CS559 Machine Learning
Decision Tree and Boosting

Tian Han

Department of Computer Science
Stevens Institute of Technology

Week 14

Outline

- Decision Tree Model
- Bagging, Random Forest and Boosting

Decision Tree Models

Going Beyond the Linear Model

Recall that we have generalized linear model using basis functions $\phi(.)$:

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

Going Beyond the Linear Model

Recall that we have generalized linear model using basis functions $\phi(.)$:

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

- In support vector machine, $\phi(\mathbf{x})$ is designed in terms of kernels.

Going Beyond the Linear Model

Recall that we have generalized linear model using basis functions $\phi(.)$:

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

- In support vector machine, $\phi(\mathbf{x})$ is designed in terms of kernels.
- In Neural Network, $\phi(\mathbf{x})$ is itself expressed as linear models of features at lower layers.

Going Beyond the Linear Model

Recall that we have generalized linear model using basis functions $\phi(.)$:

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

- In support vector machine, $\phi(\mathbf{x})$ is designed in terms of kernels.
- In Neural Network, $\phi(\mathbf{x})$ is itself expressed as linear models of features at lower layers.
- In decision trees and boosting machines, $\phi(\mathbf{x})$s are indicator functions or shallow trees/weak classifiers.

A learning problem: predict fuel efficiency

Table: From the UCI repository

cylinders	displacement	horsepower	weight	acceleration	modelyear	maker	mpg
4	low	low	low	high	75-78	asia	good
6	medium	medium	medium	medium	70-74	america	bad
8	high	high	high	low	70-74	america	bad
4	medium	medium	medium	low	75-78	europe	bad
...	...	...	...	...	...	...	...
4	low	medium	low	medium	75-78	europe	good

- 40 data points
- Goal: predict MPG (good or bad)
- Need to find: $f : X \to Y$
- **Discrete** features/attributes, **classification**

Decision trees $f : X \to Y$

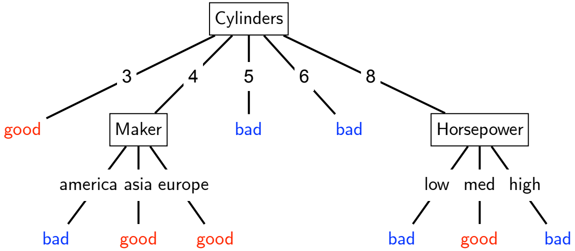


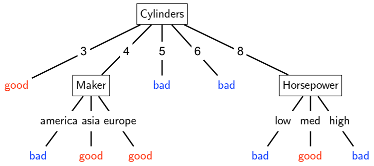
Figure: Human interpretable!

- Each internal node tests an attribute x_i
- Each branch assigns an attribute value $x_i = v$
- Each leaf assigns a class
- To classify input $\mathbf{x}$: traverse the tree from root to leaf, output the labeled y

Hypothesis space

Table: From the UCI repository

cylinders	displacement	horsepower	weight	acceleration	modelyear	maker	mpg
4	low	low	low	high	75-78	asia	good
6	medium	medium	medium	medium	70-74	america	bad
8	high	high	high	low	70-74	america	bad
4	medium	medium	medium	low	75-78	europe	bad
...	...	...	...	...	...	...	...
4	low	medium	low	medium	75-78	europe	good

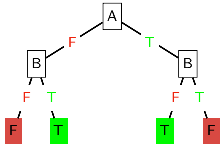


- How many possible hypotheses (how many trees can be constructed)?
- What functions can be represented?

What functions can be represented?

- Decision trees can represent any function of the input attributes
- For Boolean functions, one path to leaf gives a truth table row
- But, it could require exponentially many nodes...

A	B	A XOR B
F	F	F
F	T	T
T	F	T
T	T	F



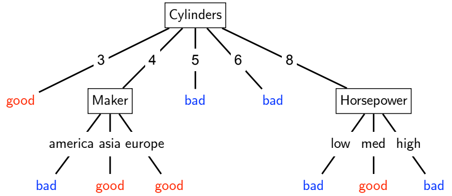


Figure: Predicting mpg=good: cyl $= 3 \vee ($cyl $= 4 \wedge ($maker $=$ asia $\vee$ maker $=$ europe$)) \vee \ldots$

Hypothesis space

- How many possible hypotheses?
- What functions can be represented?
- How many will be consistent with a given dataset?
- How will we choose the best one?
  - First look at how to split nodes, then consider how to find the best tree.

