CS182: Introduction to Machine Learning – Ensemble Methods: Boosting & Bagging + Kmeans

Yujiao Shi
SIST, ShanghaiTech

Spring, 2025

ADABOOST

Comparison

Weighted Majority Algorithm

- an example of an ensemble method
- assumes the classifiers are learned ahead of time
- only learns (majority vote) weight for each classifiers

AdaBoost

- an example of a boosting method
- simultaneously learns:
  - the classifiers themselves
  - (majority vote) weight for each classifiers

AdaBoost

上海科技大学 ShanghaiTech University

- Definitions
  - Def: a **weak learner** is one that returns a hypothesis that is not much better than random guessing
  - Def: a **strong learner** is one that returns a hypothesis of arbitrarily low error

- AdaBoost answers the following question:
  - *Does that exist an efficient learning algorithm that can combine weak learners to obtain a strong learner?*

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:

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

- Output: an aggregated hypothesis

$$g_T(\boldsymbol{x}) = \operatorname{sign}\left(H_T(\boldsymbol{x})\right)$$

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

Setting α_t

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

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

Intuition: we want good weak learners to have high importances

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

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

Setting α_t

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

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

Intuition: we want good weak learners to have high importances

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

- How does the importance of a very bad/mostly incorrect weak learner compare to the importance of a very good/mostly correct weak learner?

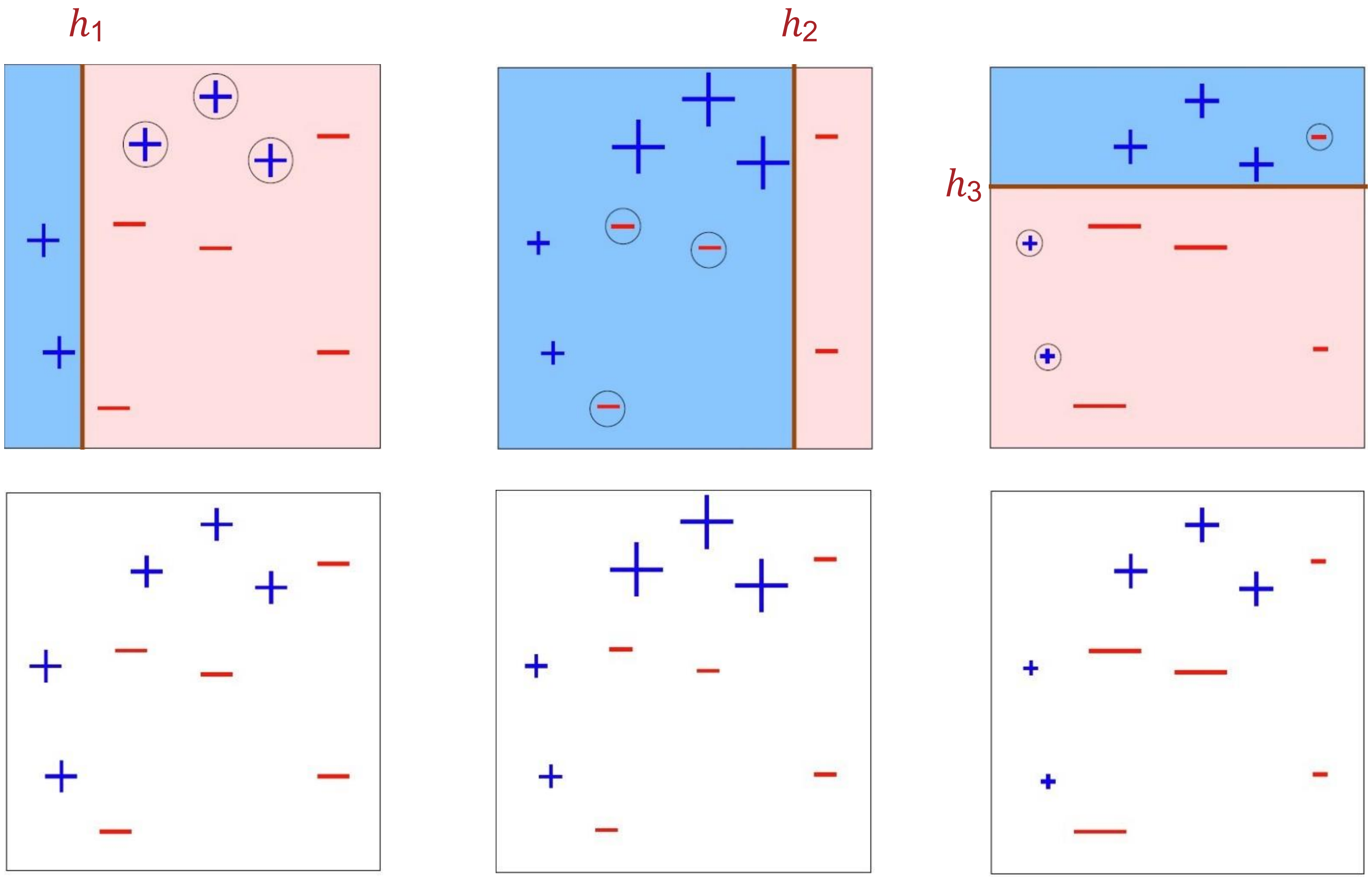
A. Similar magnitude, same sign **(TOXIC)**
B. Similar magnitude, different sign
C. Different magnitude, same sign
D. Different magnitude, different sign

