11.1 What's Integrated Algorithm?

Integrated Algorithm can also be called ensemble algorithm, which is used for classification and regression. **The core idea of integrated algorithm is to use various algorithm together so that we can solve a problem well.** There are three main parts for integrated algorithm: bagging, boosting and stacking.

11.1.1 Bagging

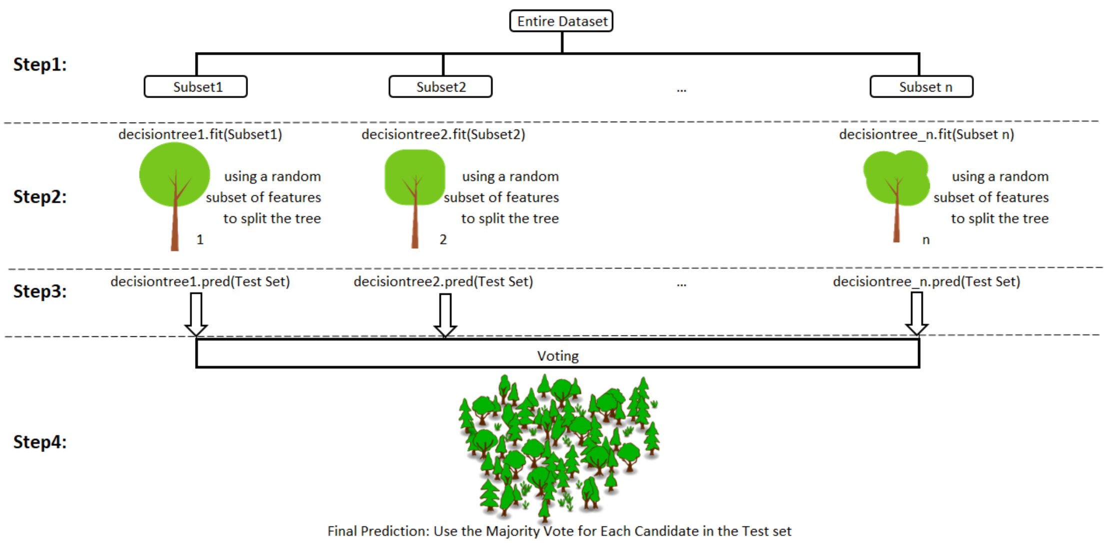
Bagging, which is short for Boostrap aggregation, is a kind of **parallel** type of integrated learning method. **The base learner can be done at the same time**. Bagging using "back" sampling method to select training set, for training set containing m sample, we randomly pick samples m times with "back". There is a close probability of 36.8% of the sample not be picked:

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

Repeated in the same way, we collect T data sets containing m samples to train T base learners, and finally combine the outputs of these T base learners and average them.