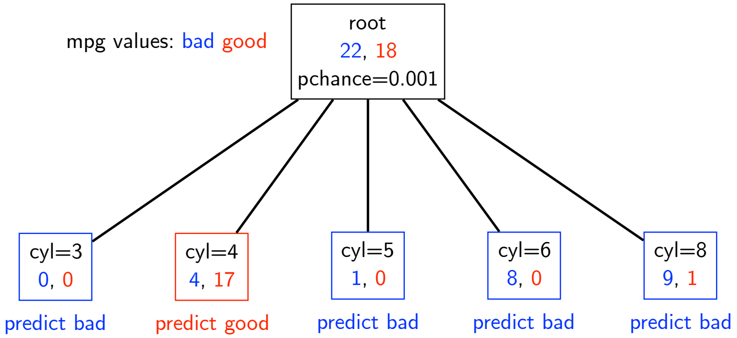
What is the simplest tree?

Table: From the UCI repository

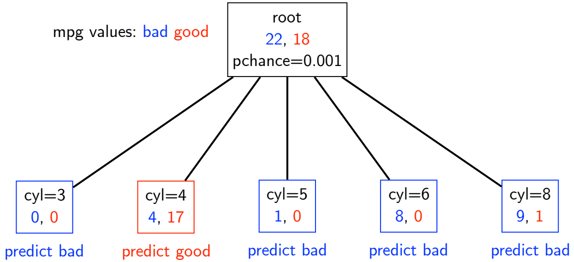
cylinders	displacement	horsepower	weight	acceleration	modelyear	maker	mpg
4	low	low	low	high	75-78	asia	good
6	medium	medium	medium	medium	70-74	america	bad
8	high	high	high	low	70-74	america	bad
4	medium	medium	medium	low	75-78	europe	bad
...	...	...	...	...	...	...	...
4	low	medium	low	medium	75-78	europe	good

- Predict mpg=bad
- Is this a good tree? Total we get $(22+, 18-)$, which means we are correct on 22 examples and wrong on 18 examples.

A simple decision tree

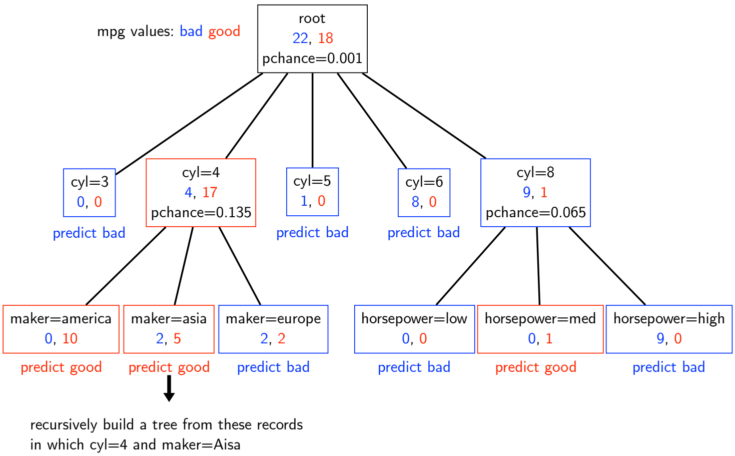


Recursive step

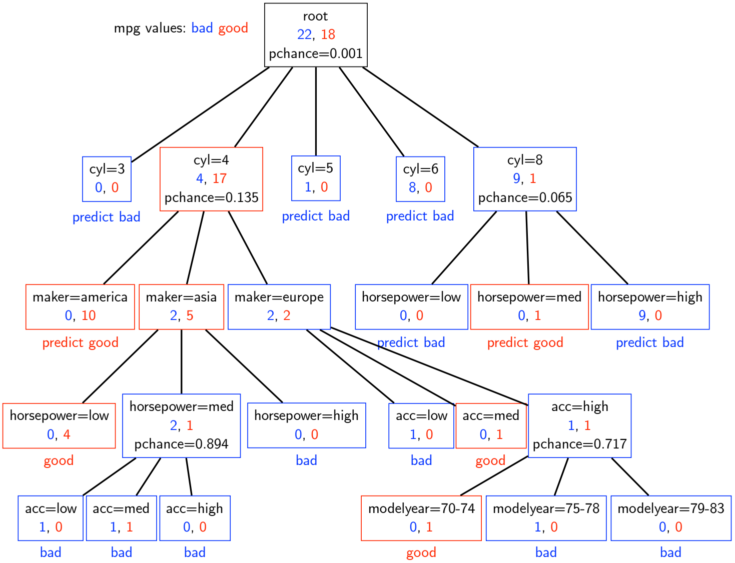


- Take the original dataset
- Partition it according to the values of the attribute we split on
- Build tree from these records (cyl=4, cyl=5, cyl=6, cyl=8)

Second level of a decision tree

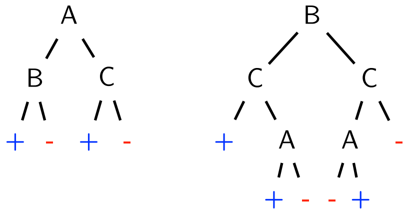


A full decision tree



Are all decision trees equal?