AdaBoost: Example

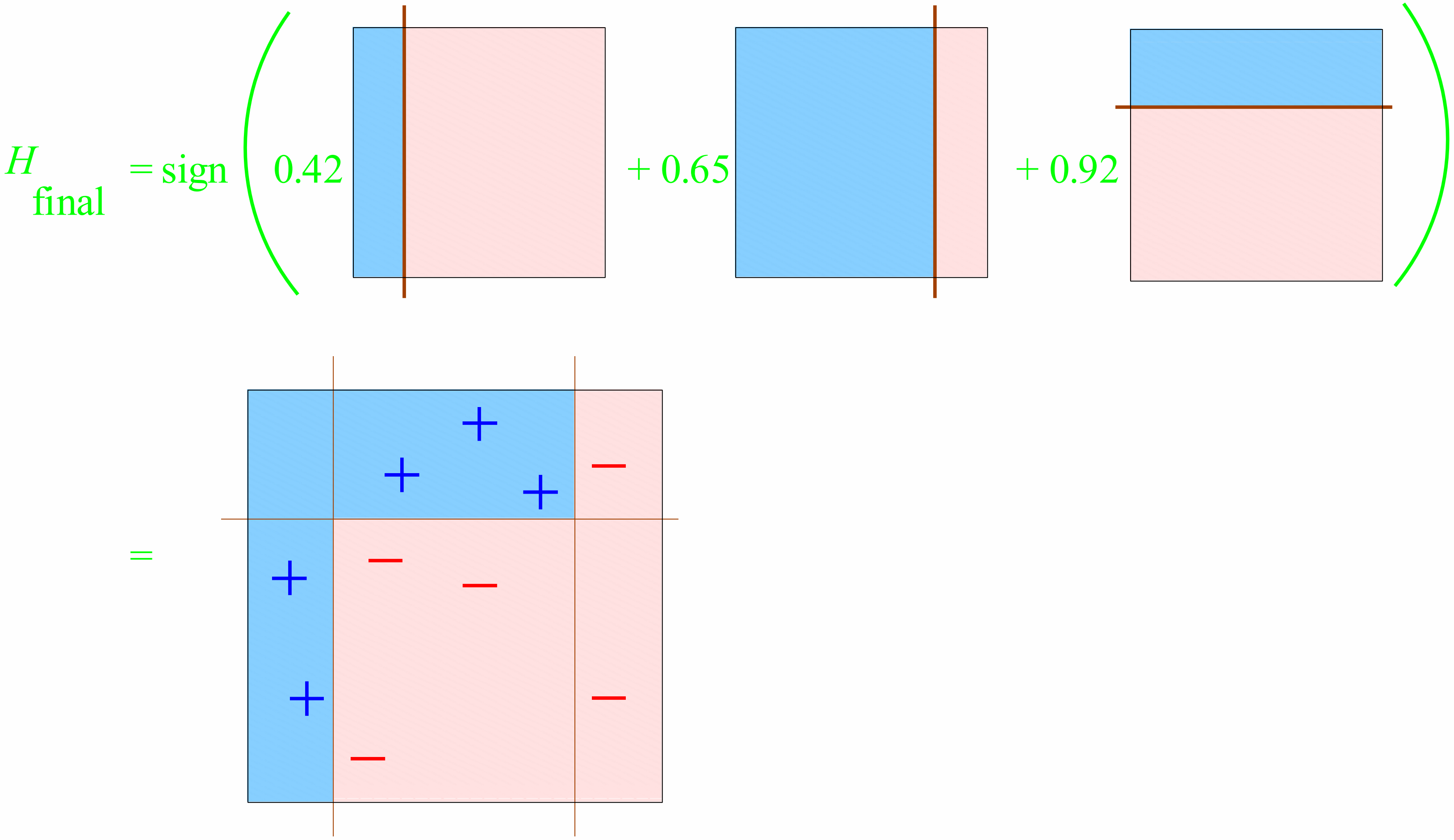
$\epsilon_1 = 0.3$
$\alpha_1 = 0.42$

$\epsilon_2 = 0.21$
$\alpha_2 = 0.65$

$\epsilon_3 = 0.14$
$\alpha_3 = 0.92$



AdaBoost: Example



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:

$$e(h(\boldsymbol{x}), y) = e^{(-y h(\boldsymbol{x}))}$$

$$h(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

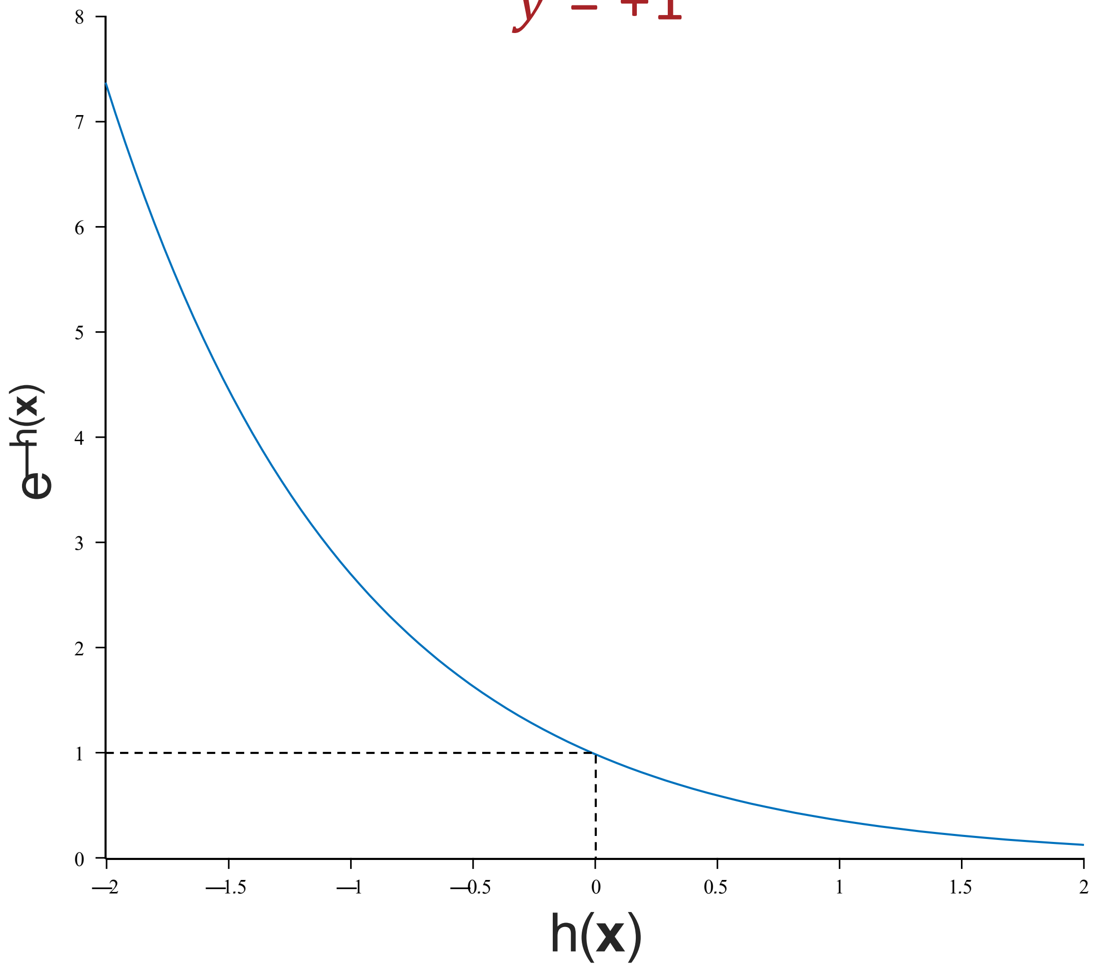
1. Because they're low variance / computational constraints
2. Because weak learners are not great on their own
3. Because the exponential loss upper bounds binary error

Exponential Loss

$$e(h(\boldsymbol{x}), y) = e^{(-y h(\boldsymbol{x}))}$$

The more $h(\boldsymbol{x})$ "agrees with" y, the smaller the loss and the more $h(\boldsymbol{x})$ "disagrees with" y, the greater the loss

$y = +1$

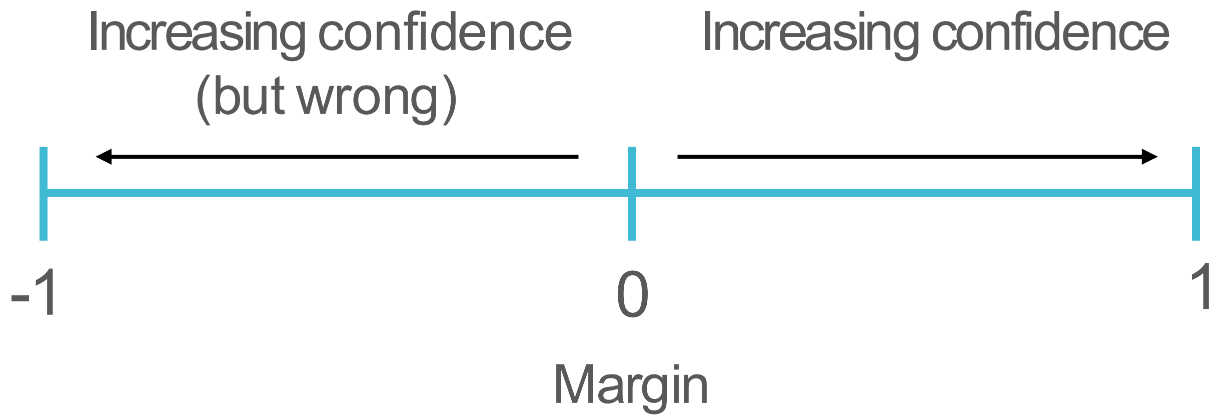


Margins

- The *margin* of training point $\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)$ is defined as:

$$m\left(\boldsymbol{x}^{(i)}, y^{(i)}\right) = \frac{y^{(i)} \sum_{t=1}^{T} \alpha_t h_t\left(\boldsymbol{x}^{(i)}\right)}{\sum_{t=1}^{T} \alpha_t}$$

- The margin can be interpreted as how confident g_T is in its prediction: the bigger the margin, the more confident.



