Decision Tree

A decision tree is a set of questions(i.e. if-then sentence) organized in a **hierarchical** manner and represented graphically as a tree.
It use 'divide-and-conquer' strategy recursively. It is easy to scale up to massive data set. The models are obtained by recursively partitioning
the data space and fitting a simple prediction model within each partition. As a
result, the partitioning can be represented graphically as a decision tree.
Visual introduction to machine learning show an visual introduction to decision tree.

Algorithm Pseudocode for tree construction by exhaustive search

1. Start at root node.
2. For each node X, find the set S that **minimizes** the sum of the node impurities in the two child nodes and choose the split $\{X^\star \in S^\star\}$ that gives the minimum overall X and S.
3. If a stopping criterion is reached, exit. Otherwise, apply step 2 to each child node in turn.

Creating a binary decision tree is actually a process of dividing up the input space according to the sum of **impurities**.

This learning process is to minimize the impurities.
C4.55 and CART6 are two later classification
tree algorithms that follow this approach. C4.5 uses
`entropy` for its impurity function, whereas CART
uses a generalization of the binomial variance called the `Gini index`.

If the training set D is divided into subsets $D_1, \ldots, D_k$, the entropy may be
reduced, and the amount of the reduction is the information gain,

$$G(D; D_1, \ldots, D_k) = Ent(D) - \sum_{i=1}^{k} \frac{|D_k|}{|D|} Ent(D_k)$$

where $Ent(D)$, the entropy of D, is defined as

$$Ent(D) = \sum_{y \in Y} P(y|D) \log(\frac{1}{P(y|D)}).$$

The information gain ratio of features A with respect of data set D is defined as

$$g_R(D, A) = \frac{G(D, A)}{Ent(D)}.$$

And another option of impurity is Gini index of probability p:

$$Gini(p) = \sum_y p_y(1-p_y) = 1 - \sum_y p_y^2.$$

Like other supervised algorithms, decision tree makes a trade-off between over-fitting and under-fitting and how to choose the hyper-parameters of decision tree such as the max depth?
The regularization techniques in regression may not suit the tree algorithms such as LASSO.

Pruning is a regularization technique for tree-based algorithm. In arboriculture, the reason to prune tree is because each cut has the potential to change the growth of the tree, no branch should be removed without a reason. Common reasons for pruning are to remove dead branches, to improve form, and to reduce risk. Trees may also be pruned to increase light and air penetration to the inside of the tree's crown or to the landscape below.

<img title = " pruning" src="https://www.treesaregood.org]
(https://www.treesaregood.org/portals/0/images/treeowner/pruning1.jpg" width="80%" />

In machine learning, we prune the decision tree to make a balance between over-fitting and under-fitting. The important step of tree pruning is to define a criterion be used to determine the correct final tree size using one of the following methods:

1. Use a distinct dataset from the training set (called validation set), to evaluate the effect of post-pruning nodes from the tree.
2. Build the tree by using the training set, then apply a statistical test to estimate whether pruning or expanding a particular node is likely to produce an improvement beyond the training set.
    - Error estimation
    - Significance testing (e.g., Chi-square test)
3. Minimum Description Length principle : Use an explicit measure of the complexity for encoding the training set and the decision tree, stopping growth of the tree when this encoding size (size(tree) + size(misclassifications(tree)) is minimized.

- https://www.saedsayad.com/decision_tree_overfitting.htm
- http://www.cs.bc.edu/~alvarez/ML/statPruning.html

When the height of a decision tree is limited to 1, i.e., it takes only one
test to make every prediction, the tree is called a decision stump. While decision trees are nonlinear classifiers in general, decision stumps are a kind
of linear classifiers.
Fifty Years of Classification and
Regression Trees and the website of Wei-Yin Loh helps much understand the decision tree.
Multivariate Adaptive Regression
Splines(MARS) is the boosting ensemble methods for decision tree algorithms.

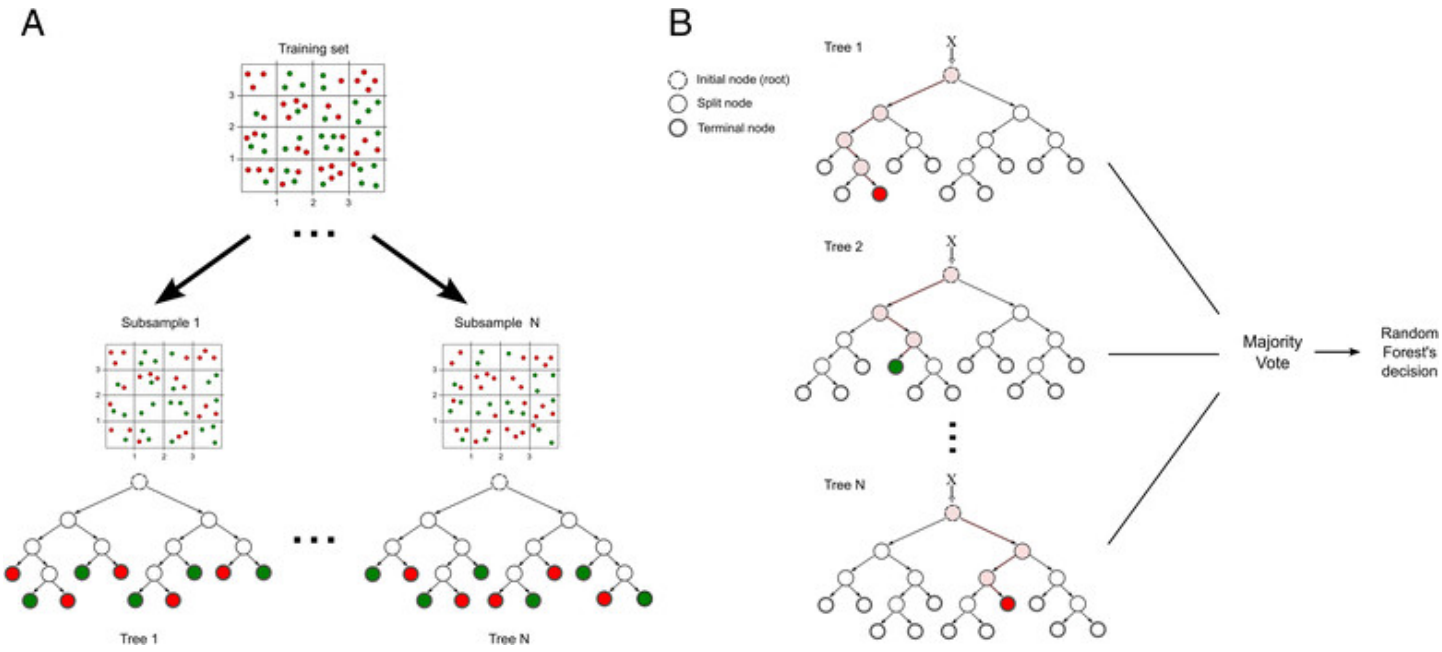
- A useful view of decision trees
- https://www.wikiwand.com/en/Decision_tree_learning
- https://www.wikiwand.com/en/Decision_tree