- Many trees can represent the same concept
- But, not all trees will have the same size!



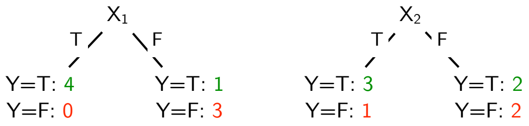
- Which tree do we prefer?

Learning decision trees is hard

- Learning the simplest (smallest) decision tree is an NP-complete problem [Hyafil & Rivest 76]
- Resort to a greedy heuristic:
  - Start from empty decision tree
  - Split on next best attribute (feature)
  - Recurse

Splitting: choosing a good attribute

- Would we prefer to split on X_1 or X_2?



- Idea: use counts as leaves to define probability distribution, so we can measure uncertainty.

X_1	X_2	**Y**
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

Splitting: choosing a good attribute

- Good split if we are more certain about classification after split
    - Deterministic good (all true or all false)
    - Uniform distribution bad
    - What about distributions in between?

$$P(X{=}A) = 1/2 \quad P(X{=}B) = 1/4 \quad P(X{=}C) = 1/8 \quad P(X{=}D) = 1/8$$
$$P(X{=}A) = 1/4 \quad P(X{=}B) = 1/4 \quad P(X{=}C) = 1/4 \quad P(X{=}D) = 1/4$$

Entropy

- Entropy $H(Y)$ of a random variable Y:

$$H(Y) = -\sum_{i=1}^{K} P(Y = y_i) \log P(Y = y_i)$$

- More uncertainty, more entropy!
- Information theory interpretation: $H(Y)$ is the expected number of bits needed to encode a randomly drawn value of Y (under most efficient code)

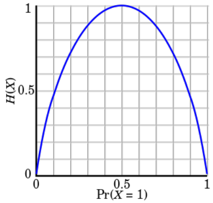


Figure: Entropy of a coin flip. [Wikipedia]

Entropy

- High Entropy
  - Y is from a uniform like distribution
  - Flat histogram
  - Values sampled from it are less predictable
- Low Entropy
  - Y is from a varied distribution(peaks and valleys)
  - Histogram has many lows and highs
  - Values sampled from it are more predictable

Entropy example

Entropy:

$$H(Y) = -\sum_{i=1}^{K} P(Y = y_i) \log P(Y = y_i)$$

In this example:

$$\begin{aligned} P(Y = T) &= 5/6 \\ P(Y = F) &= 1/6 \\ H(Y) &= -5/6 \log 5/6 - 1/6 \log 1/6 \\ &= 0.65 \end{aligned}$$

X_1	X_2	**Y**
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F

Conditional entropy

Conditional entropy $H(Y|X)$ of a random variable Y conditioned on a random variable X:

$$H(Y|X) := \mathbb{E}_x H(Y|X=x)$$
$$= -\sum_{j=1}^{v} P(X=x_j) \sum_{i=1}^{K} P(Y=y_i|X=x_j) \log P(Y=y_i|X=x_j)$$

In this example:

$$P(X_1 = T) = 4/6$$
$$P(X_1 = F) = 2/6$$
$$H(Y|X_1) = -4/6(1\log 1 + 0 \log 0)$$
$$-2/6(1/2 \log 1/2 + 1/2 \log 1/2)$$
$$= 2/6$$

X_1	X_2	**Y**
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F

Information gain

Used by the various (ID3, C4.5 and C5.0) tree-generation algorithms. Decrease in entropy (uncertainty) after splitting:

$$IG(X) = H(Y) - H(Y|X)$$

In this example:

$$IG(X_1) = H(Y) - H(Y|X_1)$$
$$= 0.65 - 0.33$$

We prefer the split ($IG(X_1) > 0$)

