Tree-based Learning Algorithms

The simple function is a real-valued function $f:\mathrm{X}\to\mathbb{R}$ if and only if it is a finite linear combination of characteristic functions:

$$f=\sum_{i=1}^{n}a_k\chi_{S_k}$$

where $a_k\in\mathbb{R}$ and the characteristic function is defined as follow

$$\chi_{S_k}=\begin{cases}1, & \text{if } x\in S_k\\ 0, & \text{otherwise}\end{cases}.$$

- The Simple Function Approximation Lemma

The tree-based learning algorithms take advantages of these universal approximators to fit the decision function.

<img title="https://cdn.stocksnap.io/" src="https://cdn.stocksnap.io/img-thumbs/960w/TIHPAM0QFG.jpg" width="80%" />

The core problem is to find the optimal parameters $a_k\in\mathbb{R}$ and the region $S_k\in\mathbb{R}^p$ when only some finite sample or training data $\{(\mathrm{x}^i,y_i)\mid i=1,2,\dots,n\}$ is accessible or available where $\mathrm{x}^i\in\mathbb{R}^p$ and $y_i\in\mathbb{R}$ or some categorical domain and the number of regions also depends on the training data set.

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 as similar as the `binary search` in the sorting problem. 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.

In brief, A decision tree is a classifier expressed as a recursive partition of the instance space as a nonparametric statistical method.

Fifty Years of Classification and Regression Trees and the website of Wei-Yin Loh helps much understand the development of decision tree methods.
Multivariate Adaptive Regression
Splines(MARS) is the boosting ensemble methods for decision tree algorithms.
`Recursive partition` is a recursive way to construct decision tree.

- An Introduction to Recursive Partitioning: Rationale, Application and Characteristics of Classification and Regression Trees, Bagging and Random Forests
- GUIDE Classification and Regression Trees and Forests (version 31.0)

- Interpretable Machine Learning: Decision Tree
- Tree-based Models
- Decision Trees and Evolutionary Programming
- Repeated split sample validation to assess logistic regression and recursive partitioning: an application to the prediction of cognitive impairment.
- A comparison of regression trees, logistic regression, generalized additive models, and multivariate adaptive regression splines for predicting AMI mortality.
- http://www.cnblogs.com/en-heng/p/5035945.html
- 高效决策树算法系列笔记
- A Brief History of Classification and Regression Trees
- https://scikit-learn.org/stable/modules/tree.html
- https://github.com/SilverDecisions/SilverDecisions
- https://publichealth.yale.edu/c2s2/publications/
- https://publichealth.yale.edu/c2s2/15_209298_5_v1.pdf
- Applications of GUIDE, CRUISE, LOTUS and QUEST classification and regression trees

A Visual and Interactive Guide

Decision tree is represented graphically as a tree as the following.

<img src="https://www.dataversity.net/wp-content/uploads/2015/07/3049155-inline-i-1-machine-learning-is-just-a-big-game-of-plinko.gif" width="60%" />

As shown above, there are differences between the length from root to the terminal nodes, which the inputs arrive at. In another word, some inputs take more tests(pass more nodes) than others.
For a given data (x^i, y_i) where $\mathrm{x}^i=(x_1^i, x_2^i, \dots, x_p^i)$.
It is obvious that $x_1^i-1<x_1^i<x_1^i+1$ and as binary search, each attribute of the sample can be sorted into some region.

<img src="https://cdn.mathpix.com/snip/images/rX89DhhiZaKwJPqDLXH3PDYtFV4rN06D3tHDwRHd1pw.original.fullsize.png" width="70%" />

To understand the construction of decision tree, we need to answer three
basic questions:

- What are the contents of the nodes?
- Why and how is a parent node split into two daughter nodes?
- When do we declare a terminal node?

The root node contains a sample of subjects from which the tree is grown.
Those subjects constitute the so-called learning sample, and the learning sample can be the entire study sample or a subset of it.
For example, the root node contains all 3,861 pregnant women who were the study subjects of the Yale Pregnancy Outcome Study.
All nodes in the same layer constitute a partition of the root node.
The partition becomes finer and finer as the layer gets deeper and deeper.
Therefore, every node in a tree is merely a subset of the learning sample.

The core idea of the leaf splitting in decision tree is to decrease the dissimilarities of the samples in the same region, i.e., to produce the terminal nodes that are homogeneous.
The goodness of a split must weigh the homogeneities (or the impurities) in the two daughter nodes.

<img src="https://computing.llnl.gov/projects/sapphire/dtrees/pol.a.gif" width="40%"/>

The recursive partitioning process may proceed until the tree is saturated in the sense that the offspring nodes subject to further division cannot be split.

- there is only one subject in a node.
- the total number of allowable splits for a node drops as we move from one layer to the next.
- the number of allowable splits eventually reduces to zero
- the nodes are terminal when they are not divided further.

The saturated tree is usually too large to be useful.

- the terminal nodes are so small that we cannot make sensible statistical inference.
- this level of detail is rarely scientifically interpretable.
- a minimum size of a node is set a priori.
- stopping rules
    - Automatic Interaction Detection(AID) (Morgan and Sonquist 1963) declares a terminal node based on the relative merit of its best split to the quality of the root node
- Breiman et al. (1984, p. 37) argued that depending on the stopping threshold, the partitioning tends to end too soon or too late.

```
These always bring the side effect - overfitting.
```

- https://flowingdata.com/
- https://github.com/parrt/dtreeviz
- https://narrative-flow.github.io/exploratory-study-2/
- https://modeloriented.github.io/randomForestExplainer/
- A Visual Introduction to Machine Learning
- How to visualize decision trees by Terence Parr and Prince Grover
- A visual introduction to machine learning
- Interactive demonstrations for ML courses, Apr 28, 2016 by Alex Rogozhnikov
- Can Gradient Boosting Learn Simple Arithmetic?
- Viusal Random Forest

Tree Construction

> *A decision tree is the function $T:\mathbb{R}^d\to\mathbb{R}$ resulting from a learning algorithm applied on training data lying in input space $\mathbb{R}^d$, which always has the following form:*

$$T(x)=\sum_{i\in\text{leaves}} g_i(x)\mathbb{I}(x\in R_i)=\sum_{i\in\text{ leaves}} g_i(x)\prod_{a\in\text{ ancestors(i)}}\mathbb{I}(S_{a(x)}=c_{a,i})$$

> *where $R_i \subset \mathbb{R}^d$ is the region associated with leaf i of the tree, ancestors(i) is the set of ancestors of leaf node i, $c_{a,i}$ is the child of node a on the path from a to leaf i, and S_a* ***is the n-array split function at node*** *a.*
> *$g_i(\cdot)$ is the decision function associated with leaf i and*
> *is learned only from training examples in R_i.*

The $g_i(x)$ can be a constant in $\mathbb{R}$ or some mathematical expression such as logistic regression. When $g_i(x)$ is constant, the decision tree is actually piecewise constant, a concrete example of simple function.

The interpretation is simple: Starting from the root node, you go to the next nodes and the edges tell you which subsets you are looking at. Once you reach the leaf node, the node tells you the predicted outcome. All the edges are connected by ‘AND’.

`Template: If feature x is [smaller/bigger] than threshold c AND … then the predicted outcome is the mean value of y of the instances in that node.`

* Decision Trees (for Classification) by Willkommen auf meinen Webseiten.
* DECISION TREES DO NOT GENERALIZE TO NEW VARIATIONS
* On the Boosting Ability of Top-Down Decision Tree Learning Algorithms
* Improving Stability of Decision Trees
* ADAPTIVE CONCENTRATION OF REGRESSION TREES, WITH APPLICATION TO RANDOM FORESTS

What is the parameters to learn when constructing a decision tree?
The value of leaves $g_i(\cdot)$ and the spliiting function $S_i(\cdot)$.
Another approach to depict a decsion tree $T_h = (N_h; L_h)$ is given the set of internal nodes, N_h, and the set of leaves, L_h.

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 $\boxed{\text{impurities}}$ in the two child nodes and choose the split $\{X^* \in S^*\}$ 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.

---

* 基于特征预排序的算法SLIQ
* 基于特征预排序的算法SPRINT
* 基于特征离散化的算法CIOUDS
* Space-efficient online computation of quantile summaries

Divisive Hierarchical Clustering **Decision Tree**

Divisive Hierarchical Clustering	Decision Tree
From Top to Down	From Top to Down
Unsupervised	Supervised
Clustering	Classification and Regression
Splitting by maximizing the inter-cluster distance	Splitting by minimizing the impurities of samples
Distance-based	Distribution-based
Evaluation of clustering	Accuracy
No Regularization	Regularization
No Explicit Optimization	Optimal Splitting

Pruning and Regularization

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/portals/0/images/treeowner/pruning1.jpg" width="40%" />

In machine learning, it is to avoid the overfitting, to make a balance between over-fitting and under-fitting and boost the generalization ability.
Pruning is to find a subtree of the saturated tree that is most “predictive" of the outcome and least vulnerable to the noise in the data

For a tree T we define

$$R(\mathcal{T})=\sum_{\tau\in\overline{\mathcal{T}}}\boldsymbol{P}\{\tau\}r(\tau)$$

where $\overline{\mathcal{T}}$ is the set of terminal nodes of $\mathcal{T}$.
$r(\tau)$ measures a certain quality of node τ.
It is similar to the sum of the squared residuals in the linear regression.
The purpose of pruning is to select the best subtree, $\mathcal{T}^*$, of an
initially saturated tree, $\mathcal{T}_0$, such that $R(T)$ is minimized.

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.

- Decision Tree - Overfitting saedsayad
- Decision Tree Pruning based on Confidence Intervals (as in C4.5)

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.

It is also useful to restrict the number of terminal nodes, the height/depth of the decision tree in order to avoid overfitting.

Cost–Complexity

$$R_\alpha(\mathcal{T}) = R(\mathcal{T}) + \alpha|\tilde{\mathcal{T}}|$$

where $\alpha(\geq 0)$ is the complexity parameter and $|\tilde{\mathcal{T}}|$ is the `number of terminal nodes` in T.

The use of tree cost-complexity allows us to construct a sequence of nested “essential” subtrees from any given tree T so that we can examine the properties of these subtrees and make a selection from them.

> *(Breiman et al. 1984, Section 3.3) For any value of the complexity parameter α, there is a unique smallest subtree of T_0 that minimizes the cost-complexity.*

Nested Optimal Subtrees

After pruning the tree using the first threshold, we seek the
second threshold complexity parameter, α_2.
We knew from our previous discussion that $\alpha_2 = 0.018$ and its
optimal subtree is the root-node tree. No more thresholds need to
be found from here, because the root-node tree is the smallest
one.
In general, suppose that we end up with m thresholds, $0 < \alpha_1 < \alpha_2 \cdots < \alpha_m$.

> *If $\alpha_1 > \alpha_2$, the optimal subtree corresponding to α_1 is a subtree of the optimal subtree corresponding to α_2.*

We need a good estimate of the misclassification costs of the subtrees.
We will select the one with the smallest misclassification cost.

- Nearest Embedded and Embedding Self-Nested Trees
- The use of classification trees for bioinformatics
- Classification and regression trees

http://support.sas.com/documentation/cdl/en/statug/68162/HTML/default/viewer.htm#statug_hpsplit_details06.htm

Missing values processing

Impact of Missing Data

- Missing data may lead to serious loss of information.
- We may end up with imprecise or even false conclusions.
- Variables change in the selected models.
- The estimated coefficients can be notably different.

Assuming the features are missing completely at random, there are a number of ways of handling missing data:

1. Discard observations with any missing values.
2. Rely on the learning algorithm to deal with missing values in its training phase.
3. Impute all missing values before training.

For most learning methods, the imputation approach (3) is necessary. The simplest tactic is to impute the missing value with the mean or median of the nonmissing values for that feature. If the features have at least some moderate degree of dependence, one can do better by estimating a predictive model for each feature given the other features and then imputing each missing value by its prediction from the model.

Some software packages handle missing data automatically, although many don't, so it's important to be aware if any pre-processing is required by the user.

Surrogate Splits

- Surrogate splits attempt to utilize the information in other predictors to assist us in splitting when the splitting variable, say, race, is missing.
- The idea to look for a predictor that is most similar to race in classifying the subjects.
- One measure of similarity between two splits suggested by Breiman et al. (1984) is the coincidence probability that the two splits send a subject to the same node.

In general, prior information should be incorporated in estimating the coincidence probability when the subjects are not randomly drawn from a general population, such as in case–control studies.

There is no guarantee that surrogate splits improve the predictive power of a particular split as compared to a random split. In such cases, the surrogate splits should be discarded.
If surrogate splits are used, the user should take full advantage of them. They may provide alternative tree structures that in principle can have a lower misclassification cost than the final tree,

because the final tree is selected in a stepwise manner and is not
necessarily a local optimizer in any sense.

- Missing values processing in CatBoost Packages
- Decision Tree: Review of Techniques for Missing Values at Training, Testing and Compatibility
- http://oro.open.ac.uk/22531/1/decision_trees.pdf
- https://courses.cs.washington.edu/courses/cse416/18sp/slides/S6_missing-data-annotated.pdf
- Handling Missing Values when Applying Classification Models
- CLASSIFICATION AND REGRESSION TREES AND FORESTS FOR INCOMPLETE DATA FROM SAMPLE SURVEYS

Regression Trees

Starting with all of the data, consider a splitting variable j and
split point s, and define the pair of half-planes

$$R_1(j,s)=\{X\mid X_j\leq s\}, R_2(j,s)=\{X\mid X_j> s\}.$$

Then we seek the splitting variable j and split point s that solve

$$\min_{j,s}[\min_{c_1}\sum_{x_i\in R_1}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2}(y_i-c_2)^2].$$

For any choice j and s, the inner minimization is solved by

$$\hat{c}_1=\operatorname{ave}(y_i|x_i\in R_1(j,s))\text{ and }\hat{c}_2=\operatorname{ave}(y_i|x_i\in R_2(j,s)).$$

For each splitting variable, the determination of the split point s can
be done very quickly and hence by scanning through all of the inputs, determination of the best pair (j,s) is feasible.
Having found the best split, we partition the data into the two resulting regions and repeat the splitting process on each of the two regions.
Then this process is repeated on all of the resulting regions.

Tree size is a tuning parameter governing the model's complexity, and the optimal tree size should be adaptively chosen from the data.
One approach would be to split tree nodes only if the decrease in sum-of-squares due to the split exceeds some threshold.
This strategy is too short-sighted, however, since a seemingly worthless split might lead to a very good split below it.

The preferred strategy is to grow a large tree T_0 , stopping the splitting process only when some minimum node size (say 5) is reached.
Then this large tree is pruned using cost-complexity pruning.

we define the cost complexity criterion

$$C_{\alpha}(T)=\sum_{m=1}^{|T|}N_mQ_m(T)+\alpha|T|.$$

The idea is to find, for each α, the subtree $T_\alpha \subset T_0$ to minimize $C_\alpha(T)$.
The tuning parameter $\alpha \geq 0$ governs the tradeoff between tree size and its goodness of fit to the data.
Large values of α result in smaller trees T_α, and conversely for smaller values of α.
As the notation suggests, with $\alpha = 0$ the
solution is the full tree T_0.

- Tutorial on Regression Tree Methods for Precision Medicine and Tutorial on Medical Product Safety: Biological Models and Statistical Methods
- ADAPTIVE CONCENTRATION OF REGRESSION TREES, WITH APPLICATION TO RANDOM FORESTS
- REGRESSION TREES FOR LONGITUDINAL AND MULTIRESPONSE DATA
- REGRESSION TREE MODELS FOR DESIGNED EXPERIMENTS

Classification Trees

If the target is a classification outcome taking values $1, 2, \dots, K$, the only
changes needed in the tree algorithm pertain to `the criteria for splitting` nodes and pruning the tree.

It turns out that most popular splitting criteria can be derived from thinking of decision trees as greedily learning a **piecewise-constant, expected-loss-minimizing approximation** to the function $f(X) = P(Y = 1|X)$. For instance, the split that maximizes information gain is also the split that produces the piecewise-constant model that maximizes the expected log-likelihood of the data. Similarly, the split that minimizes the Gini node impurity criterion is the one that minimizes the Brier score of the resulting model. Variance reduction corresponds to the model that minimizes mean squared error.