True Error (Freund & Schapire, 1995)

- For AdaBoost, with high probability:

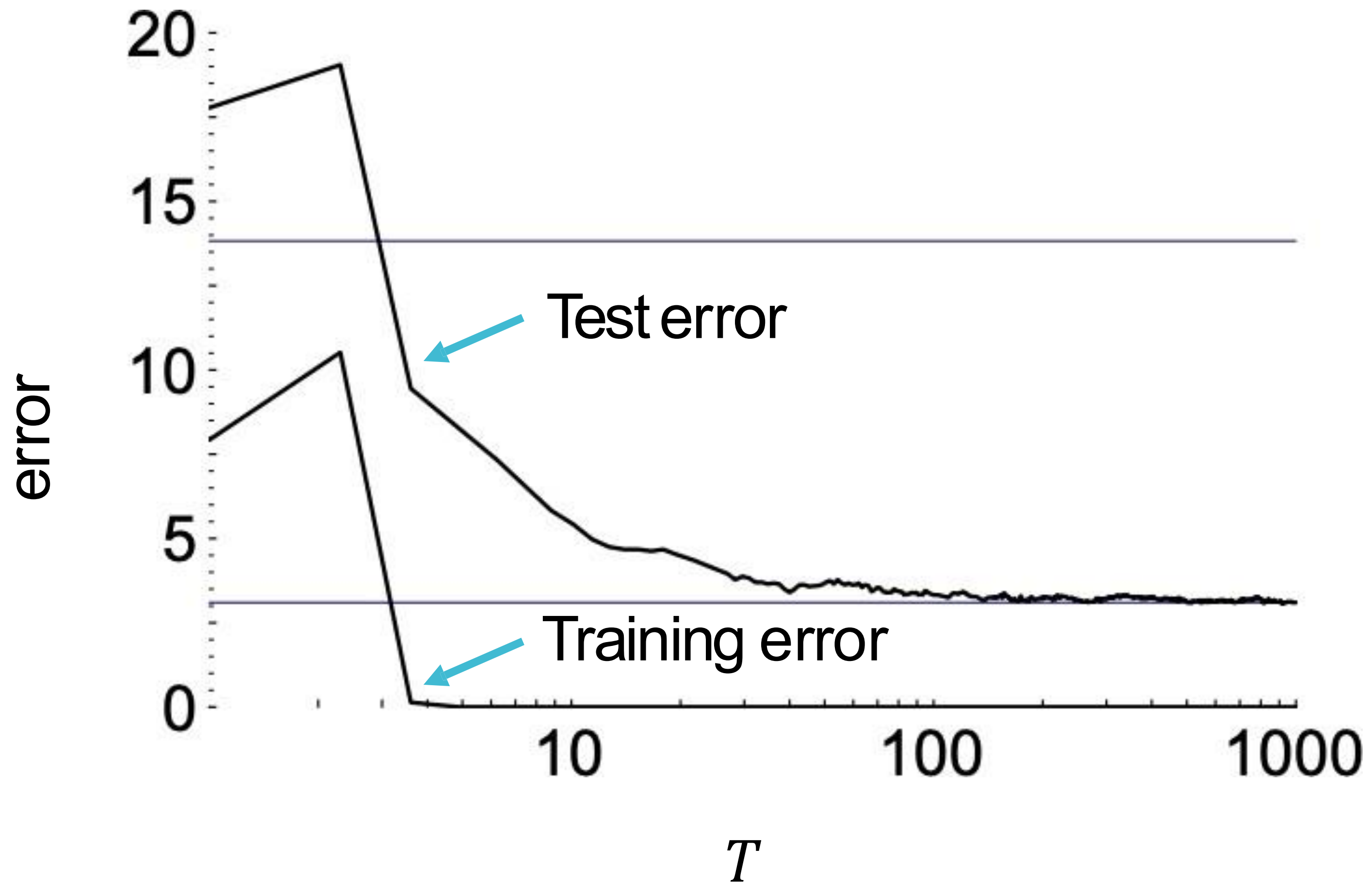
$$\text{True Error} \leq \text{Training Error} + O\left(\sqrt{\frac{d_{vc}(\mathcal{H})T}{N}}\right)$$

where $d_{vc}(\mathcal{H})$ is the VC-dimension of the weak learners and T is the number of weak learners.

- Empirical results indicate that increasing T does not lead to overfitting as this bound would suggest!

Source: http://rob.schapire.net/papers/FreundSc95.pdf

Test Error (Schapire, 1989)



- The training error often drops to zero quickly — but even after that, the test error continues to improve!

- Adaboost aims to maximize the margin, i.e., minimize the exponential loss

$$e(h(\boldsymbol{x}), y) = e^{(-y h(\boldsymbol{x}))}$$

$$h(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

Source: http://rob.schapire.net/papers/msri.pdf

Why Adaboost Minimizes the Exponential Loss?

上海科技大学
ShanghaiTech University

$$\mathcal{L} = \sum_{i=1}^{n} e^{(-y_i F(x_i))} \qquad \text{where} \qquad F(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

At step t:

$$F_t(x) = F_{t-1}(x) + \alpha_t h_t(x)$$

$$\begin{aligned}
\mathcal{L} &= \sum_{i=1}^{n} e^{(-y_i F_t(x))} \\
&= \sum_{i=1}^{n} e^{\left(-y_i\left(F_{t-1}(x)+\alpha_t h_t(x)\right)\right)} \\
&= \sum_{i=1}^{n} e^{(-y_i F_{t-1}(x))} e^{(-y_i \alpha_t h_t(x))} \\
&= \sum_{i=1}^{n} w_i e^{(-y_i \alpha_t h_t(x))}
\end{aligned}$$

$$\mathcal{L} = \sum_{i=1}^{n} w_i e^{(-y_i \alpha_t h_t(x))} = \begin{cases} \sum_{i=1}^{n} w_i e^{(-\alpha_t)} & \text{if correct} \\ \sum_{i=1}^{n} w_i e^{(\alpha_t)} & \text{if wrong} \end{cases}$$

Why Adaboost Minimizes the Exponential Loss?

上海科技大学
ShanghaiTech University

$$\mathcal{L} = \sum_{i=1}^{n} e^{(-y_i F(x_i))} \qquad \text{where} \qquad F(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

At step t:

$$\mathcal{L} = \sum_{i=1}^{n} w_i e^{(-y_i \alpha_t h_t(x))}$$

$$= \begin{cases} \sum_{i=1}^{n} w_i e^{(-\alpha_t)} & if\ correct \\ \sum_{i=1}^{n} w_i e^{(\alpha_t)} & if\ wrong \end{cases}$$

$$= \sum_{correct} w_i e^{(-\alpha_t)} + \sum_{wrong} w_i e^{(\alpha_t)}$$

$$\frac{\partial \mathcal{L}}{\partial \alpha_t} = 0$$

$$\Rightarrow \quad -\sum_{correct} w_i e^{(-\alpha_t)} + \sum_{wrong} w_i e^{(\alpha_t)} = 0$$

$$\Rightarrow \quad \alpha_t = \frac{1}{2} ln\left(\frac{\sum_{correct} w_i}{\sum_{wrong} w_i}\right)$$

Ensemble Methods: Boosting

You should be able to…

1. Explain how a weighted majority vote over linear classifiers can lead to a non-linear decision boundary
2. Implement AdaBoost
3. Describe a surprisingly common empirical result regarding Adaboost train/test curves

Ensemble Methods

Ensemble methods learn a collection of models (i.e. the **ensemble**) and combine their predictions on a test instance.