X_1	X_2	**Y**
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F

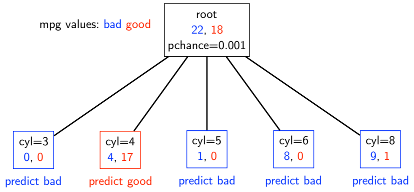
Learning decision trees

- Start from empty decision tree
- Split on next best attribute (feature) – Use information gain to select attribute
- Recurse

Information Gains - example

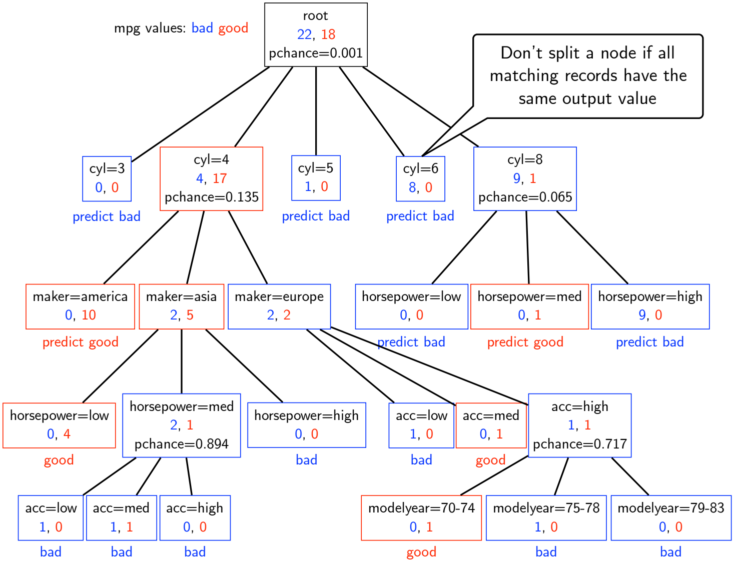
Table: Information gains using the training data set (40 records)

Input	value	Info Gain
cylinders	(3,4,5,6,8)	0.507
displacement	(low, medium, high)	0.223
horsepower	(low, medium, high)	0.388
weight	(low, medium, high)	0.304
acceleration	(low, medium, high)	0.064
modelyear	(70-74, 75-78, 79-83)	0.268

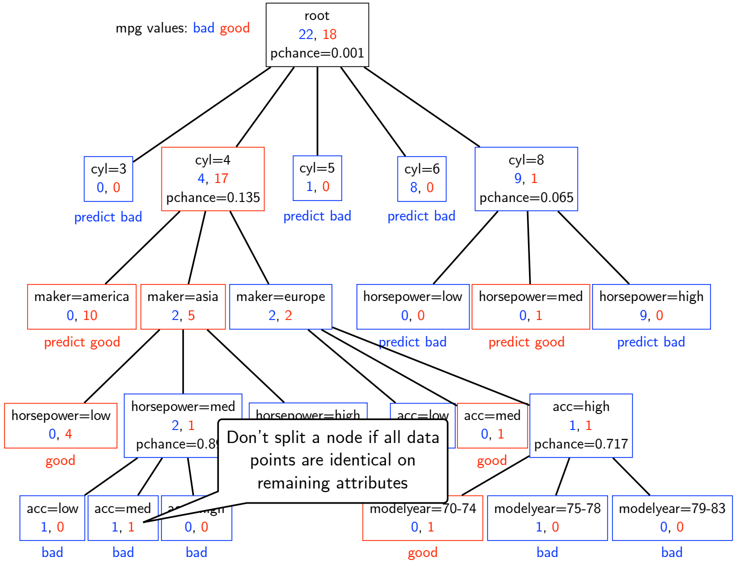


First split decided (cylinder), but when do we stop?

Base case one



Base case two



Other Metrics

- Gini index, used by the CART (classification and regression tree) algorithm.
    - a perfect separation results in a Gini score of 0

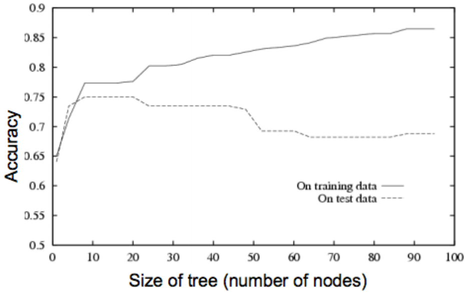
Summary: Building decision trees

BuildTree(dataset, output)

- If all output values are the same in the dataset, return a leaf node that say predict this unique output.
- If all input values are the same, return a leaf node that says "predict the majority output"
- Else find attribute X with highest Info Gain (or lowest Gini Index)
- Suppose X has n_x distinct values
    - Create a non-leaf node with n_x children
    - The i the child should be built by calling BuildTree(DS_i, output) where DS_i contains the records in dataset where $X = i$th value of X.

