MTH 221

Fundamentals of Machine Learning

Batuhan Bardak

Lecture 10: Decision Tree & Ensemble Learning

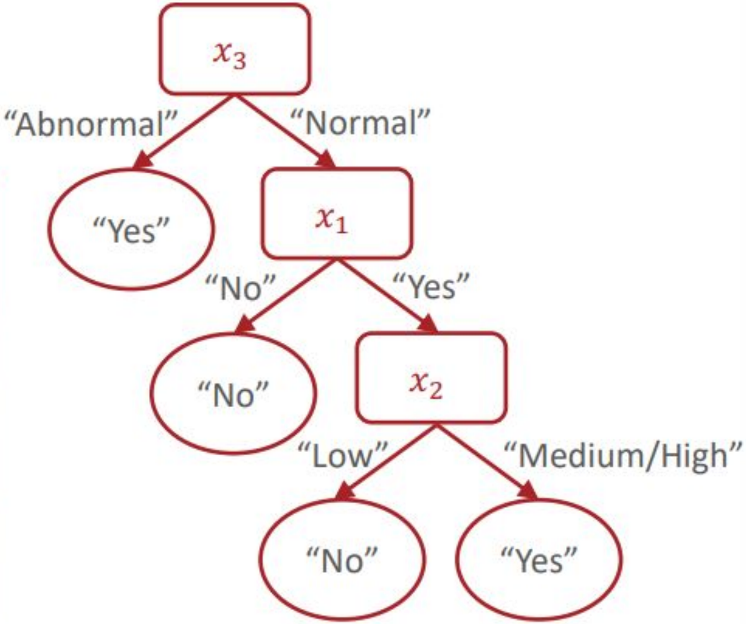
Date: 12.12.2023

Plan for today

- Decision Tree
- Bagging
  - Random Forest
- Boosting
  - AdaBoost

Decision Tree

x_1 Family History	x_2 Resting Blood Pressure	x_3 Cholesterol	y Heart Disease?
Yes	Low	Normal	No
No	Medium	Normal	No
No	Low	Abnormal	Yes
Yes	Medium	Normal	Yes
Yes	High	Abnormal	Yes



Slide Credit: Henry Chai

Notation

- Feature space, $\mathcal{X}$
- Label space, $\mathcal{Y}$
- (Unknown) Target function, $c^*: \mathcal{X} \to \mathcal{Y}$
- Training dataset:

$$\mathcal{D} = \{(\boldsymbol{x}^{(1)}, c^*(\boldsymbol{x}^{(1)}) = y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}) \dots, (\boldsymbol{x}^{(N)}, y^{(N)})\}$$

- Data point:

$$(\boldsymbol{x}^{(n)}, y^{(n)}) = \left(x_1^{(n)}, x_2^{(n)}, \dots, x_D^{(n)}, y^{(n)}\right)$$

- Classifier, $h : \mathcal{X} \to \mathcal{Y}$
- Goal: find a classifier, h, that best approximates c^*

Slide Credit: Henry Chai

Evaluation

- Loss function, $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$
  - Defines how "bad" predictions, $\hat{y} = h(\boldsymbol{x})$, are compared to the true labels, $y = c^*(\boldsymbol{x})$
  - Common choices

  1. Squared loss (for regression): $\ell(y, \hat{y}) = (y - \hat{y})^2$
  2. Binary or 0-1 loss (for classification):

     $\ell(y, \hat{y}) = \mathbb{1}(y \neq \hat{y})$

- Error rate:

$$err(h, \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}\left(y^{(n)} \neq \hat{y}^{(n)}\right)$$

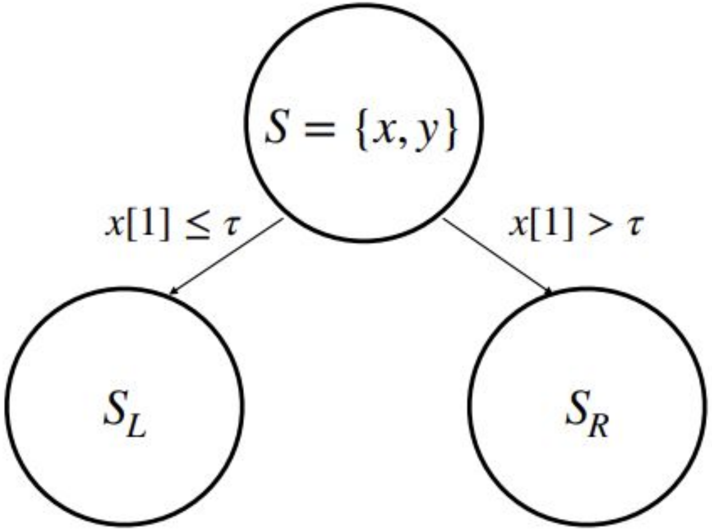
Slide Credit: Henry Chai

Overview of the Decision Tree algorithm

Slide Credit: Cornell CS4780

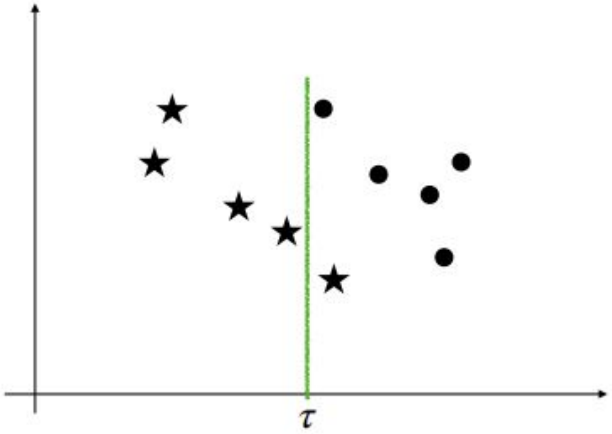
How to split a tree node

Consider k-class classification, i.e., $y \in \{1,2,\ldots,k\}$



$S_L + S_R = S,\ S_L \cap S_R = \emptyset$

Goal: do an axis aligned split such that diversity of labels in leafs are reduced



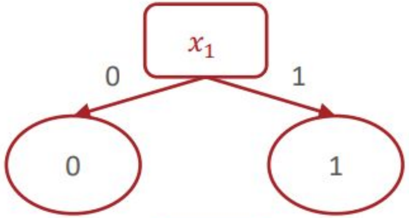
How to mathematically quantify "diversity"?

