Ensemble Methods

Prof.Mingkui Tan

South China University of Technology
Southern Artificial Intelligence Laboratory(SAIL)

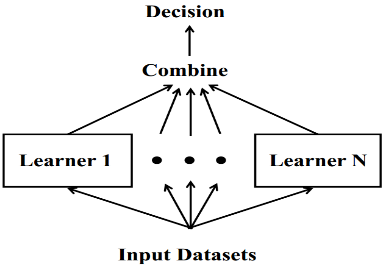
November 27, 2018

Content

Contents

Ensemble Learning

- Ensemble learning: Combine numerous weak learners to a strong learner
- Main methods: Bagging , Boosting



Contents

Decision Tree Example

Learning the Play Tennis Decision Tree

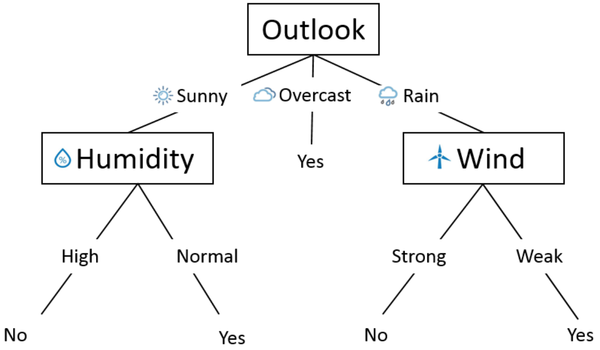
4 Attributes

Day	Outlook	Temp	Humidity	Wind	*PlayTennis*
D1	*Sunny*	*Hot*	*High*	*Weak*	*No*
D2	*Sunny*	*Hot*	*High*	*Strong*	*No*
D3	*Overcast*	*Hot*	*High*	*Weak*	*Yes*
D4	*Rain*	*Mild*	*High*	*Weak*	*Yes*
D5	*Rain*	*Cool*	*Normal*	*Weak*	*Yes*
D6	*Rain*	*Cool*	*Normal*	*Strong*	*No*
D7	*Overcast*	*Cool*	*Normal*	*Strong*	*Yes*
D8	*Sunny*	*Mild*	*High*	*Weak*	*No*
D9	*Sunny*	*Cool*	*Normal*	*Weak*	*Yes*
D10	*Rain*	*Mild*	*Normal*	*Weak*	*Yes*
D11	*Sunny*	*Mild*	*Normal*	*Strong*	*Yes*
D12	*Overcast*	*Mild*	*High*	*Strong*	*Yes*
D13	*Overcast*	*Hot*	*Normal*	*Weak*	*Yes*
D14	*Rain*	*Mild*	*High*	*Strong*	*No*

Decision Tree Example

Play Tennis: Yes or No?

- Learned function is a tree
- Classify instances by sorting them down from the root to the leaf node



Algorithm 1: Decision Tree

Input: Training set $D = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$; Attribute set $A = \{a_1, a_2, ..., a_d\}$.

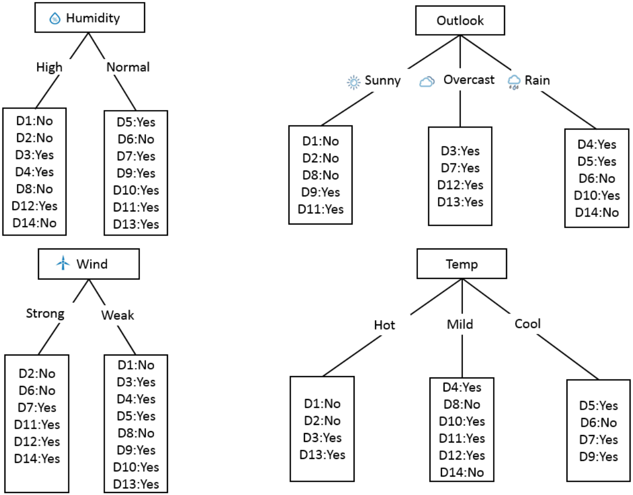
Procedure: Function TreeGenerate(D, A)