Decision trees overfit

- Standard decision trees have no learning bias.
  - Training set error is almost zero
  - Lots of variance
  - Prefer simpler trees
- Many strategies for picking simpler trees
  - Fixed depth
  - Fixed number of leaves
  - Or something smarter



Real-valued inputs

What should we do if some of the inputs are real-valued?

- Infinite number of possible split values
- Finite dataset, only finite number of relevant splits.

Proposed solution:

- Threshold splits.

Threshold splits

- Binary tree: spilt on attribute X at value t:
  - One branch: $X < t$
  - Other branch: $X \geq t$
- Allow repeated splits on same variable.

The set of possible thresholds

- Binary tree: spilt on attribute X at value t:
    - One branch: $X < t$
    - Other branch: $X \geq t$
- Search through possible values of t (seems hard!!!)
- But only a finite number of t's are important:
    - Sort data according to X into $\{x_1, ..., x_m\}$
    - Consider split points of the form $x_i + \frac{x_{i+1} - x_i}{2}$
    - Moreover, only splits between examples of **different classes** matter.

Picking the best threshold

- Suppose X is real valued with threshold t:
- Want $IG(Y|X:t)$, the information gain for Y when testing if X is greater than or less than t
- Define
  - $H(Y|X:t) = P(X<t)H(Y|X<t) + P(X \geq t)H(Y|X \geq t)$
  - $IG(Y|X:t) = H(Y) - H(Y|X:t)$
  - $IG^*(Y|X) = \max_t IG(Y|X:t)$
- Use $IG^*(Y|X)$ for continuous variables.

Summary

- Presented for classification, can be used for regression and density estimation too
- The process of recursive splitting/partitioning eventually divide the whole region into M sub-regions $R_m, m = 1, ..., M$.
- The model:

$$f(\mathbf{x}) = \sum_{m=1}^{M} C_m 1(\mathbf{x} \in R_m)$$

- Decision trees will overfit
    - Must use tricks to find "simple trees"
    - Fixed depth/early stopping, pruning, hypothesis testing

Bagging, Random Forest, Boosting

Ensemble: Reduce variance

- Averaging reduces variance (when predictions are independent):

$$Var(\bar{X}) = \frac{Var(X)}{N}$$

- Average models to reduce model variance
  - In any network, the bias can be reduced at the cost of increased variance
  - In a group of networks, the variance can be reduced at no cost to bias

Ensemble Classifiers

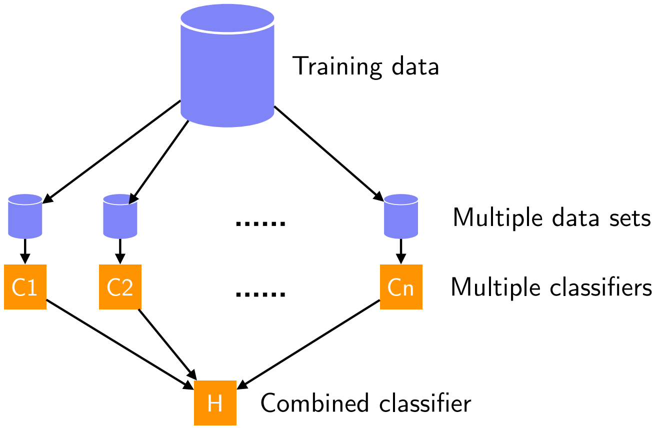
- Basic idea: build different “experts” and let them vote
- Advantages:
  - Improve predictive performance
  - Different types of classifiers can be directly included
  - Easy to implement
  - Not too much parameter tuning
- Disadvantages:
  - The combined classifier is not transparent (black box)
  - Not a compact representation

Ensemble Methods

Predict class label for unseen data by aggregating a set of predictions (classifiers learned from the training data)

- Bagging (Breiman 1994 “Bagging Predictors”)
- Random forests (Breiman 2001 “Random Forests”)
- Boosting (Freund and Schapire 1995, Friedman et al. 1998,)

General idea



Components of an Ensemble

- A method to generate the individual classifiers of the ensemble
- A method for combining the outputs of these classifiers

Diversity and Accuracy

- The individual classifiers must be diverse (errors on different data)
- If they make the same errors, such mistakes will be carried into the final prediction
- The component classifiers need to be “reasonably accurate” to avoid poor classifiers to obtain the majority of votes.