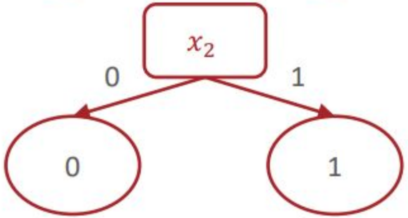
Slide Credit: Cornell CS4780

Splitting Criteria?

x_1	x_2	y
1	0	0
1	0	0
1	0	1
1	0	1
1	1	1
1	1	1
1	1	1
1	1	1

- Which feature would you split on using training error rate as the splitting criterion?





Training error rate: 2/8

Slide Credit: Henry Chai

Splitting Criteria

- A **splitting criterion** is a function that measures how good or useful splitting on a particular feature is *for a specified dataset*
- Insight: use the feature that optimizes the splitting criterion for our decision stump.
- Potential splitting criteria:
  - Training error rate (minimize)
  - Gini impurity (minimize) → CART algorithm
  - Mutual information (maximize) → ID3 algorithm

Slide Credit: Henry Chai

Entropy

- Entropy describes the purity or uniformity of a collection of values: the lower the entropy, the more pure

$$H(S) = -\sum_{v \in V(S)} \frac{|S_v|}{|S|} \log_2\left(\frac{|S_v|}{|S|}\right)$$

where S is a collection of values,

$V(S)$ is the set of unique values in S

S_v is the collection of elements in S with value v

- If all the elements in S are the same, then

$$\mathrm{H}(S) = -1 \log_2(1) = 0$$

Slide Credit: Henry Chai

Entropy

- Entropy describes the purity or uniformity of a collection of values: the lower the entropy, the more pure

$$H(S) = -\sum_{v \in V(S)} \frac{|S_v|}{|S|} \log_2 \left(\frac{|S_v|}{|S|}\right)$$

where S is a collection of values,

$V(S)$ is the set of unique values in S

S_v is the collection of elements in S with value v

- If S is split fifty-fifty between two values, then

$$\mathrm{H}(S) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = -\log_2\left(\frac{1}{2}\right) = 1$$

Slide Credit: Henry Chai

Entropy

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)

= Entropy (0.36, 0.64)

= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)

= 0.94

Mutual Information

- Mutual information describes how much information or clarity a particular feature provides about the label

$$I(x_d, Y) = H(Y) - \sum_{v \in V(x_d)} (f_v)\left(H\left(Y_{x_d=v}\right)\right)$$

where x_d is a feature

Y is the collection of all labels

$V(x_d)$ is the set of unique values of x_d

f_v is the fraction of inputs where $x_d = v$

$Y_{x_d=v}$ is the collection of labels where $x_d = v$

Slide Credit: Henry Chai

Mutual Information Example

x_d	y
1	1
1	1
0	0
0	0

$$I(x_d, Y) = H(Y) - \sum_{v \in V(x_d)} (f_v)\left(H\left(Y_{x_d=v}\right)\right)$$

$$= 1 - \frac{1}{2} H\left(Y_{x_d=0}\right) - \frac{1}{2} H\left(Y_{x_d=1}\right)$$

$$= 1 - \frac{1}{2}(0) - \frac{1}{2}(0) = 1$$

Slide Credit: Henry Chai

Mutual Information Example

x_d	y
1	1
0	1
1	0
0	0

$$I(x_d, Y) = H(Y) - \sum_{v \in V(x_d)} (f_v)\left(H\left(Y_{x_d=v}\right)\right)$$

$$= 1 - \frac{1}{2} H\left(Y_{x_d=0}\right) - \frac{1}{2} H\left(Y_{x_d=1}\right)$$

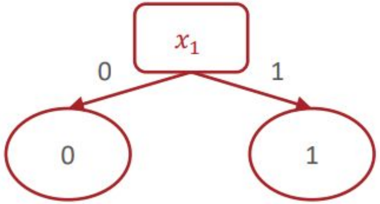
$$= 1 - \frac{1}{2}(1) - \frac{1}{2}(1) = 0$$

Slide Credit: Henry Chai

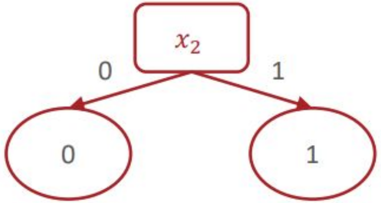
Mutual Information as a Splitting Criteria

x_1	x_2	y
1	0	0
1	0	0
1	0	1
1	0	1
1	1	1
1	1	1
1	1	1
1	1	1

- Which feature would you split on using mutual information as the splitting criterion?



Mutual Information: 0



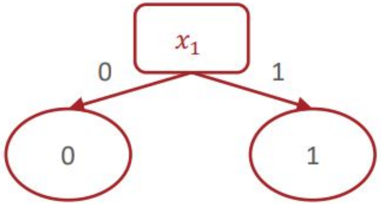
Mutual Information: $H(Y) - \frac{1}{2}H\left(Y_{x_2=0}\right) - \frac{1}{2}H\left(Y_{x_2=1}\right)$

Slide Credit: Henry Chai

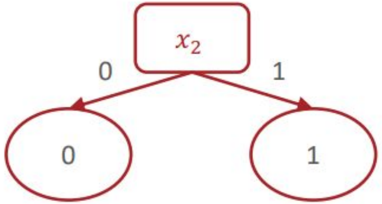
Mutual Information as a Splitting Criteria

x_1	x_2	y
1	0	0
1	0	0
1	0	1
1	0	1
1	1	1
1	1	1
1	1	1
1	1	1

- Which feature would you split on using mutual information as the splitting criterion?



