Introduction to Big Data Analysis
Ensemble Methods

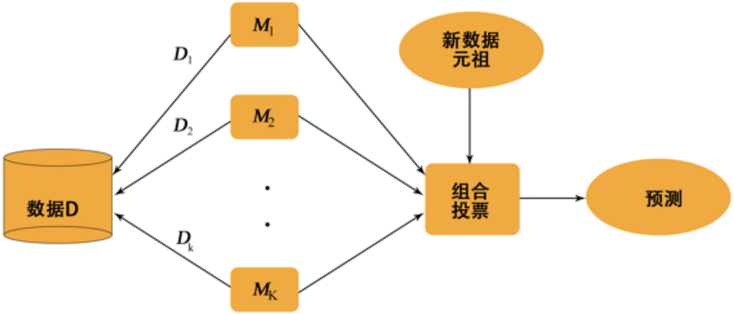
Zhen Zhang

Southern University of Science and Technology

Outlines

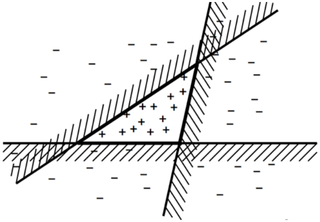
Ensemble Methods

- Wisdom of Crowds (“三个臭皮匠，顶个诸葛亮”)
- Multiple weak learners (base learners, may be heterogenous) can improve learning performance



Why it can improve the performance

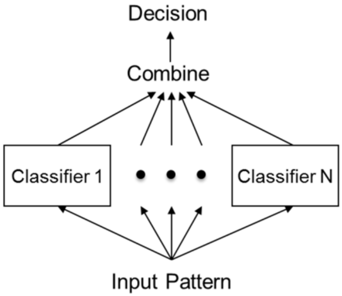
- More flexible to interpret
    - Single linear classifier (perceptron) does not work
    - Try multiple classifiers



- Reduce misclassfication rate
    - Misclassfication rate of single classifier is p
    - Choose N classifiers, same but independent, voting
    - Error rate of majority vote $= \sum_{k>N/2} \binom{N}{k} p^k (1-p)^{N-k}$
    - When $N = 5, p = 0.1$, Error rate < 0.01

Two commonly used ensemble methods

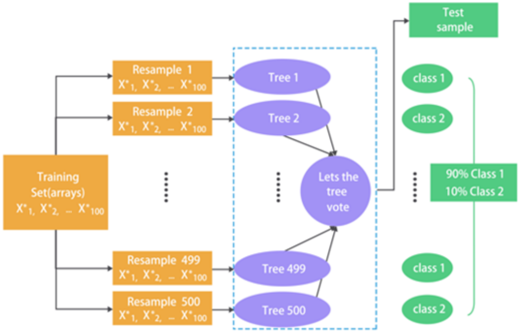
- Bagging
  - Random sampling : generating independent models, and averaging for regressions (making majority vote for classifications)
  - Reducing variances
  - Example : Random forests
- Boosting
  - Sequential training : training the subsequent models based on the errors of previous models
  - Reducing bias
  - Examples : AdaBoost and GBDT



Outlines

Bagging

- Bagging is short for bootstrap aggregation
- Bagging generates a committee of predictors and combine them in a certain manner to the final model
- Single predictor suffers from instability, while bagging could improve the stability by majority vote (classification) or averaging (regression) over all single predictors


Training Set(arrays) $X^*_1, X^*_2, \ldots X^*_{100}$
Resample 1 $X^*_1, X^*_2, \ldots X^*_{100}$
Resample 2 $X^*_1, X^*_2, \ldots X^*_{100}$
Resample 499 $X^*_1, X^*_2, \ldots X^*_{100}$
Resample 500 $X^*_1, X^*_2, \ldots X^*_{100}$
Tree 1
Tree 2
Tree 499
Tree 500
Lets the tree vote
Test sample
class 1
class 2
90% Class 1
10% Class 2
class 1
class 2

Sampling

- Given a dataset D of n samples, at the iteration $m = 1, \ldots, M$, the training set D_m is obtained by sampling from D with replacement. Then D_m is used to construct classifier $\hat{f}_m(x)$.
- Sampling with replacement : some samples in D may be missing in D_m, while some other samples may occur more than once
- On average, 63.2% of the samples in D could be selected into D_m. In fact, for each sample, the probability that it is not selected in one round is $1 - \frac{1}{n}$. Then it is not selected in all n rounds with probability $\lim_{n \to \infty} (1 - \frac{1}{n})^n = 0.368$.