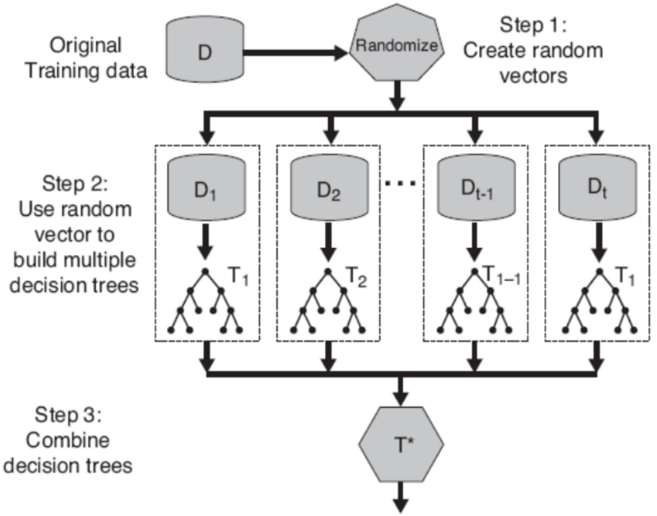
Bagging: Bootstrap Aggregation

- Take repeated bootstrap samples from training set D (Breiman, 1994)
- Bootstrap sampling: Given set D containing N training examples, create D' by drawing N examples at random **with replacement** from D
- Bagging:
  - create k bootstrap samples $D_1, ..., D_k$
  - Train distinct classifier on each D_i
  - Classify new instances by majority vote/average

When is Bagging (bootstrapping) effective?

- To ensure diverse classifiers, the base classifier should be unstable, that is, small changes in the training set should lead to large changes in the classifier output.
- Bagging is not effective with nearest neighbor classifiers. NN classifiers are highly stable with respect to variations of the training data.
- When the errors are highly correlated, and bagging becomes ineffective.

Random Forests


Original Training data
D
Randomize
Step 1: Create random vectors
Step 2: Use random vector to build multiple decision trees
D_1
D_2
...
D_{t-1}
D_t
T_1
T_2
T_{1-1}
T_1
Step 3: Combine decision trees
T^*

Random Forests

- Ensemble method specifically designed for *decision tree classifiers*.
- Two sources of randomness: "**bagging**" and "**random input vectors**"
- Use bootstrap aggregation to train many decision trees
    - Randomly subsample N examples
    - Train decision tree on subsample
    - Use average or majority vote among learned trees as prediction
- Also *randomly subsample features*: best split at each node is chosen from a random sample of m attributes instead of all attributes

Random forest algorithm

- For $b = 1$ to B
  - Draw a bootstrap sample of size N from the data
  - Grow a tree T_b using the bootstrap sample as follows
    - Choose m *attributes* uniformly at random from the data
    - Choose the best attribute among the m to split on
    - Split on the best attribute and recurse until partitions have fewer than $s_{\min}$ number of nodes
- Prediction for a new data point x
  - Regression: $\frac{1}{B} \sum_b T_b(x)$
  - Classification: choose the majority class label among $T_1(x), ..., T_B(x)$

Boosting

- Bagging reduces variance by averaging
- Bagging has little effect on bias
- Can we average and reduce bias?
- Yes: Boosting

Example: Email Spam

How would you classify an email as SPAM or not? using following criteria. If:

1. Email has only one image file (promotional image), SPAM
2. Email has only link(s), SPAM
3. Email body consist of sentence like "You won a prize money of $ xxxxxx", SPAM
4. Email from our official domain "stevens.edu" , Not a SPAM
5. Email from known source, Not a SPAM

The Boosting Approach

- devise computer program for deriving rough rules
- apply procedure to subset of examples
- obtain a simple rule
- apply to 2nd subset of examples
- obtain a 2nd rule
- repeat T times

Key details

- How to choose examples on each round?
  - concentrate on "hardest" examples (those most often misclassified by previous rule)
- How to combine the rules into single prediction rule?
  - take (weighted) majority vote of rules

Boosting

- boosting = general method of converting rough rules into highly accurate prediction rule
- technically
  - assume given "weak" learning algorithm that can consistently find classifiers at least slightly better than random, say, accuracy $\geq 55\%$
  - given sufficient data, a boosting algorithm can provably construct single classifier with very high accuracy say, 99%