Mutual Information: 0



Mutual Information: $-\frac{2}{8}\log_2\frac{2}{8} - \frac{6}{8}\log_2\frac{6}{8} - \frac{1}{2}(1) - \frac{1}{2}(0) \approx 0.31$

Slide Credit: Henry Chai

Pseudocode

```
def predict(x'):
 - walk from root node to a leaf node
   while(true):
      if current node is internal (non-leaf):
            check the associated attribute, x_d
            go down branch according to x'_d
      if current node is a leaf node:
            return label stored at that leaf
```

Pseudocode

def train($\mathcal{D}$):

    store root = tree_recurse($\mathcal{D}$)

def tree_recurse($\mathcal{D}'$):

    q = new node()

    base case – if (SOME CONDITION):

    recursion – else:

        find best attribute to split on, x_d

        q.split = x_d

        for v in $V(x_d)$, all possible values of x_d:

            $\mathcal{D}_v = \left\{\left(x^{(i)}, y^{(i)}\right) \in \mathcal{D} \mid x_d^{(i)} = v\right\}$

            q.children(v) = tree_recurse($\mathcal{D}_v$)

    return q

Slide Credit: Henry Chai

Pseudocode

```
def train(𝒟):
    store root = tree_recurse(𝒟)
def tree_recurse(𝒟′):
    q = new node()
    base case – if (𝒟′ is empty OR
      all labels in 𝒟′ are the same OR
      all features in 𝒟′ are identical OR
      some other stopping criterion):
      q.label = majority_vote(𝒟′)


    recursion – else:
    return q
```

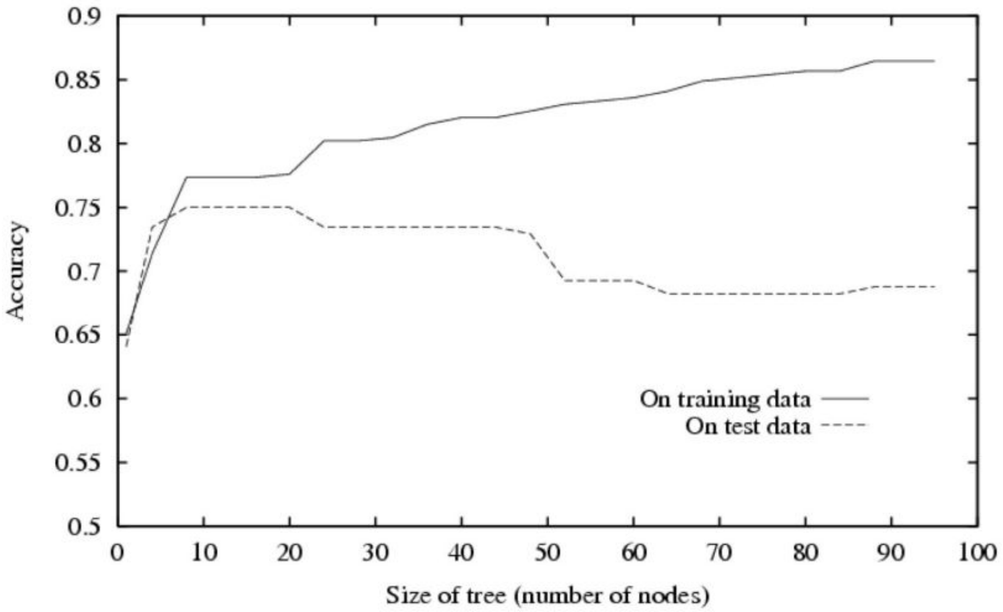
Slide Credit: Henry Chai

Pros & Cons

- Pros
  - Interpretable
  - Efficient (computational cost and storage)
  - Can be used for classification and regression tasks
  - Compatible with categorical and real-valued features
- Cons
  - Learned greedily: each split only considers the immediate impact on the splitting criterion
    - Not guaranteed to find the smallest (fewest number of splits) tree that achieves a training error rate of 0.
  - Liable to overfit!

Slide Credit: Henry Chai

Overfitting in Decision Trees



Slide Credit: Henry Chai

Combatting Overfitting in Decision Trees

- Heuristics:
  - Do not split leaves past a fixed depth, δ
  - Do not split leaves with fewer than c data points
  - Do not split leaves where the maximal information gain is less than τ
  - Take a majority vote in impure leaves

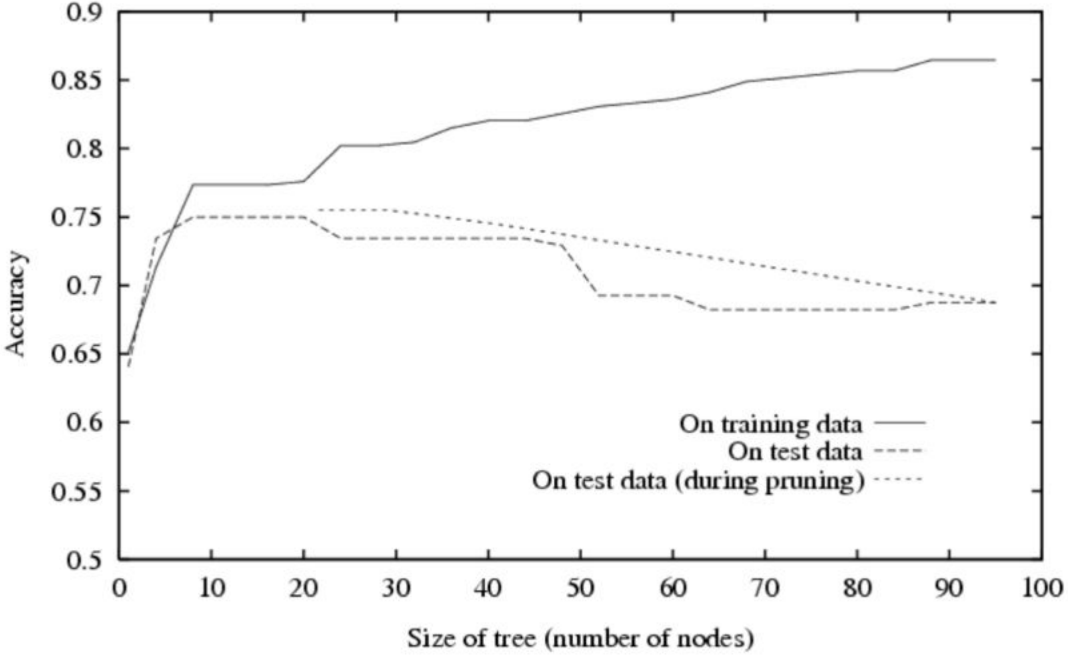
Slide Credit: Henry Chai

Combatting Overfitting in Decision Trees

- Pruning:
  - First, learn a decision tree
  - Then, evaluate each split using a “validation” dataset by comparing the validation error rate with and without that split
  - Greedily remove the split that most decreases the validation error rate
  - Stop if no split is removed