- https://www.wikiwand.com/en/Recursive_partitioning
- An Introduction to Recursive Partitioning: Rationale, Application and Characteristics of Classification and Regression Trees, Bagging and Random Forests
- ADAPTIVE CONCENTRATION OF REGRESSION TREES, WITH APPLICATION TO RANDOM FORESTS
- GUIDE Classification and Regression Trees and Forests (version 31.0)
- How to visualize decision trees by Terence Parr and Prince Grover
- CART
- A visual introduction to machine learning
- Interpretable Machine Learning: Decision Tree
- Tree-based Models
- http://ai-depot.com/Tutorial/DecisionTrees-Partitioning.html
- https://www.ncbi.nlm.nih.gov/pubmed/16149128
- http://www.cnblogs.com/en-heng/p/5035945.html

Random Forest

Random forests (Breiman, 2001) is a substantial modification of bagging
that builds a large collection of de-correlated trees, and then averages them.
On many problems the performance of random forests is very similar to boosting, and they are simpler to train and tune.

- For $t = 1, 2, \ldots, T$:
    - Draw a bootstrap sample $Z^\star$ of size N from the training data.
    - Grow a random-forest tree T_t to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
        - Select m variables at random from the p variables.
        - Pick the best variable/split-point among the m.
        - Split the node into two daughter nodes.
- Vote for classification and average for regression.



- https://mi2datalab.github.io/randomForestExplainer/index.html
- https://github.com/kjw0612/awesome-random-forest
- https://blog.datadive.net/interpreting-random-forests/
- https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
- https://dimensionless.in/author/raghav/
- http://www.rhaensch.de/vrf.html
- https://www.wikiwand.com/en/Random_forest
- https://sktbrain.github.io/awesome-recruit-en.v2/
- https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf
- https://dimensionless.in/introduction-to-random-forest/
- https://www.elderresearch.com/blog/modeling-with-random-forests

Ensemble methods

There are many competing techniques for solving the problem, and each technique is characterized by choices and meta-parameters: when this flexibility is taken into account, one easily ends up with a very large number of possible models for a given task.

- Zhou Zhihua's publication on ensemble methods
- Ensemble Learning literature review
- KAGGLE ENSEMBLING GUIDE
- Ensemble Machine Learning: Methods and Applications
- MAJORITY VOTE CLASSIFIERS: THEORY AND APPLICATION
- LambdaMART 不太简短之介绍
- Neural Random Forests
- Generalized Random Forests
- Additive Models, Boosting, and Inference for Generalized Divergences

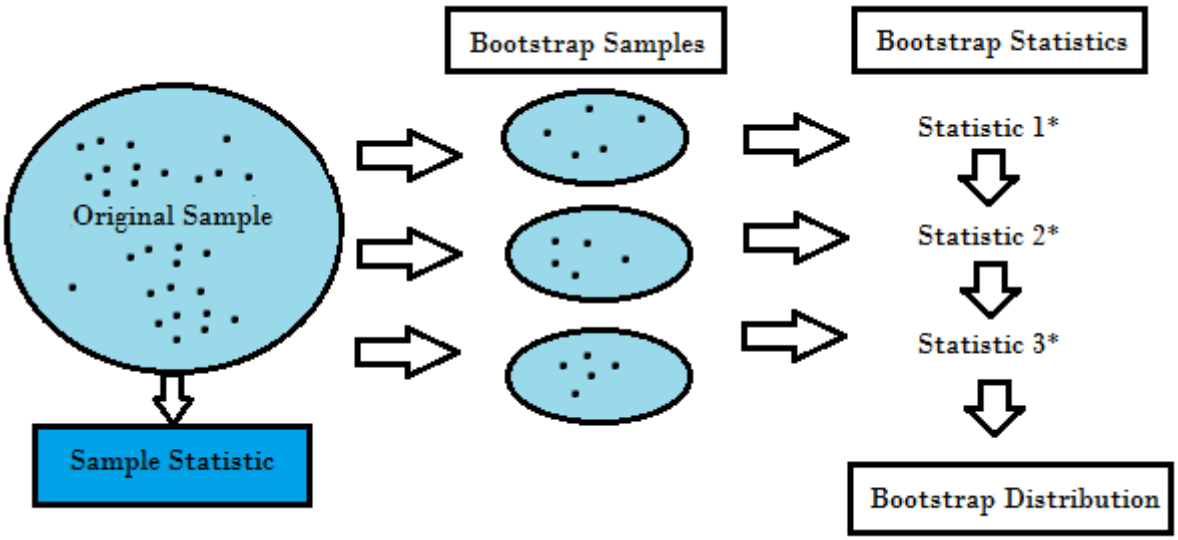
- Boosting as Entropy Projection
- Weak Learning, Boosting, and the AdaBoost algorithm

Bagging

Bagging, short for 'bootstrap aggregating', is a simple but highly effective ensemble method that creates diverse models on different random bootstrap samples of the original data set.
Random forest is the application of bagging to decision tree algorithms.