A formal description of boosting

- Given training set $(x_1, y_1), ..., (x_N, y_N)$
- $y_i \in \{-1, +1\}$ correct labels of instance $x_i \in X$
- for $t = 1, ..., T$:
  - construct weight distribution $u^{(t)}$ on $\{1, ..., N\}$
  - find weak classifier:

$$f_t : X \to \{-1, +1\}$$

  with error ϵ_t on $u^{(t)}$:

$$\epsilon = P_{i \sim u^{(t)}}[f_t(x_i) \neq y_i]$$

- Output final/combined classifier H_{final}

The Idea of AdaBoost

- Training $f_2(x)$ on the new training set that *fails* $f_1(x)$
- How to find a new training set that fails $f_1(x)$? ϵ_1: the error rate of $f_1(x)$ on its training data

$$\epsilon_1 = \frac{\sum_n u_n^{(1)} \delta(f_1(x_n) \neq y_n)}{Z_1}$$

where $Z_1 = \sum_n u_n^{(1)}$, $\epsilon_1 < 0.5$

- Change the weights from $u_n^{(1)}$ to $u_n^{(2)}$ such that the misclassified examples obtain larger weights, and correct examples get smaller weights.
- Training $f_2(x)$ based on the new weights $u_n^{(2)}$

AdaBoost

- constructing $u^{(t)}$
    - $u_i^{(1)} = 1/n$
    - given $u^{(t)}$ and f_t:

$$
\begin{aligned}
u_i^{(t+1)} &= \frac{u_i^{(t)}}{Z^t} \times \begin{cases} e^{-\alpha_t} & \text{if } y_i = f_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq f_t(x_i) \end{cases} \\
&= \frac{u_i^{(t)}}{Z^t} \times e^{-\alpha_t y_i f_t(x_i)}
\end{aligned}
$$

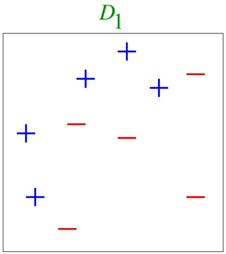
where

$$
\begin{aligned}
Z^t &= \text{ the normalization factor} \\
\alpha_t &= \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) > 0
\end{aligned}
$$

- final classifier

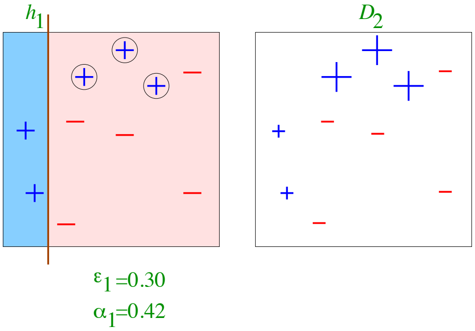
$$
H_{\text{final}}(x) = \text{sign}\left(\sum_t \alpha_t f_t(x)\right)
$$