Generate node;
if all the samples in D belong to class C **then**
    mark this node as class C leaf node; **return**
end
if $A = \emptyset$ **OR** samples in D have the same value on A **then**
    mark this node as leaf node, and the class should be the most frequent occurrence class; **return**
end
Select the best partition attribute a_* from A;
for each value a_*^v of attribute a_* **do**
    D_v is the sample subset of D with $a_* = a_*^v$;
    if $D_v = \emptyset$ **then**
        mark this node as the leaf node, and the class should be the most frequent occurrence class; **return**
    else
        generate a branch for this node, TreeGenerate$(D_v, A \backslash \{a_*\})$
    end
end

Output: A decision tree

Decision Tree Example

Which Is The Best Partition Attribute?



Decision Tree Example

A Good Attribute

An attribute is good when:

- For one value we get all instances as positive
- For other value we get all instances as negative

Decision Tree Example

Poor Attribute

An attribute is poor when:

- It provides no discrimination
- Attribute is immaterial to the decision
- For each value we have same number of positive and negative instances

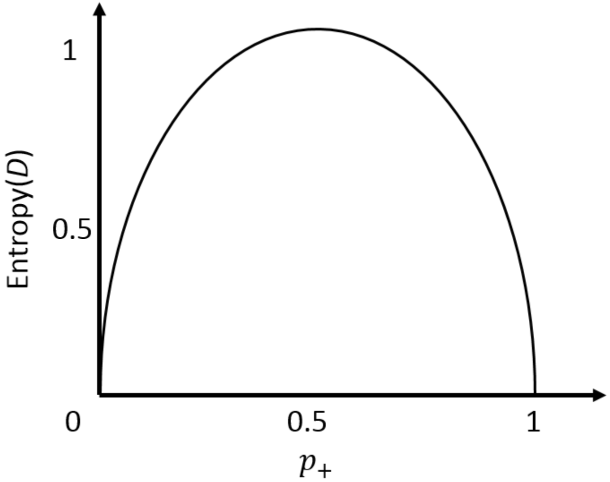
Measure of Homogeneity of Examples

- Entropy: Characterizes the (im)purity of an arbitrary collection of examples.
- Given a collection D of positive and negative examples, entropy of D relative to boolean classification is

$$Entropy(D) \equiv -p_+ log_2 p_+ - p_- log_2 p_-$$

Where p_+ is proportion of positive examples and p_- is proportion of negative examples.

Entropy Function Relative to A Boolean Classification



Entropy

- Illustration:
- D is a collection of 14 examples with 9 positive and 5 negative examples
- Entropy of D relative to the Boolean classification:

$$Entropy(9+,5-) = -\frac{9}{14}log_2\frac{9}{14} - \frac{5}{14}log_2\frac{5}{14} = 0.940$$

- Entropy is zero if all members of D belong to the same class

Entropy for Multi-Valued Target function

- If the target attribute can take on c different values, the entropy of D relative to this c-wise classification is

$$Entropy(D) \equiv \sum_{i=1}^{c} -p_i log_2 p_i$$

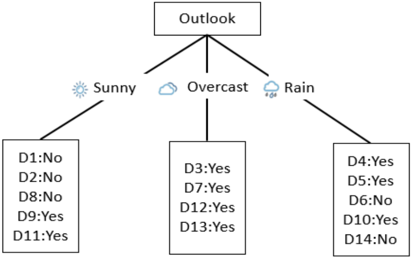
Information Gain Measures the Expected Reduction in Entropy

- Entropy measures the impurity of a collection
- Information gain of attribute A is the reduction in entropy caused by partitioning the set of examples D

$$Gain(D, A) \equiv Entropy(D) - \sum_{v \in Values(A)} \frac{|D_v|}{|D|} Entropy(D_v)$$