The basic motivation of parallel ensemble methods is to exploit the independence between the base learners, since the error can be reduced dramatically by combining independent base learners.
Bagging adopts the most popular strategies for aggregating the outputs of the base learners, that is, voting for classification and averaging for regression.

- Draw `bootstrap samples` $B_1, B_2, \ldots, B_n$ independently from the original training data set for base learners;
- Train the ith base learner F_i at the B_i;
- Vote for classification and average for regression.



bootstrap-sample

It is a sample-based ensemble method.

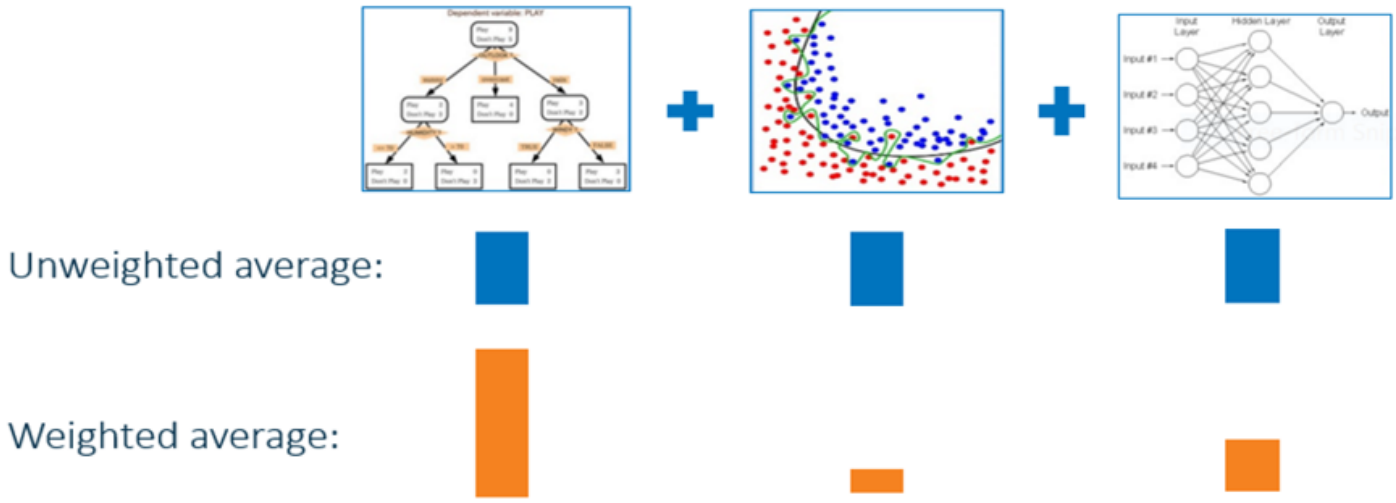
There is an alternative of bagging called combining ensemble method. It trains a linear combination of learner:

$$F = \sum_{i=1}^{n} w_i F_i$$

where the weights $w_i \geq 0, \sum_{i=1}^{n} w_i = 1$. The weights $w = \{w_i\}_{i=1}^{n}$ are solved by minimizing the ensemble error

$$w = \arg\min_{w} \sum_{k}^{K} (F(x_k) - y_k)^2$$

if the training data set $\{x_k, y_k\}_{k=1}^{K}$ is given.



weighted-unweighted

- http://www.machine-learning.martinsewell.com/ensembles/bagging/
- https://www.cnblogs.com/earendil/p/8872001.html
- https://www.wikiwand.com/en/Bootstrap_aggregating
- Bagging Regularizes
- Bootstrap Inspired Techniques in Computational Intelligence

Random Subspace Methods

Abstract: "Much of previous attention on decision trees focuses on the splitting criteria and optimization of tree sizes. The dilemma between overfitting and achieving maximum accuracy is seldom resolved. A method to construct a decision tree based classifier is proposed that maintains highest accuracy on training data and improves on generalization accuracy as it grows in complexity. The classifier consists of multiple trees constructed systematically by pseudorandomly selecting subsets of components of the feature vector, that is, trees constructed in randomly chosen subspaces. The subspace method is compared to single-tree classifiers and other forest construction methods by experiments on publicly available datasets, where the method's superiority is demonstrated. We also discuss independence between trees in a forest and relate that to the combined classification accuracy."

- http://www.machine-learning.martinsewell.com/ensembles/rsm/

Boosting

The term boosting refers to a family of algorithms that are able to convert weak learners to strong learners.
It is kind of similar to the "trial and error" scheme: if we know that the learners perform worse at some given data set S, the learner may pay more attention to the data drawn from S.
For the regression problem, of which the output results are continuous, it progressively reduce the error by trial.
In another word, we will reduce the error at each iteration.

Chinese_herb_clinic

Reweighting with Boosted Decision Trees

- https://betterexplained.com/articles/adept-method/
- https://web.stanford.edu/~hastie/Papers/buehlmann.pdf
- https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/
- http://www.machine-learning.martinsewell.com/ensembles/boosting/
- Boosting at Wikipedia

AdaBoost

AdaBoost is a boosting methods for supervised classification algorithms, so that the labeled data set is given in the form $D = \{(x_i, \mathrm{d}_i)\}_{i=1}^{N}$.
AdaBoost is to change the distribution of training data and learn from the shuffled data.
It is an iterative trial-and-error in some sense.