Creating a binary decision tree is actually a process of dividing up the input space according to the sum of **impurities**, which is different from other learning mthods such as support vector machine.

- Intuitively, the least impure node should have only one class of outcome (i.e., $PY = 1|\tau = 0 \text{ or } 1$), and its impurity is zero.
- Node τ is most impure when $P\{Y = 1|\tau\} = \frac{1}{2}$.
- The impurity function has a concave shape and can be formally defined as

$$i(\tau) = \phi(P(Y = 1 \mid \tau))$$

  where the function ϕ has the properties (i) $\phi \geq 0$ and (ii) for any
  $p \in (0, 1)$, $\phi(p) = \phi(1 - p)$ and $\phi(0) = \phi(1) < \phi(p)$.

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 $D_1, \dots, D_k$, the entropy may be
reduced, and the amount of the reduction is the information gain,

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

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

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

where $y \in Y$ is the class index.

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:

$$Gini(p) = \sum_{y} p_y(1 - p_y) = 1 - \sum_{y} p_y^2.$$

$\color{red}{\text{PS: all above impurities}}$ are based on the probability $\boxed{\text{distribuion}}$ of data.
So that it is necessary to estimate the probability distribution of each attribute.

<img src="https://www.projectrhea.org/rhea/images/5/52/Impurity_Old_Kiwi.jpg">

Algorithm	Splitting Criteria	Loss Function
ID3	Information Gain	
C4.5	Normalized information gain ratio	
CART	Gini Index	

If your splitting criterion is information gain, this corresponds to a log-likelihood loss function. This works as follows.

If you have a constant approximation $\hat{f}$ to f on some regions S, then the approximation that maximizes the expected log-likelihood of the data (that is, the probability of seeing the data if your approximation is correct)is

$$L(\hat{f}) = E(\log P(Y = Y_{observed}|X, f = \hat{f})) = \sum_{X_i} Y_i \log \hat{f}(X_i) + (1 - Y_i) \log(1 - \hat{f}(X_i))$$

for a binary classification problem where $Y_i \in \{0, 1\}$ is its classification.
Here Suppose Y is determined from X by some function $f(X) = P(Y = 1|X)$.
First we need to find the constant value $\hat{f}(X) = f$ that maximizes this value.

Suppose that you have n total instances, p of them positive ($Y = 1$) and the rest negative. Suppose that you predict some arbitrary probability f – we'll solve for the one that maximizes expected log-likelihood. So we take the expected log-likelihood $\mathbb{E}_X(logP(Y = Y_{observed}|X))$, and break up the expectation by the value of $Y_{observed}$:

$$L(\hat f)
= (\log P(Y=Y_{observed} | X, Y_{observed}=1)) P(Y_{observed} = 1) \

- (\log P(Y=Y_{observed} | X, Y_{observed}=0)) P(Y_{observed} = 0)$$

Substituting in some variables gives
$$L(\hat f)
= \frac{p}{n} \log f

- \frac{n - p}{n} \log (1 - f).$$

And $\arg\max_f L(\hat f) = \frac{p}{n}$ by setting its derivative 0.
Let's substitute this back into the likelihood formula and shuffle some variables around:

$$L(\hat f) = (f \log f + (1-f)\log(1-f))\,.$$

A similar derivation shows that Gini impurity corresponds to a Brier score loss function. The Brier score for a candidate model

$$B(\hat f) = E((Y - \hat f(X))^2).$$

Like log-likelihood, the predictions that f^ should make to minimize the Brier score are simply $f = p/n$.

Now let's take the expected Brier score and break up by Y, like we did before:
$$B(\hat f)
= (1 - \hat f(X))^2 P(Y_{observed} = 1)

- \hat f(X)^2 P(Y_{observed} = 0)

$$Plugginginsomevalues:$$

    B(\hat f) = (1 - f)^2 f + f^2(1 - f) = f(1-f)$$
    which is exactly (proportional to) the Gini impurity in the 2-class setting. (A similar result holds for multiclass learning as well.)

---	---
QUEST	?
Oblivious Decision Trees	?
Online Adaptive Decision Trees	?
Lazy Tree	?
Option Tree	?
Oblique Decision Trees	?
MARS	?

* Building Classification Models: id3-c45
* Data Mining Tools See5 and C5.0
* 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
* TimeSleuth is an open source software tool for generating temporal rules from sequential data

Oblique Decision Trees

In this paper, we consider that the instances take the form $(x_1, x_2, \cdots, x_d, c_j)$, where the x_i are real-valued attributes, and the c_j is a discrete value that represents the class label of the instance. Most tree inducers consider tests of the form $x_i > k$ that are equivalent to axis-parallel hyperplanes in the attribute space. The task of the inducer is to find appropriate values for i and k. Oblique decision trees consider more general tests of the form

$$\sum_{i=1}^{d}\alpha_i x_i + \alpha_{d+1} > 0$$

where the α_{d+1} are real-valued coefficients

In a compact way, the general linear test can be rewriiten as

$$\langle\alpha, x\rangle + b > 0$$

where $\alpha = (\alpha_1, \cdots, \alpha_d)^T$ and $x = (x_1, x_2, \cdots, x_d)$.

<img src="https://computing.llnl.gov/projects/sapphire/dtrees/pol.o.gif" width="50%"/>

* https://computing.llnl.gov/projects/sapphire/dtrees/oc1.html
* Decision Forests with Oblique Decision Trees
* Global Induction of Oblique Decision Trees: An Evolutionary Approach
* On Oblique Random Forests

It is natural to generalized to nonlinear test, which can be seen as feature engineering of the input data.

Classification and Regression Tree

Classification and regression trees (CART) are a non-parametric decision tree learning technique that produces either classification or regression trees, depending on whether the dependent variable is categorical or numeric, respectively. CART is both a generic term to describe tree algorithms and also a specific name for Breiman's original algorithm for constructing classification and regression trees.

* Classification and Regression Tree Methods(In Encyclopedia of Statistics in Quality and Reliability)
* Classification And Regression Trees for Machine Learning
* Classification and regression trees
* http://homepages.uc.edu/~lis6/Teaching/ML19Spring/Lab/lab8_tree.html

- CLASSIFICATION AND REGRESSION TREES AND FORESTS FOR INCOMPLETE DATA FROM SAMPLE SURVEYS
- Classification and Regression Tree Approach for Prediction of Potential Hazards of Urban Airborne Bacteria during Asian Dust Events

https://publichealth.yale.edu/c2s2/8_209304_5_v1.pdf

Incremental Decision Tree

- https://github.com/doubleplusplus/incremental_decision_tree-CART-Random_Forest_python
- https://people.cs.umass.edu/~utgoff/papers/mlj-id5r.pdf
- https://people.cs.umass.edu/~lrn/iti/index.html
- A Streaming Parallel Decision Tree Algorithm

Very Fast Decision Tree

`Hoeffding Tree` or `VFDT` is the standard decision tree algorithm for data stream classification. VFDT uses the Hoeffding bound to decide the minimum number of arriving instances to achieve certain level of confidence in splitting the node. This confidence level determines how close the statistics between the attribute chosen by VFDT and the attribute chosen by decision tree for batch learning.

<img src="https://cdn.mathpix.com/snip/images/ljyeLweXgJeZdukPq98Sswt5cz3q87X5uozthZSJWr4.original.fullsize.png">

- Very Fast Decision Tree (VFDT) classifier
- Mining High-Speed Data Streams
- http://huawei-noah.github.io/streamDM/docs/HDT.html
- http://www.cs.washington.edu/dm/vfml/vfdt.html
- VFDT Algorithm for Decision Tree Generation

Extremely Fast Decision Tree

`Hoeffding Anytime Tree` produces the asymptotic batch tree in the limit, is naturally resilient to concept drift, and can be used as a higher accuracy replacement for Hoeffding Tree in most scenarios, at a small additional computational cost.

- Extremely Fast Decision Tree

Although exceedingly simple conceptually, most implementations of tree-based models do not efficiently utilize `modern superscalar processors`. By laying out data structures in memory in a more cache-conscious fashion, removing branches from the execution flow using a technique called predication, and micro-batching predictions using a technique called vectorization, we are able to better exploit modern processor architectures.

- https://www.cs.upc.edu/~gavalda/DataStreamSeminar/files/Lecture7.pdf
- Runtime Optimizations for Tree-based Machine Learning Models
- Optimized very fast decision tree with balanced classification accuracy and compact tree size
- Distributed Decision Trees with Heterogeneous Parallelism

Decision Stream

Decision stream is a statistic-based supervised learning technique that generates a deep directed acyclic graph of decision rules to solve classification and regression tasks. This decision tree based method avoids the problem of data exhaustion in terminal nodes by merging of leaves from the same/different levels of predictive model.

Unlike the classical decision tree approach, this method builds a predictive model with high degree of connectivity by merging statistically indistinguishable nodes at each iteration. The key advantage of decision stream is an efficient usage of every node, taking into account all fruitful feature splits. With the same quantity of nodes, it provides higher depth than decision tree, splitting and merging the data multiple times with different features. The predictive model is growing till no improvements are achievable, considering different data recombinations, and resulting in deep directed acyclic graph, where decision branches are loosely split and merged like natural streams of a waterfall. Decision stream supports generation of extremely deep graph that can consist of hundreds of levels.

- https://metacademy.org/roadmaps/Prof.Kee/Decision_Stream
- https://arxiv.org/pdf/1704.07657.pdf

Oblivious Decision Trees

- Fast Ranking with Additive Ensembles of Oblivious and Non-Oblivious Regression Trees
- https://www.ijcai.org/Proceedings/95-2/Papers/008.pdf
- http://www.aaai.org/Papers/Workshops/1994/WS-94-01/WS94-01-020.pdf

Decision Graph

- Decision Graphs : An Extension of Decision Trees
- https://www.projectrhea.org/rhea/index.php/ECE662:Glossary_Old_Kiwi

Properties of Decision Tree
Linearly separable data points.
Classification and Regression Decision Trees Explained
linearly separable data points.
Limitations of Trees
Tree structure is prone to instability even with minor data perturbations.
To leverage the richness of a data set of massive size, we need to broaden the classic statistical view of “one parsimonious model" for a given data set.
Due to the adaptive nature of the tree construction, theoretical inference based on a tree is usually not feasible. Generating more trees may provide an empirical solution to statistical inference.

Random Forest

YOSHUA BENGIO, OLIVIER DELALLEAU, AND CLARENCE SIMARD demonstrates some theoretical limitations of decision trees. And they can be seriously hurt by the curse of dimensionality in a sense that is a bit different
from other nonparametric statistical methods, but most importantly, that they cannot generalize to variations not seen in the training set.
This is because a decision tree creates a partition of the input space and needs at least one example in each of the regions

associated with a leaf to make a sensible prediction in that region.
A better understanding of the fundamental reasons for this limitation suggests that one should use forests or even deeper architectures instead of trees,
which provide a form of distributed representation and can generalize to variations not encountered in the training data.

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.

- RANDOM FORESTS by Leo Breiman
- Decision Trees do not generalize to new variations

- For $t=1,2,\dots,T$:
    - Draw a bootstrap sample Z^* 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.

<img src="https://dimensionless.in/wp-content/uploads/RandomForest_blog_files/figure-html/voting.png" width="80%" />

Forest Size

Because of so many trees in a forest, it is impractical to present a forest or interpret a forest.
Zhang and Wang (2009): a tree is removed if its removal from the
forest has the minimal impact on the overall prediction accuracy

- Calculate the prediction accuracy of forest F, denoted by p_F. For every tree, denoted by T, in forest F, calculate the prediction accuracy of forest F_{-T} that excludes T, denoted by $p_{F_{-T}}$.
- Let Δ_{-T} be the difference in prediction accuracy between F and $F_{-T}: \Delta_{-T} = p_F - p_{F_{-T}}$.
- The tree T^p with the smallest Δ_{-T} is the least important one and hence subject to removal: $T^p = \arg\min_{T\in F}(\Delta_{-T})$.

Optimal Size Subforest

- Search for the smallest random forest
- https://publichealth.yale.edu/c2s2/8_209304_5_v1.pdf

properties of random forest

Robustness to Outliers

properties of random forest
Scale Tolerance
Ability to Handle Missing Data
Ability to Select Features
Ability to Rank Features

- randomForestExplainer
- https://modeloriented.github.io/randomForestExplainer/
- Awesome Random Forest
- Interpreting random forests
- Random Forests by Leo Breiman and Adele Cutler
- https://dimensionless.in/author/raghav/
- https://koalaverse.github.io/machine-learning-in-R/random-forest.html
- https://www.wikiwand.com/en/Random_forest
- https://sktbrain.github.io/awesome-recruit-en.v2/
- Introduction to Random forest by Raghav Aggiwal
- Jump Start your Modeling with Random Forests by Evan Elg
- Complete Analysis of a Random Forest Model
- Analysis of a Random Forests Model
- Narrowing the Gap: Random Forests In Theory and In Practice
- Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling

<img title="Data Mining with Decision Tree" src="https://www.worldscientific.com/na101/home/literatum/publisher/wspc/books/content/smpai/2014/9097/9097/20140827-01/9097.cover.jpg" width= "30%" />

MARS and Bayesian MARS

MARS

Multivariate adaptive regression splines (MARS) provide a convenient approach to capture the nonlinearity aspect of polynomial regression by assessing cutpoints (knots) similar to step functions. The procedure assesses each data point for each predictor as a knot and creates a linear regression model with the candidate feature(s).

Multivariate Adaptive Regression Splines (MARS) is a non-parametric regression method that builds multiple linear regression models across the range of predictor values. It does this by `partitioning the data`, and run a `linear regression model` on each different partition.

Whereas polynomial functions impose a global non-linear relationship, step functions break the range of x into bins, and fit a different constant for each bin. This amounts to converting a continuous variable into an ordered categorical variable such that our linear regression function is converted to Equation 1：

$$y_i = \beta_0 + \beta_1 C_1(x_i) + \beta_2 C_2(x_i) + \beta_3 C_3(x_i) \cdots + \beta_d C_d(x_i) + \epsilon_i, \tag{1}$$

where $C_n(x)$ represents x values ranging from $c_n \leq x < c_{n+1}$ for $n = 1, 2, \ldots, d$.

The MARS algorithm builds a model in two steps. First, it creates a collection of so-called basis functions (BF). In this procedure, the range of predictor values is partitioned in several groups. For each group, a separate linear regression is modeled, each with its own slope. The connections between the separate regression lines are called knots. The MARS algorithm automatically searches for the best spots to place the knots. Each knot has a pair of basis functions. These basis functions describe the relationship between the environmental variable and the response. The first basis function is 'max(0, env var - knot), which means that it takes the maximum value out of two options: 0 or the result of the equation 'environmental variable value – value of the knot'. The second basis function has the opposite form: max(0, knot - env var).

<img
src="https://s3.amazonaws.com/cdn.freshdesk.com/data/helpdesk/attachments/production/6018214220/original/MARS.png
" width="70%" />

To highlight the progression from recursive partition regression to MARS we start by giving the partition regression model,

$$\hat{f}(x) = \sum_{i=1}^{k} a_i B_i(x) \tag{2}$$

where $x \in D$ and $a_i (i = 1, 2, \ldots, k)$ are the suitably chosen coefficients of the basis functions B_i and k is the number of basis functions in the model.
These basis functions are such that $B_i(x) = \mathbb{I}(x \in R_i)$ where $\mathbb{I}$ is the indicator function
which is one where the argument is true, zero elsewhere and
the $R_i (i = 1, \ldots, k)$ form a partition of D.

The usual MARS model is the same as that given in (2) except that the basis functions are different. Instead the B_i are given by

$$B_i(X) = \begin{cases} 1, & i = 1 \\ \Pi_{j=1}^{J_i} [s_{ji}(x_{\nu(ji)} - t_{ji})]_+, & i = 2, 3, \ldots \end{cases}$$

where $[\cdot]_+ = \max(0, \cdot)$; J_i is the degree of the interaction of basis B_i, the s_{ji}, which we shall call the sign indicators, equal ± 1, $\nu(ji)$ give the index of the predictor variable
which is being split on the t_{ji} (known as knot points) give
the position of the splits.

