh misclassified points more heavily.
- Point is misclassified if $y_i H_t(\vec{x}^{(i)}) < 0$

Q2: Updating Weights

- This will do the trick:

$$\omega_i^{(t+1)} \propto \omega_i^{(t)} \cdot \exp\left(-\alpha_t y_i H_t(\vec{x}^{(i)})\right)$$

- $\propto$ because we normalize.

Q3: Combining Classifiers

- The final classifier:

$$H_t(\vec{x}) = \sum_{t=1}^{T} \alpha_t H_t(\vec{x})$$

AdaBoost

Given data $(\vec{x}^{(1)}, y_1), \ldots, (\vec{x}^{(n)}, y_n)$, labels in $\{-1, 1\}$.

- Initialize weight vector, $\vec{\omega}^{(1)} = (\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n})^T$
- Repeat:
  - Give data and weights $\vec{\omega}^{(t)}$ to weak learner. Receive a classifier, $H_t : \mathcal{X} \to \{-1, 1\}$ back.
  - Calculate "performance", $\alpha_t = \frac{1}{2} \ln \frac{1+r_t}{1-r_t}$
  - Update $\vec{\omega}^{(t+1)} \propto \omega_i^{(t)} \cdot \exp\left(-\alpha_t y_i H_t(\vec{x}^{(i)})\right)$
- Final classifier: $H_t(\vec{x}) = \sum_{t=1}^{T} \alpha_t H_t(\vec{x})$

Example: Decision Stumps

- To learn decision stump, given data and $\vec{\omega}^{(t)}$.
- Try all features, thresholds.
- Choose that which maximizes the margin:

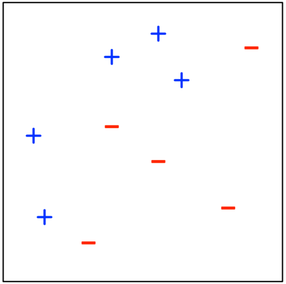
$$r_t = \sum_{i=1}^{n} \omega_i^{(t)} y_i H_t(\vec{x}^{(i)}) \in [-1, 1]$$

Example: Decision Stumps

- To learn decision stump, given data and $\vec{\omega}^{(t)}$.
- Try all features, thresholds.
- Equivalently, choose that which maximizes the performance:

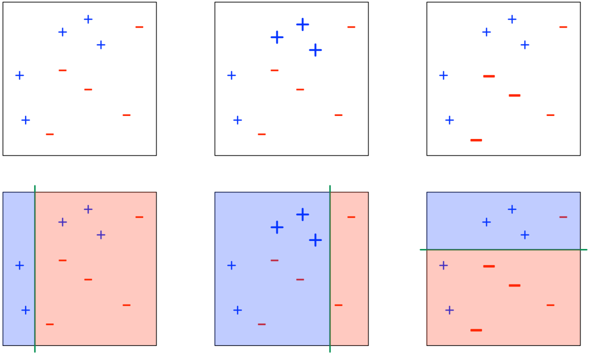
$$\alpha_t = \frac{1}{2} \ln \frac{1 + r_t}{1 - r_t}$$

Example



Example

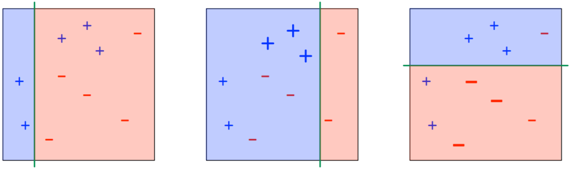
D_1 D_2 D_3



h_1 h_2 h_3

$r_1 = 0.40, \alpha_1 = 0.42$ $r_2 = 0.58, \alpha_2 = 0.65$ $r_3 = 0.72, \alpha_3 = 0.92$

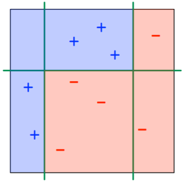
Example



h_1 $\alpha_1 = 0.42$ | h_2 $\alpha_2 = 0.65$ | h_3 $\alpha_3 = 0.92$

Final classifier:

$\text{sign}\left(0.42h_1(x) + 0.65h_2(x) + 0.92h_3(x)\right)$

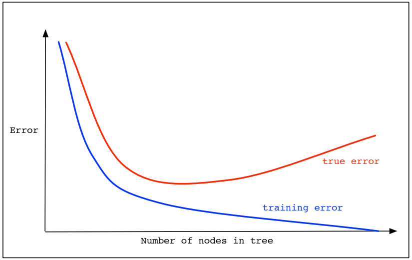


Theory

Suppose that on each round t, the weak learner returns a rule H_t whose error on the step t weighted data is $\leq \frac{1}{2} - \gamma$. Then after T rounds, the training error of the combined rule H is at most $e^{-\gamma^2 T/2}$.