- Where $Values(A)$ is the set of all possible values for attribute A and D_v is the subset of D for which attribute A has value v

Measure of Purity: Information (bits)



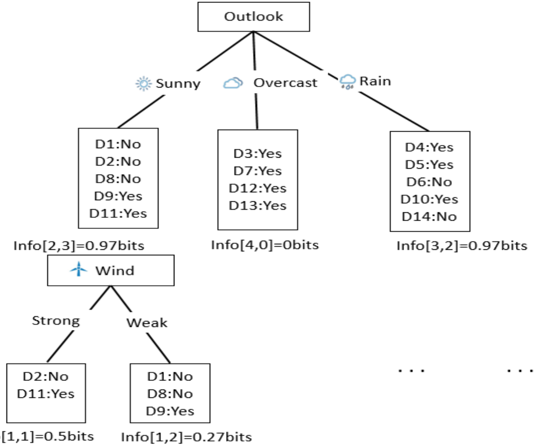
Info[2,3]=0.97bits Info[4,0]=0bits Info[3,2]=0.97bits

$$
\begin{aligned}
Info[2,3] &= entropy(2,3) \\
&= -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} \\
&= 0.97bits
\end{aligned}
$$

Information Gain for Each Attribute

- Gain(outlook) = 0.94 - 0.693 = 0.247
- Gain(temperature)= 0.94 - 0.911 = 0.029
- Gain(humidity)= 0.94 - 0.788 = 0.152
- Gain(windy)= 0.94 - 0.892 = 0.048
- $\arg\max_A \{0.247, 0.029, 0.152, 0.048\}$ = outlook
- Select outlook as the splitting attribute of tree

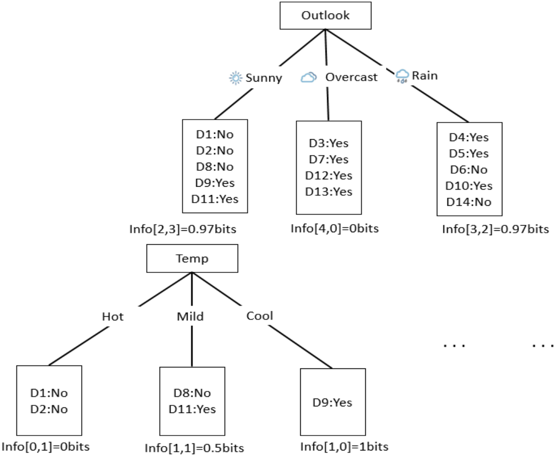
Expanded Tree Stumps for PlayTennis for Outlook=Sunny



Ave Info=$\frac{2}{5}\times 0.5+\frac{3}{5}\times 0.27$=0.36bits

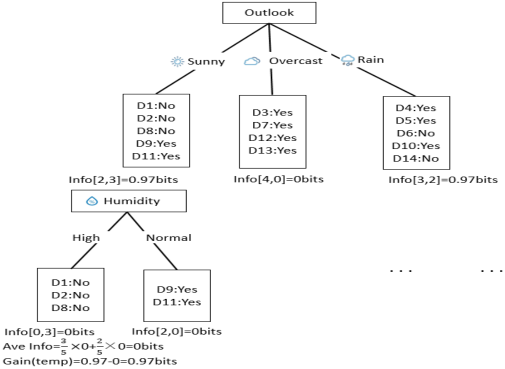
Gain(temp)=0.97-0.36=0.61bits

Expanded Tree Stumps for PlayTennis for Outlook=Sunny



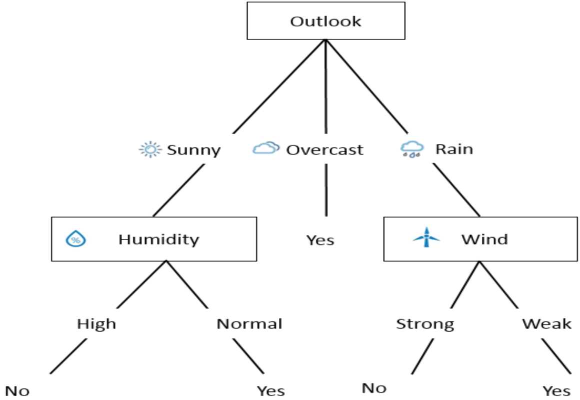
Ave Info=0+0+$\frac{1}{5}\times 1$=0.2bits
Gain(temp)=0.97-0.2=0.77bits