- http://uc-r.github.io/mars
- OVERVIEW OF SDM METHODS IN BCCVL
- https://projecteuclid.org/download/pdf_1/euclid.aos/1176347963
- Using multivariate adaptive regression splines to predict the distributions of New Zealand's freshwater diadromous fish
- http://www.stat.ucla.edu/~cocteau/stat204/readings/mars.pdf
- Multivariate Adaptive Regression Splines (MARS)
- https://en.wikipedia.org/wiki/Multivariate_adaptive_regression_splines
- https://github.com/cesar-rojas/mars
- Earth: Multivariate Adaptive Regression Splines (MARS)
- A Python implementation of Jerome Friedman's Multivariate Adaptive Regression Splines

- https://bradleyboehmke.github.io/HOML/mars.html
- http://www.cs.rtu.lv/jekabsons/Files/ARESLab.pdf
- https://asbates.rbind.io/2019/03/02/multivariate-adaptive-regression-splines/

Bayesian MARS

A Bayesian approach to multivariate adaptive regression spline (MARS) fitting (Friedman, 1991) is proposed. This takes the form of a probability distribution over the space of possible MARS models which is explored using reversible jump Markov chain Monte Carlo methods (Green, 1995). The generated sample of MARS models produced is shown to have good predictive power when averaged and allows easy interpretation of the relative importance of predictors to the overall fit.

The BMARS basis function can be written as

$$B(\vec{x}) = \beta_0 + \sum_{k=1}^{\mathrm{K}} \beta_k \prod_{l=0}^{\mathrm{I}} (x_l - t_{k,l})_{+}^{o_{k,l}} \tag{1}$$

where $\vec{x}$ is a vector of input, $t_{k,l}$ is the knot point in the l^{th} dimension of the k^{th} component,
the function $(y)_+$ evaluates to y if $y > 0$, else it is 0, o is the polynomial degree in the l^{th} dimension of the k^{th} component, β_k is the coefficient of the k^{th} component, K is the maximum number of components of the basis function,
and I is the maximum allowed number of interactions between the L dimensions of the input space.

<img src="http://www.milbo.users.sonic.net/gallery/plotmo-example1.png" width="70%" />

- Bayesian MARS
- An Implementation of Bayesian Adaptive Regression Splines (BARS) in C with S and R Wrappers
- Classification with Bayesian MARS
- http://www.drryanmc.com/presentations/BMARS.pdf
- Bayesian methods for nonlinear classification and regression. (2002). Denison, Holmes, Mallick and Smith: Wiley.
- Gradient Enhanced Bayesian MARS for Regression and Uncertainty Quantification

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.

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking).

- Computer Science 598A: Boosting: Foundations & Algorithms
- 4th Workshop on Ensemble Methods
- Zhou Zhihua's publication on ensemble methods
- Online Ensemble Learning: An Empirical Study
- Ensemble Learning literature review
- KAGGLE ENSEMBLING GUIDE
- Ensemble Machine Learning: Methods and Applications
- MAJORITY VOTE CLASSIFIERS: THEORY AND APPLICATION

- Neural Random Forests
- Generalized Random Forests
- Selective Ensemble of Decision Trees
- An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants
- DIFFERENT TYPES OF ENSEMBLE METHODS

Bagging	Boosting	Stacking
<img src="https://datavedas.com/wp-content/uploads/2018/04/image015.jpg" />	<img src="https://www.datavedas.com/wp-content/uploads/2018/05/3.1.1.1.6-ENSEMBLE-METHODS.jpg" />	<img src="https://datavedas.com/wp-content/uploads/2018/04/image045-2.jpg" />

- ML-Ensemble: High performance ensemble learning in Python
- https://github.com/flennerhag/mlens
- https://mlbox.readthedocs.io/en/latest/
- Ensemble Systems & Learn++ by Robi Polikar

- Applications of Supervised and Unsupervised Ensemble Methods
- Boosting-Based Face Detection and Adaptation (Synthesis Lectures on Computer Vision #2)
- Feature Selection and Ensemble Methods for Bioinformatics: Algorithmic Classification and Implementations
- Outlier Ensembles: An Introduction

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` $B_1, B_2, \dots, B_n$ 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.

<img title="bootstrap-sample" src="https://www.statisticshowto.datasciencecentral.com/wp-content/uploads/2016/10/bootstrap-sample.png" width="70%"/>

It is a sample-based ensemble method.

- 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
- ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R

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 pseudo-randomly 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

- http://rob.schapire.net/papers
- https://cseweb.ucsd.edu/~yfreund/papers
- http://www.boosting.org/
- FastForest: Learning Gradient-Boosted Regression Trees for Classification, Regression and Ranking
- Additive Models, Boosting, and Inference for Generalized Divergences
- Weak Learning, Boosting, and the AdaBoost algorithm
- http://jboost.sourceforge.net/

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.

<img title = Chinese_herb_clinic src = http://www.stat.ucla.edu/~sczhu/Vision_photo/Chinese_herb_clinic.jpg width=50% />

- https://mlcourse.ai/articles/topic10-boosting/
- Reweighting with Boosted Decision Trees
- https://betterexplained.com/articles/adept-method/
- BOOSTING ALGORITHMS: REGULARIZATION, PREDICTION AND MODEL FITTING
- What is the difference between Bagging and Boosting?
- Boosting and Ensemble Learning
- Boosting at Wikipedia
- Tree, Forest and Ensemble
- An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants
- Online Parallel Boosting

Methods	Overfit-underfitting	Training Type
Bagging	avoid over-fitting	parallel
Boosting	avoid under-fitting	sequential

AdaBoost

`AdaBoost` is a boosting methods for supervised classification algorithms, so that the labeled data set is given in the form $D=\{(x_i,\mathrm{y}_i)\}_{i=1}^{N}$.
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.

Discrete AdaBoost

- Input $D=\{(x_i,\mathrm{y}_i)\}_{i=1}^{N}$ where $x\in\mathcal{X}$ and $\mathrm{y}\in\{+1,-1\}$.
- Initialize the observation weights $w_i=\frac{1}{N}, i=1,2,\dots,N$.
- For $t=1,2,\dots,T$:
  - Fit a classifier $G_t(x)$ to the training data using weights w_i.
  - Compute

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

  - Compute $\alpha_t=\log(\frac{1-err_t}{err_t})$.
  - Set $w_i\leftarrow w_i\exp[\alpha_t\mathbb{I}(G_t(x_i)\neq \mathrm{y}_i)], i=1,2,\dots,N$ and renormalize so that $\sum_i w_i=1$.
- Output $G(x)=sign[\sum_{t=1}^{T}\alpha_t G_t(x)]$.

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

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

Note that the weight updating $w_i\leftarrow w_i\exp[\alpha_t\mathbb{I}(G_t(x_i)\neq \mathrm{y}_i)]$ so that the weight does not vary if the prediction is correct $\mathbb{I}(G_t(x_i)\neq \mathrm{y}_i)=0$ and the weight does increase if the prediction is wrong $\mathbb{I}(G_t(x_i)\neq \mathrm{y}_i)=1$.

<img title="reweighting" src="https://arogozhnikov.github.io/images/reweighter/1-reweighting.png" width= "80%" />

<img src="https://cdn-images-1.medium.com/max/1600/0*WOo4d8oNmb85y_Eb.png" width="60%">

- AdaBoost at Wikipedia
- BrownBoost at Wikipedia
- CoBoosting at Wikipedia

- CSDN blog: Adaboost 算法的原理与推导
- On the Convergence Properties of Optimal AdaBoost
- Some Open Problems in Optimal AdaBoost and Decision Stumps
- Parallelizing AdaBoost by weights dynamics

<img src ="https://cseweb.ucsd.edu/~yfreund/portraitsmall.jpg" width="30%" />
<img src=https://www.microsoft.com/en-us/research/wp-content/uploads/2017/09/avatar_user_33549_1504711750-180x180.jpg width=40% />

For a two-class problem, an `additive logistic model` has the form

$$\log\frac{P(y=1\mid x)}{P(y=-1\mid x)}=\sum_{m=1}^{M}f_m=F(x)$$

where $P(y=-1\mid x)+P(y=1\mid x)=1$; inverting we obtain

$$p(x)=P(y=1\mid x)=\frac{\exp(F(x))}{1+\exp(F(x))}.$$

Consider the exponential criterion

$$\arg\min_{F(x)}\mathbb{E}[\exp(-yF(x))]\iff F(x)=\frac{1}{2}\log\frac{P(y=1\mid x)}{P(y=-1\mid x)}.$$

> *The Discrete AdaBoost algorithm (population version) builds an additive logistic regression model via Newton-like updates for minimizing* $\mathbb{E}[\exp(-yF(x))]$.

$$\mathbb{E}[\exp(-yF(x))]=\exp(F(x))P(y=-1\mid x)+\exp(-F(x))P(y=+1\mid x)$$

so that

$$\frac{\partial\mathbb{E}[\exp(-yF(x))]}{\partial F(x)}=-\exp(-F(x))P(y=+1\mid x)+\exp(F(x))P(y=-1\mid x)\tag{1}$$

where and $\mathbb{E}$ represents expectation.
Setting the equation (1) to 0, we get

$$F(x)=\frac{1}{2}\log\frac{P(y=1\mid x)}{P(y=-1\mid x)}.$$

So that

$$\operatorname{sign}(H(x)) = \operatorname{sign}(\frac{1}{2}\log\frac{P(y=1\mid x)}{P(y=-1\mid x)})$$
$$= \begin{cases} 1, & \text{if } \frac{P(y=1|x)}{P(y=-1|x)} > 1 \\ -1, & \text{if } \frac{P(y=1|x)}{P(y=-1|x)} < 1 \end{cases} = \arg\max_{x\in\{+1,-1\}} P(f(x)=y\mid x).$$

- Input $D=\{(x_i,\mathrm{y}_i)\}_{i=1}^{N}$ where $x\in\mathcal{X}$ and $\mathrm{y}\in\{+1,-1\}$.
- Initialize the observation weights $w_i=\frac{1}{N}, i=1,2,\dots,N$.
- For $t=1,2,\dots,T$:
  - Fit a classifier $G_t(x)$ to the training data using weights w_i.
  - Compute

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

  - Compute $\alpha_t=\log(\frac{1-err_t}{err_t})$.
  - Set $w_i\leftarrow w_i\exp[-\alpha_t G_t(x_i)\mathrm{y}_i)], i=1,2,\dots,N$ and renormalize so that $\sum_i w_i=1$.
- Output $G(x)=sign[\sum_{t=1}^{T}\alpha_t G_t(x)]$.

Real AdaBoost

In `AdaBoost`, the error is binary- it is 0 if the classification is right otherwise it is 1. It is not precise for some setting. The output of decision trees is a class probability estimate $p(x)=P(y=1|x)$, the probability that x is in the positive class

Real AdaBoost

- Input $D=\{(x_i,\mathrm{y}_i)\}_{i=1}^{N}$ where $x\in\mathcal{X}$ and $y\in\{+1,-1\}$.
- Initialize the observation weights $w_i=\frac{1}{N}, i=1,2,\dots,N$;
- For $m=1,2,\dots,M$:
  - 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 $w_i\leftarrow w_i\exp[-y_i f_m(x_i)], i=1,2,\dots,N$ and renormalize so that $\sum_{i=1} w_i=1$.
- Output $G(x)=sign[\sum_{t=1}^{M}\alpha_t G_m(x)]$.

> *The Real AdaBoost algorithm fits an additive logistic regression model by stagewise and approximate optimization of* $\mathbb{E}[\exp(-yF(x))]$.

- Additive logistic regression: a statistical view of boosting

Gentle AdaBoost

- Input $D=\{(x_i,\mathrm{y}_i)\}_{i=1}^{N}$ where $x\in\mathcal{X}$ and $y\in\{+1,-1\}$.
- Initialize the observation weights $w_i=\frac{1}{N}, i=1,2,\dots,N, F(x)=0$;
- For $m=1,2,\dots,M$:
    - Fit a classifier $f_m(x)$ by weighted least-squares of y_i to x_i with weights w_i.
    - Update $F(x)\leftarrow F(x)+f_m(x)$.
    - UPdate $w_i\leftarrow w_i\exp(-\mathrm{y}_i f_m(x_i))$ and renormalize.
- Output the classifier $sign[F(x)]=sign[\sum_{t=1}^{M}\alpha_t f_m(x)]$.

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 $y_i\in\{1,2,\dots,K\}$ 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 $F(x)=\sum_{m=1}^{M}h(x\mid\theta_m)$.

LogitBoost (two classes)

- Input $D=\{(x_i,\mathrm{y}_i)\}_{i=1}^{N}$ where $x\in\mathcal{X}$ and $y\in\{+1,-1\}$.
- Initialize the observation weights $w_i=\frac{1}{N}, i=1,2,\dots,N, F(x)=0, p(x_i)=\frac{1}{2}$;
- For $m=1,2,\dots,M$:
    - Compute the working response and weights:

$$\begin{aligned}z_i&=\frac{y_i^*-p_i}{p_i(1-p_i)},\\ w_i&=p_i(1-p_i).\end{aligned}$$

    - Fit a classifier $f_m(x)$ by `weighted least-squares` of y_i to x_i with weights w_i.
    - Update $F(x)\leftarrow F(x)+\frac{1}{2}f_m(x)$.
    - UPdate $w_i\leftarrow w_i\exp(-\mathrm{y}_i f_m(x_i))$ and renormalize.
- Output the classifier $sign[F(x)]=sign[\sum_{t=1}^{M}\alpha_t f_m(x)]$.

Here y^* represents the outcome and $p(y^*=1)=p(x)=\frac{\exp(F(x))}{\exp(F(x))+\exp(-F(x))}$.

<img title ="logitBoost" src="https://img-blog.csdn.net/20151028220708460" width="80%" />

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

<img title ="robust logitBoost" src="https://img-blog.csdn.net/20151029111043502" width="80%" />

- 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

arc-x4 Algorithm

Recent work has shown that combining multiple versions of unstable classifiers such as trees or neural nets results in reduced test set error. One of the more effective is bagging (Breiman [1996a]) Here, modified training sets are formed by resampling from the original training set, classifiers constructed using these training sets and then combined by voting. Freund and Schapire [1995,1996] propose an algorithm the basis of which is to adaptively resample and combine (hence the acronym--arcing) so that the weights in the resampling are increased for those cases most often misclassified and the combining is done by weighted voting. Arcing is more successful than bagging in test set error reduction. We explore two arcing algorithms, compare them to each other and to bagging, and try to understand how arcing works. We introduce the definitions of bias and variance for a classifier as components of the test set error. Unstable classifiers can have low bias on a large range of data sets. Their problem is high variance. Combining multiple versions either through bagging or arcing reduces variance significantly.

Breiman proposes a boosting algorithm called `arc-x4` to investigate whether the success of AdaBoost roots in its technical details or in the resampling scheme it uses.
The difference between AdaBoost and arc-x4 is twofold.
First, the weight for object z_j at step k is calculated as the proportion of times z_j has been misclassified by the $k-1$ classifiers built so far.
Second, the final decision is made by plurality voting rather than weighted majority voting.

`arc` represents `adaptively resample and combine`.

<img src="https://cdn.mathpix.com/snip/images/-OxsjINF-1pNoCPvQ-z1OJwSyJ2ref2JyCdqtBD_D0M.original.fullsize.png" width="70%">

- Combining Pattern Classifiers: Methods and Algorithms
- BIAS, VARIANCE , AND ARCING CLASSIFIERS
- Online Ensemble Learning: An Empirical Study
- Arcing Classifiers

Properties of AdaBoost
AdaBoost is inherently sequential.
The training classification error has to go down exponentially fast if the weighted errors of the component classifiers are strictly better than chance.
A crucial property of AdaBoost is that it almost never overfits the data no matter how many iterations it is run.

Multiclass Boosting

In binary classification we learn a function $f(x)$ which is positive for one class and negative for the other class. This means that $f(x)$ is a transformation that pushes example of the first class toward direction of vector +1 and examples of the other class toward direction of vector -1.

<img src="http://www.svcl.ucsd.edu/projects/mcboost/figs/binary_pushing.png">

Using this observation, for multiclass boosting, e.g. 3 classes, we need to push examples in three different directions.

<img src="http://www.svcl.ucsd.edu/projects/mcboost/figs/3_class_pushing.png" width="60%"/>

* http://www.multiboost.org/

multiBoost

Similar to AdaBoost in the two class case, this new algorithm combines weak classifiers and only requires the performance of each weak classifier be better than random guessing (rather than 1/2).