$$f(x) = \frac{1}{T}\sum_{m=1}^{T} f_m(x) \tag{2}$$

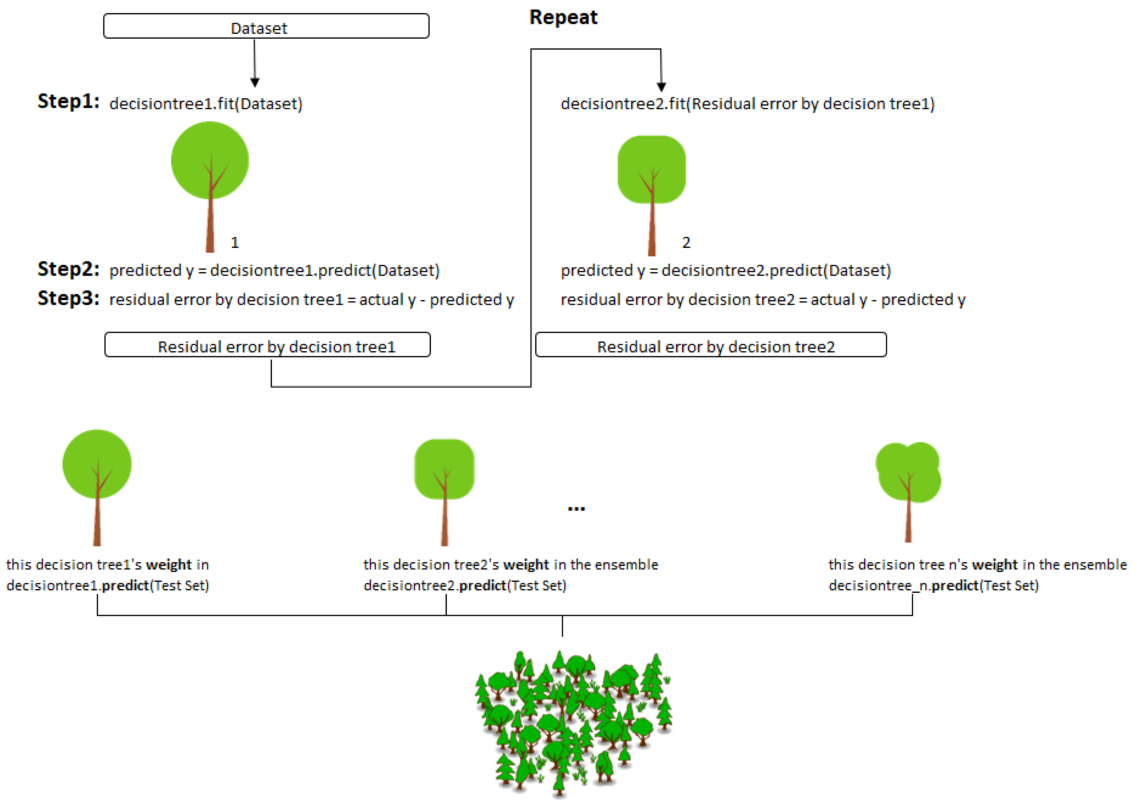
Random Forest is a classic bagging algorithm whose base learner is consisted of with many decision trees. And the "Random" means:
(1) random sample selection, freely choose samples from the whole data set (for example, every time 80% as the input data for different learners) and put them back into dataset and repick
(2) random feature selection, randomly select features as the input data for different learners. Based on the above random selection, we can make different learner with different models, which makes more sense for bagging algorithm. Random forest structure is shown in the following figure:


Entire Dataset
Step1:
Subset1
Subset2
...
Subset n
decisiontree1.fit(Subset1)
decisiontree2.fit(Subset2)
decisiontree_n.fit(Subset n)
Step2:
using a random subset of features to split the tree
1
2
n
Step3:
decisiontree1.pred(Test Set)
decisiontree2.pred(Test Set)
decisiontree_n.pred(Test Set)
Voting
Step4:

Final Prediction: Use the Majority Vote for Each Candidate in the Test set

11.1.2 Boosting

Boosting is a kind of **serial working mechanism**. The basic idea is: to increase the weight for samples which got predicted wrong, make the follow-up learner to pay more attention to the marking error of the training sample, as far as possible to correct these errors. Then we add all learners as a serial model. Boosting structure is shown in the following figure:


Dataset
Repeat
Step1: decisiontree1.fit(Dataset)
decisiontree2.fit(Residual error by decision tree1)
1
2
Step2: predicted y = decisiontree1.predict(Dataset)
Step3: residual error by decision tree1 = actual y - predicted y
predicted y = decisiontree2.predict(Dataset)
residual error by decision tree2 = actual y - predicted y
Residual error by decision tree1
Residual error by decision tree2
...
this decision tree1's weight in decisiontree1.predict(Test Set)
this decision tree2's weight in the ensemble decisiontree2.predict(Test Set)
this decision tree n's weight in the ensemble decisiontree_n.predict(Test Set)

Final Prediction: **Weighted Majority** Vote for Each Candidate in the Test set

- **Adaboost:**
  Specifically, the whole Adaboost iterative algorithm is divided into three steps:

1. Initializes the weight distribution of the training data. If there are N samples, each training sample is initially given the same weight Value: 1 / N.

$$P_0 = \{p_{0,1}, p_{0,2}, \dots, p_{0,N}\} \tag{3}$$

2. Train the weak classifier. In the specific training process, if a sample point has been accurately classified, the next training set needs to reduce its weight; Conversely, if a sample point is not accurately classified, its weight should be increased. Then, we update weights to train the next classifier, and the whole training process continues iteratively (1, 2, 3, …, k).