Slide Credit: Henry Chai

Combatting Overfitting in Decision Trees



Slide Credit: Henry Chai

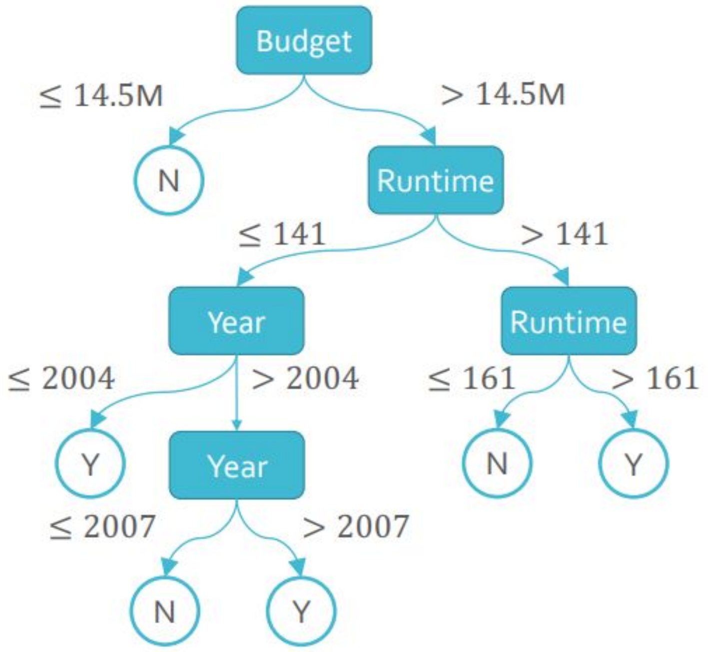
The Wisdom Of Crowds

- In 1906, Francis Galton asked ~800 people at a farmer's fair to guess the weight of a cow, including "experts"
  - Actual weight: 1198 lbs
  - Mean guess: 1197 lbs
  - Mean guess was more accurate than any single guess, even the experts