Generalization

- Boosted decision stumps are really resistant to overfitting.



Generalization

- Boosted decision stumps are really resistant to overfitting.

Why not?

- Why use weak learners?

- What if we replace decision stumps with SVMs or logistic regression?

Why not?

- Why use weak learners?
- What if we replace decision stumps with SVMs or logistic regression?
- You can, but weak learners are **fast** to learn.
- The point of boosting is that weak learners are "just as good" as strong learners.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 18 | Part 2

Random Forests

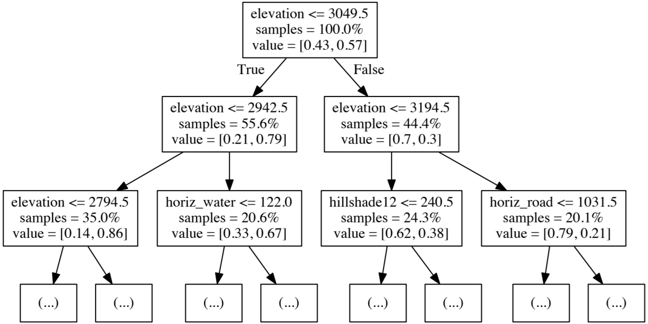
Let’s Try

- Decision trees are susceptible to overfitting.
- Let’s try using boosted decision trees.

Example: Forest Cover Type

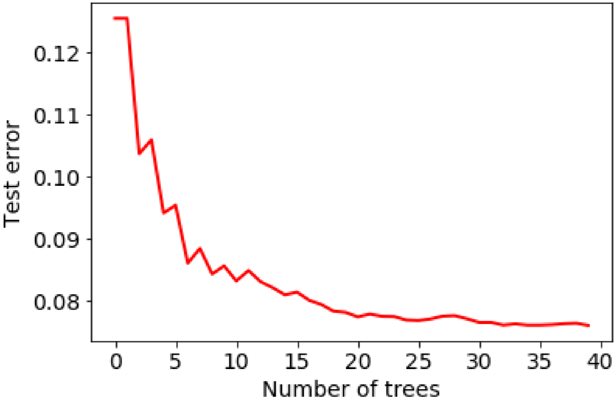
- **Goal**: predict forest type.
  - Spruce-fir
  - Lodgepole pine
  - etc. 7 classes in total.
- 54 cartographic/geological features.
  - Elevation, slope, amount of shade, distance to water, etc.

Decision Tree

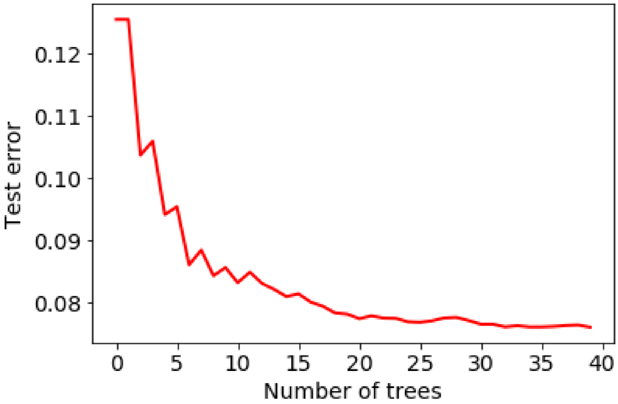


Depth 20. Training error: 1%. Test error: 12.6%.

Boosted Decision Trees



Boosted Decision Trees



Depth 20: Test error: 8.7%. **Slow!**

Another Idea

- Prevent decision trees from overfitting by “hiding data” randomly.
- Train a bunch of trees, quickly.
- Average them to make a final prediction.

Random Forests

- For $t = 1$ to T
  - Choose n' training points randomly, with replacement.
  - Fit a decision tree, H_t.
    - At each node, restrict to one of k features, chosen randomly.
- Final classifier: majority vote of $H_1, \ldots, H_T$.
- Common settings: $n' = n$ (bootstrap), $k = \sqrt{d}$.

Forest Cover Type

- Decision trees: 12.6% error.
- Boosted decision trees: 8.7% error (but **slow**!)
- Random forests: 8.8% error.
  - 50% of features dropped.
  - Each individual tree $H_1, \ldots, H_t$ has test error around 15%.