   - According to the current weights, we pick the classifier with the minimum error rate:

$$e_k = \sum_{i=1}^{N} p_{k-1,i} \cdot I(H_k(x_i) \neq y_i) \tag{4}$$

$$\text{Where } I(H_k(x_i) \neq y_i) = \begin{cases} 1, & H_k(x_i) \neq y_i \\ 0, & H_k(x_i) = y_i \end{cases}$$

   - Compute the weight of the current classifier β_k in the final strong classifiers:

$$\beta_k = \frac{1}{2} \ln \left(\frac{1 - e_k}{e_k}\right) \tag{5}$$

   Update weight distribution P_k of training samples:

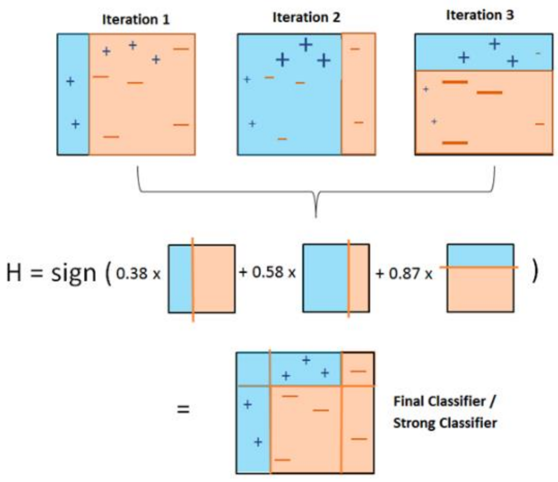
$$P_k = \{p_{k,1}, p_{k,2}, \dots, p_{k,N}\} \tag{6}$$

   Where

$$p_{k,i} = \frac{p_{k-1,i} \cdot e^{-\beta_k y_i H_k(x_i)}}{\gamma^k}$$

3. The weak classifier from each training is combined into strong classifier.

$$H = sign(\sum_{i=1}^{K} \beta_i H_i) \tag{7}$$



- **Xgboost: Extreme Gradient Boosting**

Firstly, xgboost is an efficient system implementation of Gradient Boosting. In addition to tree(gbtree), linear classifier (gblinear) can also be used for base learner in xgboost. GBDT in particular refers to the gradient boosting decision tree algorithm. We need to add a new model on the condition of the previous model. But the requirement for the new model is to satisfy the overall effect after adding it. https://www.zhihu.com/question/41354392

$$
\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\cdots \\
\underbrace{\hat{y}_i^{(t)}}_{\text{第t轮的模型预测}} &= \sum_{k=1}^{t} f_k(x_i) = \underbrace{\hat{y}_i^{(t-1)}}_{\text{保留前面t-1轮的模型预测}} + \underbrace{f_t(x_i)}_{\text{加入一个新的函数}}
\end{aligned}
$$

Try to find a new model to optimize the loss function:

$$
\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\
&= \underbrace{\sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t)}_{\text{目标：找到 } f_t \text{ 来优化这一目标}} + constant
\end{aligned}
$$

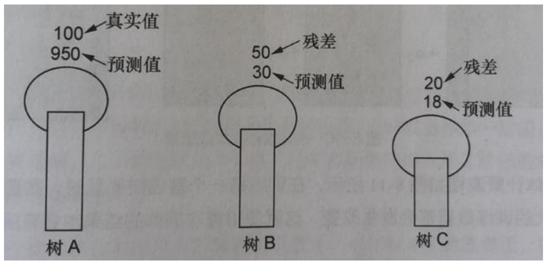
Regularization: because too many leaves are prone to overfitting, we need to limit the number of leaves:

$$
\Omega(f_t) = \underbrace{\gamma T}_{\text{叶子的个数}} + \underbrace{\frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2}_{\text{w的L2模平方}}
$$

Using Mean Square Error to represent the loss function:

$$
\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} \left(y_i - (\hat{y}_i^{(t-1)} + f_t(x_i))\right)^2 + \Omega(f_t) + const \\
&= \sum_{i=1}^{n} \left[\underbrace{2(\hat{y}_i^{(t-1)} - y_i)}_{\text{一般叫做残差}} f_t(x_i) + f_t(x_i)^2\right] + \Omega(f_t) + const
\end{aligned}
$$

Let's take an example of predicting bank loan. Assume the real loan is 1000, and first model predict 950, then we just need to use the second model to predict the residual, and so on. At last, we add up then together, which is 950 + 30 + 18 = 998.