Algorithm

- Input : training set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
- Output : additive model $\hat{f}_{bag}(x)$

1. For $m = 1$ to M :
   1.1 Sample from D with replacement to obtain D_m
   1.2 Train a model $\hat{f}_m(x)$ from the dataset D_m : for classification, $\hat{f}_m(x)$ returns a K-class 0-1 vector e_k ; for regression, it is just a value
2. Compute bagging estimate $\hat{f}_{bag}(x) = \frac{1}{M} \sum_{m=1}^{M} \hat{f}_m(x)$: for classification, make majority vote $\hat{G}_{bag}(x) = \arg\max_k \hat{f}_k(x)$; for regression, just return the average value

Variance Reduction

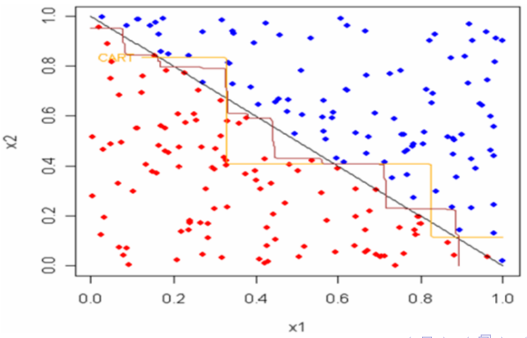
- In bagging, we use the same model to train different sample set in each iteration ; assume the models $\{\hat{f}_m(x)\}_{m=1}^{M}$ have the same variance $\sigma^2(x)$, while the correlation of each pair is $\rho(x)$
- Then the variance of the final model is :

$$\begin{aligned} \mathrm{Var}(\hat{f}_{bag}(x)) &= \frac{1}{M^2}\Big(\sum_{m=1}^{M}\mathrm{Var}(\hat{f}_m(x)) + \sum_{t\neq m}\mathrm{Cov}(\hat{f}_t(x)\hat{f}_m(x))\Big) \\ &= \rho(x)\sigma^2(x) + \frac{1-\rho(x)}{M}\sigma^2(x) \end{aligned}$$

- As $M \to \infty$, $\mathrm{Var}(\hat{f}_{bag}(x)) \to \rho(x)\sigma^2(x)$. This usually reduces the variance.
- If $\rho(x) = 0$, the variance could approach zero
- The random sampling in bagging is to reduce the correlation $\rho(x)$, i.e., make the sub-predictors as independent as possible

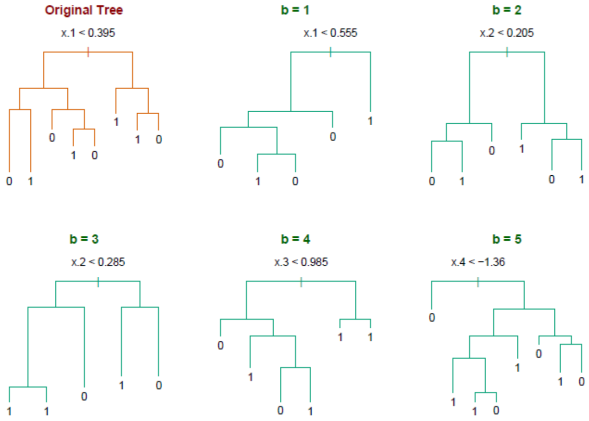
Limitations of Decision Tree

- Stuck at local optimum : The greedy algorithm makes it stop at the local optimum, as it seeks the maximal information gain in each tree split
- Decision boundary : Use one feature in each split, the decision boundary is parallel to the coordinate axes
- Bad interpretability and instability