- Initialize the observation weights $w_i = \frac{1}{N}, i = 1, 2, \ldots, N$;
- For $t = 1, 2, \ldots, T$:

- Fit a classifier $G_m(x)$ to the training data using weights w_i;
- Compute

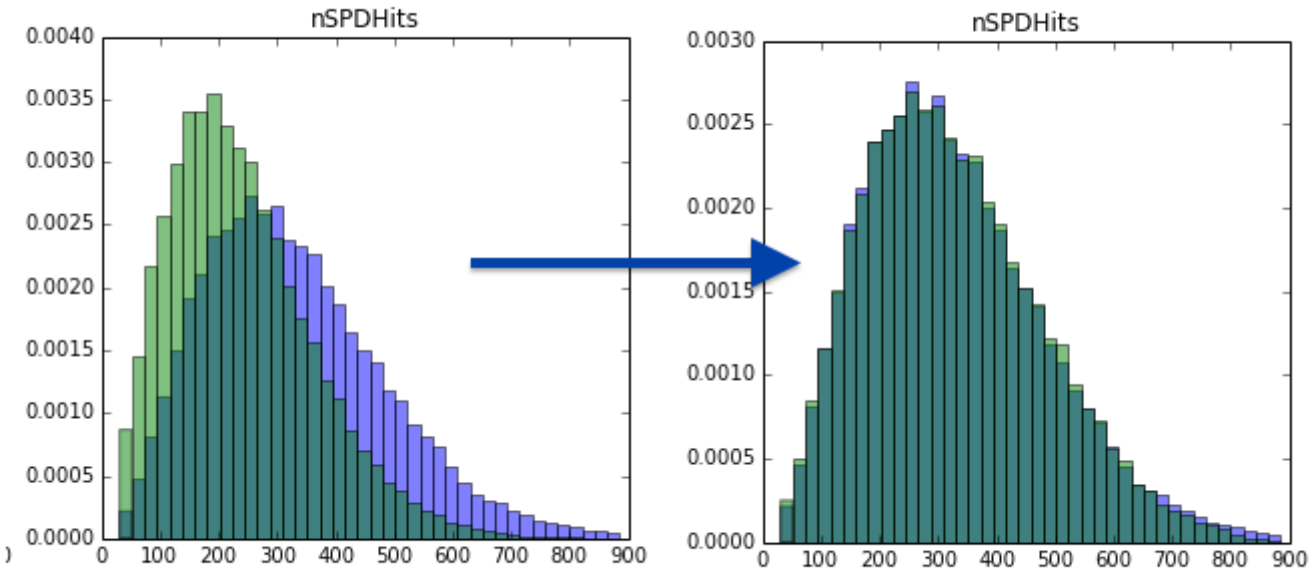
$$err_t = \frac{\sum_{i=1}^{N} \mathbb{I}(G_t(x_i) \neq \mathrm{d}_i)}{\sum_{i=1}^{N} w_i}.$$

- Compute $\alpha_t = \log(\frac{1-err_t}{err_t})$.
- Set $w_i \leftarrow w_i \exp[\alpha_t \mathbb{I}(\mathrm{d}_i \neq G_t(x_i))], i = 1, 2, \ldots, N$.

- Output $G(x) = sign[\sum_{t=1}^{T} \alpha_t G_t(x)]$.

The indicator function $\mathbb{I}(x \neq y)$ is defined as

$$\mathbb{I}(x \neq y) = \begin{cases} 1, \text{if } x \neq y \\ 0, \text{otherwise.} \end{cases}$$



1-reweighting

- AdaBoost at Wikipedia
- CSDN blog

Real AdaBoost

- Initialize the observation weights $w_i = \frac{1}{N}, i = 1, 2, \ldots, N$;
- For $t = 1, 2, \ldots, T$:
  - Fit a classifier $G_m(x)$ to obtain a class probability estimate $p_m(x) = \hat{P}_w(y = 1 \mid x) \in [0, 1]$, using weights w_i.
  - Compute $f_m(x) \leftarrow \frac{1}{2} \log p_m(x)/(1 - p_m(x)) \in \mathbb{R}$.
  - Set $w_i \leftarrow w_i \exp[-y_i f_m(x_i)], i = 1, 2, \ldots, N$ and renormalize so that $\sum_{i=1} w_i = 1$.

- Output $G(x) = sign[\sum_{t=1}^{T} \alpha_t f_m(x)]$.

- Additive logistic regression: a statistical view of boosting

Gentle AdaBoost

1. Start with weights $w_i = 1/N, i = 1, 2, \ldots, N, F(x) = 0$.
2. Repeat for $m = 1, 2, \ldots, M$:
   (a) Fit the regression function $f_m(x)$ by weighted least-squares of y_i to x_i with weights w_i.
   (b) Update $F(x) \leftarrow F(x) + f_m(x)$.
   (c) Update $w_i \leftarrow w_i \exp(-y_i f_m(x_i))$ and renormalize.
3. Output the classifier $\text{sign}[F(x)] = \text{sign}[\sum_{m=1}^{M} f_m(x)]$.

gentle AdaBoost

Gradient Boosting Decision Tree

One of the frequently asked questions is `What's the basic idea behind gradient boosting?` and the answer from [https://explained.ai/gradient-boosting/faq.html] is the best one I know:

> *Instead of creating a single powerful model, boosting combines multiple simple models into a single **composite model**. The idea is that, as we introduce more and more simple models, the overall model becomes stronger and stronger. In boosting terminology, the simple models are called weak models or weak learners.*
> *To improve its predictions, gradient boosting looks at the difference between its current approximation,$\hat{y}$, and the known correct target vector y, which is called the residual, $y - \hat{y}$. It then trains a weak model that maps feature vector x to that residual vector. Adding a residual predicted by a weak model to an existing model's approximation nudges the model towards the correct target. Adding lots of these nudges, improves the overall models approximation.*

Gradient Boosting

It is the first solution to the question that if weak learner is equivalent to strong learner.

We may consider the generalized additive model, i.e.,

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)$$

where $\{f_k\}_{k=1}^{K}$ is regression decision tree rather than polynomial.
The objective function is given by

$$obj = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where $\sum_{k=1}^{K} \Omega(f_k)$ is the regular term.

The additive training is to train the regression tree sequentially.
The objective function of the tth regression tree is defined as

$$obj^{(t)} = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_k)$$
$$= \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + C$$

where C is constant and $C = \sum_{k=1}^{t-1} \Omega(f_k)$.
Particularly, we take $L(x, y) = (x - y)^2$, and the objective function is given by

$$obj^{(t)} = \sum_{i=1}^{n} [y_i - (\hat{y}_i^{(t-1)} + f_t(x_i))]^2 + \Omega(f_t) + C$$
$$= \sum_{i=1}^{n} [(y_i - \hat{y}_i^{(t-1)}) f_t(x_i)) + f_t(x_i)^2] + \Omega(f_t) + C'$$

where $C' = \sum_{i=1}^{n} (y_i - \hat{y}_i^{(t-1)})^2 + \sum_{k=1}^{t-1} \Omega(f_k)$.