We consider two types:

- **Bagging:** learns models in parallel by taking many subsets of the training data
- **Boosting:** learns models serially by reweighting the training data

BAGGING

Bagging

“BAGGing” is also called Boostrap AGGregretion

Bagging answers the question:

How can I obtain many classifiers/regressors to ensemble together?

We’ll consider three possible answers:

1. **(sample) bagging**
2. **feature bagging** (aka. random subspace method)
3. **random forests** (which combine sample bagging and feature bagging to train a “forest” of decision trees)

(Sample) Bagging

Key idea: Repeatedly sample with replacement a collection of training examples and train a model on that sample.

Return an ensemble of the trained models; combine predictions by majority vote for classification and by averaging for regression.

Algorithm 1 (Sample) Bagging

```
procedure SampleBagging(D, T, S)
    for t = 1,..., T do
        for s = 1,..., S do
            i_s ~ Uniform(1,..., N)
        S_t = {(x^(i_s), y^(i_s))}_{s=1}^S        ◃ Bootstrap sample
        h_t = train(S_t)                          ◃ Classifier
    return ĥ(x) = aggregate(h_1,..., h_T)         ◃ Ensemble
```

for classification: $\hat{h}(\mathbf{x}) = \operatorname{argmax}_{y \in Y} \sum_{t=1}^{T} \mathrm{I}[y = h_t(\mathbf{x})]$ ◃ Majority vote

for regression: $\hat{h}(\mathbf{x}) = \frac{1}{T} \sum_{t=1}^{T} h_t(\mathbf{x})$ ◃ Average

(Sample) Bagging

training data D

i	x_1	x_2	x_3	y
1	1	0	1	+
2	0	1	1	-
3	1	1	0	+
4	0	1	0	+
5	1	0	0	-

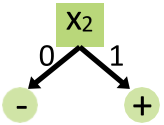
test instance

x_1	x_2	x_3
0	0	0

bootstrap sample S_1

i	x_1	x_2	x_3	y
3	1	1	0	+
5	1	0	0	-
3	1	1	0	+

classifier h_1

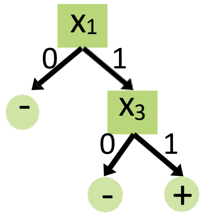


-

bootstrap sample S_2

i	x_1	x_2	x_3	y
2	0	1	1	-
5	1	0	0	-
1	1	0	1	+

classifier h_2

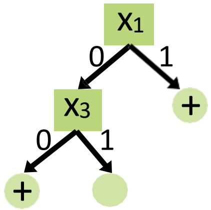


-

bootstrap sample S_3

i	x_1	x_2	x_3	y
2	0	1	1	-
4	0	1	0	+
1	1	0	1	+

classifier h_3



-

+

majority vote

Feature Bagging

Key idea: Repeatedly sample with replacement a subset of the features, create a copy of the training data with only those features, and train a model on the copy.

Return an ensemble of the trained models; combine predictions by majority vote for classification and by averaging for regression.

Algorithm 2 Feature Bagging

---

procedure SampleBagging(D, T, S)
    for $t = 1, \dots, T$ **do**
        for $s = 1, \dots, S$ **do**
            $m_s \sim \text{Uniform}(1, \dots, M)$
        for $i = 1, \dots, N$ **do**
            $\tilde{\mathbf{x}}^{(i)} = [x^{(i)}_{m_1}, x^{(i)}_{m_2}, \dots, x^{(i)}_{m_S}]^T$
        $D_t = \{(\tilde{\mathbf{x}}^{(i)}, y^{(i)})\}_{i=1}^{N}$ ◃Random subspace
        $h_t = \text{train}(D_t)$ ◃Classifier
    return $\hat{h}(\mathbf{x}) = \text{aggregate}(h_1, \dots, h_T)$ ◃Ensemble