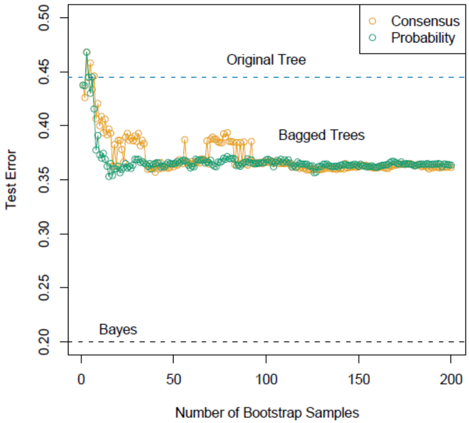
Bagging Tree vs. Single Tree

- The true split point is $x_1 = 0.5$ and the Bayes error is 0.2 ($P(Y = 1|x_1 \leqslant 0.5) = 0.2$ and $P(Y = 1|x_1 > 0.5) = 0.8$). # of features is p=5 and # of samples is n=2000.



Error of Bagging Trees

- Bagging trees reduce the test error by reducing variance, leaving bias unchanged
- Consensus : majority vote ; Probability : average the probability predicted by sub-models



Random Forest

- Random Forest further reduces the variance by adding independency to the committee of decision trees
- This is achieved by introducing more randomness.
- Three level of randomness :
  - Sampling on the training data with replacement
  - Select features at random
  - Select split point at random
- No pruning is needed.
- Example : RF consisting of 3 independent trees, each with an error rate of 40%. Then the probability that more than one tree misclassify the samples is $0.4^3 + 3 * 0.4^2 * (1 - 0.4) = 0.352$

Random Forest Algorithm

- Input : training set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
- Output : additive model $\hat{f}_{rf}(x)$

1. For $m = 1$ to M :
   1.1 Sample from D with replacement to obtain D_m
   1.2 Grow a random-forest tree T_m to the dataset D_m : by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached
      1.2.1 Select q features at random from the p features
      1.2.2 Pick the best feature/split-point among the q
      1.2.3 Split the node into two daughter nodes
2. Output the ensemble of trees $\{T_m\}_{m=1}^{M}$: for regression, $\hat{f}_{rf}(x) = \frac{1}{M}\sum_{m=1}^{M} T_m(x)$: for classification, make majority vote

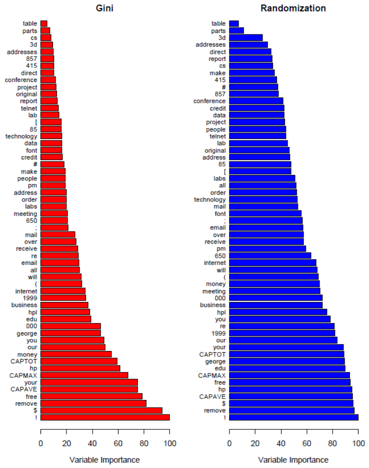
- Small value of q increases the independency of trees ; empirically, $q = \log_2 p + 1$

Model Evaluation

- Margins : The difference between the percentage of decision trees that correctly classify the samples and the percentage of trees misclassifying the samples
- Out-of-bag (OOB) erros : The observation is called out-of-bag sample to some trees if it is not sampled for those trees. Denote the training set in the m-th sampling by D_m. OOB error is computed as :
  1. For each observation (x_i, y_i), find the trees which treat it as OOB sample : $\{\hat{T}_m(\mathbf{x}) : (\mathbf{x}_i, y_i) \notin D_m\}$
  2. Use those trees to classify this observation and make majority vote as the label of this observation :
     $$\hat{f}_{oob}(\mathbf{x}_i) = \arg\max_{y \in \mathcal{Y}} \sum_{m=1}^{M} \mathrm{I}(\hat{f}_m(\mathbf{x}_i) = y)\mathrm{I}(\mathbf{x}_i \notin D_m)$$
  3. Compute the number of misclassified samples, and take the ratio of this number to the total number of samples as OOB error : $Err_{oob} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{I}(\hat{f}_{oob}(\mathbf{x}_i) \neq y_i)$

Feature Importance

- Using split criteria
  - The improvement in the split-criterion as feature importance
  - It is accumulated over all the trees for each variable
- Using OOB randomization
  - Randomly permute the values of each feature in the OOB samples, and compute the prediction accuracy
  - The decrease in accuracy as a result of this permutation is averaged over all trees as feature importance



Proximity Matrix

- For every tree, any pair of OOB observations sharing a terminal node has their proximity increased by one
- The more two OOB samples sharing the same node, the more similar the two samples are
- Proximity matrix could be used to define distance between two samples