SAMME

---

* Initialize the observation weights $w_i=\frac{1}{N}, i=1,2,\dots,N$.
* For $t=1,2,\dots,T$:
    * Fit a classifier $G_t(x)$ to the training data using weights w_i.
    * Compute

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

    * Compute $\alpha_t=\log(\frac{1-err_t}{err_t})+\log(K-1)$.
    * Set $w_i\leftarrow w_i\exp[\alpha_t\mathbb{I}(G_t(x_i)\neq \mathrm{y}_i)], i=1,2,\dots,N$ and renormalize so that $\sum_i w_i=1$.
* Output $G(x)=\arg\max_k[\sum_{t=1}^{T}\alpha_t\mathbb{I}_{G_t(x)=k}]$.

* https://web.stanford.edu/~hastie/Papers/samme.pdf

<img src="https://cdn.mathpix.com/snip/images/aoxWmzifAs8sfUHfXVHlUDeDB_C3XDh6i-P5OtAitCA.original.fullsize.png">

MCBoost

In our paper we showed that for a M-class classification problem, the optimal set of directions will form a simplex in M-1 dimensions. Below you can see examples of those directions for $M=2,3,4$.

<img src="http://www.svcl.ucsd.edu/projects/mcboost/figs/simplex_codes.png" />

* http://www.svcl.ucsd.edu/projects/mcboost/
* https://www.lri.fr/~kegl/research/publications.html
* MultiBoost: A Multi-purpose Boosting Package
* A theory of multiclass boosting
* The return of AdaBoost.MH: multi-class Hamming trees
* Multi-class AdaBoost
* https://github.com/tizfa/sparkboost
* multiclass boosting: theory and algorithms
* LDA-AdaBoost.MH: Accelerated AdaBoost.MH based on latent Dirichlet allocation for text categorization

* http://www.svcl.ucsd.edu/projects/sop_boost/
* Multiclass Boosting: MCBoost

Bonsai Boosted Decision Tree

bonsai BDT (BBDT):

1. $\boxed{\text{discretizes}}$ input variables before training which ensures a fast and robust implementation
2. converts decision trees to n-dimentional table to store
3. prediction operation takes one reading from this table

The first step of preprocessing data limits where the splits of the data can be made and, in effect, permits the grower of the tree to
control and shape its growth; thus, we are calling this a bonsai BDT (BBDT).
The discretization works by enforcing that the smallest keep interval that can be created when training the BBDT is:

$$\Delta x_{min} > \delta_x \forall x \text{ on all leaves}$$

where $\delta_x = \min\{|x_i - x_j| : x_i, x_j \in x_{discrete}\}$.

Discretization means that the data can be thought of as being binned, even though many of the possible bins may not form leaves in the BBDT; thus, there are a finite number, n_{keep}^{max}, of possible keep regions that can be defined. If the n_{keep}^{max} BBDT response values can be stored in memory, then the extremely large number of if/else statements that make up a BDT can be converted into a one-dimensional array of response values. One-dimensional array look-up speeds are extremely fast; they take, in essense, zero time.
If there is not enough memory available to store all of the response values, there are a number of simple alternatives that can be used. For example, if the cut value is known then just the list of indices for keep regions could be stored.

---

* Efficient, reliable and fast high-level triggering using a bonsai boosted decision tree
* Boosting bonsai trees for efficient features combination : Application to speaker role identification
* Bonsai Trees in Your Head: How the Pavlovian System Sculpts Goal-Directed Choices by Pruning Decision Trees
* HEM meets machine learning
* BDT: Gradient Boosted Decision Tables for High Accuracy and Scoring Efficiency

<img src="http://www.arvinzyy.cn/2017/10/03/Gradient-Boosted-Decision-Tables/3.png" />

Gradient Boosting Decision Tree

* Gradient Boosted Feature Selection
* Gradient Regularized Budgeted Boosting
* Open machine learning course. Theme 10. Gradient boosting
* GBM in Machine Learning in R

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
<img title =golf src=https://explained.ai/gradient-boosting/images/golf-MSE.png width=60% />

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

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

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

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

where $C=\sum_{k=1}^{t-1}\Omega(f_k)$.

Particularly, we take $L(x,y)=(x-y)^2$, and the objective function is given by

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

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 simplified to

$$\arg\min_{f_t}\sum_{i=1}^{n}[-2(y_i-\hat{y}_i^{(t-1)})f_t(x_i)+f_t^2(x_i)] \implies f_t(x_i)=(y_i-\hat{y}_i^{(t-1)})$$

where $i\in\{1,\cdots,n\}$ and $(y_i-\hat{y}_i^{(t-1)})=-\frac{1}{2}[\frac{\partial L(\mathrm{y}_i,f(x_i))}{\partial f(x_i)}]_{f=f_{t-1}}$.

Boosting for Regression Tree

- Input training data set $\{(x_i,\mathrm{y}_i)\mid i=1,\cdots,n\},x_i\in x\subset\mathbb{R}^n,y_i\in\mathcal{Y}\subset\mathbb{R}$.
- Initialize $f_0(x)=0$.
- For $t=1,2,\dots,T$:
  - For $i=1,2,\dots,n$ compute the residuals
  $$r_{i,t}=y_i-f_{t-1}(x_i)=y_i-\hat{y}_i^{t-1}.$$
  - Fit a regression tree to the targets $r_{i,t}$ giving **terminal regions**
  $$R_{j,m},j=1,2,\dots,J_m.$$
  - For $j=1,2,\dots,J_m$ compute
  $$\boxed{\gamma_{j,t}=\arg\min_{\gamma}\sum_{x_i\in R_{j,m}}L(\mathrm{d}_i,f_{t-1}(x_i)+\gamma)}.$$
  - Update $f_t=f_{t-1}+\nu\sum_{j=1}^{J_m}\gamma_{j,t}\mathbb{I}(x\in R_{j,m}),\nu\in(0,1)$.
- Output $f_T(x)$.

For general loss function, it is more common that $(y_i-\hat{y}_i^{(t-1)})\neq-\frac{1}{2}[\frac{\partial L(\mathrm{y}_i,f(x_i))}{\partial f(x_i)}]_{f=f_{t-1}}$.

Gradient Boosting for Regression Tree

- Input training data set $\{(x_i, \mathrm{y}_i) \mid i=1,\cdots,n\}, x_i \in x \subset \mathbb{R}^n, y_i \in \mathcal{Y} \subset \mathbb{R}$.
- Initialize $f_0(x) = \arg\min_\gamma L(\mathrm{y}_i, \gamma)$.
- For $t = 1, 2, \dots, T$:
    - For $i = 1, 2, \dots, n$ compute

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

    - Fit a regression tree to the targets $r_{i,t}$ giving **terminal regions**

$$R_{j,m}, j = 1, 2, \dots, J_m.$$

    - For $j = 1, 2, \dots, J_m$ compute

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

    - Update $f_t = f_{t-1} + \nu \sum_{j=1}^{J_m} \gamma_{j,t} \mathbb{I}(x \in R_{j,m}), \nu \in (0,1)$.
- Output $f_T(x)$.

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

$$\begin{aligned} f_t &= f_{t-1} + \nu \underbrace{\sum_{j=1}^{J_m} \gamma_{j,t} \mathbb{I}(x \in R_{j,m})}_{\text{to fit the gradient}}, \\ &\approx f_{t-1} + \nu \underbrace{\sum_{i=1}^{n} -[\frac{\partial L(\mathrm{y}_i, f(x_i))}{\partial f(x_i)}]_{f=f_{t-1}}}_{\text{fitted by a regression tree}}, \nu \in (0,1). \end{aligned}$$

Note that the incremental tree is approximate to the negative gradient of the loss function, i.e.,

$$\boxed{\sum_{j=1}^{J_m} \gamma_{j,t} \mathbb{I}(x \in R_{j,m}) \approx \sum_{i=1}^{n} -[\frac{\partial L(\mathrm{y}_i, f(x_i))}{\partial f(x_i)}]_{f=f_{t-1}}}$$

where J_m is the number of the terminal regions and n is the number of training samples/data.

Method	Hypothesis space	Update formulea	Loss function
Gradient Descent	parameter space Θ	$\theta_t = \theta_{t-1} - \rho_t \underbrace{\nabla_\theta L\mid_{\theta=\theta_{t-1}}}_{\text{Computed by Back-Propagation}}$	$L(f) = \sum_i \ell(y_i, f(x_i \mid \theta))$

Method	Hypothesis space	Update formulea	Loss function
Gradient Boost	function space $\mathcal{F}$	$F_t = F_{t-1} - \rho_t \underbrace{\nabla_F L\|_{F=F_{t-1}}}_{\text{Approximated by Decision Tree}}$	$L(F) = \sum_i \ell(y_i, F(x_i))$

- Greedy Function Approximation: A Gradient Boosting Machine
- 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://github.com/benedekrozemberczki/awesome-gradient-boosting-papers
- https://github.com/talperetz/awesome-gradient-boosting
- https://github.com/parrt/dtreeviz

<img src="https://raw.githubusercontent.com/benedekrozemberczki/awesome-gradient-boosting-papers/master/boosting.gif">

Ensemble Methods	Training Data	Decision Tree Construction	Update Formula
AdaBoost	$(x_i, y_i, w_{i,t})$	Fit a `classifier` $G_t(x)$ to the training data using weights w_i	$f_t = f_{t-1} + \alpha_t G_t$
Gradient Boost	$(x_i, y_i, r_{i,t})$	Fit a `tree` $G_t(x)$ to the targets $r_{i,t}$	$f_t = f_{t-1} + \nu G_t$.

Note that α_t is computed as $\alpha_t = \log(\frac{1-err_t}{err_t})$ while the shrinkage parameter ν is chosen in $(0, 1)$.
AdaBoost is desined for any classifier while Gradient Boost methods is usually applied to decision tree.

Stochastic Gradient Boost

A minor modification was made to gradient boosting to incorporate randomness as an integral part of the procedure. Specially, at each iteration a subsample of the training data is drawn at random (without replacement) from the full training data set. This randomly selected subsamples is then used, instead of the full sample, to fit the base learner and compute the model update for the current iteration.

Stochastic Gradient Boosting for Regression Tree

- Input training data set $\{(x_i, \mathrm{y}_i) \mid i = 1, \cdots, n\}, x_i \in x \subset \mathbb{R}^n, y_i \in \mathcal{Y} \subset \mathbb{R}$. Amd $\{\pi(i)\}_1^N$ be a random permutation of their integers $\{1, \dots, n\}$. Then a random sample of size $\hat{n} < n$ is given by $\{(x_{\pi(i)}, \mathrm{y}_{\pi(i)}) \mid i = 1, \cdots, \hat{n}\}$.

- Initialize $f_0(x)=\arg\min_{\gamma} L(\mathrm{y}_i,\gamma)$.
- For $t=1,2,\dots,T$:
  - For $i=1,2,\dots,\hat{n}$ compute

$$r_{\pi(i),t}=-{[\frac{\partial L(\mathrm{y}_{\pi(i),t},f(x_{\pi(i),t}))}{\partial f(x_{\pi(i),t})}]}_{f=f_{t-1}}.$$

  - Fit a regression tree to the targets $r_{\pi(i),t}$ giving **terminal regions**

$$R_{j,m}, j=1,2,\dots,J_m.$$

  - For $j=1,2,\dots,J_m$ compute

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

  - Update $f_t=f_{t-1}+\nu\sum_{j=1}^{J_m}\gamma_{j,t}\mathbb{I}(x\in R_{j,m}),\nu\in(0,1)$.
- Output $f_T(x)$.

- https://statweb.stanford.edu/~jhf/ftp/stobst.pdf
- https://statweb.stanford.edu/~jhf/
- Stochastic gradient boosted distributed decision trees

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 as following

$$f(x+h)=f(x)+f'(x)h+f^{(2)}(x)h^2/2!+\cdots+f^{(n)}(x)h^{(n)}/n!+\cdots$$

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.

<img src="http://zhanpengfang.github.io/fig_418/gbt_example.jpg" width="80%" />

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

$$
\begin{aligned}
obj^{(t)} &= \sum_{i=1}^{n} L[y_i, \hat{y}_i^{(t-1)} + f_t(x_i)] + \Omega(f_t) + C \\
&\simeq \sum_{i=1}^{n} \underbrace{[L(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{h_i f_t^2(x_i)}{2}]}_{\text{2ed Order Taylor Expansion}} + \Omega(f_t) + C'
\end{aligned}
$$

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

$$
\begin{gathered}
f_t(x) = w_{q(x)} = \sum\nolimits_{i=1}^{T} w_i \mathbb{I}(q(x) = i), \\
w \in \mathbb{R}^T, q : \mathbb{R}^d \Rightarrow \{1, 2, \dots, T\}.
\end{gathered}
$$

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

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

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

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

where $I_j = \{i \mid q(x_i) = j\}$ is the set of indices of data points assigned to the j-th leaf.
The equation hold because only the leaves or terminal nodes output the results.
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:

$$w_j^* = -(\underbrace{\overbrace{H_j + \lambda}^{\text{Hessian of objective function}}}_{\text{learning rate}})^{-1}\underbrace{G_j}_{\text{gradient}},$$

$$obj^* = \frac{1}{2}\sum_{j=1}^{T} -(H_j + \lambda)^{-1}G_j^2 + \gamma T.$$

Method	Hypothesis space	Update formulea
Newton's Method	parameter space Θ	$\theta_t = \theta_{t-1} - \rho_t(\overbrace{\nabla_\theta^2 L\|_{\theta=\theta_{t-1}}}^{\text{Hessian Matrix}})^{-1}\underbrace{\nabla_\theta L\|_{\theta=\theta_{t-1}}}_{\text{Gradient}}$
xGBoost	function space $\mathcal{F}$	$F_t = F_{t-1} - \rho_t\underbrace{(\overbrace{\nabla_F^2 L\|_{F=F_{t-1}}}^{\text{Hessian}})^{-1}\overbrace{\nabla_F L\|_{F=F_{t-1}}}^{\text{Gradient}}}_{\text{Approximated by Decision Tree}}$

Another key of xGBoost is how to a construct a tree fast and painlessly.
We will try to optimize `one level` of the tree at a time. Specifically we try to split a leaf into two leaves, and the score it gains is

$$Gain = \frac{1}{2}\left[\underbrace{\frac{G_L^2}{H_L+\lambda}}_{\text{from left leaf}} + \underbrace{\frac{G_R^2}{H_R+\lambda}}_{\text{from the right leaf}} - \underbrace{\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}}_{\text{from the original leaf}}\right] - \gamma$$

This formula can be decomposed as 1) the score on the new left leaf 2) the score on the new right leaf 3) The score on the original leaf 4) regularization on the additional leaf.
We can see an important fact here: if the gain is smaller than γ, we would do better not to add that branch. This is exactly the **pruning techniques** in tree based models! By using the principles of supervised learning, we can naturally come up with the reason these techniques work.

<img src="https://raw.githubusercontent.com/dmlc/web-data/master/xgboost/model/struct_score.png" width="70%">

<img src="https://pic3.zhimg.com/80/v2-46792243acd6570c3416df14a8d0bb1e_hd.jpg" width="80%" />
<img src="https://pic3.zhimg.com/80/v2-6cd871031772e6ab3005b3166731bae2_hd.jpg" width="80%" />

Other features include:

- shrinkages： Shrinkage scales newly added weights by a factor η after each step of tree boosting. Similar to a learning rate in tochastic optimization, shrinkage reduces the influence of each individual tree and leaves space for future trees to improve the model.
- column (feature) subsampling: This technique is used in RandomForest.
- row sub-sampling: speeds up computations of the parallel algorithm.