---

Feature Bagging

training data D

i	x_1	x_2	x_3	x_4	y
1	1	0	1	0	+
2	0	1	1	1	-
3	1	1	0	0	+

bootstrap sample S_1

i	x_4	x_2	y
1	0	0	+
2	1	1	-
3	0	1	+

bootstrap sample S_2

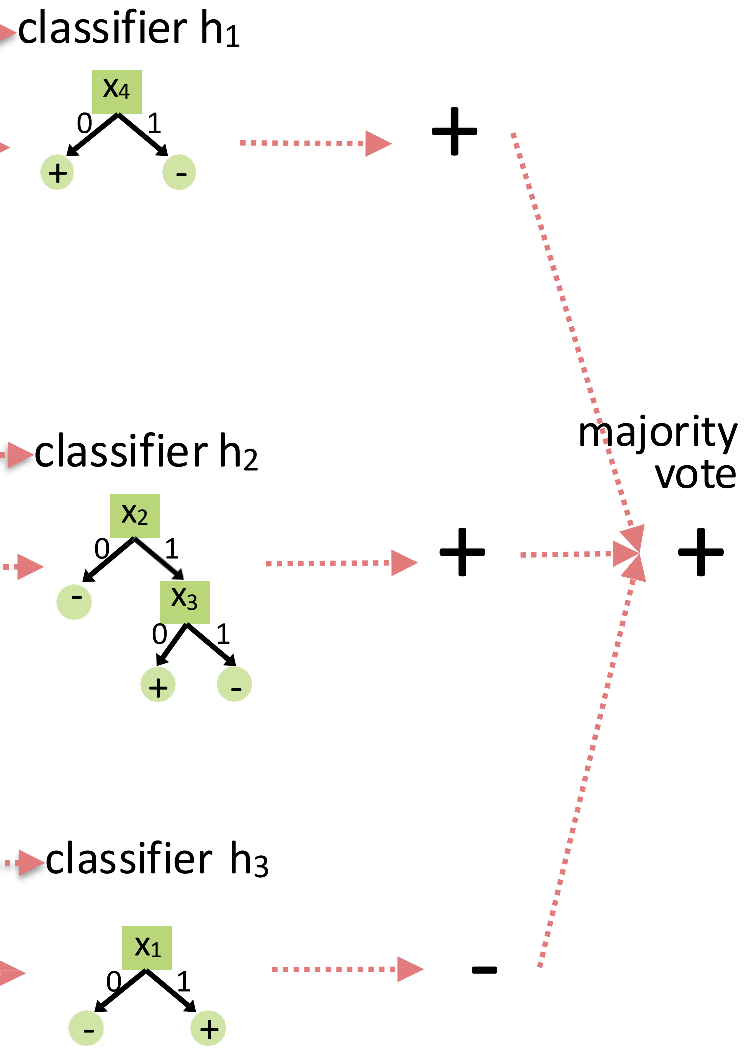
i	x_2	x_3	y
1	0	1	+
2	1	1	-
3	1	0	+

bootstrap sample S_3

i	x_1	x_3	y
1	1	1	+
2	0	1	-
3	1	0	+

test instance

x_1	x_2	x_3	x_4
0	1	0	0

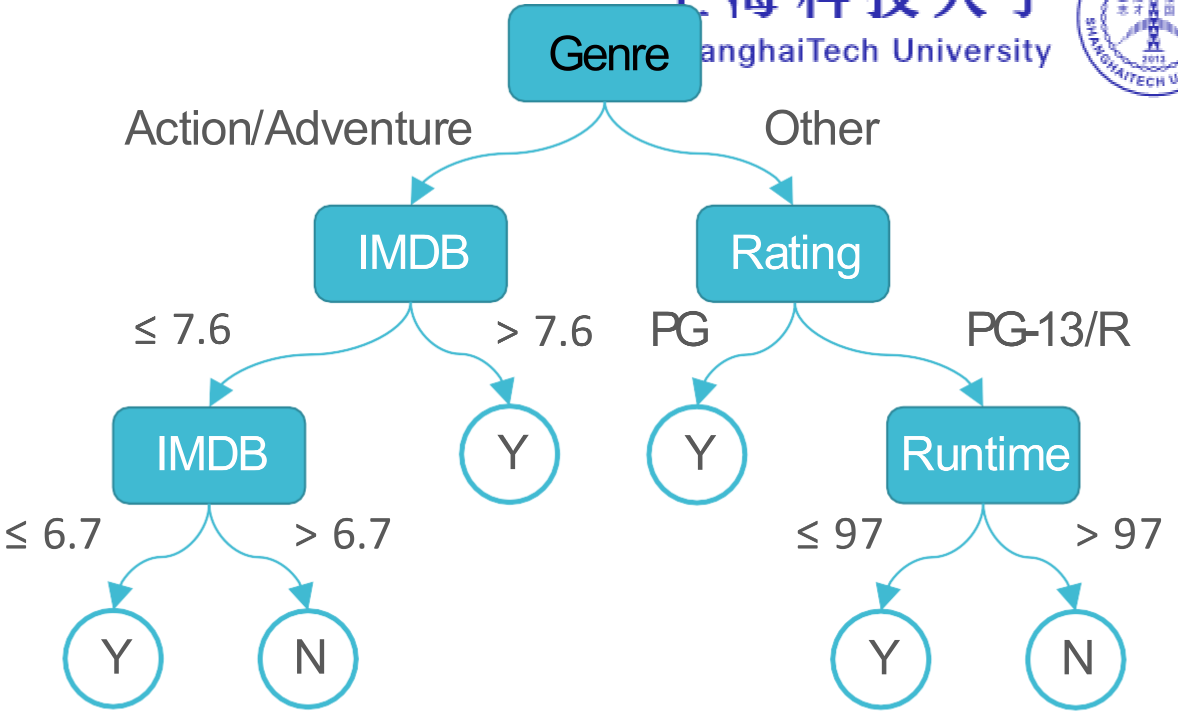


RANDOM FORESTS

Decision Trees: 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
  - Limited expressivity (especially short trees, i.e., stumps)
    - Can be addressed via boosting
  - Highly variable
    - Can be addressed via bagging → random forests

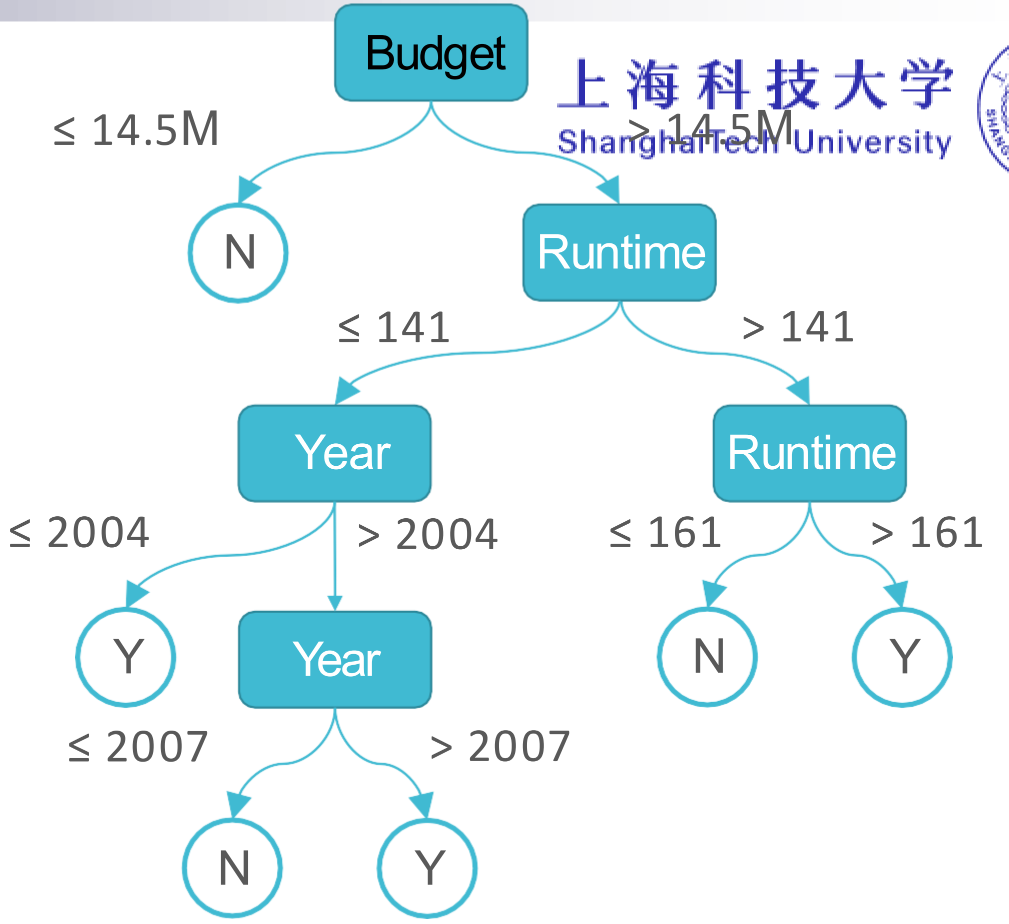
MovieID	Runtime	Genre	Budget	Year	IMDB	Rating	Liked?
1	124	Action	18M	1980	8.7	PG	Y
2	105	Action	30M	1984	7.8	PG	Y
3	103	Comedy	6M	1986	7.8	PG-13	N
4	98	Adventure	16M	1987	8.1	PG	Y
5	128	Comedy	16.4M	1989	8.1	PG	Y
6	120	Comedy	11M	1992	7.6	R	N
7	120	Drama	14.5M	1996	6.7	PG-13	N
8	136	Action	115M	1999	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	N
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

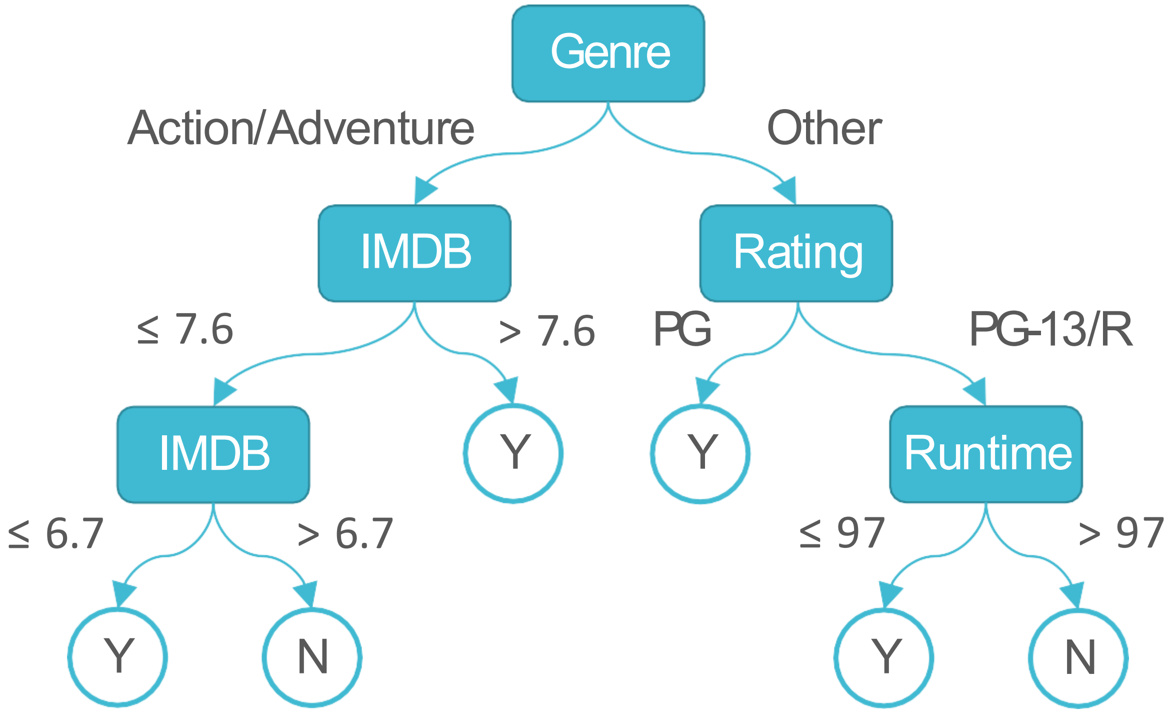
Source: https://www.kaggle.com/datasets/danielgrijalvas/movies