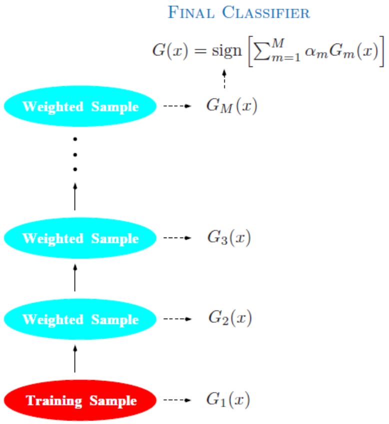
Pros and Cons

- Where it is good
  - Bagging or random forest (RF) work for models with high variance but low bias
  - Better for nonlinear estimators
  - RF works for very high-dimensional data, and no need to do feature selection as RF gives the feature importance
  - Easy to do parallel computing
- Disadvantage
  - Overfitting when the samples are large-sized with great noise, or when the dimension of data is low
  - Slow computing performance comparing to single tree
  - Hard to interpret

Outlines

Boosting

- Boosting : combines the outputs of many "weak" classifiers to produce a powerful "committee"
- Weak classifier : error rate < 0.5 (random guessing)
- Sequentially apply the weak classifiers to the repeatedly modified data, emphasizing the misclassified samples
- Combine weak classifiers through a weighted majority vote or averaging to produce the final prediction



Boosting Fits an Additive Model

- Additive model : $f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$
- Possible choices for basis function $b(x; \gamma)$:
  - Neural networks : $\sigma(\gamma_0 + \gamma_1^T x)$, where $\sigma(t) = 1/(1 + e^{-t})$
  - Wavelets
  - Cubic spline basis
  - Trees
  - Eigenfunctions in reproducing kernel Hilbert space (RKHS)
- Parameter fitting : $\min_{\{\beta_m, \gamma_m\}} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m))$
- Loss function : squared error $L(y, f(x)) = (y - f(x))^2$ or likelihood-based loss

Forward Stagewise Additive Modeling

- Input : training set $D = \{(x_1, y_1), \dots, (x_N, y_N)\}$
- Output : additive model $f_M(x)$

```
1. Initialize $f_0(x) = 0$
2. For $m = 1$ to $M$ :
    2.1 Compute $(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$
    2.2 Update $f_m(x) = f_{m-1}(x) + \beta_m b(x_i; \gamma_m)$
```

- Squared error loss : in step 2.1,
$$L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) = (\underbrace{y_i - f_{m-1}(x_i)}_{\text{residual}} - \beta b(x_i; \gamma))^2$$

Exponential Loss and AdaBoost

- Exponential loss : $L(y, f(x)) = \exp(-yf(x))$
- Classifier as basis function : $b(x;\gamma) = G(x) \in \{-1, 1\}$
- Let $w_i^{(m)} = \exp(-y_i f_{m-1}(x_i))$, then step 2.1 turns to be :

$$
\begin{aligned}
(\beta_m, G_m) &= \arg\min_{\beta, G} \sum_{i=1}^{n} w_i^{(m)} \exp(-\beta y_i G(x_i)) \\
&= \arg\min_{\beta, G} \Big[\sum_{y_i \neq G(x_i)} w_i^{(m)} (e^{\beta} - e^{-\beta}) + e^{-\beta} \sum_{i=1}^{n} w_i^{(m)} \Big]
\end{aligned}
$$

- $G_m = \arg\min_{G} \sum_{i=1}^{n} w_i^{(m)} I(y_i \neq G(x_i))$.
- $\beta_m = \arg\min_{\beta} \left[\epsilon_m (e^{\beta} - e^{-\beta}) + e^{-\beta}\right] = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$ where

  $\epsilon_m = (\sum_{i=1}^{n} w_i^{(m)} I(y_i \neq G(x_i))) / \sum_{i=1}^{n} w_i^{(m)}$ is weighted error rate