If there is no regular term $\sum_{k=1}^{t} \Omega(f_k)$, the problem is simplfied to $\arg\min_{f_t} \sum_{i=1}^{n} [(y_i - \hat{y}_i^{(t-1)}) f_t(x_i)) + f_t(x_i)^2]$.

- Initialize $f_0(x) = \arg\min_{\gamma} L(\mathrm{d}_i, \gamma)$;
- For $t = 1, 2, \ldots, T$:
    - Compute

$$r_{i,t} = -[\frac{\partial L(\mathrm{d}_i, f(x_i))}{\partial f(x_i)}]_{f=f_{t-1}}.$$

    - Fit a regression tree to the targets $r_{i,t}$ giving terminal regions $R_{j,m}, j = 1, 2, \ldots, J_m$.
    - For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{j,t} = \arg\min_{\gamma} \sum_{x_i \in R_{j,m}} L(\mathrm{d}_i, f_{t-1} + \gamma)$$

- Output $f_T(x)$.

An important part of gradient boosting method is regularization by shrinkage which consists in modifying the update rule as follows:

$$f_t = f_{t-1} + \nu \sum_{j=1}^{J_m} \gamma_{j,t} 1_{R_{j,m}}, \nu \in (0, 1).$$

- Gradient Boosting at Wikipedia
- Gradient Boosting Explained
- Gradient Boosting Interactive Playground
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3885826/
- https://explained.ai/gradient-boosting/index.html
- https://explained.ai/gradient-boosting/L2-loss.html
- https://explained.ai/gradient-boosting/L1-loss.html
- https://explained.ai/gradient-boosting/descent.html
- https://explained.ai/gradient-boosting/faq.html
- GBDT算法原理 - 飞奔的猫熊的文章 - 知乎
- https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf
- https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/
- https://data-flair.training/blogs/gradient-boosting-algorithm/
- https://arxiv.org/abs/1803.02042
- Greedy Function Approxiamtion: A Gradient Boosting Machine
- LambdaMART 不太简短之介绍

A general gradient descent “boosting” paradigm is developed for additive expansions based on any fitting criterion. It is not only for the decision tree.

- Gradient Boosting Machines
- https://machinelearningmastery.com/start-with-gradient-boosting/
- https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/

LogitBoost

Given a training data set $\{\mathrm{X}_i, y_i\}_{i=1}^N$, where $\mathrm{X}_i \in \mathbb{R}^p$ is the feature and $y_i \in \{1, 2, \ldots, K\}$ is the desired categorical label. The classifier F learned from data is a function

$$\begin{aligned} F : \mathbb{R}^P &\to y \\ X_i &\mapsto y_i. \end{aligned}$$

And the function F is usually in the additive model $F(x) = \sum_{m=1}^M h(x \mid \theta_m)$.

The Original LogitBoost Algorithm

1: $F_{i,k} = 0,\ p_{i,k} = \frac{1}{K},\quad k = 0$ to $K-1,\ i = 1$ to N

2: For $m = 1$ to M Do

3: For $k = 0$ to $K-1$, Do

4: $w_{i,k} = p_{i,k}\,(1 - p_{i,k}),\qquad z_{i,k} = \frac{r_{i,k} - p_{i,k}}{p_{i,k}\left(1 - p_{i,k}\right)}.$

5: Fit the function $f_{i,k}$ by a weighted least-square of $z_{i,k}$ to $\mathbf{x}_i$ with weights $w_{i,k}$.

6: $F_{i,k} = F_{i,k} + \nu \frac{K-1}{K}\left(f_{i,k} - \frac{1}{K}\sum_{k=0}^{K-1} f_{i,k}\right)$

7: End

8: $p_{i,k} = \exp(F_{i,k}) / \sum_{s=0}^{K-1} \exp(F_{i,s}),\quad k = 0$ to $K-1,\ i = 1$ to N

9: End

where $r_{i,k} = 1$ if $y_i = k$ otherwise 0.

Robust LogitBoost

1: $F_{i,k} = 0, p_{i,k} = \frac{1}{K}, k = 0$ to $K-1, i = 1$ to N

2: For $m = 1$ to M Do

3: For $k = 0$ to $K-1$ Do

4: $\{R_{j,k,m}\}_{j=1}^{J} = J$-terminal node regression tree from $\{r_{i,k} - p_{i,k},\ \mathbf{x}_i\}_{i=1}^{N}$,

: with weights $p_{i,k}(1-p_{i,k})$.

5: $\beta_{j,k,m} = \frac{K-1}{K} \frac{\sum_{\mathbf{x}_i \in R_{j,k,m}} r_{i,k} - p_{i,k}}{\sum_{\mathbf{x}_i \in R_{j,k,m}} (1-p_{i,k}) p_{i,k}}$

6: $F_{i,k} = F_{i,k} + \nu \sum_{j=1}^{J} \beta_{j,k,m} 1_{\mathbf{x}_i \in R_{j,k,m}}$

7: End

8: $p_{i,k} = \exp(F_{i,k}) / \sum_{s=0}^{K-1} \exp(F_{i,s}), \quad k = 0$ to $K-1,\ i = 1$ to N

9: End

robust logitBoost

- LogitBoost used first and second derivatives to construct the trees;
- LogitBoost was believed to have numerical instability problems.