Expanded Tree Stumps for PlayTennis for Outlook=Sunny



- Since Gain(humidity) is highest, select humidity as splitting attribute
- No need to split further

Decision Tree for the Weather Data



Contents

Bagging

Bootstrap Sampling

- Giveing a dataset D with m samples, we take samples to make dataset $D^{'}$
- Select a sample from D to $D^{'}$ and put it back to the data set D each time, so the sample has the probability to be selected next time
- Repeat the procedure m times to get $D^{'}$ with m samples

Bagging

Bootstrap Sampling

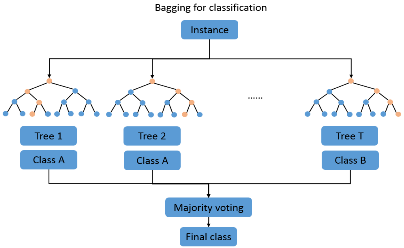
The probablility that a sample will not be selected:

$$\lim_{m \to +\infty} \left(1 - \frac{1}{m}\right)^m \to \frac{1}{e} \approx 0.368$$

- There are approximately $\frac{1}{3}$ samples used for testing which is called out-of-bag estimate
- Bootstrap sampling is useful in the case of small dataset and getting testing samples difficultly

Bagging

- Get T sampling sets through bootstrap sampling
- Train T base learners through the sampling sets respectively



- For classification:
  The class with the most votes becomes the final class
- For regression:
  The final output is the average output of every base learner

Random Forest

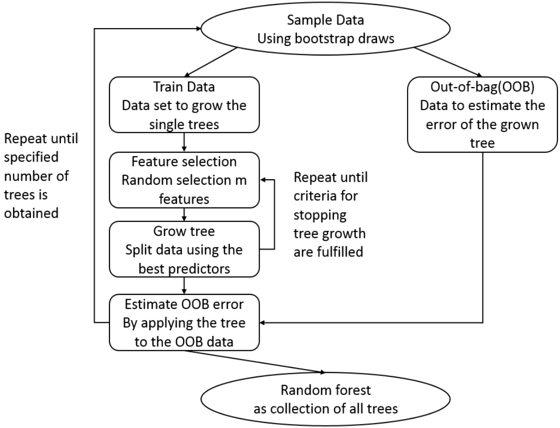
- Random forest is an extension of bagging using decision tree as base learner
- Randomly select m out of p features to get the optimal partition feature

Comparison between bagging and random forest

- The training efficiency of random forest is higher than bagging
- Bagging uses decision tree with definite structure
- Random forest uses decision tree with random structure

Random Forest

An example of the process flow is depicted below

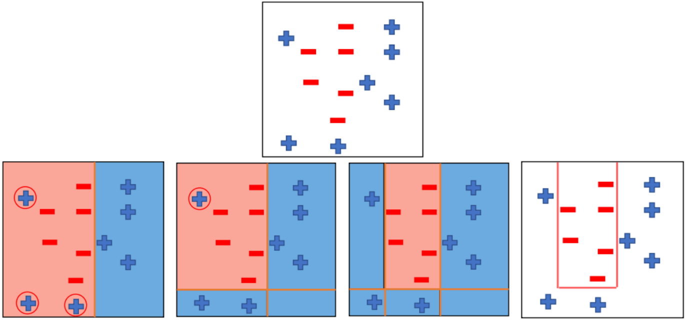


Contents

Adaboost

How to train the base leaner?