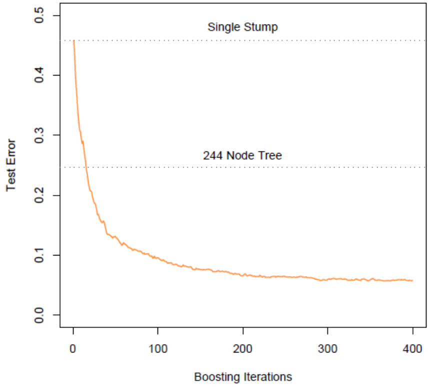
AdaBoost Algorithm

- Input : training set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, loss function $L(y, f(x))$
- Output : Weighted classifier $G(x)$

1. Initialize $w_i = 1/N$, $i = 1, \ldots, N$
2. For $m = 1$ to M :
   2.1 Fit a classifier $G_m(x)$ to the training data D with weight $\{w_i\}$
   2.2 Compute the error $\epsilon_m = (\sum_{i=1}^{n} w_i^{(m)} I(y_i \neq G(x_i))) / \sum_{i=1}^{n} w_i^{(m)}$
   2.3 Compute $\alpha_m = \log \frac{1-\epsilon_m}{\epsilon_m}$ $(\alpha_m = 2\beta_m > 1)$
   2.4 Update the weight $w_i^{(m+1)} = w_i^{(m)} \exp(\alpha_m I(y_i \neq G_m(x_i))$, for $i = 1, \ldots, N$
3. Output $G(x) = \text{sign}\Big[\sum_{m=1}^{M} \alpha_m G_m p(x)\Big]$

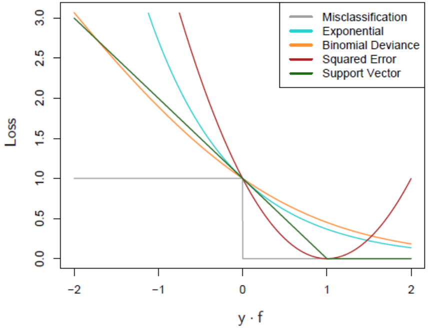
Illustration

- Weights of weak classifiers : the better the classifier is, the larger its weight is
- Weights of samples : Re-weighting after each step, increase the weights for misclassified samples
- Simulation : 2-class classification, 1000 training samples from each class, 10,000 test samples ; two-leaf classification tree (stump) as base learner



Loss Functions

- For classification, exponential loss and binomial negative log-likelihood (deviance) loss $\log(1 + \exp(-2yf))$ share the same population minimizer ; thus it is equivalent to MLE rule
- For classification, squared error loss is not good (not monotonically decreasing) ; the exponential loss is good and binomial deviance is better (less penalty for large $-yf$)



Pros and Cons

- Where it is good
  - AdaBoost improve the classification performance comparing to weak classifiers
  - Many choices for weak classifiers : trees, SVMs, kNNs, etc.
  - Only one tuning parameter M : # of weak classifiers
  - prevent overfitting suffered by single weak classifiers (e.g. complex decision tree)
- Disadvantage
  - Weak interpretability
  - Overfitting when using very bad weak classifiers
  - Sensitive to outliers
  - Not easy for parallel computing

Outlines

Boosting Tree

- Using classification trees or regression trees as base learners
- $f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m)$ where $T(x; \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j)$
- Parameter set $\Theta = \{R_j, \gamma_j\}_{j=1}^{J}$
- Parameter finding : minimizing the empirical risk

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j) \qquad \text{(Combinatorial optimizationp)}$$

- Approximate suboptimal solutions :
  1. Finding γ_j given R_j : $\gamma_j = \bar{y}_j = \frac{1}{|R_j|} \sum_{y_i \in R_j} y_i$ for L^2 loss ; and

     $\gamma_j =$ modal class in R_j for misclassification loss
  2. Finding R_j given γ_j : Difficult, need to estimate γ_j as well ; greedy, top-down recursive partitioning algorithm

Boosting Tree as Forward Stagewise Algorithm

- $\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$
  1. $\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})$
  2. Finding R_{jm} is more difficult than for a single tree in general.
- Squared-error loss : fit a tree to the residual $L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)) = (\underbrace{y_i - f_{m-1}(x_i)}_{\text{residual}} - T(x_i; \Theta_m))^2$
- Two-class classification and exponential loss : AdaBoost for trees, $\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} w_i^{(m)} \exp[-y_i T(x_i; \Theta_m)]$
  1. $\hat{\gamma}_{jm} = \log \frac{\sum_{x_i \in R_{jm}} w_i^{(m)} I(y_i = 1)}{\sum_{x_i \in R_{jm}} w_i^{(m)} I(y_i = -1)}$
- Absolute error or the Huber loss : robust but slow