Toy example

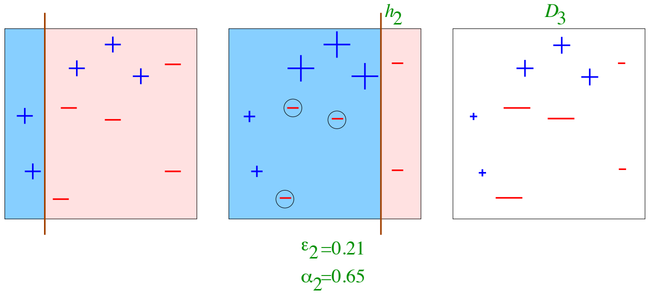


weak classifiers = vertical or horizontal half-planes

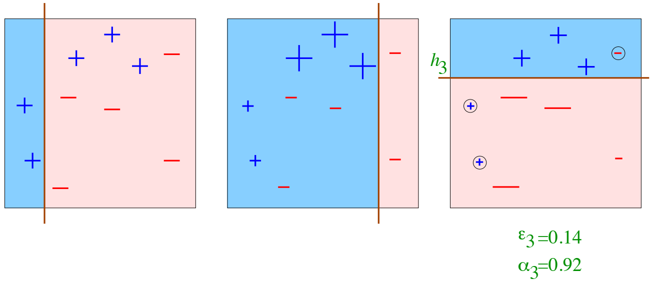
Round 1



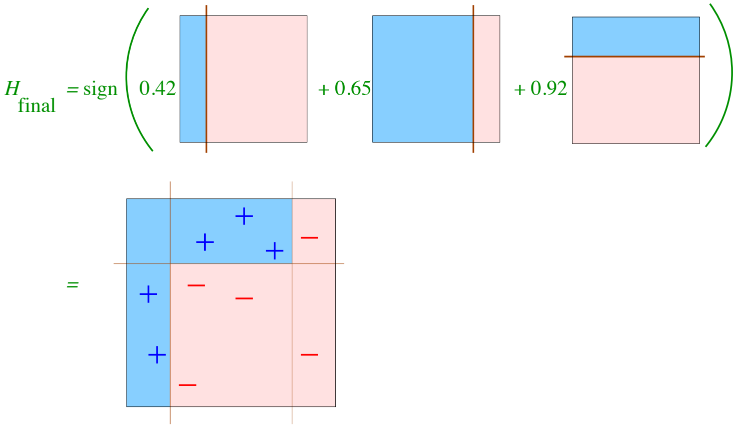
Round 2



Round 3



Final Classifier


$H_{\text{final}} = \text{sign}\left(0.42 \;\; + 0.65 \;\; + 0.92\right)$
=
+
−

Weak classifier as basis function

- The general problem here is to try to combine many simple "weak" classifiers into a single "strong" classifier
- We consider voted combinations of simple binary component classifiers

$$f_T(x) = \alpha_1 f(\mathbf{x};\theta_1) + \ldots + \alpha_T f(\mathbf{x};\theta_T)$$

  where θ is the model parameter and the (non-negative) votes α_i can be used to emphasize component classifiers that are more reliable than others. (The basis functions $f(\mathbf{x};\theta_t)$ takes the forms of weak classifier)
- It can be shown that Adaboost minimize the following exponential error: (adding one classifier at a time)

$$L = \sum_{n=1}^{N} \exp\{-y_n f_T(\mathbf{x}_n)\}$$

The AdaBoost algorithm

1. Set $u_i^{(0)} = 1/n$ for $i = 1, ..., n$
2. At the m^{th} iteration we find (any) classifier $f(\mathbf{x}; \hat{\theta}_m)$ for which the weighted classification error ϵ_m

$$\epsilon_m = 0.5 - \frac{1}{2}\Big(\sum_{i=1}^{n} u_i^{(m-1)} y_i f(\mathbf{x}_i; \hat{\theta}_m)\Big)$$

   is better than chance
3. The new component is assigned votes based on its error (smaller error rate, larger weight for voting)

$$\hat{\alpha}_m = \frac{1}{2} \ln \frac{1 - \epsilon_m}{\epsilon_m}$$

4. The weights are updated according to (Z_m is chosen so that the new weights $u_i^{(m)}$ sum to one):

$$u_i^{(m)} = \frac{1}{Z_m} \cdot u_i^{(m-1)} \exp(-y_i \hat{\alpha}_m f(\mathbf{x}_i; \hat{\theta}_m))$$

Bagging vs Boosting

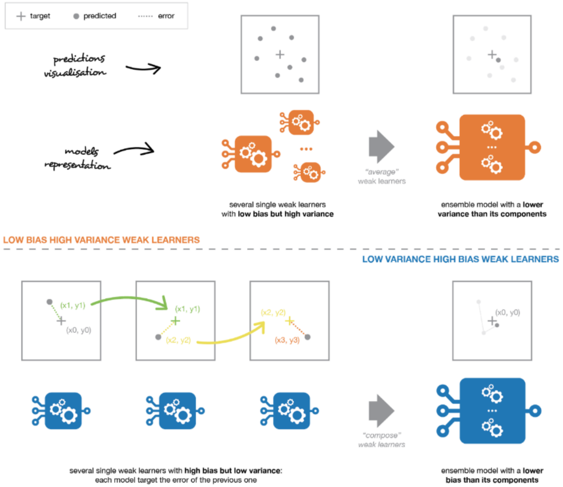

+ target
● predicted
······ error
predictions visualisation
models representation
several single weak learners with low bias but high variance
"average" weak learners
ensemble model with a lower variance than its components
LOW BIAS HIGH VARIANCE WEAK LEARNERS
LOW VARIANCE HIGH BIAS WEAK LEARNERS
(x1, y1)
(x0, y0)
(x2, y2)
(x3, y3)
"compose" weak learners
several single weak learners with high bias but low variance: each model target the error of the previous one
ensemble model with a lower bias than its components

Figure: [Joseph Rocca]

Acknowledgement and Further Reading

Slides are adapted from Dr. Y. Ning's Spring 19 offering of CS-559.

Further Reading:
Chapter 14.2, 14.3 of *Pattern Recognition and Machine Learning* by C. Bishop.