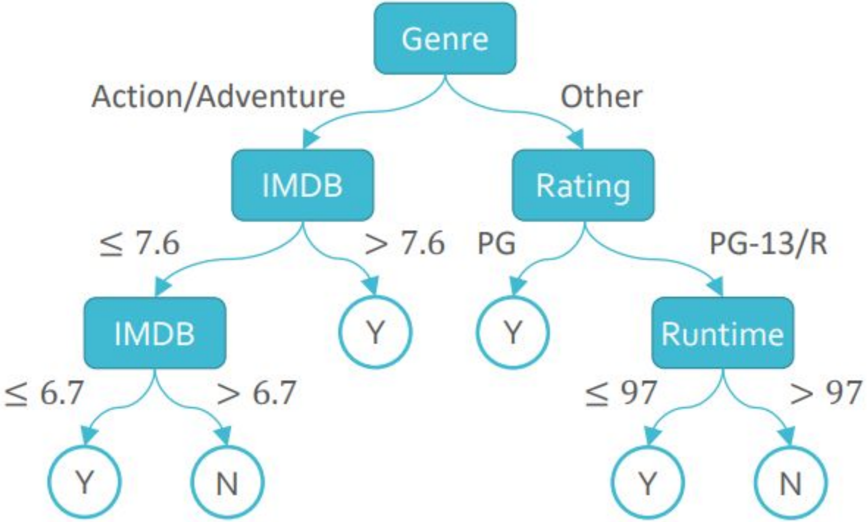
Slide Credit: Henry Chai

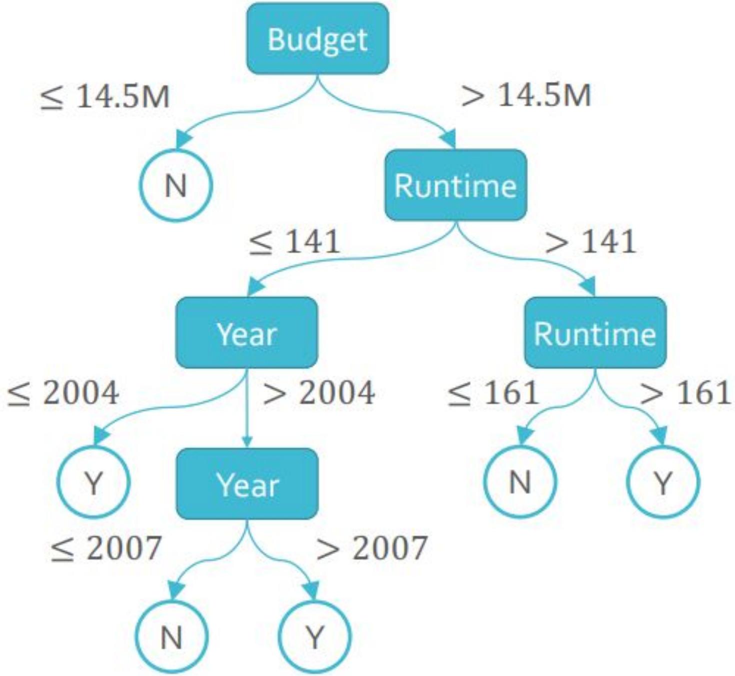
Decision Trees

MovieID	Runtime	Genre	Budget	Year	IMDB	Rating	Liked?
1	124	Action	18M	1980	8.7	PG	Y
2	105	Action	30M	1984	7.8	PG	Y
3	103	Comedy	6M	1986	7.8	PG-13	N
4	98	Adventure	16M	1987	8.1	PG	Y
5	128	Comedy	16.4M	1989	8.1	PG	Y
6	120	Comedy	11M	1992	7.6	R	N
7	120	Drama	14.5M	1996	6.7	PG-13	N
8	136	Action	115M	1999	6.5	PG	Y
9	90	Action	90M	2001	6.6	PG-13	Y
10	161	Adventure	100M	2002	7.4	PG	N
11	201	Action	94M	2003	8.9	PG-13	Y
12	94	Comedy	26M	2004	7.2	PG-13	Y
13	157	Biography	100M	2007	7.8	R	N
14	128	Action	110M	2007	7.1	PG-13	N
15	107	Drama	39M	2009	7.1	PG-13	N
16	158	Drama	61M	2012	7.6	PG-13	*Y*
17	169	Adventure	165M	2014	8.6	PG-13	Y
18	100	Biography	9M	2016	6.7	R	N
19	130	Action	180M	2017	7.9	PG-13	Y
20	141	Action	275M	2019	6.5	PG-13	Y



Decision Trees





Slide Credit: Henry Chai

Decision Trees

- Pros
  - Interpretable
  - Efficient (computational cost and storage)
  - Can be used for classification and regression tasks
  - Compatible with categorical and real-valued features
- Cons
  - Learned greedily: each split only considers the immediate impact on the splitting criterion
    - Not guaranteed to find the smallest (fewest number of splits) tree that achieves a training error rate of 0.
  - Prone to overfit
  - High variance
    - Can be addressed via ensembles → random forests

Slide Credit: Henry Chai

Random Forests

- Combines the prediction of many diverse decision trees to reduce their variability
- If B independent random variables $x^{(1)}, x^{(2)}, \dots, x^{(B)}$ all have variance σ^2, then the variance of $\frac{1}{B}\sum_{b=1}^{B} x^{(b)}$ is $\frac{\sigma^2}{B}$
- Random forests = bagging + split-feature randomization

  = **b**ootstrap **agg**regat**ing** + split-feature randomization

Slide Credit: Henry Chai

Aggregating

- How can we combine multiple decision trees, $\{t_1, t_2, \dots, t_B\}$, to arrive at a single prediction?
- Regression - average the predictions:

$$\bar{t}(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B} t_b(\boldsymbol{x})$$

- Classification - plurality (or majority) vote; for binary labels encoded as $\{-1, +1\}$:

$$\bar{t}(\boldsymbol{x}) = \operatorname{sign}\left(\frac{1}{B}\sum_{b=1}^{B} t_b(\boldsymbol{x})\right)$$

Slide Credit: Henry Chai

Bootstrapping

- Insight: one way of generating different decision trees is by changing the training data set
- Issue: often, we only have one fixed set of training data
- Idea: resample the data multiple times ***with replacement***

MovieID	⋯
1	⋯
2	⋯
3	⋯
⋮	⋮
19	⋯
20	⋯

Training data

MovieID	⋯
1	⋯
1	⋯
1	⋯
⋮	⋮
14	⋯
19	⋯

Bootstrapped Sample 1

MovieID	⋯
4	⋯
4	⋯
5	⋯
⋮	⋮
16	⋯
16	⋯

Bootstrapped Sample 2

…

…

Slide Credit: Henry Chai

Bootstrapping

- Idea: resample the data multiple times ***with replacement***
  - Each bootstrapped sample has the same number of data points as the original data set
  - Duplicated points cause different decision trees to focus on different parts of the input space

MovieID	⋯
1	⋯
2	⋯
3	⋯
⋮	⋮
19	⋯
20	⋯

Training data

MovieID	⋯
1	⋯
1	⋯
1	⋯
⋮	⋮
14	⋯
19	⋯

Bootstrapped Sample 1

MovieID	⋯
4	⋯
4	⋯
5	⋯
⋮	⋮
16	⋯
16	⋯

Bootstrapped Sample 2

…

Slide Credit: Henry Chai

Split-feature Randomization

- Issue: decision trees trained on bootstrapped samples still behave similarly
- Idea: in addition to sampling the data points (i.e., the rows), also sample the features (i.e., the columns)
- Each time a split is being considered, limit the possible features to a randomly sampled subset

Random Forests

- Input: $\mathcal{D} = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$, B, ρ
- For $b = 1, 2, \ldots, B$
  - Create a dataset, $\mathcal{D}_b$, by sampling N points from the original training data $\mathcal{D}$ ***with replacement***
  - Learn a decision tree, t_b, using $\mathcal{D}_b$ and the ID3 algorithm ***with split-feature randomization***, sampling ρ features for each split
- Output: $\bar{t} = f(t_1, \ldots, t_B)$, the aggregated hypothesis

Slide Credit: Henry Chai

Recall: Validation Sets

- Suppose we want to compare multiple hyperparameter settings $\theta_1, \ldots, \theta_K$
- For $k = 1, 2, \ldots, K$
  - Train a model on D_{train} using θ_k
- Evaluate each model on D_{val} and find the best hyperparameter setting, θ_{k^*}
- Compute the error of a model trained with θ_{k^*} on D_{test}

Slide Credit: Henry Chai

Out-of-bag Error

- For each training point, $\boldsymbol{x}^{(n)}$, there are some decision trees which $\boldsymbol{x}^{(n)}$ was not used to train (roughly B/e trees or 37%)
  - Let these be $t^{(-n)} = \left\{t_1^{(-n)}, t_2^{(-n)}, \dots, t_{N_{-n}}^{(-n)}\right\}$
- Compute an aggregated prediction for each $\boldsymbol{x}^{(n)}$ using the trees in $t^{(-n)}$, $\bar{t}^{(-n)}\left(\boldsymbol{x}^{(n)}\right)$
- Compute the out-of-bag (OOB) error, e.g., for regression

$$E_{OOB} = \frac{1}{N}\sum_{n=1}^{N}\left(\bar{t}^{(-n)}\left(\boldsymbol{x}^{(n)}\right) - y^{(n)}\right)^2$$