Gradient Descent for General Loss

- Supervised learning is equivalent to the optimization problem

$$\min_f L(f) = \min_f \sum_{i=1}^{N} L(y_i, f(x_i))$$

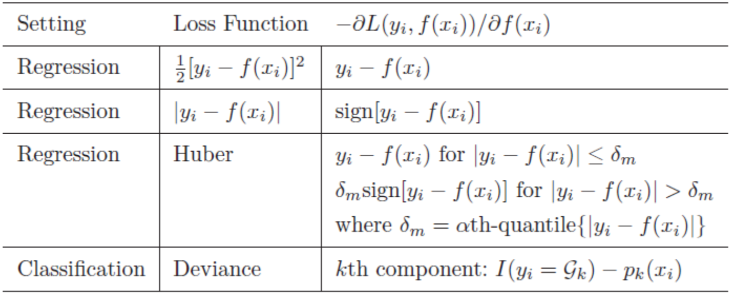
- Numerical optimization : $\hat{\mathbf{f}} = \arg\min_{\mathbf{f}} L(\mathbf{f})$ where $\mathbf{f} = \{f(x_1), f(x_2), \ldots, f(x_N)\}$,
- Approximate $\hat{\mathbf{f}}$ by $\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m$, where $\mathbf{f}_0 = \mathbf{h}_0$ is initial guess
- Gradient descent method : $\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m$, where $g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$, and $\mathbf{h}_m = -\rho_m \mathbf{g}_m$

Gradient Boosting Decision Tree (GBDT)

- Find a tree $T(x;\Theta_m)$ by minimization problem

$$\tilde{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} (-g_{im} - T(x_i;\Theta_m))^2$$

In general $\tilde{R}_{jm} \neq R_{jm}$



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

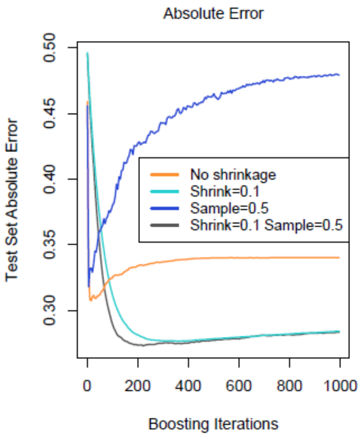
GBDT Algorithm

- Input : training set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, loss function $L(y, f(x))$
- Output : boosting tree $\hat{f}(x)$

```
1. Initialize $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$
2. For $m = 1$ to $M$ :
    2.1 For $i = 1, 2, \ldots, N$ compute $r_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f = f_{m-1}}$
    2.2 Fit a regression tree to the target (residual) $r_{im}$, giving
        terminal regions $R_{jm}$, $j = 1, \ldots, J_m$
    2.3 For $j = 1, \ldots, J_m$, compute
        $\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$
    2.4 Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x_i \in R_{jm})$
3. $\hat{f}(x) = f_M(x)$
```

Regularization Techniques

- Shrinkage : the step 2.4 is modified as $f_m(x) = f_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x_i \in R_{jm})$
- Subsampling : at each iteration, sample a fraction η of the training set and grow the next tree using the subsample
- Shrinkage + subsampling : best performance



Feature importance and Partial Dependence Plots

- Feature importance
  - When fitting a single tree T, at each node t, one feature $X_{v(t)}$ and one separate value $X_{v(t)} = c_{v(t)}$ are chosen to improve a certain quantity of criterion (e.g. GINI, entropy, squared error, etc.)
  - Sum all these improvements i_t brought by each feature X_k over all internal nodes : $I_k(T) = \sum_{t=1}^{J-1} i_t I(v(t) = k)$
  - Average the improvements of all trees $\Rightarrow$ importance of that feature : $I_k = \frac{1}{M} \sum_{m=1}^{M} I_k(T_m)$
- Partial Dependence Plots
  - Partial dependence of $f(X)$ on X_S : $f_S(X_S) = \mathrm{E}_{X_c} f(X_S, X_C)$
  - Estimate by empirical mean : $\bar{f}_S(X_S) = \frac{1}{N} \sum_{i=1}^{N} f(X_S, x_{iC})$