SYSTEM DESIGN

- Column Block for Parallel Learning

- Cache-aware Access
- Blocks for Out-of-core Computation

<img title = "Tianqi Chen" src="https://tqchen.com/data/img/tqchen-new.jpg" width="30%" />

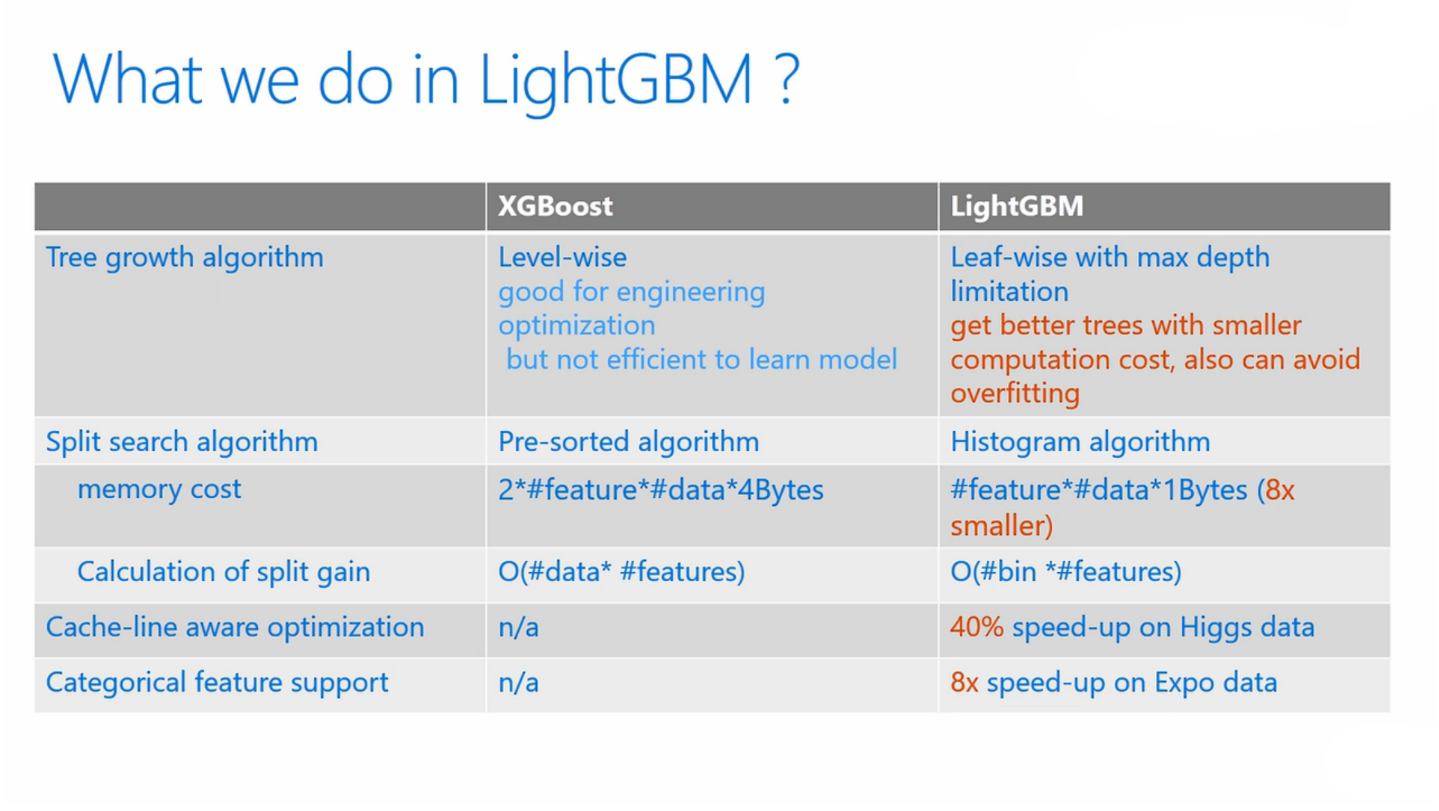
- https://xgboost.readthedocs.io/en/latest/tutorials/model.html
- https://xgboost.ai/
- A Kaggle Master Explains Gradient Boosting
- Extreme Gradient Boosting with R
- XGBoost: A Scalable Tree Boosting System
- xgboost的原理没你想像的那么难
- How to Visualize Gradient Boosting Decision Trees With XGBoost in Python
- Awesome XGBoost
- Story and lessons from xGBoost
- Awesome XGBoost

<img src=https://pic2.zhimg.com/50/v2-d8191a1191979eadbd4df191b391f917_hd.jpg />

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

A major reason is that for each feature, they need to scan all the data instances to estimate the information gain of all possible split points, which is very time consuming. To tackle this problem, the authors of lightGBM propose two novel techniques: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB).

<img title="GOSS" src="https://pic3.zhimg.com/80/v2-1608d8b79f3d605e111878b715254d3e_hd.jpg" width="69%" />
<img title="EFB" src="https://pic1.zhimg.com/80/v2-4d7875a088e184c694474be2bec26698_hd.jpg" width="90%" />

`Leaf-wise` learning is to split a leaf(with max delta loss) into 2 leaves rather than split leaves in one level into 2 leaves.

<img src=http://zhoutao822.coding.me/2019/01/13/LightGBM/1.png width=80% />

And it limits the depth of the tree in order to avoid over-fitting.

<img src=http://zhoutao822.coding.me/2019/01/13/LightGBM/7.png width=80% />

Instead of one-hot encoding, the optimal solution is to split on a `categorical feature` by partitioning its categories into 2 subsets. If the feature has k categories, there are $2^{(k-1)} - 1$ possible partitions. But there is an efficient solution for regression trees[8]. It needs about $O(k \times log(k))$ to find the optimal partition.

The basic idea is to sort the categories according to the training objective at each split. More specifically, LightGBM sorts the `histogram` (for a categorical feature) according to its `accumulated values` (sum_gradient / sum_hessian) and then finds the best split on the sorted histogram.

`Histogram` is an un-normalized empirical cumulative distribution function, where the continuous features (in flow point data structure) is split into k buckets by threahold values such as if $x \in [0, 2)$ then x will be split into bucket 1. It really reduces the complexity to store the data and compute the impurities based on the distribution of features.

<img src="http://zhoutao822.coding.me/2019/01/13/LightGBM/5.png" width="80%" />

Optimization in parallel learning

Feature Parallel in LightGBM, Data Parallel in LightGBM

<img src="https://www.msra.cn/wp-content/uploads/2017/06/4caedc7agw1fbfnmnhh0qj20fh071q3e.jpg" width="80%" />
<img src="https://www.msra.cn/wp-content/uploads/2017/06/4caedc7agw1fbfnp8a5suj20q20hs41i.jpg" width="80%" />
<img src="https://www.msra.cn/wp-content/uploads/2017/06/4caedc7agw1fbfnp8c1jjj20wy0dd41o.jpg" width="80%"/>
<img src="https://www.msra.cn/wp-content/uploads/2017/06/4caedc7agw1fbfnp86gk4j20xq0hlwjo.jpg" width="80%" />
<img src="https://zhoutao822.coding.me/2019/01/13/LightGBM/6.png" width="80%">

- A Communication-Efficient Parallel Algorithm for Decision Tree
- 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
- Reference papers of lightGBM
- https://lightgbm.readthedocs.io/en/latest/Features.html
- https://aichamp.wordpress.com/tag/lightgbm/

CatBoost

`CatBoost` is an algorithm for gradient boosting on decision trees. Developed by Yandex researchers and engineers, it is the successor of the `MatrixNet` algorithm that is widely used within the company for ranking tasks, forecasting and making recommendations. It is universal and can be applied across a wide range of areas and to a variety of problems such as 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.

Two critical algorithmic advances introduced in CatBoost are the implementation of `ordered boosting`, a permutation-driven alternative to the classic algorithm, and
an innovative algorithm for processing `categorical features`.
Both techniques were created to fight a prediction shift caused by a special kind of target leakage present in all currently existing implementations of gradient boosting algorithms.

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

Before learning, the possible values of objects are divided into disjoint ranges ($\boxed{\text{buckets}}$) delimited by the threshold values ($\boxed{\text{splits}}$). 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.

The most widely used technique which is usually applied to low-cardinality categorical features
is one-hot encoding; another way to deal with categorical features is to compute some statistics using the label values of the examples.
Namely, assume that we are given a dataset of observations $D=\{(\mathrm{X}_i,\mathrm{Y}_i)\mid i=1,2,\cdots,n\}$,
where $\mathrm{X}_i=(x_{i,1},x_{i,2},\cdots,x_{i,m})$ is a vector of m features, some numerical, some categorical, and $\mathrm{Y}_i\in\mathbb{R}$ is a label value.
The simplest way is to substitute the category with the *average* label value on the whole train dataset. So, $x_{i;k}$ is substituted with $\frac{\sum_{j=1}^{n}[x_{j;k}=x_{i;k}]\cdot\mathrm{Y}_j}{\sum_{j=1}^{n}[x_{j;k}=x_{i;k}]}$; where $[\cdot]$ denotes Iverson

brackets, i.e., $[x_{j;k} = x_{i;k}]$ equals 1 if $x_{j;k} = x_{i;k}$ and 0 otherwise.
This procedure, obviously, leads to overfitting.

CatBoost uses a more efficient strategy which reduces overfitting and allows to use the whole dataset for training.

Before each split is selected in the tree (see Choosing the tree structure), `categorical features are transformed to numerical`. This is done using various statistics on combinations of categorical features and combinations of categorical and numerical features.

The method of transforming categorical features to numerical generally includes the following stages:

- Permutating the set of input objects in a random order.
- Converting the label value from a floating point to an integer.

Namely, we perform a random permutation of the dataset and
for each example we compute average label value for the example with the same category value placed before the given one in the permutation.
Let $\sigma = (\sigma_1, \cdots, \sigma_n)$ be the permutation, then $x_{\sigma_p;k}$ is substituted with

$$\frac{\sum_{j=1}^{p-1}[x_{\sigma_j;k} = x_{\sigma_p;k}] \cdot \mathrm{Y}_{\sigma_j} + a \cdot P}{\sum_{j=1}^{p-1}[x_{\sigma_j;k} = x_{\sigma_p;k}]}$$

where we also add a prior value P and a parameter $a > 0$, which is the weight of the prior.

The method depends on the machine learning problem being solved (which is determined by the selected loss function).

The tree depth and other rules for choosing the structure are set in the starting parameters.

How a “feature-split” pair is chosen for a leaf:

- A list is formed of possible candidates (“feature-split pairs”) to be assigned to a leaf as the split.
- A number of penalty functions are calculated for each object (on the condition that all of the candidates obtained from step 1 have been assigned to the leaf).
- The split with the smallest penalty is selected.

The resulting value is assigned to the leaf.

This procedure is repeated for all following leaves (the number of leaves needs to match the depth of the tree).

CatBoost implements an algorithm that allows to fight usual gradient boosting biases.

Assume that we take one random permutation σ of the training examples and maintain n different supporting models $M_1, \cdots, M_n$ such that the model M_i is learned using only the `first i examples in the permutation`.
At each step, in order to obtain the residual for j-th sample, we use the model M_{j-1}.

<img src="https://cdn.mathpix.com/snip/images/kVsAOTih7qFTcp3z-BPJIH8cKdrq5GpvkYivu0XRg-o.original.fullsize.png" witdh= "80%"/>

Three Steps

Three Steps
<img src = "https://s2.51cto.com/oss/201808/30/108338cbd6df1a13dd9ed6d14c9da35d.png" width="100%" />
<img src = "https://s1.51cto.com/oss/201808/30/e0ac1ddc9b9c0e513e2669f56151edc7.png" width="100%" />
<img src = "https://s1.51cto.com/oss/201808/30/4183b4bba0529b55e5f4f1bef8072ab5.png" width="100%" />

Andrey Gulin

- https://tech.yandex.com/catboost/
- How is CatBoost? Interviews with developers
- Reference papers of CatBoost
- CatBoost: unbiased boosting with categorical features
- Efficient Gradient Boosted Decision Tree Training on GPUs
- CatBoost：比XGBoost更优秀的GBDT算法

More: TencentBoost, ThunderGBM and Beyond

There are more gradient boost tree algorithms such as ThubderGBM, TencentBoost, GBDT on angle and H2o.

TencentBoost

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 data sets with medium-dimensional features, they can suffer performance degradation for many industrial applications where the trained data sets contain high dimensional features. The performance degradation derives from their inefficient mechanisms for model aggregation-either map-reduce 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.

- TencentBoost: A Gradient Boosting Tree System with Parameter Server
- GBDT on Angel

- The purposes of using parameter server in GBDT

ThunderGBM

`ThunderGBM` is dedicated to helping users apply GBDTs and Random Forests to solve problems efficiently and easily using GPUs. Key features of ThunderGBM are as follows.

- Support regression, classification and ranking.
- Use same command line options as XGBoost, and support Python (scikit-learn) interface.
- Supported Operating Systems: Linux and Windows.
- ThunderGBM is often 10 times faster than XGBoost, LightGBM and CatBoost. It has excellent performance on handling high dimensional and sparse problems.

Methods	Tree Construction	Update Formula	Training Methods
XGBoost	Newton-like		
LightGBM	leaf-wise		
CatBoost	ordered boosting		
TencentBoost			
ThunderGBM			

- ThunderGBM: Fast GBDTs and Random Forests on GPUs
- ThunderGBM：快成一道闪电的梯度提升决策树
- Efficient Gradient Boosted Decision Tree Training on GPUs
- Gradient Boosted Categorical Embedding and Numerical Trees
- 一步一步理解GB、GBDT、xgboost
- 从结构到性能，一文概述XGBoost、Light GBM和CatBoost的同与不同
- 从决策树、GBDT到XGBoost/lightGBM/CatBoost

Parallel Gradient Boosting Decision Trees

Distributed implementations of GBT generally follow the following procedures.

1. The training instances are partitioned onto a set of workers.
2. To split one tree node, each worker computes the gradient statistics of the instances. For each feature, an individual gradient histogram needs to be built.
3. A coordinator aggregates the gradient histograms of all workers, and finds the best split feature and split value.
4. The coordinator broadcasts the split result. Each worker splits the current tree node, and proceeds to new tree nodes

Since more features generally yield higher predictive accuracy in practice, many datasets used in industrial applications often contain hundreds of thousands or even millions of
features. Considering that GBT requires merging of gradient
histograms for all the features during each iteration, when the

dimension of features increases, the communication cost of
model aggregation proportionally increases meanwhile.

- Parallel Gradient Boosting Decision Trees
- https://zeyiwen.github.io/papers.html
- PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce
- FastForest: Learning Gradient-Boosted Regression Trees for Classification, Regression and Ranking
- Efficient Distributed Decision Trees for Robust Regression
- Communication and Memory Efficient Parallel Decision Tree Construction

<img src="https://media.springernature.com/full/springer-static/image/art%3A10.1186%2Fs40537-019-0186-3/MediaObjects/40537_2019_186_Fig1_HTML.png" width="70%" />

Parallelize Node Building at Each Level

<img src="http://zhanpengfang.github.io/fig_418/sequential_alg.jpg" />

Parallelize Split Finding on Each Node

<img src="http://zhanpengfang.github.io/fig_418/split_parallel_alg.jpg" width="80%"/>

Parallelize Split Finding at Each Level by Features

<img src="http://zhanpengfang.github.io/fig_418/feature_parallel_alg.jpg" />

- Tiny Gradient Boosting Tree
- Programmable Decision Tree Framework
- bonsai-dt: Programmable Decision Tree Framework
- Treelite : model compiler for decision tree ensembles
- Block-distributed Gradient Boosted Trees
- Distributed decision tree ensemble learning in Scala
- Yggdrasil: An Optimized System for Training Deep Decision Trees at Scale
- TF Boosted Trees: A scalable TensorFlow based framework for gradient boosting
- Runtime Optimizations for Tree-based Machine Learning Models
- Exploiting Thread-Level Parallelism to Build Decision Trees

Fast Traversal of Large Ensembles of Regression Trees

- Fast Traversal of Large Ensembles of Regression Trees
- Parallelizing the Traversal of Large Ensembles of Decision Trees

QuickScorer

`QuickScorer` was designed by Lucchese, C., Nardini, F. M., Orlando, S., Perego, R., Tonellotto, N., and Venturini, R. with the support of Tiscali S.p.A.

It adopts a novel bitvector representation of the tree-based ranking model, and performs an interleaved traversal of the ensemble by means of simple logical bitwise operations. The performance of the proposed algorithm are unprecedented, due to its cache-aware approach, both in terms of data layout and access patterns, and to a control flow that entails very low branch mis-prediction rates.

All the nodes whose Boolean conditions evaluate to *False* are called false nodes, and true nodes otherwise.
The scoring of a document represented by a feature vector x requires the traversing of all the trees in the ensemble, starting at their root nodes.
If a visited node in N is a false one, then the right branch is taken, and the left branch otherwise.
The visit continues recursively until a leaf node is reached, where the value of the prediction is returned.