- [ ] Fundamental Techniques in Big Data Data Streams, Trees, Learning, and Search by Li Ping
- [ ] LogitBoost python package
- [ ] ABC-LogitBoost for Multi-Class Classification
- [ ] LogitBoost学习
- [ ] 几种Boost算法的比较
- [ ] Robust LogitBoost and Adaptive Base Class (ABC) LogitBoost

xGBoost

In Gradient Boost, we compute and fit a regression a tree to

$$r_{i,t} = -[\frac{\partial L(\mathrm{d}_i, f(x_i))}{\partial f(x_i)}]_{f=f_{t-1}}.$$

Why not the error $L(\mathrm{d}_i, f(x_i))$ itself?

Recall the Taylor expansion
$f(x+h) = f(x) + f'(x)h + f^{(2)}(x)h^2/2! + \cdots + f^{(n)}(x)h^{(n)}/n! + \cdots$ so that the non-convex error function can be expressed as a polynomial in terms of h,
which is easier to fit than a general common non-convex function.
So that we can implement additive training to boost the supervised algorithm.

In general, we can expand the objective function at x^{t-1} up to the second order

$$obj^{(t)} = \sum_{i=1}^{n} L[y_i, \hat{y}_i^{(t-1)} + f_t(x_i)] + \Omega(f_t) + C$$
$$\simeq \sum_{i=1}^{n} [L(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{h_i f_t^2(x_i)}{2}] + \Omega(f_t) + C'$$

where $g_i = \partial_{\hat{y}_i^{(t-1)}} L(y_i, \hat{y}_i^{(t-1)})$, $h_i = \partial^2_{\hat{y}_i^{(t-1)}} L(y_i, \hat{y}_i^{(t-1)})$.

After we remove all the constants, the specific objective at step t becomes

$$obj^{(t)} \approx \sum_{i=1}^{n} [L(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{h_i f_t^2(x_i)}{2}] + \Omega(f_t)$$

One important advantage of this definition is that the value of the objective function only depends on g_i and h_i. This is how XGBoost supports custom loss functions.

In order to define the complexity of the tree $\Omega(f)$, let us first refine the definition of the tree $f(x)$ as

$$f_t(x) = w_{q(x)}, w \in \mathbb{R}^T, q : \mathbb{R}^d \Rightarrow \{1, 2, \ldots, T\}.$$

Here w is the vector of scores on leaves, q is a function assigning each data point to the corresponding leaf, and T is the number of leaves. In XGBoost, we define the complexity as

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{i=1}^{n} w_i^2.$$

After re-formulating the tree model, we can write the objective value with the t-th tree as:

$$obj^{(t)} = \sum_{i=1}^{n} [g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2}\lambda \sum_{i=1}^{n} w_i^2]$$
$$= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T$$

where $I_j = \{i | q(x_i) = j\}$ is the set of indices of data points assigned to the j-th leaf.
We could further compress the expression by defining $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$:

$$obj^{(t)} = \sum_{j=1}^{T} [(G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2] + \gamma T$$

In this equation, w_j are independent with respect to each other, the form $G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2$ is quadratic and the best w_j for a given structure $q(x)$ and the best objective reduction we can get is:

$$
\begin{aligned}
w_j^* &= -(H_j + \lambda I)^{-1} G_j \\
obj^* &= \frac{1}{2} \sum_{j=1}^{T} -(H_j + \lambda I)^{-1} G_j^2 \ + \gamma T.
\end{aligned}
$$

- https://xgboost.readthedocs.io/en/latest/tutorials/model.html
- https://xgboost.ai/
- A Kaggle Master Explains Gradient Boosting
- Extreme Gradient Boosting with R
- XGBoost: A Scalable Tree Boosting System
- xgboost的原理没你想像的那么难
- 一步一步理解GB、GBDT、xgboost
- How to Visualize Gradient Boosting Decision Trees With XGBoost in Python
- Awesome XGBoost

LightGBM

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.
- Support of parallel and GPU learning.
- Capable of handling large-scale data.

What we do in LightGBM ?

	XGBoost	LightGBM
Tree growth algorithm	Level-wise good for engineering optimization but not efficient to learn model	Leaf-wise with max depth limitation get better trees with smaller computation cost, also can avoid overfitting
Split search algorithm	Pre-sorted algorithm	Histogram algorithm
memory cost	2*#feature*#data*4Bytes	#feature*#data*1Bytes (8x smaller)
Calculation of split gain	O(#data* #features)	O(#bin *#features)
Cache-line aware optimization	n/a	40% speed-up on Higgs data
Categorical feature support	n/a	8x speed-up on Expo data

lightGBM

- LightGBM, Light Gradient Boosting Machine
- LightGBM: A Highly Efficient Gradient Boosting Decision Tree
- Python3机器学习实践：集成学习之LightGBM - AnFany的文章 - 知乎
- https://lightgbm.readthedocs.io/en/latest/
- https://www.msra.cn/zh-cn/news/features/lightgbm-20170105
- LightGBM

CatBoost

`CatBoost` is an algorithm for gradient boosting on decision trees. It is developed by Yandex researchers and engineers, and is used at Yandex for search, recommendation systems, personal assistant, self-driving cars, weather prediction and many other tasks. It is in open-source and can be used by anyone now.

`CatBoost` is based on gradient boosted decision trees. During training, a set of decision trees is built consecutively. Each successive tree is built with reduced loss compared to the previous trees.

The number of trees is controlled by the starting parameters. To prevent overfitting, use the overfitting detector. When it is triggered, trees stop being built.

Before learning, the possible values of objects are divided into disjoint ranges (buckets) delimited by the threshold values (splits). The size of the quantization (the number of splits) is determined by the starting parameters (separately for numerical features and numbers obtained as a result of converting categorical features into numerical features).

Quantization is also used to split the label values when working with categorical features. A random subset of the dataset is used for this purpose on large datasets.

- https://tech.yandex.com/catboost/

- https://catboost.ai/
- Efficient Gradient Boosted Decision Tree Training on GPUs

More

Gradient boosting tree (GBT), a widely used machine learning algorithm, achieves state-of-the-art performance in academia, industry, and data analytics competitions. Although existing scalable systems which implement GBT, such as XGBoost and MLlib, perform well for datasets with medium-dimensional features, they can suffer performance degradation for many industrial applications where the trained datasets contain highdimensional features. The performance degradation derives from their inefficient mechanisms for model aggregation-either mapreduce or all-reduce. To address this high-dimensional problem, we propose a scalable execution plan using the parameter server architecture to facilitate the model aggregation. Further, we introduce a sparse-pull method and an efficient index structure to increase the processing speed. We implement a GBT system, namely TencentBoost, in the production cluster of Tencent Inc. The empirical results show that our system is 2-20× faster than existing platforms.