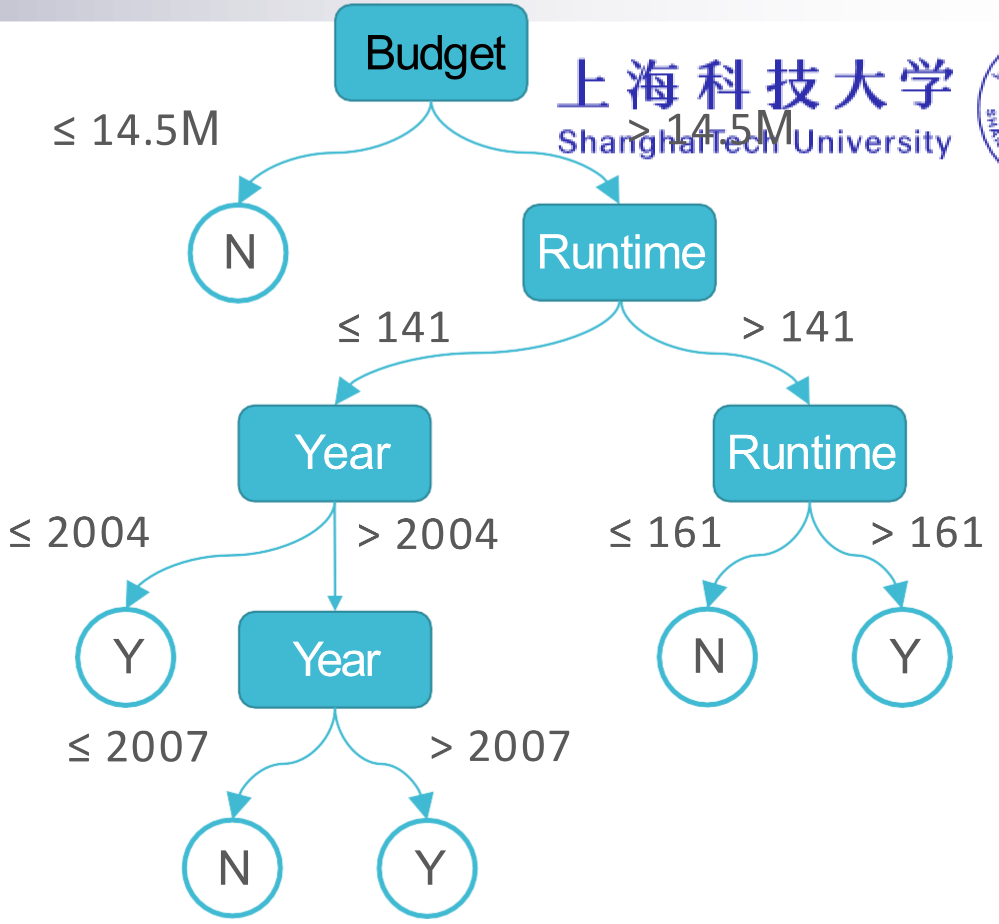
MovieID	Runtime	Genre	Budget	Year	IMDB	Rating	Liked?
1	124	Action	18M	1980	8.7	PG	Y
2	105	Action	30M	1984	7.8	PG	Y
3	103	Comedy	6M	1986	7.8	PG-13	N
4	98	Adventure	16M	1987	8.1	PG	Y
5	128	Comedy	16.4M	1989	8.1	PG	Y
6	120	Comedy	11M	1992	7.6	R	N
7	120	Drama	14.5M	1996	6.7	PG-13	N
8	136	Action	115M	1999	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

Source: https://www.kaggle.com/datasets/danielgrijalvas/movies





Decision Trees

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 = sample bagging + feature bagging

  = <u>**b**</u>ootstrap <u>**agg**</u>regat<u>**ing**</u> + split-feature randomization

Random Forests

Key idea: Combine (sample) bagging and a specific variant of feature bagging to train decision trees.

Repeat the following to train many decision trees:

- draw a sample with replacement from the training examples,
- recursively learn the decision tree
- but at each node when choosing a feature on which to split, first randomly sample a subset of the features, then pick the best feature from among that subset.

Return an ensemble of the trained decision trees.