The building block of this approach is an alternative method for tree traversal based on `bit-vector computations`.
Given a tree and a vector of document features,
this traversal processes all its nodes and produces a bitvector
which encodes the exit leaf for the given document.

Given an input feature vector x and a tree $T_h = (N_h; L_h)$, where N_h is a set of internal nodes and L_h is a set of leaves,
our tree traversal algorithm processes the internal nodes of
T_h with the goal of identifying a set of candidate exit leaves, denoted by C_h with $C_h \subset L_h$,
which includes the actual exit leaf e_h.
Initially C_h contains all the leaves in L_h, i.e., $C_h = L_h$.
Then, the algorithm evaluates one after the other in an arbitrary order the test conditions of all the internal nodes of T_h.
Considering the result of the test for a certain internal node $n \in N_h$,
the algorithm is able to infer that some leaves cannot be the exit leaf and, thus, it can safely remove them from C_h.
Indeed, if n is a false node (i.e., its test condition is false), the leaves in the left subtree of n cannot be the exit leaf and they can be safely removed from C_h.
Similarly, if n is a true node, the leaves in the right subtree of n can be removed from C_h.

The second refinement implements the operations on C_h with fast bit-wise operations.
The idea is to represent C_h with a bitvector $\mathrm{leaf\_index}_h$, where each bit corresponds to a distinct leaf in L_h, i.e., $\mathrm{leaf\_index}_h$ is the characteristic vector of C_h.
Moreover, every internal node n is associated with a bit mask of the same length encoding
(with 0's) the set of leaves to be removed from C_h whenever n turns to be a false node.
In this way, the bitwise `logical AND` between $\mathrm{leaf\_index}_h$ and the bit mask of a false node n corresponds to the removal of the leaves in the left subtree of n from C_h.
Once identified all the false nodes in a tree and performed the associated AND operations over $\mathrm{leaf\_index}_h$, the exit leaf of the tree corresponds to the leftmost bit set to 1 in $\mathrm{leaf\_index}_h$.

One important result is that `Quick Scorer` computes
$s(x)$ by only identifying the branching nodes whose test evaluates to false, called false nodes.
For each false node detected in $T_h \in T$, `Quick Scorer` updates a bitvector associated with T_h, which stores information that is eventually exploited to identify
the exit leaf of T_h that contributes to the final score $s(x)$.
To this end, `Quick Scorer` maintains for each tree $T_h \in T$ a bitvector *leafidx[h]*, made of $\wedge$ bits, one per leaf.
Initially, every bit in *leafidx[h]* is set to 1. Moreover, each branching node is
associated with a bitvector mask, still of $\wedge$ bits, identifying the set of unreachable
leaves of T_h in case the corresponding test evaluates to false.
Whenever a false node is visited, the set of unreachable leaves *leafidx[h]* is updated through a logical $AND(\wedge)$ with mask.
Eventually, the leftmost bit set in *leafidx[h]* identifies the leaf corresponding to the score contribution of T_h, stored in the lookup table *leafvalues*.

ALGORITHM 1: Scoring a feature vector x using a binary decision tree T_h

```
- **Input**:
    - $x$: input feature vector
    - $T_h = (N_h, L_h)$: binary decision tree, with
        - $N_h = \{n_0, n_1, \cdots\}$: internal nodes of $T_h$
        - $L_h = \{l_0, l_1, \cdots\}$: leaves of $T_h$
        - $n.mask$: node bit mask associated with $n \in N_h$
        - $l_j.val$: score contribution associated with leaf $l_j \in L_h$
- **Output**:
    - tree traversal output value
- $score(x, T_h)$:
    - $\text{leaf\_index}_h \leftarrow (1, 1, \ldots, 1)$
    - $U \leftarrow FindFalse(x, T_h)$
    - **foreach node** $n \in U$ **do**
        - $\text{leaf\_index}_h \leftarrow \text{leaf\_index}_h \wedge n.mask$
    - $j \leftarrow$ index of leftmost bit set to 1 of $\text{leaf\_index}_h$
    - **return** $l_j.val$
```

ALGORITHM 2: : The QUICKSCORER Algorithm

```
- **Input**:
    - $x$: input feature vector
    - $\mathcal{T}$: ensemble of binary decision trees, with
        - $\{w_0, w_1, \cdots, w_{|\mathcal{T}|-1}\}$: weights, one per tree
        - $thresholds$: sorted sublists of thresholds, one sublist per feature
        - $treeids$: tree's ids, one per node/threshold
        - $nodemasks$: node bit masks, one per node/threshold
        - $offsets$: offsets of the blocks of triples
        - $leafindexes$: result bitvectors, one per each tree
        - $leafvalues$: score contributions, one per each tree leaf
- **Output**:
    - final score of $x$
- $\text{QUICKSCORER}(x, T_h)$:
    - **foreach node** $h \in \{0, 1 \cdots, |T| - 1\}$ **do**
        - $\text{leaf\_index}_h \leftarrow (1, 1, \ldots, 1)$
    - **foreach node** $k \in \{0, 1 \cdots, |\mathcal{F}| - 1\}$ **do**
        - $i \leftarrow offsets[k]$
        - $end \leftarrow offsetsets[k + 1]$
        - **while** $x[k] > thresholds[i]$ do
            - $h \leftarrow treeids[i]$
            - $\text{ leafindexes } [h] \leftarrow \text{ leafindexes } [h] \wedge \text{ nodemasks } [i]$
            ▪ $i \leftarrow i + 1$
            ▪ **if** $i \geq end$ **then**
                ▪ **break**
    ○ $score \leftarrow 0$
    ○ **foreach node** $h \in \{0, 1 \cdots, |T| - 1\}$ **do**
        ▪ $j \leftarrow$ index of leftmost bit set to 1 of $leafindexes[h]$
        ▪ $l \leftarrow h \cdot |L_h| + j$
        ▪ $\text{score} \leftarrow \text{score} + w_h \cdot \text{leafvalues}[l]$
    ○ **return** $score$
```

- [ ] QuickScorer: a fast algorithm to rank documents with additive ensembles of regression trees
- [ ] Official repository of Quickscorer
- [ ] QuickScorer: Efficient Traversal of Large Ensembles of Decision Trees
- [ ] Fast Ranking with Additive Ensembles of Oblivious and Non-Oblivious Regression Trees

Tree traversal

- [ ] GPU-based Parallelization of QuickScorer to Speed-up Document Ranking with Tree Ensembles

https://github.com/hpclab/gpu-quickscorer

https://github.com/hpclab/multithread-quickscorer

https://github.com/hpclab/vectorized-quickscorer

https://patents.google.com/patent/WO2016203501A1/en

<img src="https://ercim-news.ercim.eu/images/stories/EN107/perego.png" width="60%" />

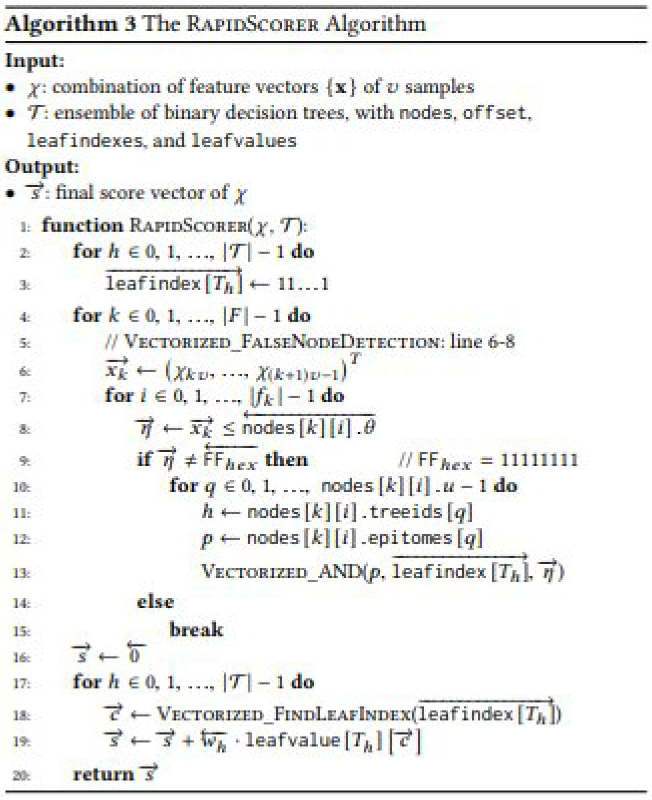
vQS

Considering that in most application scenarios the same tree-based model is applied to a multitude of items, we recently introduced further optimisations in QS. In particular, we introduced vQS [3], a parallelised version of QS that exploits the SIMD capabilities of mainstream CPUs to score multiple items in parallel. Streaming SIMD Extensions (SSE) and Advanced Vector Extensions (AVX) are sets of instructions exploiting wide registers of 128 and 256 bits that allow parallel operations to be performed on simple data types. Using SSE and AVX, vQS can process up to eight items in parallel, resulting in a further performance improvement up to a factor of 2.4x over QS. In the same line of research we are finalising the porting of QS to GPUs, which, preliminary tests indicate, allows impressive speedups to be achieved.

- Exploiting CPU SIMD Extensions to Speed-up Document Scoring with Tree Ensembles

RapidScorer

`RapidScorer` is a novel framework
for speeding up the scoring process of industry-scale tree ensemble models, without hurting the quality of scoring results.
`RapidScorer` introduces a modified run length encoding called `epitome` to the bitvector representation of the tree nodes.
Epitome can greatly reduce the computation cost to traverse the tree ensemble, and work with several other proposed strategies to maximize the compactness of data units in memory.
The achieved compactness makes it possible to fully utilize data parallelization to improve model scalability.



Algorithm 3 The RapidScorer Algorithm

Input:
- χ: combination of feature vectors $\{\mathbf{x}\}$ of υ samples
- $\mathcal{T}$: ensemble of binary decision trees, with `nodes`, `offset`, `leafindexes`, and `leafvalues`

Output:
- $\overrightarrow{s}$: final score vector of χ

```
1:  function RapidScorer($\chi$, $\mathcal{T}$):
2:      for $h \in 0, 1, \ldots, |\mathcal{T}| - 1$ do
3:          $\overrightarrow{\texttt{leafindex}[T_h]} \leftarrow 11\ldots1$
4:      for $k \in 0, 1, \ldots, |F| - 1$ do
5:          // Vectorized_FalseNodeDetection: line 6-8
6:          $\overrightarrow{x_k} \leftarrow (\chi_{k\upsilon}, \ldots, \chi_{(k+1)\upsilon-1})^T$
7:          for $i \in 0, 1, \ldots, |f_k| - 1$ do
8:              $\overrightarrow{\eta} \leftarrow \overrightarrow{x_k} \leq \overleftarrow{\texttt{nodes}[k][i].\theta}$
9:              if $\overrightarrow{\eta} \neq \overleftarrow{\texttt{FF}_{hex}}$ then            // $\texttt{FF}_{hex} = 11111111$
10:                 for $q \in 0, 1, \ldots, \texttt{nodes}[k][i].u - 1$ do
11:                     $h \leftarrow \texttt{nodes}[k][i].\texttt{treeids}[q]$
12:                     $p \leftarrow \texttt{nodes}[k][i].\texttt{epitomes}[q]$
13:                     Vectorized_AND($p$, $\overrightarrow{\texttt{leafindex}[T_h]}$, $\overrightarrow{\eta}$)
14:             else
15:                 break
16:     $\overrightarrow{s} \leftarrow \overleftarrow{0}$
17:     for $h \in 0, 1, \ldots, |\mathcal{T}| - 1$ do
18:         $\overrightarrow{c} \leftarrow$ Vectorized_FindLeafIndex($\overrightarrow{\texttt{leafindex}[T_h]}$)
19:         $\overrightarrow{s} \leftarrow \overrightarrow{s} + \overleftarrow{w_h} \cdot \texttt{leafvalue}[T_h][\overrightarrow{c}]$
20:     return $\overrightarrow{s}$
```

RapidScorer

- http://ai.stanford.edu/~wzou/kdd_rapidscorer.pdf
- RapidScorer: Fast Tree Ensemble Evaluation by Maximizing Compactness in Data Level Parallelization

AdaQS

This article extends the work of quickscorer by proposing a novel adaptive algorithm (i.e., AdaQS) for sparse data and regression trees with default directions as trained by XGBoost.

For each tree node with default direction going right, we adaptively swap its left child and right child. The swap operation is to ensure every default direction going left, thus the absent features of sparse data lead to no false node.

However, the swap has a side effect that changes the Boolean condition from '<' (less than) operation to '>=' (more than or equal to) operation. To preserve the efficiency of quickscorer's search strategy we transform the regression trees into two separate suites of flat structures. One corresponds to the tree nodes with '>' operation and the other corresponds to the tree nodes with '<=' operation. When a sparse instance queries the score, we search in both the two suites and integrate the results.

<img src="https://pic1.zhimg.com/80/v2-a911464197f0eb281ca742c0ea954e98_hd.jpg" width="80%" />

- https://zhuanlan.zhihu.com/p/54932438
- https://github.com/qf6101/adaqs

Gradient Boosting Machine: Beyond Boost Tree

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

<img src="https://cdn.mathpix.com/snip/images/gHJzRtK3cCxjlpKHDepZs7KqKIFe9_y_x7E2042UOrA.original.fullsize.png">

Note that any fitting criterion that estimates conditional expectation (given } $\mathbf{x}$) could in principle be used to estimate the (smoothed) negative gradient (7) at line 4 of Algorithm 1, i.e.,

$$\mathbf{a}_m=\arg\min_{\mathbf{a},\beta}\sum_{i=1}^{N}\left[\tilde{y}_i-\beta h\left(\mathbf{x}_i;\mathbf{a}\right)\right]^2$$

where the model $h(\cdot;\cdot)$ is parameterized by $\mathbf{a}$.
Technically, $h(\cdot;\cdot)$ can be any function - smooth or non-smooth, differentiable or non-differentiable, convex or non-convex-not only the decision tree.

Least-square Regression	**M-regression**
<img src="https://cdn.mathpix.com/snip/images/ufROD3iCdX_-wh7MhVgdKsdB6tdb4lLcukdAirZzmiY.original.fullsize.png">	<img src="https://cdn.mathpix.com/snip/images/BPg0lCcHQNq9HIFI

<img src="https://cdn.mathpix.com/snip/images/lpBjBOiYuKK_kq22TO-HXiDjd2jm_cocCM_snzz-hf4.original.fullsize.png">
<img src="https://cdn.mathpix.com/snip/images/EkdmqKT9xvrdi5whxIBSd7rMTYuUVp9TwmmCOxPA2Ak.original.fullsize.png">

Another improvment of this framework is to find the `profitable diretion` instead of the negative gradients $\tilde{y}_i$.