Make the wrong predictive samples more important, and handle it in next round:



Adaboost

Sample weight updating formula

$$w_{m+1}(i) = \frac{w_m(i)}{z_m} e^{-\alpha_m y_i h_m(\mathbf{x}_i)}$$

$z_m = \sum_{i=1}^{n} w_m(i) e^{-\alpha_m y_i h_m(\mathbf{x}_i)}$ is normalization term, makes $w_m(i)$ become probability distributions

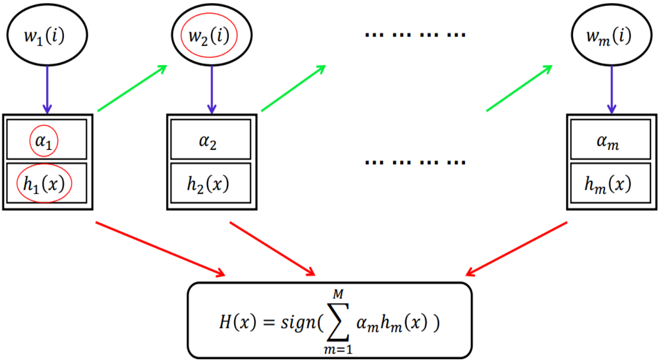
$$w_{m+1}(i) = \begin{cases} \dfrac{w_m(i)}{z_m} e^{-\alpha_m} & \text{for right predictive sample} \\ \dfrac{w_m(i)}{z_m} e^{\alpha_m} & \text{for wrong predictive sample} \end{cases}$$

so in next round, $\frac{w_{wrong}(i)}{w_{right}(i)} = e^{2\alpha_m} = \dfrac{1-\epsilon_m}{\epsilon_m}$ and $\epsilon_m < 0.5$, wrong samples will be more important

Adaboost

How to combine the base learner?

Every iteration generates a new base learner $h_m(\mathbf{x})$ and its importance score α_m



Adaboost

Evaluate the performance of the base learner

- Base learner

$$h_m(\mathbf{x}) : \mathbf{x} \mapsto \{-1, 1\}$$

- Error rate

$$\epsilon_m = p(h_m(\mathbf{x}_i) \neq y_i) = \sum_{i=1}^{n} w_m(i)\mathbb{I}(h_m(\mathbf{x}_i) \neq y_i)$$

$\epsilon_m < 0.5$, or the performance of Adaboost is weaker than random classfication.

Adaboost

Importance score of base learner

Make the base learner with lower ϵ_m more important

$$\alpha_m = \frac{1}{2} \log \frac{1 - \epsilon_m}{\epsilon_m}$$

Adaboost

Additive model

- Final learner

$$H(\mathbf{x}) = \mathrm{sign}(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x}))$$

Note: $h_m(\mathbf{x}) = \mathrm{sign}(\mathbf{w}^\top \mathbf{x})$ is a nonlinear function, so the Adaboost can deal with nonlinear problem

Algorithm

Algorithm 2: Adaboost

Input: $D = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, **where** $\mathbf{x}_i \in X, y_i \in \{-1, 1\}$

Initialize: Sample distribution w_m

Base learner: $\mathcal{L}$

$w_1(i) = \frac{1}{n}$

for m=1,2,...,M **do**

$\quad h_m(x) = \mathcal{L}(D, w_m)$

$\quad \epsilon_m = \sum_{i=1}^{n} w_m(i)\mathbb{I}(h_m(\mathbf{x}_i) \neq y_i)$

$\quad$ **if** $\epsilon_m > 0.5$ **then**

$\quad\quad$ **break**

$\quad$ **end**

$\quad \alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$

$\quad w_{m+1}(i) = \frac{w_m(i)}{z_m} e^{-\alpha_m y_i h_m(\mathbf{x}_i)}$,**where** $i = 1, 2, ..., n$ **and**

$\quad z_m = \sum_{i=1}^{n} w_m(i) e^{-\alpha_m y_i h_m(\mathbf{x}_i)}$

end

Output: $H(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})$

Contents

Gradient Boosting Decision Trees

GBDT is a decision tree algorithm with iteration

Example: What is the difference between regression tree and GBDT?