Pros and Cons

- Where it is good
  - For all regression problems
  - Better for two-class classification, possible for multi-class problems (not suggested)
  - Various nonlinearity, strong representability
- Disadvantage
  - Sequential process, inconvenient for parallel computing
  - High computational complexity, not suitable for high-dimensional problems with sparse features

Outlines

Introduction

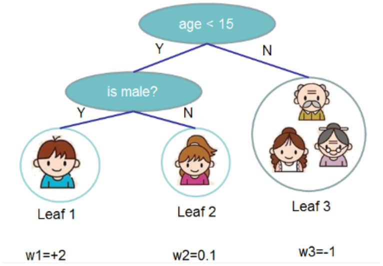
- Developed by Tianqi Chen (http ://homes.cs.washington.edu/∼tqchen/)
- Distributed gradient boosting : can be parallelized
- Highly efficient
- Good performance
- Out-of-Core Computing for big dataset
- Cache Optimization of data structures and algorithms

Cost Functions

- Cost function :
  $F(\Theta_m) = \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)) + R(\Theta_m)$, where
  $R(\Theta)$ is regularization term (L^0, L^1 or L^2 penalties)
- Taylor expansion up to second order :
  $F(\Theta_m) \approx \sum_{i=1}^{N} \Big[L(y_i, f_{m-1}(x_i)) + g_i^{(m)} T(x_i; \Theta_m) +$
  $\frac{1}{2} h_{ii}^{(m)} T(x_i; \Theta_m)^2 \Big] + R(\Theta_m)$, where
  $g_i^{(m)} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x_i) = f_{m-1}(x_i)}$ is the gradient of loss
  function, and $h_{ii}^{(m)} = \left[\frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2} \right]_{f(x_i) = f_{m-1}(x_i)}$ is the diagonal
  of the Hessian of loss function (off-diagonals are zeros).

Penalties

- Take regression trees as examples : Let J_m be the number of leaf nodes (number of rectangles in the partition), γ_{jm} is the approximate constant (weight w) in the leaf node (region) R_{jm}
- The complexity of tree is the sum of L^0 and L^2 norm of $\{\gamma_{jm}\}$: $R(\Theta_m) = \frac{1}{2}\lambda\sum_{j=1}^{J_m}\gamma_{jm}^2 + \mu J_m$



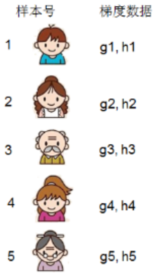
$$R = \frac{1}{2}\lambda(4 + 0.01 + 1) + 3\mu$$

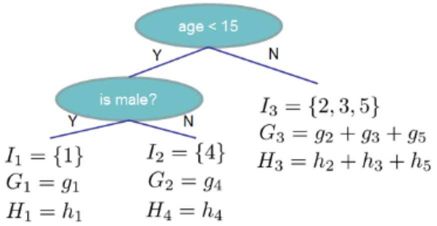
Optimal solutions

- Reformulation of approximated cost function :
  $F(\Theta_m) \approx \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i)) + \sum_{j=1}^{J_m} \Big[\Big(\sum_{x_i \in R_{jm}} g_i^{(m)} \Big) \gamma_{jm} + \frac{1}{2} \Big(\sum_{x_i \in R_{jm}} h_{ii}^{(m)} + \lambda \Big) \gamma_{jm}^2 \Big] + \mu J_m = \sum_{j=1}^{J_m} \Big[G_j^{(m)} \gamma_{jm} + \frac{1}{2} (H_j^{(m)} + \lambda) \gamma_{jm}^2 \Big] + \mu J_m + constant$, where $G_j^{(m)} = \sum_{x_i \in R_{jm}} g_i^{(m)}$ and $H_j^{(m)} = \sum_{x_i \in R_{jm}} h_{ii}^{(m)}$
- By differentiation w.r.t. γ_{jm}, we have the optimal solution :
  $\hat{\gamma}_{jm} = -\frac{G_j^{(m)}}{H_j^{(m)} + \lambda}$
- Simplified cost function :
  $F(\Theta_m) = -\frac{1}{2} \sum_{j=1}^{J_m} \frac{(G_j^{(m)})^2}{H_j^{(m)} + \lambda} + \mu J_m + constant$