- Greedy Function Approximation: A Gradient Boosting Machine
- Gradient Boosting Machines in UC Business Analytics R Programming Guide
- Start With Gradient Boosting, Results from Comparing 13 Algorithms on 165 Datasets
- A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning
- Gradient Boosting Algorithm – Working and Improvements
- BOOSTING ALGORITHMS: REGULARIZATION, PREDICTION AND MODEL FITTING

`AdaBoost` is related with so-called exponential loss $\exp(-y_i p(x_i))$ where $x_i\in\mathbb{R}^p, y_i\in\{-1,+1\}, p(\cdot)$ is the input features, labels and prediction function, respectively.
And the weight is update via the following formula:

$$w_i\leftarrow w_i\exp[-y_ip(x_i)], i=1,2,\dots,N.$$

The gradient-boosting algorithm is general in that it only requires the analyst specify a `loss function` and its `gradient`.
When the labels are multivariate, Alex Rogozhnikova et al define a more
general expression of the AdaBoost criteria

$$w_i\leftarrow w_i\exp[-y_i\sum_j a_{ij}p(x_j)], i=1,2,\dots,N,$$

where a_{ij} are the elements of some square matrix A. For the case where A is the identity matrix, the AdaBoost weighting procedure is recovered. Other choices of A will induce `non-local effects`, e.g., consider the sparse matrix A_{knn} given by

$$a_{ij}^{knn} = \begin{cases} 1, & j \in knn(i); \text{ events } i \text{ and } j \text{ belong to the same class} \\ 0, & \text{otherwise.} \end{cases}$$

- New approaches for boosting to uniformity
- uBoost: A boosting method for producing uniform selection efficiencies from multivariate classifiers

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

- Complete Machine Learning Guide to Parameter Tuning in Gradient Boosting (GBM) in Python
- https://bradleyboehmke.github.io/HOML/gbm.html
- Leveraging k-NN for generic classification boosting
- Constructing Boosting Algorithms from SVMs: an Application to One-Class Classification

<img src="https://cdn.mathpix.com/snip/images/nroGBasdD7HoKNgfFquOqY2U5D006PoqfS699AGw_zA.original.fullsize.png">
<img src="https://cdn.mathpix.com/snip/images/c2HBE74ZdlTSjMZHToap4mv82cKTqZpTWZch-LL4DAc.original.fullsize.png">

- Boosting algorithms as gradient descent

Optimization and Boosting

Gradient descent methods, as `numerical optimization methods`, update the values of the parameters at each iteration while the size of parameter is fixed so that the complexity of the model is limited.
In the end, these methods output some optimal or sub-optimal parameters of the model θ_T where

$$\theta_T = \theta_0 - \sum_{t=1}^{T-1} \rho_t \nabla_\theta L \mid_{\theta=\theta_{t-1}}, \quad \theta_t = \theta_{t-1} - \rho_t \nabla_\theta L \mid_{\theta=\theta_{t-1}}.$$

The basic idea of gradient descent methods is to find

$$\theta_t = \arg\min_{\alpha\in\mathbb{R}} L(f(\theta_{t-1} + \alpha\Delta)), \Delta \in span(\nabla_\theta L \mid_{\theta=\theta_{t-1}}).$$

So that $L(f(\theta_t)) \leq L(f(\theta_{t-1}))$. In some sense, it requires the model is expressive enough to solve the problems or the size of the parameter is large.

Gradient boost methods, as `boost methods`, are in the additive training form and the size of the models increases after each iteration so that the model complexity grows. In the end, these methods output some optimal or sub-optimal models F_T where

$$F_T = f_0 + \sum_{t=1}^{T}\rho_t f_t, \quad f_t \approx -\nabla_F L \mid_{F=F_{t-1}}, F_{t-1} = \sum_{i=0}^{t-1} f_i.$$

The basic idea of gradient boosts methods is to find

$$f_t = \arg\min_{f\in\mathcal{F}} L(F_{t-1} + f).$$

so that $L(F_t)\leq L(F_{t-1})$. In some sense, it requires the model f is easy to construct.

- Greedy Function Approximation: A Gradient Boosting Machine
- OPTIMIZATION BY GRADIENT BOOSTING
- boosting as optimization@metacademy
- Boosting, Convex Optimization, and Information Geometry
- Generalized Boosting Algorithms for Convex Optimization
- Survey of Boosting from an Optimization Perspective

However, there are 2 drawbacks:

- The loss function $L(\cdot,\cdot)$ may be non-smooth as in numercial optimization problems, whcih requires the non-smooth methods extend to be boost methods.
- The nengative gradient diretions may not the steepest direction. `xGBoost` takes the advantages of the Newton's methods. It may bring some superise when ocnsidering more fast or accelerated methods.

- CGBoost: Conjugate Gradient in Function Space
- TaylorBoost: First and Second Order Boosting Algorithms
- Historical GBM – Momentum
- BrownBoost
- Fully Corrective Boosting with Arbitrary Loss and Regularization
- http://www.work.caltech.edu/
- http://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote19.html

To be more general, how to connect the numerical optimization methods such as fixed pointed iteration methods and the boosting algorithms?
Is it possible to combine $\boxed{\text{Anderson Acceleration}}$ and $\boxed{\text{Gradinet Boosting}}$?

<img src="https://cdn.mathpix.com/snip/images/x2SuuD7mVNaXTVizaQ6l7IY6yrQpVuj1TeZCqOriwqk.original.fullsize.png">

<img src="https://cdn.mathpix.com/snip/images/wwnWSNc1Dz8Z2s58qGidk3BNN_SRAZXYhmAyiMmOrh4.original.fullsize.png">

Boosting	**Optimization**

Boosting	Optimization
Decision Tree	Coordinate-wise Optimization
AdaBoost	???
Stochastic Gradient Boost	Stochastic Gradient Descent
Gradient Boost	Gradient Descent
???	Accelerated Gradient Methods
???	Mirror Gradient Methods
???	Proximal Gradient Methods
???	ADMM

AdaBoost stepwise minimizes a function

$$L(G_t)=\sum_{n=1}^{N}\exp(-\mathrm{y}_n G_t(x_n))$$

The gradient of $L(G_t)$ gives the example weights used for AdaBoost:

$$\frac{\partial L(G_t)}{\partial G_t(x_n)}=-\mathrm{y}_n\exp(-\mathrm{y}_n G_t(x_n)).$$

Compared with `entropic descent method`, in each iteration of AdaBoost:

$$w_i\leftarrow w_i\exp[-\alpha_t(\mathrm{y}_i G_t(x_i))]>0, i=1,2,\dots,N,\sum_{n=1}^{N}w_n=1.$$

$Note$: given the input feature x_i, the label y_i is a fixed constant and the model is modified with the training data set and the distribution (D,w) i.e., $\{(x_i,y_i,w_i)\mid i=1,2,\cdots,N\}$.

We give a unified account of boosting and logistic regression in which each learning problem is cast in terms of optimization of Bregman distances. The striking similarity of the two problems in this framework allows us to design and analyze algorithms for both simultaneously, and to easily adapt algorithms designed for one problem to the other. For both problems, we give new algorithms and explain their potential advantages over existing methods. These algorithms are iterative and can be divided into two types based on whether the parameters are updated sequentially (one at a time) or in parallel (all at once). We also describe a parameterized family of algorithms that includes both a sequential- and a parallel-update algorithm as special cases, thus showing how the sequential and parallel approaches can themselves be unified. For all of the algorithms, we give convergence proofs using a general formalization of the auxiliary-function proof technique. As one of our sequential-update algorithms is equivalent to AdaBoost, this provides the first general proof of convergence for AdaBoost. We show that all of our algorithms generalize easily to the multiclass case, and we contrast the new algorithms with the iterative scaling algorithm. We conclude with a few experimental results with synthetic data that highlight the behavior of the old and newly proposed algorithms in different settings.

- Boost: Foundations and Algorithms
- 机器学习算法中GBDT与Adaboost的区别与联系是什么？
- Boosting as Entropy Projection
- Logistic Regression, AdaBoost and Bregman Distances

Accelerated Gradient Boosting

The difficulty in accelerating GBM lies in the fact that weak (inexact) learners are commonly used, and therefore the errors can accumulate in the momentum term. To overcome it, we design a "corrected pseudo residual" and fit best weak learner to this corrected pseudo residual, in order to perform the z-update. Thus, we are able to derive novel computational guarantees for AGBM. This is the first GBM type of algorithm with theoretically-justified accelerated convergence rate.

- Initialize $f_0(x)=g_0(x)=0$;
- For $m=1,2,\dots,M$:
    - Compute a linear combination of f and h: $g^{m}(x)=(1-\theta_m)f^{m}(x)+\theta_m h^{m}(x)$ and $\theta_m=\frac{2}{m+2}$
    - For $i=1,2,\dots,n$ compute

$$r_{i,m}=-[\frac{\partial L(\mathrm{y}_i, g^{m}(x_i))}{\partial g^{m}(x_i)}].$$

    - Find the best weak-learner for pseudo residual:

$$\tau_{m,1}=\arg\min_{\tau\in\mathcal{T}}\sum\nolimits_{i=1}^{n}(r_{i,m}-b_{\tau}(x_i))^2$$

    - Update the model: $f^{m+1}(x)=g^{m}(x)+\eta b_{\tau_{m,1}}$.
    - Update the corrected residual:

$$c_{i,m}=\begin{cases} r_{i,m} & \text{if m=0,}\\ r_{i,m}+\frac{m+1}{m+2}(c_{i,m-1}-b_{\tau_{m,2}}) & \text{otherwise.}\end{cases}$$

    - Find the best weak-learner for the corrected residual: $b_{\tau_{m,2}}=\arg\min_{\tau\in\mathcal{T}}\sum_{i=1}^{n}(c_{i,m}-b_{\tau}(x_i))^2$.
    - Update the momentum model: $h^{m+1}=h^{m}+\frac{\gamma\eta}{\theta_m}b_{\tau_{m,2}}(x)$.
- Output $f^{M}(x)$.

- Accelerated Gradient Boosting

Accelerating Gradient Boosting Machine

Gradient Boosting Machine (GBM) is an extremely powerful supervised learning algorithm that is widely used in practice. GBM routinely features as a leading algorithm in machine learning competitions such as Kaggle and the KDDCup. In this work, we propose Accelerated Gradient Boosting Machine (AGBM) by incorporating Nesterov's acceleration techniques into the design of GBM. The difficulty in accelerating GBM lies in the fact that weak (inexact) learners are commonly used, and therefore the errors can accumulate in the momentum term. To overcome it, we design a "corrected pseudo residual" and fit best weak learner to this corrected pseudo residual, in order to perform the z-update. Thus, we are able to derive novel computational guarantees for AGBM. This is the first GBM type of algorithm with theoretically-justified accelerated convergence rate. Finally we demonstrate with a number of numerical experiments the effectiveness of AGBM over conventional GBM in obtaining a model with good training and/or testing data fidelity.

<img src="https://cdn.mathpix.com/snip/images/K2A8aE0RgnOsjuJSyVmwIHcMEwP9gi6NVZGR6-czP20.original.fullsize.png">

- Accelerating Gradient Boosting Machine

Historical Gradient Boosting Machine

We introduce the Historical Gradient Boosting Machine with the objective of improving
the convergence speed of gradient boosting. Our approach is analyzed from the perspective
of numerical optimization in function space and considers gradients in previous steps, which
have rarely been appreciated by traditional methods. To better exploit the guiding effect
of historical gradient information, we incorporate both the accumulated previous gradients and the current gradient into the computation of descent direction in the function space. By fitting to the descent direction given by our algorithm, the weak learner could enjoy
the advantages of historical gradients that mitigate the greediness of the steepest descent direction. Experimental results show that our approach improves the convergence speed of gradient boosting without significant decrease in accuracy.

<img src="https://cdn.mathpix.com/snip/images/o3JoaeBZKpduHXV72sQJ45OKo4ZftCArb_AAy_FTwHQ.original.fullsize.png">

- Historical Gradient Boosting Machine

Parallel Boosting with Momentum

We describe a new, simplified, and general analysis of a fusion of Nesterov's accelerated gradient with parallel coordinate descent. The resulting algorithm, which we call BOOM, for boosting with momentum, enjoys the merits of both techniques. Namely, BOOM retains the momentum and convergence properties of the accelerated gradient method while taking into account the curvature of the objective function. We describe a distributed implementation of BOOM which is suitable for massive high dimensional datasets. We show experimentally that BOOM is especially effective in large scale learning problems with rare yet informative features.

<img src="https://cdn.mathpix.com/snip/images/exGkrKTVJNhntO3IrjRSqhZY0gLyXPKcHhCUksHsEHc.original.fullsize.png">

- https://ai.google/research/pubs/pub41341

Accelerated proximal boosting

Gradient boosting is a prediction method that iteratively combines weak learners to produce a complex and accurate model. From an optimization point of view, the learning procedure of gradient boosting mimics a gradient descent on a functional variable. This paper proposes to build upon the proximal point algorithm when the empirical risk to minimize is not differentiable. In addition, the novel boosting approach, called accelerated proximal boosting, benefits from Nesterov's acceleration in the same way as gradient boosting [Biau et al., 2018]. Advantages of leveraging proximal methods for boosting are illustrated by numerical experiments on simulated and real-world data. In particular, we exhibit a favorable comparison over gradient boosting regarding convergence rate and prediction accuracy.

<img src="https://cdn.mathpix.com/snip/images/YTfPvnaxJMcdXpBbgdRjwRHC0tdHdxeb4nuqBf_MrHY.original.fullsize.png">

- Accelerated proximal boosting

Translation Optimization Methods to Boost Algorithm

The following steps are the keys to a constructed a decision tree in gardient boost methods:

- For $i = 1, 2, \dots, n$ compute

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

- Fit a regression tree to the targets $r_{i,t}$ giving **terminal regions**

$$R_{j,m}, j = 1, 2, \dots, J_m.$$

Here we compute the gradient of loss function with respective to each prediction $f(x_i)$ and it is why we call it `gradient boost`.
If we fit a regression tree to a subsample of the targets $r_{i,t}$ randomly, it is `stochastic gradient boost`.

All variants of gradient boost methods mainly modify $\boxed{\text{the steps to construct a new decision tree}}$. And it is trained in additive way.

Mirror gradient descent update formulea can be transferred to be

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

then fit a regression tree to the targets $r_{i,t}$.

What is the alternative of gradient descent in order to combine `ADMM` as an operator splitting methods for numerical optimization and `Boosting` such as gradient boosting/extreme gradient boosting?
Can we do leaves splitting and optimization in the same stage?

The core transfer from ADMM to Boost is how to change the original optimization to one linearly constrained convex optimization so that it adjusts to ADMM:

$$\arg\min_{f_t\in\mathcal{F}} \sum_{i=1}^{n} \ell[y_i, \hat{y}_i^{(t-1)} + f_t(x_i)] + \gamma T + \frac{\lambda}{2}\sum_{i=1}^{T} w_i^2 \iff \boxed{???} \quad ?$$

where $f_t(x) = \sum_{i=1}^{T} w_i \mathbb{I}(q(x) = i)$ is a decision tree.

As in `xGboost`, we can write the objective value with the t-th tree as:

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

where $I_j = \{i \mid q(x_i) = j\}$ is the set of indices of data points assigned to the j-th leaf.

Note that the samples output in the same leaf has the same output result, so they share the same gradients and Hessian.
And it is natural to $\boxed{\text{split the leaves as operators}}$.

It seems attractive to me to understand the analogy between
$\boxed{\text{operator splitting in ADMM}}$ and $\boxed{\text{leaves splitting in Decision Tree}}$.

To be simple, we do not take the number of leaves into consideration.
And the objective function is simplied to be

$$f(w)=\sum_{j=1}^{T}f_j(w_j)$$

where $f_j(w_j)=(\sum_{i\in I_j}g_i)w_j+\frac{1}{2}(\sum_{i\in I_j}h_i+\lambda)w_j^2=G_jw_j+\frac{1}{2}(H_j+\lambda)w_j^2$.
Note that the gradients and Hessian are constant as well as λ.
Obviously, it is a `concensus optimziation problem`.

$\boxed{\text{ADMM for consensus optimization}}$
<img src="https://image.slidesharecdn.com/admmslides-110829200511-phpapp02/95/alternating-direction-36-728.jpg?cb=1314648375" width="80%"/>
<img src="https://image.slidesharecdn.com/admmslides-110829200511-phpapp02/95/alternating-direction-37-728.jpg?cb=1314648375" width="80%"/>

The objective function can be regualrized as

$$\sum_{j=1}^{T}f_j(w_j)+\|\theta\|_{\ell_1},\\ \text{s.t. } \theta-w=0$$

which is the convex structure optimization problem with linear constraints.
The augmented Langrage is

$$L_{\beta}(w,\theta,\lambda)=f(w)+\|\theta\|_{\ell_1}-\lambda^T(\theta-w)+\frac{\beta}{2}\|\theta-w\|_2^2.$$

More generally, if the objective function is `not differebtiable`, we can also regard the samples in the same leaf as one operator.
Addtionally, ℓ_1 regularization may take the role as sparse regularization.

Suppose that we obtain the optimal solution of the objective function f via ADMM, i.e., $w^*=\arg\min_w f$, then we can update the tree.

$\boxed{\text{ADMM-Boost}}$

- $w^t=\arg\min_w L_{\beta}(w,\theta^{t-1},\lambda^{t-1})$;
- Construct a decision tree with the parameter w^t;
- $\theta^t=\arg\min_{\theta} L_{\beta}(w^t,\theta,\lambda^{t-1})$;
- Construct a decision tree with the parameter θ^t;
- $\lambda^t=\lambda^{t-1}-\beta(\theta^t-w^t)$.

Leveraging `distributed ADMM`, it is easy to extend to distributed version.