- ThunderGBM: Fast GBDTs and Random Forests on GPUs
- TencentBoost: A Gradient Boosting Tree System with Parameter Server
- GBDT on Angel
- Gradient Boosted Categorical Embedding and Numerical Trees

- https://arxiv.org/abs/1901.04055
- https://arxiv.org/abs/1901.04065

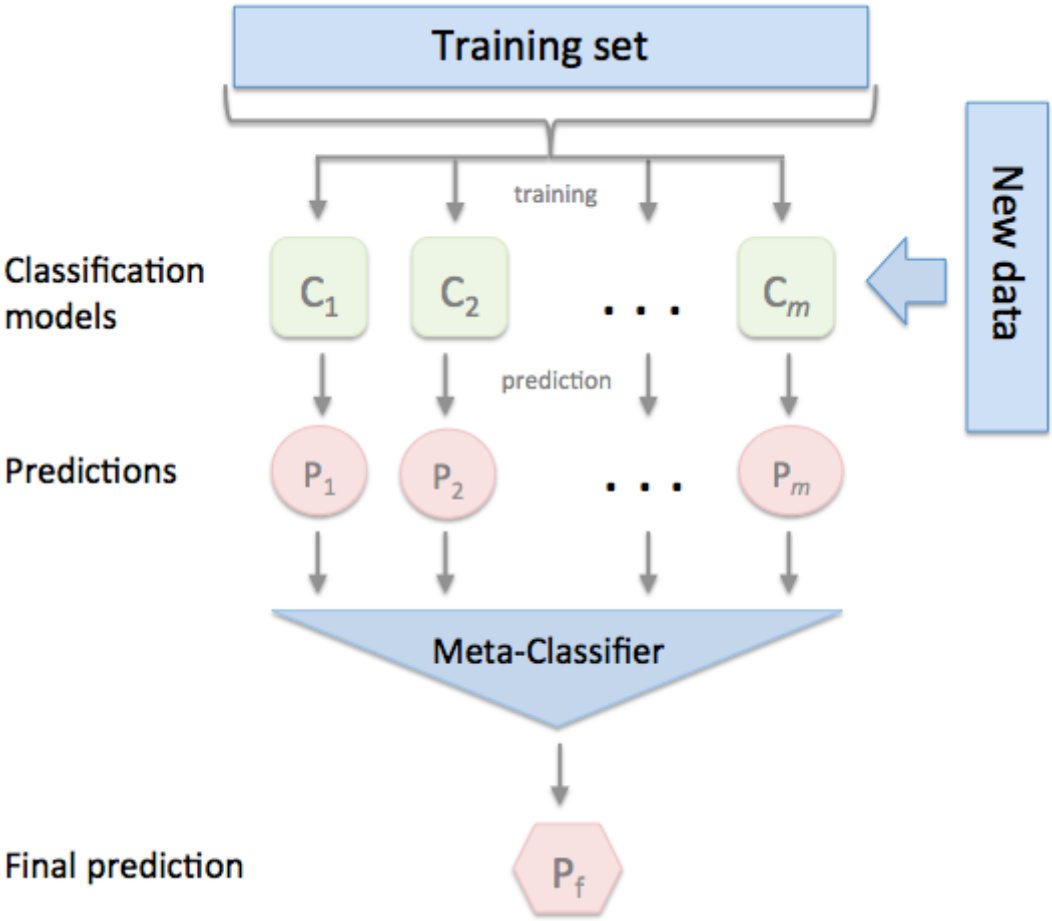
Stacking

Stacked generalization (or stacking) is a different way of combining multiple models, that introduces the concept of a meta learner. Although an attractive idea, it is less widely used than bagging and boosting. Unlike bagging and boosting, stacking may be (and normally is) used to combine models of different types.

The procedure is as follows:

1. Split the training set into two disjoint sets.
2. Train several base learners on the first part.
3. Test the base learners on the second part.
4. Using the predictions from 3) as the inputs, and the correct responses as the outputs, train a higher level learner.

Note that steps 1) to 3) are the same as cross-validation, but instead of using a winner-takes-all approach, we train a meta-learner to combine the base learners, possibly non-linearly. It is a little similar with **composition** of functions in mathematics.