Out-of-bag Error

- For each training point, $\boldsymbol{x}^{(n)}$, there are some decision trees which $\boldsymbol{x}^{(n)}$ was not used to train (roughly B/e trees or 37%)
  - Let these be $t^{(-n)} = \left\{t_1^{(-n)}, t_2^{(-n)}, \dots, t_{N_{-n}}^{(-n)}\right\}$
- Compute an aggregated prediction for each $\boldsymbol{x}^{(n)}$ using the trees in $t^{(-n)}$, $\bar{t}^{(-n)}\left(\boldsymbol{x}^{(n)}\right)$
- Compute the out-of-bag (OOB) error, e.g., for classification

$$E_{OOB} = \frac{1}{N} \sum_{n=1}^{N} \left[\!\left[\bar{t}^{(-n)}\left(\boldsymbol{x}^{(n)}\right) \neq y^{(n)}\right]\!\right]$$

- E_{OOB} can be used for hyperparameter optimization!

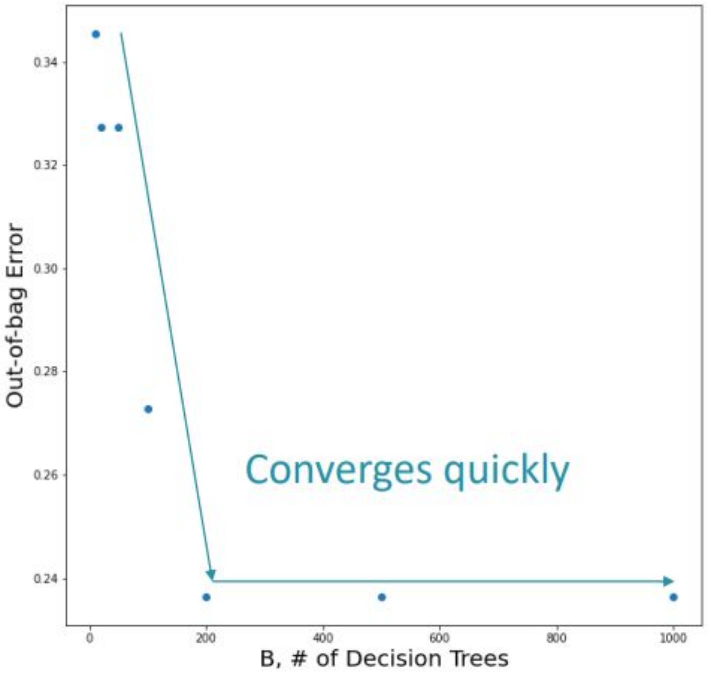
Slide Credit: Henry Chai

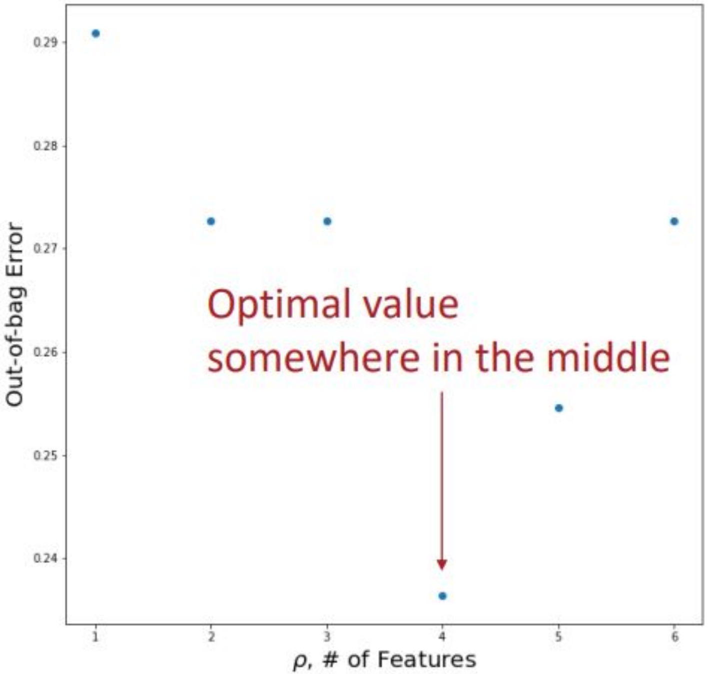
Out-of-bag Error

- Suppose we want to compare different numbers of trees in our random forest $B_1, \dots, B_K$
- For $k = 1, 2, \dots, K$
  - Train a random forest on D_{train} with B_k trees
- Compute E_{OOB} for each random forest and find the best number of trees, B_{k^*}
- Evaluate the random forest with B_{k^*} trees on D_{test}

Slide Credit: Henry Chai

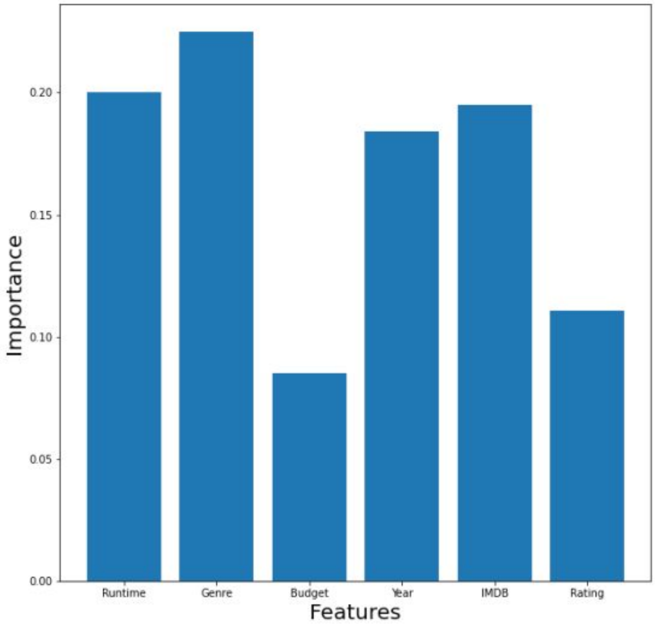
Setting Hyperparameters





Feature Importance

- Some of the interpretability of decision trees gets lost when switching to random forests
- Random forests allow for the computation of "feature importance", a way of ranking features based on how useful they are at predicting the target
- Initialize each feature's importance to zero
- Each time a feature is chosen to be split on, add the reduction in Gini impurity (weighted by the number of data points in the split) to its importance