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

…

…

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

…

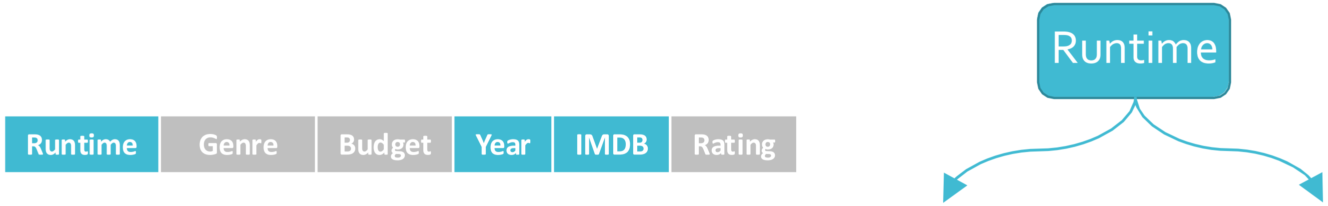
Split-feature Randomization

- 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

Runtime	Genre	Budget	Year	IMDB	Rating

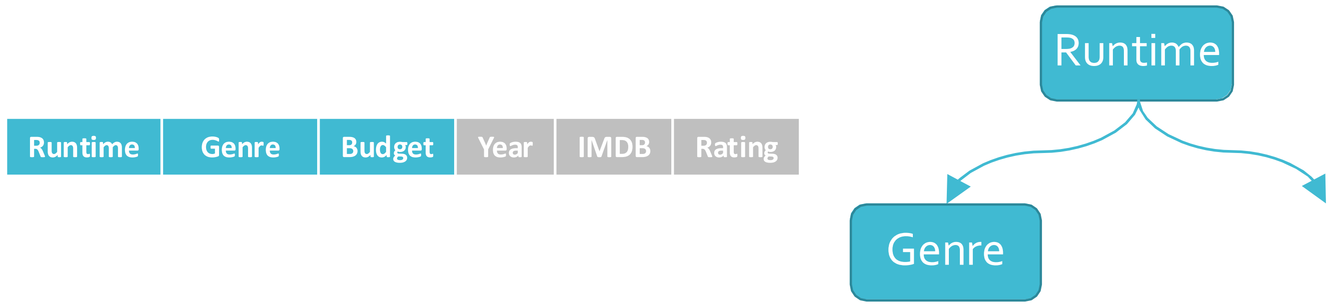
Split-feature Randomization

- 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



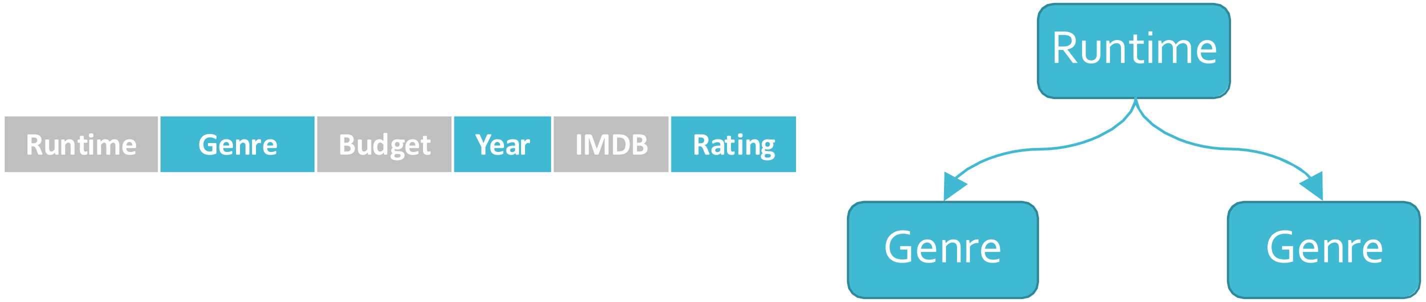
Split-feature Randomization

- 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



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: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, B, \rho$
- 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

How can we set B and ρ?

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, B, \rho$
- 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

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}

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 $t^{(-n)} = \left\{t_1^{(-n)}, t_2^{(-n)}, \dots, t_{N\text{-}n}^{(-n)}\right\}$
- 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

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

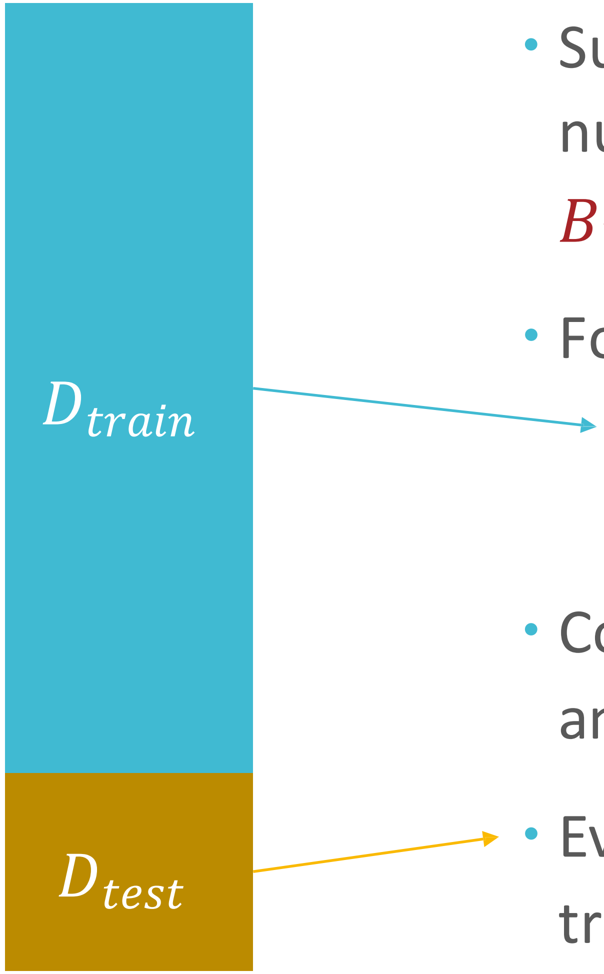
Out-of-bag Error

- 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 $t^{(-n)} = \left\{t_1^{(-n)}, t_2^{(-n)}, \ldots, t_{N_{-n}}^{(-n)}\right\}$
- 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

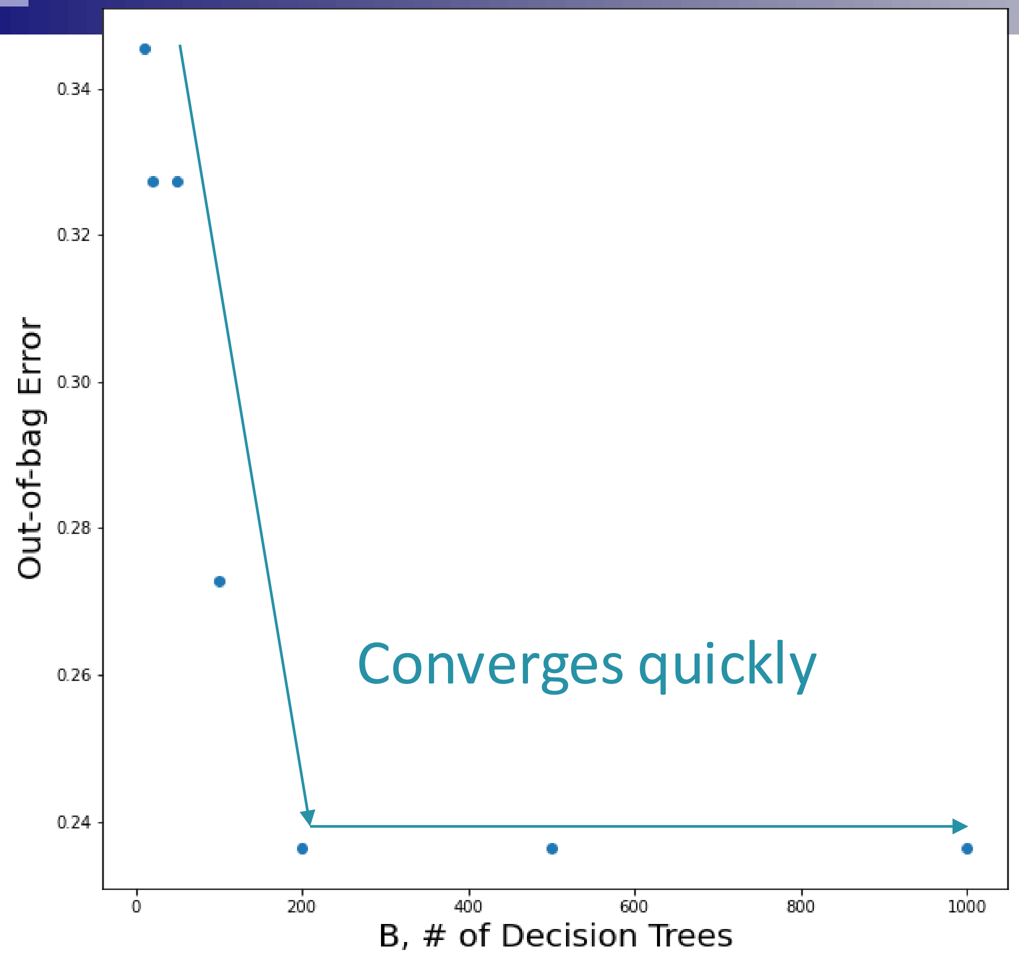
$$E_{OOB} = \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}\left(\bar{t}^{(-n)}\left(\boldsymbol{x}^{(n)}\right) \neq y^{(n)}\right)$$