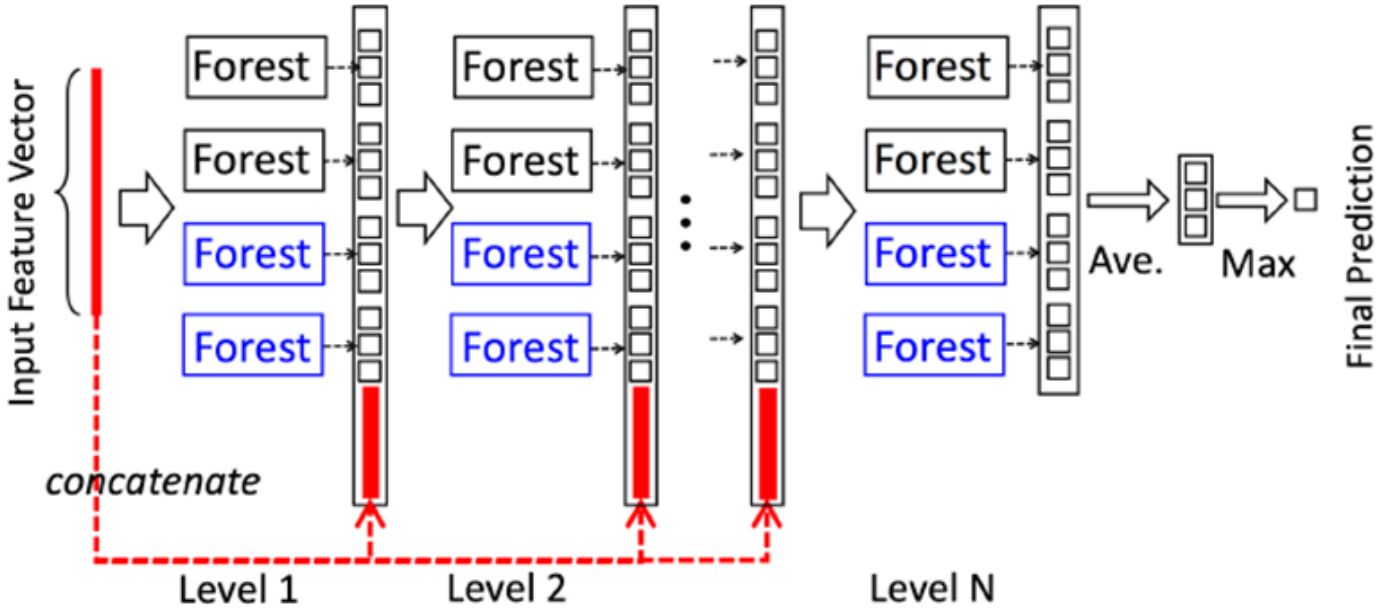


Stacking, Blending and and Stacked Generalization are all the same thing with different names. It is a kind of ensemble learning.

- http://www.machine-learning.martinsewell.com/ensembles/stacking/
- https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/
- http://www.machine-learning.martinsewell.com/ensembles/stacking/
- Stacking与神经网络 - 微调的文章 - 知乎
- http://www.chioka.in/stacking-blending-and-stacked-generalization/
- https://blog.csdn.net/willduan1/article/details/73618677
- 今我来思，堆栈泛化(Stacked Generalization)

In the sense of stacking, deep neural network is thought as the stacked `logistic regression`. And `Boltzman machine` can be stacked in order to construct more expressive model for discrete random variables.

Deep Forest

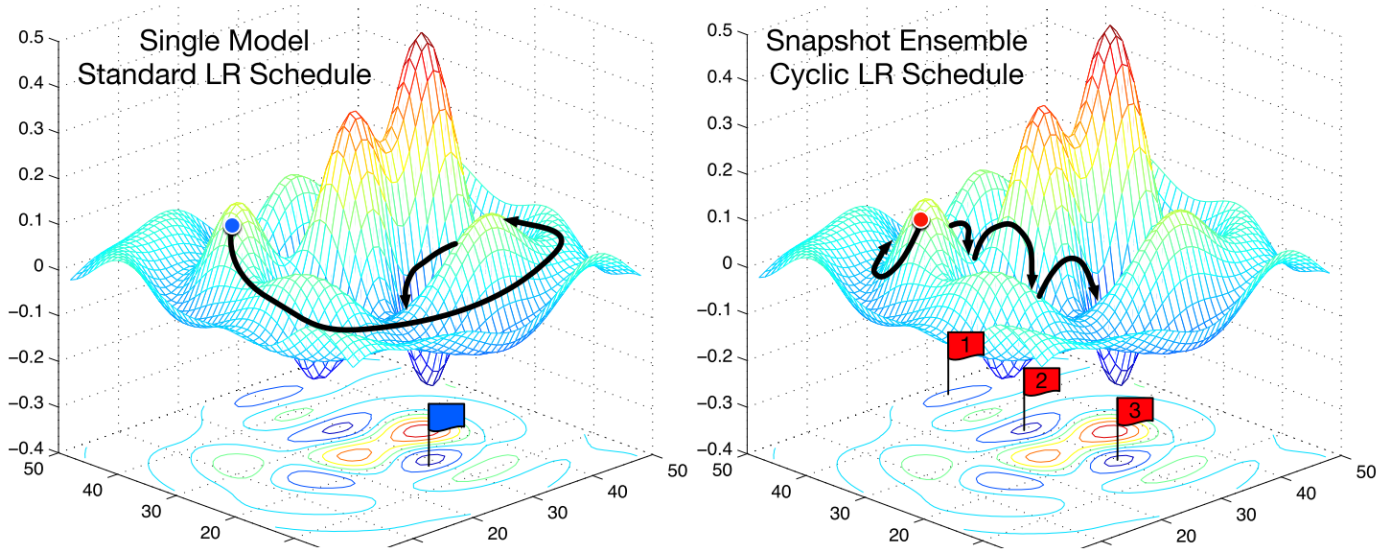


Deep Forest

- Deep forest
- https://github.com/kingfengji/gcForest
- https://zhuanlan.zhihu.com/p/37492203
- https://arxiv.org/abs/1806.00007
- https://arxiv.org/abs/1502.00712
- http://adsabs.harvard.edu/abs/2015arXiv150200712P
- https://cosx.org/2018/10/python-and-r-implementation-of-gcforest/

Deep Learning Ensemble

Deep learning and ensemble learning share some similar guide line.



snapshot

- [ ] Neural Network Ensembles
- [ ] https://machinelearningmastery.com/ensemble-methods-for-deep-learning-neural-networks/
- [ ] http://ruder.io/deep-learning-optimization-2017/
- [ ] https://arxiv.org/abs/1704.00109v1
- [ ] http://jtleek.com/advdatasci/17-blending.html
- [ ] https://arxiv.org/abs/1708.03704
- [ ] https://www.isca-speech.org/archive/archive_papers/interspeech_2014/i14_1915.pdf

---

Other ensemble methods include clustering methods ensemble, dimensionality reduction ensemble, regression ensemble, ranking ensemble.

- https://www.wikiwand.com/en/Ensemble_learning
- https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/
- https://www.toptal.com/machine-learning/ensemble-methods-machine-learning
- https://machinelearningmastery.com/products/
- https://blog.csdn.net/willduan1/article/details/73618677#
- http://www.scholarpedia.org/article/Ensemble_learning
- https://arxiv.org/abs/1505.01866