---

* A Duality View of Boosting Algorithms
* New understanding of boosting methods, Chunhua Shen, School of Computer Science, University of Adelaide
* Consensus optimization
* Asynchronous Distributed ADMM for Consensus Optimization

Another interesting question is how to boost the composite/multiplicative models rather than the additive model?
Is it necessary to approxiamte the negative gradient using decision tree?

Deep Gradient Boosting

It shows that each iteration of the backpropagation algorithm can be viewed as fitting a
weak linear regressor to the gradients at the output of each layer, before non-linearity is applied.
We call this approach `deep gradient boosting (DGB)`, since it's effectively a `layer-wise boosting approach` where the typical decision trees are replaced by linear regressors. Under this model, SGD naturally emerges as an extreme case where the network weights are highly regularized, in the L2 norm sense.
In addition, DGB takes into account the correlations between training samples (features), just like regular regression would, when calculating the weight updates while SGD does not.
Intuitively, it makes sense to ignore the correlations between training samples considering that the most difficult
test samples would be the ones that show low correlations with the training set.

This work suggests an alternative explanation for why SGD generalizes so well when training neural networks.
We show that each iteration of the backpropagation algorithm can be viewed as fitting a weak linear regressor to the gradients at the output of each layer, before non-linearity is applied.

The classic backpropagation algorithm minimizes an error function E of a multi-layer neural network using gradient descent and the chain rule.
The resulting weight updates at a given layer are

$$\Delta w_{ij} = x_i y_j$$

where x_i is the output from the previous layer at node i while $y_j = \frac{\partial E}{\partial \mathrm{net}_j}$ is the derivative with respect to the input at node j calculated using the chain rule from the last layer to the current one.
We can interpret y_i as a pseudo-residual and infer the weight updates $\Delta w_{ij} = v_{ij}$ such that $\sum_i x_i v_{i,j} = y_j, \forall j$.

For the extreme case of a single sample update we need to use L2 regularization which leads to the following optimization problem:

$$\begin{array}{ll} \underset{v_{i,j}}{\text{minimize}} & \frac{1}{2}\sum_i v_{i,j}^2 \\ \text{subject to} & \sum_i x_i v_{i,j} = y_j, \quad \forall j \end{array}$$

* https://arxiv.org/pdf/1907.12608.pdf
* The Nuclear Engineering, Statistical Forecasting, Data Science, and Oxford Commas of Ryan McClarren

The Generic Leveraging Algorithm

Let us assume the loss function $G(f, D)$ has the following additive form

$$G(f, D)=\sum_{n=1}^{N} g(f(x_n), y_n),$$

and we would like to solve the optimization problem

$$\min_{f\in\mathcal{F}} G(f, D)=\min_{w}\sum_{n=1}^{N} g(f_w(x_n), y_n).$$

And $g'(f_w(x_n), y_n))=\frac{\partial g(f_w(x_n), y_n)}{\partial f_w(x_n)}$ for $n=1,2,\cdots, N$.

`Leveraging methods` are designed under a subsampling framework, in which one samples a small proportion of the data (subsample) from the full sample, and then performs intended computations for the full sample using the small subsample as a surrogate. The key of the success of the leveraging methods is to construct nonuniform sampling probabilities so that influential data points are sampled with high probabilities

$\boxed{\text{Leveraging = Boosting without PAC Boosting property}}$.

- Input $D=\{(x_i,y_i)\}_{i=1}^{N}$;Loss function $G:\mathbb{R}^n\to\mathbb{R}$.
- Initialize the observation weights $f_o=0, d_n^1=g'(f_0(x_n), y_n), n=1,2,\dots, N$.
- For $t=1,2,\dots, T$:
    - Train classifier on $\{D, \mathbf{d}^t\}$ and obtain hypothesis $h_t:\mathbb{X}\to\mathbb{Y}$
    - Set $\alpha_t=\arg\min_{\alpha\in\mathbb{R}} G[f_t+\alpha h_t]$
    - Update $f_{t+1}=f_t+\alpha_t h_t$ and $d_n^{t+1}=g'(f_{t+1}(x_n), y_n), n=1,2,\dots, N$
- Output f_T.

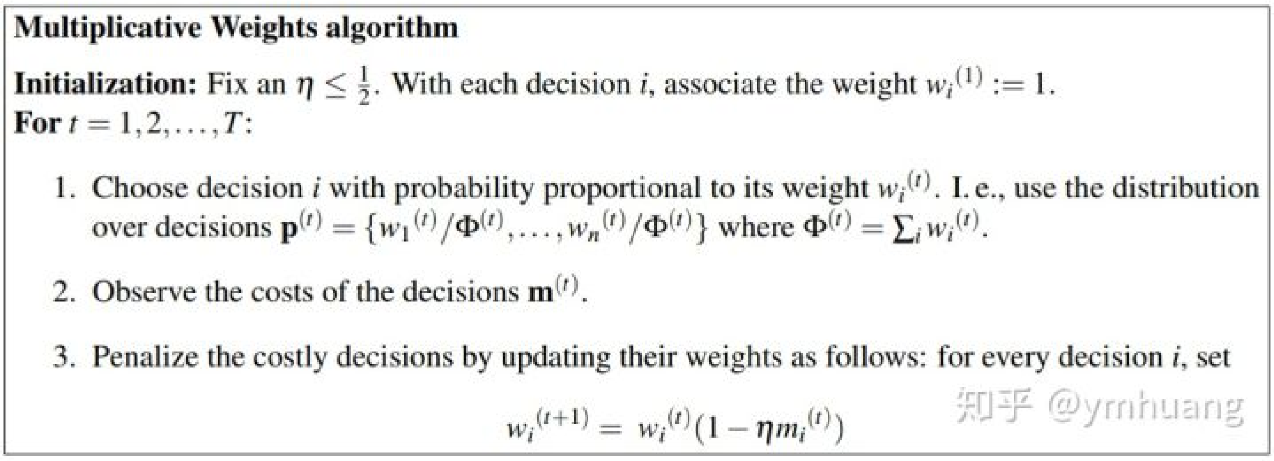
Here $\mathbf{d}^t=(d_1^t, d_2^t,\cdots, d_N^t)$ for $t=1,2,\cdots, T$.

- An Introduction to Boosting and Leveraging
- FACE RECOGNITION HOMEPAGE
- Leveraging for big data regression
- A Statistical Perspective on Algorithmic Leveraging
- http://homepages.math.uic.edu/~minyang/BD.htm
- Some Theory for Generalized Boosting Algorithms

Matrix Multiplicative Weight Algorithms

`Matrix Multiplicative Weight` can be considered as an ensemble method of optimization methods.
The name “multiplicative weights” comes from how we implement the last step: if the weight of the chosen object at step t is w_t before the event, and G represents how well the object did in the event, then we’ll update the weight according to the rule:

$$w_{t+1} = w_t(1+G).$$



Multiplicative Weights algorithm

Initialization: Fix an $\eta \leq \frac{1}{2}$. With each decision i, associate the weight $w_i{}^{(1)} := 1$.
For $t = 1, 2, \ldots, T$:

1. Choose decision i with probability proportional to its weight $w_i{}^{(t)}$. I. e., use the distribution over decisions $\mathbf{p}^{(t)} = \{w_1{}^{(t)}/\Phi^{(t)}, \ldots, w_n{}^{(t)}/\Phi^{(t)}\}$ where $\Phi^{(t)} = \sum_i w_i{}^{(t)}$.
2. Observe the costs of the decisions $\mathbf{m}^{(t)}$.
3. Penalize the costly decisions by updating their weights as follows: for every decision i, set

$$w_i{}^{(t+1)} = w_i{}^{(t)}(1 - \eta m_i{}^{(t)})$$

Matrix Multiplicative Weight

Jeremy wrote a blog on this topic:

> *In general we have some set X of objects and some set Y of “event outcomes” which can be completely independent. If these sets are finite, we can write down a table M whose rows are objects, whose columns are outcomes, and whose i, j entry $M(i, j)$ is the reward produced by object x_i when the outcome is y_j. We will also write this as $M(x, y)$ for object x and outcome y. The only assumption we’ll make on the rewards is that the values $M(x, y)$ are bounded by some small constant B (by small I mean B should not require exponentially many bits to write down as compared to the size of X). In symbols, $M(x, y) \in [0, B]$. There are minor modifications you can make to the algorithm if you want negative rewards, but for simplicity we will leave that out. Note the table M just exists for analysis, and the algorithm does not know its values. Moreover, while the values in M are static, the choice of outcome y for a given round may be nondeterministic.*
>
> *The `MWUA` algorithm randomly chooses an object $x \in X$ in every round, observing the outcome $y \in Y$, and collecting the reward $M(x, y)$ (or losing it as a penalty). The guarantee of the MWUA theorem is that the expected sum of rewards/penalties of MWUA is not much worse than if one had picked the best object (in hindsight) every single round.*

Theorem (from Arora et al): The cumulative reward of the MWUA algorithm is, up to constant multiplicative factors, at least the cumulative reward of the best object minus $\log(n)$, where n is the number of objects.

- The Reasonable Effectiveness of the Multiplicative Weights Update Algorithm
- Matrix Multiplicative Weight （1）
- Matrix Multiplicative Weight （2）
- Matrix Multiplicative Weight （3）
- The Multiplicative Weights Update framework
- The Multiplicative Weights Update Method: a Meta Algorithm and Applications
- Nonnegative matrix factorization with Lee and Seung's multiplicative update rule
- A Combinatorial, Primal-Dual approach to Semidefinite Programs
- Milosh Drezgich, Shankar Sastry. "Matrix Multiplicative Weights and Non-Zero Sum Games".
- The Matrix Multiplicative Weights Algorithm for Domain Adaptation by David Alvarez Melis
- https://lcbb.epfl.ch/algs16/notes/notes-04-14.pdf

Application

News lists some news on CatBoost.
See XGBoost Resources Page for a complete list of use cases of XGBoost, including machine learning challenge winning solutions, data science tutorials and industry adoptions.

- 拍拍贷教你如何用GBDT做评分卡
- LambdaMART 不太简短之介绍
- https://catboost.ai/news
- Finding Influential Training Samples for Gradient Boosted Decision Trees
- Parallel Boosted Regression Trees for Web Search Ranking

- Efficient, reliable and fast high-level triggering using a bonsai boosted decision tree
- CERN boosts its search for antimatter with Yandex’s MatrixNet search engine tech
- MatrixNet as a specific Boosted Decision Tree algorithm which is available as a service
- Bagging and Boosting statistical machine translation systems
- A Novel, Gradient Boosting Framework for Sentiment Analysis in Languages where NLP Resources Are Not Plentiful: A Case Study for Modern Greek
- EGBMMDA: Extreme Gradient Boosting Machine for MiRNA-Disease Association prediction
- Awesome gradient boosting

Selective Ensemble

Selective ensemble naturally bears two goals simultaneously, i.e., maximizing the generalization performance and minimizing the number of learners. When pushing to the limit, the two goals are conflicting, as overly fewer individual learners lead to poor performance. To achieve both good performance and a small ensemble size, previous selective ensemble approaches solve some objectives that mix the two goals.

- Selective Ensemble of Decision Trees
- Growing and Pruning Selective Ensemble Regression over Drifting Data Stream
- Selective Ensemble under Regularization Framework

* Selecting a representative decision tree from an ensemble of decision-tree models for fast big data classification

<img src="https://media.springernature.com/full/springer-static/image/art%3A10.1186%2Fs40537-019-0186-3/MediaObjects/40537_2019_186_Figa_HTML.png" width="70%"/>

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.

<img src="https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier_files/stackingclassification_overview.png" width="65%" />

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

When a lot of different models are applied to a data simultaneously then such a method of meta-ensemble modeling is known as Stacking. Here, there is no single function, rather we have meta-level where a function is used to combine the outputs of different functions. Thus the information from various models is combined to come up with a unique model. This is among the most advanced form of data modeling used commonly in data hackathons and other online competitions where maximum accuracy is required. Stacking models can have multiple levels and can be made very complex by using various combinations of features and algorithms. There are many forms of Stacking method and in this blog post, a stacking method known as blending has been explored.

* Stacked Regression
* Stacked Ensemble Models for Improved Prediction Accuracy
* https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/
* Stacked Generalization (Stacking)

<img src="https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier_files/stacking_algorithm.png" width="80%"/>

* 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)
* 我爱机器学习:集成学习（一）模型融合与Bagging
* https://github.com/ikki407/stacking
* Python package for stacking (machine learning technique)
* Signal Processing and Pattern Recognition Laboratory

- https://blog.csdn.net/mrlevo520/article/details/78161590
- http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/stacked-ensembles.html#id3
- https://web.njit.edu/~avp38/projects/multi_projects/ensemble.html
- http://ml-ensemble.com/
- https://rasbt.github.io/mlxtend/
- https://shodhganga.inflibnet.ac.in/bitstream/10603/7989/15/15_chapter 6.pdf

Issues in Stacked Generalization

Stacked generalization is a general method of using a high-level model to combine lower-level models to achieve greater predictive accuracy. In this paper we address two crucial issues which have been considered to be a `black art' in classification tasks ever since the introduction of stacked generalization in 1992 by Wolpert: the type of generalizer that is suitable to derive the higher-level model, and the kind of attributes that should be used as its input. We find that best results are obtained when the higher-level model combines the confidence (and not just the predictions) of the lower-level ones. We demonstrate the effectiveness of stacked generalization for combining three different types of learning algorithms for classification tasks. We also compare the performance of stacked generalization with majority vote and published results of arcing and bagging.

- Issues in Stacked Generalization

Learning to Efficiently Rank with Cascades

Our core idea is to consider the ranking problem as a "cascade", where ranking is broken into a finite number of distinct stages. Each stage considers successively richer and more complex features, but over successively smaller candidate document sets. The intuition is that although complex features are more time-consuming to compute, the additional overhead is offset by examining fewer documents. In other words, the cascade model views retrieval as a multi-stage progressive refinement problem, where each stage balances the cost of exploiting various features with the potential gain in terms of better results. We have explored this notion in the context of linear models and tree-based models.

- Learning to Efficiently Rank with Cascades, University of Maryland

Theory

- Spatial Pyramids and Two-layer Stacking SVM Classifiers for Image Categorization: A Comparative Study
- Cascaded classifiers and stacking methods for classification of pulmonary nodule characteristics
- Stacking与神经网络 - 微调的文章 - 知乎
- Blending and deep learning

Linear Blending

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.

<img title = weighted-unweighted src=https://blogs.sas.com/content/subconsciousmusings/files/2017/05/weighted-unweighted.png width=80%/>

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.

<img src="http://www.chioka.in/wp-content/uploads/2013/09/stacking.png" width="80%" />

- Mixture of Experts
- Hierarchical Mixture of Experts and the EM Algorithms

$\boxed{\text{partition} + \text{stacking}}$: Different data activates different algorithms.

Stacked AutoEncoder

<img scr="https://uploads.toptal.io/blog/image/335/toptal-blog-image-1395721542588.png" width="80%"/>

- http://deeplearning.net/tutorial/SdA.html
- http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders
- http://akosiorek.github.io/ml/2019/06/23/stacked_capsule_autoencoders.html
- https://github.com/cazala/synaptic

Stacked GAN

https://khanrc.tistory.com/entry/Deep-Learning-Tutorial

- https://github.com/hanzhanggit/StackGAN
- https://arxiv.org/abs/1612.03242
- https://arxiv.org/abs/1710.10916

Deep Forest

In this paper, we propose gcForest, a decision tree ensemble approach with performance highly competitive to deep neural networks.

<img title="Deep Forest" src="https://raw.githubusercontent.com/DataXujing/Cos_pic/master/pic2.png" width="80%" />
<img src="https://raw.githubusercontent.com/DataXujing/Cos_pic/master/pic3.png" width="80%" />

- Deep forest
- https://github.com/kingfengji/gcForest
- 周志华团队和蚂蚁金服合作：用分布式深度森林算法检测套现欺诈
- Multi-Layered Gradient Boosting Decision Trees
- Deep Boosting: Layered Feature Mining for General Image Classification
- gcForest 算法原理及 Python 与 R 实现

- https://www.wikiwand.com/en/Ensemble_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
- http://users.rowan.edu/~polikar/publications.html
- CAREER: An Ensemble of Classifiers Based Approach for Incremental Learning