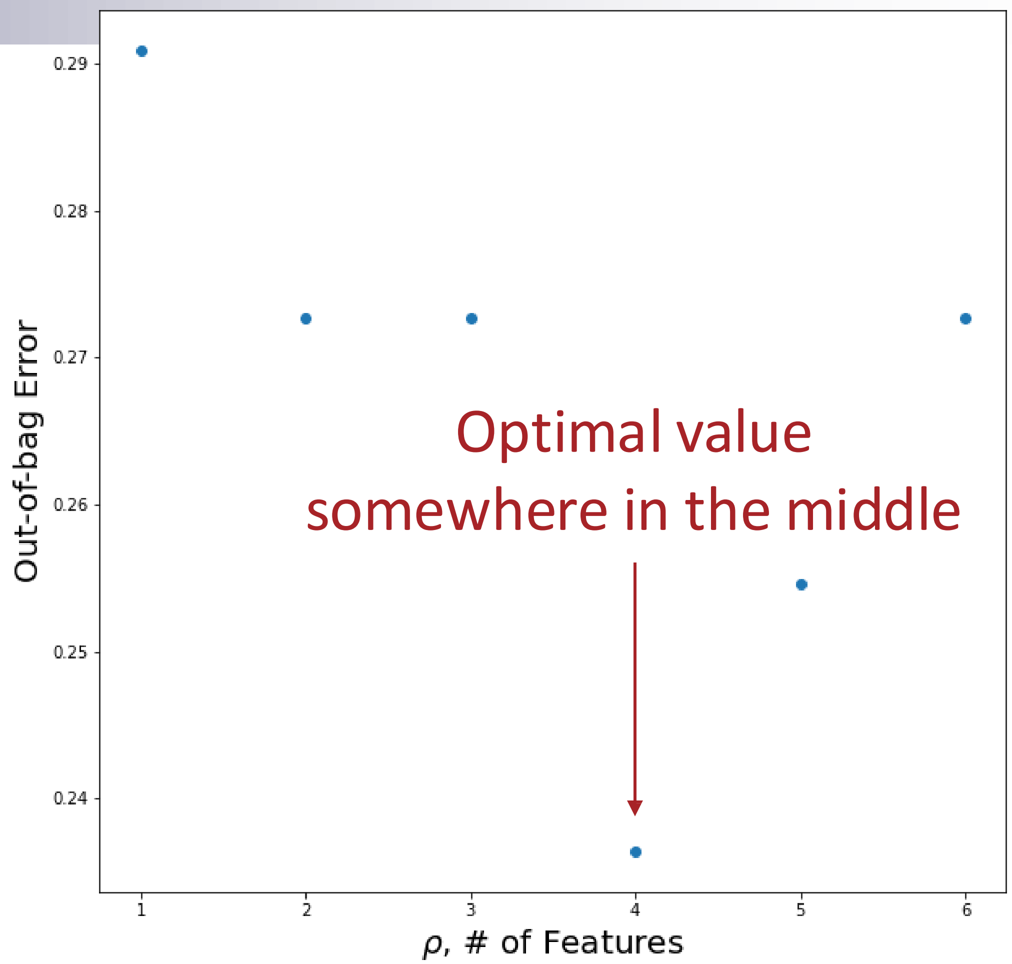
- E_{OOB} can be used for hyperparameter optimization!

Out-of-bag Error



- Suppose we want to compare different numbers of trees in our random forest $B_1, \ldots, B_K$
- For $k = 1, 2, \ldots, K$
  - 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}



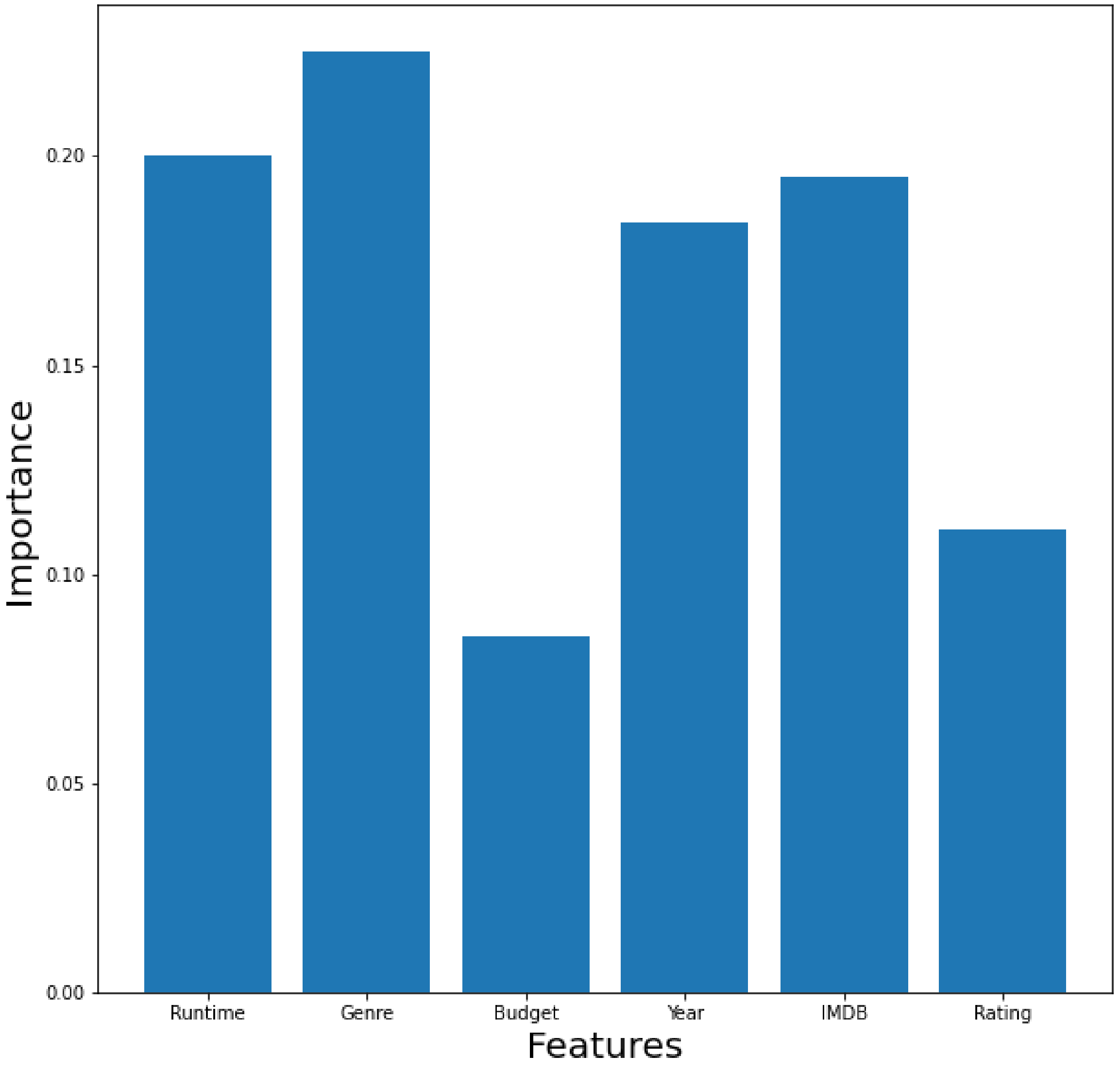


Setting Hyperparameters

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 entropy (weighted by the number of data points in the split) to its importance

Feature Importance



Key Takeaways

- 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

Question

上海科技大学
ShanghaiTech University

- What is the difference between AdaBoost and Random Forest?

- Can we design an aglorithm that combines the merits of both methods?
  - a potential suggestion for Project

Learning Objectives

Ensemble Methods: Bagging

You should be able to…

1. Distinguish between (sample) bagging, the random subspace method, and random forests.
2. Implement (sample) bagging for an arbitrary base classifier/regressor.
3. Implement the random subspace method for an arbitrary base classifier/ regressor.
4. Implement random forests.
5. Contrast out-of-bag error with cross-validation error.
6. Differentiate boosting from bagging, when to use which.
7. Compare and contrast weighted and unweighted majority vote of a collection of classifiers.
8. Discuss the relation in bagging between the sample size and variance of the base classifier/regressor (project suggestion).
9. Bound the generalization error of a random forest classifier (project suggestion).