Suppose: There are 4 peoples A, B, C and D, whose age are 14, 16, 24, 26 respectively.

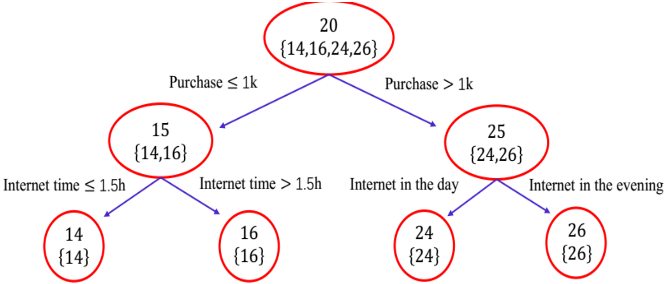
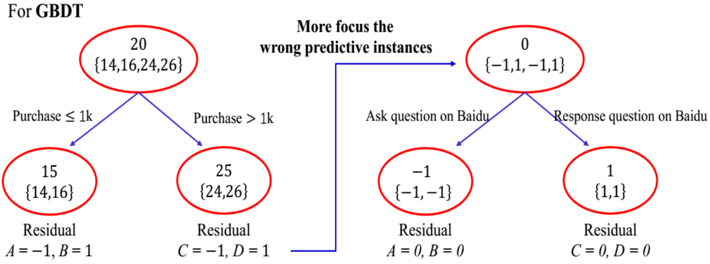


Figure: Single regression tree

Gradient Boosting Decision Trees

- The key of GBDT is that trees learn all the results and residuals of all trees before.
- The residual is the difference of predictive value and real value, so the predictive value is the sum of all results of trees.



- So,
  A=15+(-1)=14 B=15+1=16 C=25+(-1)=24 D=25+1=26

Gradient Boosting Decision Trees

Question

Q1:when results of these two algorithms are same, Why do we choose GBDT?

- The motivation of this algorithm is that every calculation of residual is to increase the weight of wrong predictive samples,and the residual of right predictive sample is zero.
- So in the next iteration, model can concentratively address these wrong predictive samples. Another function is to prevent over fitting.

Gradient Boosting Decision Trees

Question

Q2: Where does this algorithm reflect gradient boosting?

- In algorithm, residual is the gradient descent direction, which is the derivation of mean square error(MSE). Actually, MSE is the loss function of CART regression tree.

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\|y_i - f(x_i)\|$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\|y_i - f(x_i)\| \leq \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\|y_i - f(x_i)\| > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\|y_i - f(x_i)\|\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Algorithm

Algorithm 3: GBDT

Input: $D = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, **where** $\mathbf{x}_i \in X, y_i \in \{-1, 1\}$

Initialize: $f_0(x) = \arg\min_\mu \sum_{i=1}^n L(y_i, \mu)$

for m=1,2,...,M **do**

  for i=1,2,...,n **do**

    $r_{im} = -\left[\frac{\partial L(y_i, f_{m-1}(\mathbf{x}_i))}{\partial f_{m-1}(\mathbf{x}_i)}\right]$

    Fit a regression tree to targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, ..., J_m$

  end

  for j=1,2,...,J_m **do**

    $\mu_{jm} = \arg\min_\mu \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, f_{m-1}(\mathbf{x}_i) + \mu), j = 1, 2, ..., J_m$

    Update $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \sum_{j=1}^{J_m} \mu_{jm} \mathbb{I}(\mathbf{x} \in R_{jm})$

  end

end

Output: $\hat{f}(\mathbf{x}) = f_M(\mathbf{x})$

THANK YOU!