Structure Score

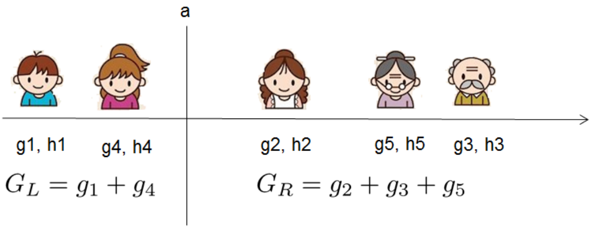
- Neglecting the constant term, we obtain the structure score :
  $SS = -\frac{1}{2}\sum_{j=1}^{J_m}\frac{(G_j^{(m)})^2}{H_j^{(m)}+\lambda} + \mu J_m$
- It is similar to information gain : minimizing the structure score leads to the best tree





Node Splitting - Greedy Algorithm

- When splitting a node into left (L) and right (R) child nodes, we are maximizing $Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$
- Enumerate all possible splits at $x < a$ (e.g., age$<$ 15) from left to right



Greedy Algorithm for split finding

- Input : training set $D = \{(x_1, y_1), \dots, (x_N, y_N)\}$, loss function $L(y, f(x))$, the index set $I = \{i | x_i \in R_{jm}\}$ of current node R_{jm}, feature dimension d
- Output : best split

```
1. Initialize $gain = 0$, $G = \sum_{i \in I} g_i$, $H = \sum_{i \in I} h_{ii}$
2. For $k = 1$ to $K$ :
    2.1 $G_L = 0$, $H_L = 0$
    2.2 For $j$ in sorted($I$, by $x_{jk}$), do
        2.2.1 $G_L = G_L + g_j$, $H_L = H_L + h_{jj}$, $G_R = G - G_L$, $H_R = H - H_L$
        2.2.2 $score = \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
3. Output split with max score
```

Loss Functions

- Square loss $L(y, f) = (y - f)^2$:
  $g_i^{(m)} = 2(f_i - y_i) = 2 \times residue$, $h_{ii}^{(m)} = 2$
- Logistic loss $L(y, f) = y \ln(1 + e^{-f}) + (1 - y) \ln(1 + e^{f})$:
  $g_i^{(m)} = -y_i \left(1 - \frac{1}{1+e^{-f_{m-1}(x_i)}} + (1 - y_i) \frac{1}{1+e^{-f_{m-1}(x_i)}}\right) =$
  $Pred - Label$, $h_{ii}^{(m)} = \frac{e^{-f_{m-1}(x_i)}}{(1+e^{-f_{m-1}(x_i)})^2} = Pred \times (1 - Pred)$

Outlines

Conclusions

- Ensemble methods have integrable abilities of single models, achieving better performance
- Easy to generalize to new data
- When there are strong noises, easy to overfit
- Computationally intensive

Python Examples

- Random forest :

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100)
# RandomForestClassifier(bootstrap=True, class_weight=None,
    pcriterion='gini', max_depth=None, max_features='auto',
    max_leaf_nodes=None, min_impurity_split=1e-07,
    min_samples_leaf=1, min_samples_split=2,
    min_weight_fraction_leaf=0.0, n_estimators=100,
    n_jobs=1, oob_score=False, random_state=None,
    verbose=0, warm_start=False)
# Feature importance in random forest
feature_imp = pd.Series(rf.feature_importances_)
rf.fit(X_train, Y_train)
Y_predict_rf = rf.predict(X_test)
oob_error = 1 - rf2.oob_score_
```

- AdaBoost :

```
from sklearn.ensemble import AdaBoostClassifier
adaboost = AdaBoostClassifier(n_estimators = 50)
adaboost.fit(X_train, Y_train)
adaboost.staged_predict(X_train)
Y_predict_ada = adaboost.predict(X_test)
```

References

- 数据分析导论，博雅大数据学院
- 周志华，机器学习，2016
- T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning : Data mining, Inference, and Prediction, 2nd Edition, 2009