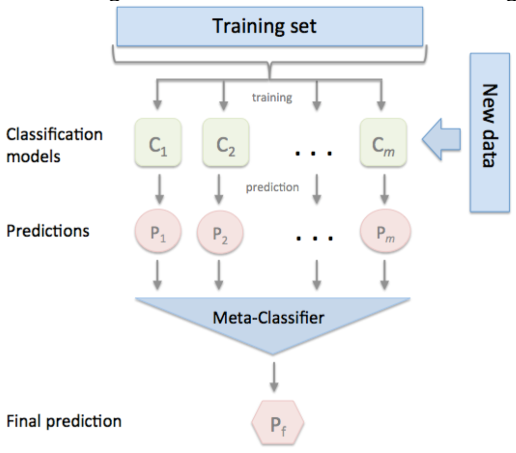
Rewrite the objective function with Taylor's formula:

$$
Obj^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t) + constant
$$

- 泰勒展开：$f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2$
- 定义：$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

11.1.3 Stacking

For bagging and boosting algorithms, their base learner must be same, especially the base learner of random forest algorithm in bagging must be decision tree. But in stacking algorithm, we can use all kinds of machine learning algorithm to build model in the first phase, and sent the outputs from first phase to the overall classifier. Stacking structure is shown in the following figure:



11.2 Combine strategies

Combining strategies refer to how to combine the outputs of the base learners to produce the final output of the integration model after training the base learners. The following are some common combining strategies:

11.2.1 Average method (regression problem)

- **Simple averaging:**

$$f(x) = \frac{1}{T}\sum_{m=1}^{T} f_m(x) \tag{3}$$

- **Weighted averaging:**

$$f(x) = \sum_{m=1}^{T} w_m \cdot f_m(x) \tag{4}$$

As the weights of each base learner are obtained in the training, generally speaking, the weighted average method is used when the performance difference of the individual learner is large, and the simple average method is used when the performance difference of the individual learner is small.

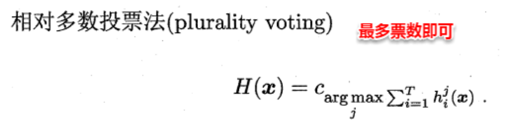
11.2.2 Voting method (classification)

- **Majority voting:**

绝对多数投票法(majority voting) 必须要占一半以上

$$H(\boldsymbol{x}) = \begin{cases} c_j, & \text{if } \sum_{i=1}^{T} h_i^j(\boldsymbol{x}) > 0.5 \sum_{k=1}^{N}\sum_{i=1}^{T} h_i^k(\boldsymbol{x}) ; \\ \text{reject}, & \text{otherwise.} \end{cases}$$

- **Plurality voting:**



相对多数投票法(plurality voting) 最多票数即可

$$H(\boldsymbol{x}) = c_{\underset{j}{\arg\max} \sum_{i=1}^{T} h_i^j(\boldsymbol{x})} .$$

11.3. Diversity

The diversity among the base learners is an important factor affecting the generalization performance of the integrator. Therefore, it’s important to increase diversity for integrated learning. The general idea is to introduce randomness into the learning process, and the common practice is to disturb data samples, input attributes and algorithm parameters.

- **data sample disturbance**: use different data sets to train different base learners. For example, there is a fallback self - sampling method, but this method is only effective for unstable learning algorithms, such as decision trees and neural networks.
- **input attribute disturbance**: a subspace of the original space is randomly selected to train the base learner. For example, random forest, extract a subset from the initial attribute set, and train the base learner based on each subset. However, if the training set contains only a few attributes, it is not appropriate to use attribute perturbation.
- **algorithm parameter disturbance**, through the random setting of different parameters, such as: neural network, random initialization weight and random setting of hidden layer node number. You will also see here that integrated learning is essentially a generic framework that can use any base learner to improve the generalization performance of a single learner.