Slide Credit: Henry Chai

Summary of Bagging/Random Forest

- Ensemble methods employ a "wisdom of crowds" philosophy
  - Can reduce the variance of high variance methods
- Random forests = bagging + split-feature randomization
  - Aggregate multiple decision trees together
  - Bootstrapping and split-feature randomization increase diversity in the decision trees
  - Use out-of-bag errors for hyperparameter optimization
  - Use feature importance to identify useful attributes

Slide Credit: Henry Chai

Decision Tree Pros & Cons

- Pros
  - Interpretable
  - Efficient (computational cost and storage)
  - Can be used for classification and regression tasks
  - Compatible with categorical and real-valued features
- Cons
  - Learned greedily: each split only considers the immediate impact on the splitting criterion
    - Not guaranteed to find the smallest (fewest number of splits) tree that achieves a training error rate of 0.
  - Prone to overfit
  - High variance
    - Can be addressed via bagging → random forests
  - High bias (especially short trees, i.e., stumps)
    - Can be addressed via boosting

Slide Credit: Henry Chai

Boosting

- Another ensemble method (like bagging) that combines the predictions of multiple hypotheses.
- Aims to reduce the bias of a “weak” or highly biased model (can also reduce variance).

Slide Credit: Henry Chai

AdaBoost

- Intuition: iteratively reweight inputs, giving more weight to inputs that are difficult-to-predict correctly
- Analogy:
  - You all have to take a test (😱) …
  - … 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

Slide Credit: Henry Chai

AdaBoost

- Input: $\mathcal{D}\ \left(y^{(n)} \in \{-1,+1\}\right), T$
- Initialize data point weights: $\omega_0^{(1)}, \dots, \omega_0^{(N)} = \frac{1}{N}$
- For $t = 1, \dots, T$
  1. Train a weak learner, h_t, by minimizing the *weighted* training error
  2. Compute the *weighted* training error of h_t:
  $$\epsilon_t = \sum_{n=1}^{N} \omega_{t-1}^{(n)} \mathbb{1}\left(y^{(n)} \neq h_t\left(\boldsymbol{x}^{(n)}\right)\right)$$
  3. Compute the **importance** of h_t:
  $$\alpha_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$
  4. Update the data point weights:

$$\omega_t^{(n)} = \frac{\omega_{t-1}^{(n)}}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t\left(\boldsymbol{x}^{(n)}\right) = y^{(n)} \\ e^{\alpha_t} & \text{if } h_t\left(\boldsymbol{x}^{(n)}\right) \neq y^{(n)} \end{cases} = \frac{\omega_{t-1}^{(n)} e^{-\alpha_t y^{(n)} h_t\left(\boldsymbol{x}^{(n)}\right)}}{Z_t}$$

- Output: an aggregated hypothesis

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

Slide Credit: Henry C

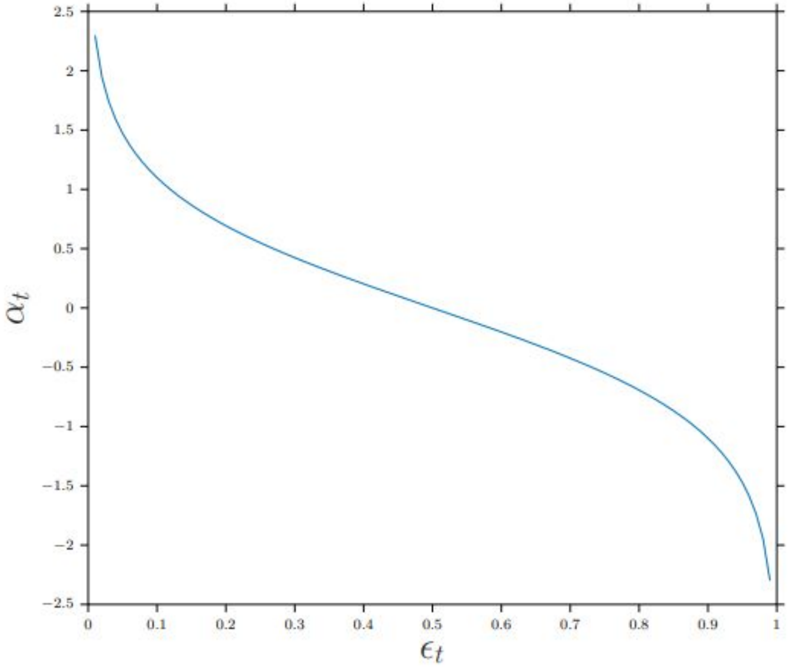
Setting α_t

α_t determines the contribution of h_t to the final, aggregated hypothesis:

$$g(\boldsymbol{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)$$

Intuition: we want good weak learners to have high importances

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



Slide Credit: Henry Chai

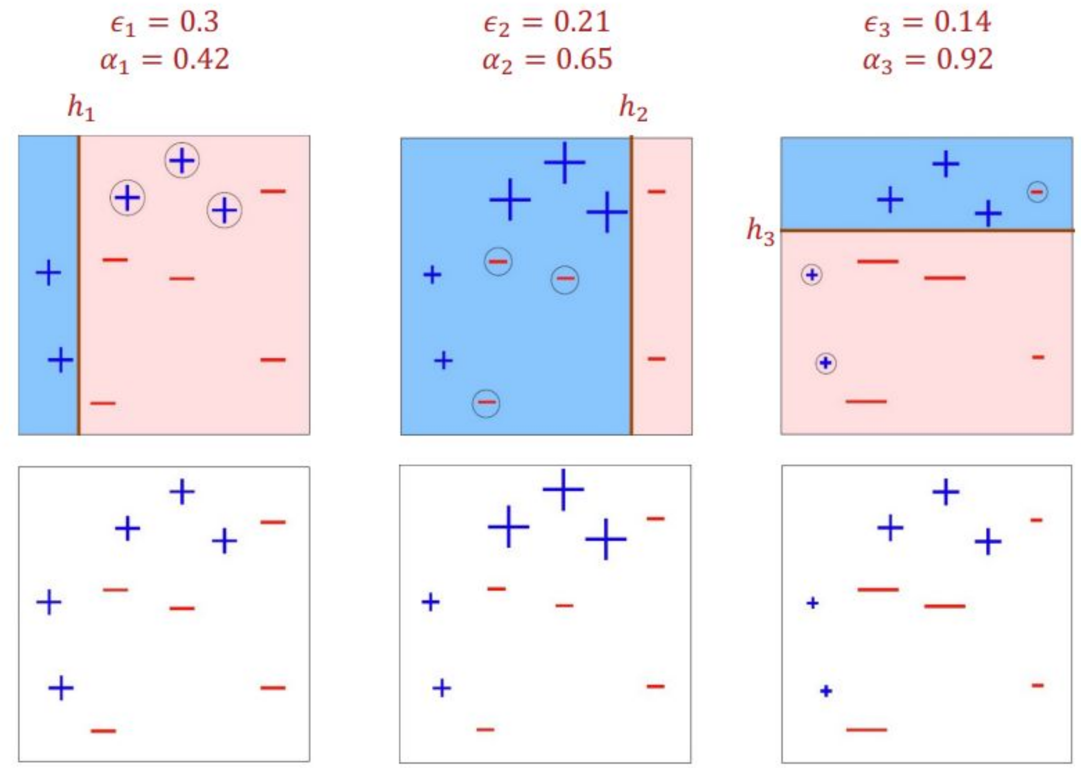
Updating w^(n)

- Intuition: we want incorrectly classified inputs to receive a higher weight in the next round

$$\omega_t^{(n)} = \frac{\omega_{t-1}^{(n)}}{Z_t} \times \begin{cases} e^{-\alpha_t} \text{ if } h_t\left(\boldsymbol{x}^{(n)}\right) = y^{(n)} \\ e^{\alpha_t} \text{ if } h_t\left(\boldsymbol{x}^{(n)}\right) \neq y^{(n)} \end{cases} = \frac{\omega_{t-1}^{(n)} e^{-\alpha_t y^{(n)} h_t\left(\boldsymbol{x}^{(n)}\right)}}{Z_t}$$

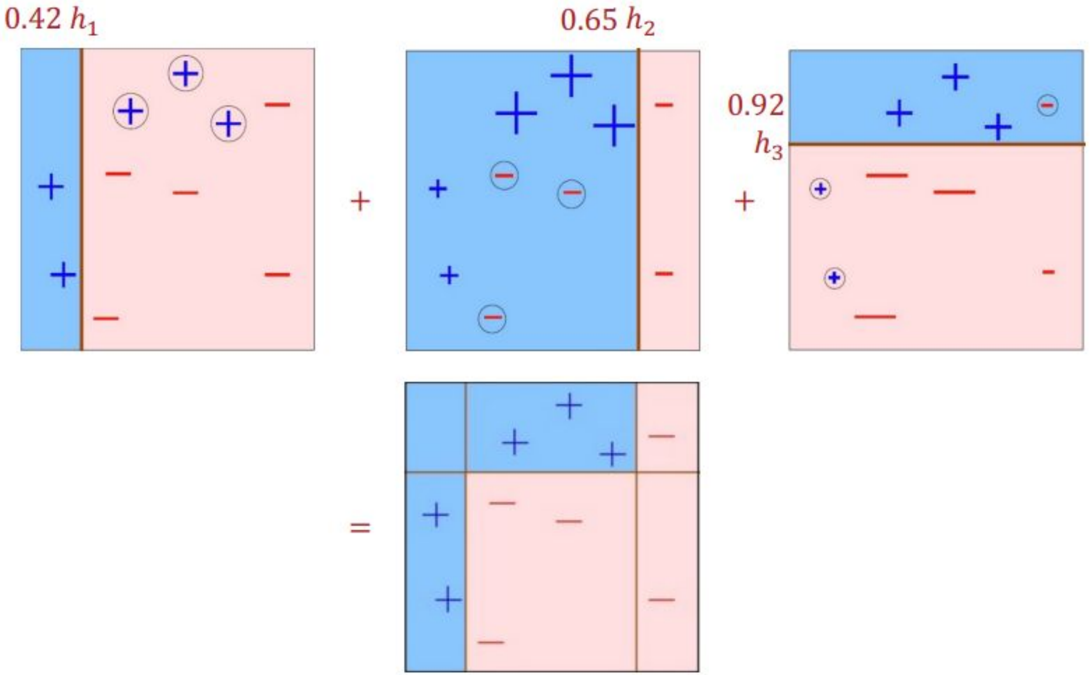
- If $\epsilon_t < \frac{1}{2}$, then $\frac{1-\epsilon_t}{\epsilon_t} > 1$
- If $\frac{1-\epsilon_t}{\epsilon_t} > 1$, then $\alpha_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right) > 0$
- If $\alpha_t > 0$, then $e^{-\alpha_t} < 1$ and $e^{\alpha_t} > 1$

Adaboost Example



Slide Credit: Henry Chai

Adaboost Example



Slide Credit: Henry Chai

Why AdaBoost?

1. If you want to use weak learners …
2. … and want your final hypothesis to be a weighted combination of weak learners, …
3. … then Adaboost greedily minimizes the exponential loss: $e(h(\boldsymbol{x}), y) = e^{(-yh(\boldsymbol{x}))}$

1. Because they're low variance / computational constraints
2. Because weak learners are not great on their own
3. Because the exponential loss upper bounds binary error

Slide Credit: Henry Chai