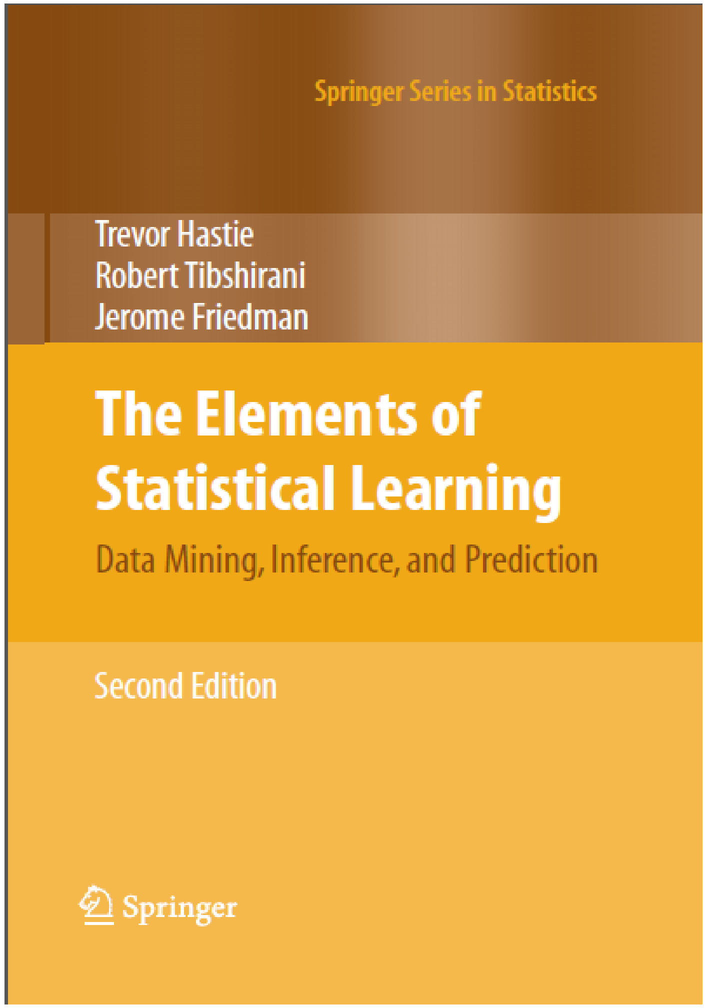
Ensemble Methods



Springer Series in Statistics

Trevor Hastie
Robert Tibshirani
Jerome Friedman

The Elements of Statistical Learning

Data Mining, Inference, and Prediction

Second Edition

Springer

coursera

Practical Machine Learning (in Coursera)

Ensemble Methods

“Wisdom of Crowds”

The collective knowledge of a diverse and independent body of people typically exceeds the knowledge of any single individual, and can be harnessed by voting.

Ensemble Methods

- Motivation: combining predictions from several models to obtain one (better) overall prediction.

- Netflix competition: two main factors to improve overall accuracy
  - individual algorithm
  - ensemble idea

Ensemble Methods

Netflix:

We looked at the two underlying algorithms with the best performance in the ensemble: *Matrix Factorization* (which the community generally called SVD,*Singular Value Decomposition*) and *Restricted Boltzmann Machines* (RBM). SVD by itself provided a 0.8914 RMSE, while RBM alone provided a competitive but slightly worse 0.8990 RMSE. A linear blend of these two reduced the error to 0.88.

https://www.techdirt.com/blog/innovation/articles/20120409/03412518422
http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html

Ensemble Methods

Ensemble Methods:

1. **Bagging**
2. **Random Forests**
3. **Boosting: Adaboost, GBM (XGBoost, LightGBM)**
4. **Stacking**

Ensemble Methods

Two central premises:

- Pooling models represents a richer model class than simply choosing one of them. Thus the weighted sum of predictions from a collection of models may give improved performance.
- Evaluation of performance is predictive rather than model-based.

1. Bagging

- Motivation: combine the predictors of bootstrap samples to improve the estimate or prediction

基于bootstrap抽样，然后整合在一起

- Bootstrap aggregating (Bagging) was proposed by Breiman (1994)

Recall Bootstrap

- Main idea: sampling with replacement
- Consider the training set z_1, …, z_n with $z_i = (x_i, y_i)$ as the "population". We generate bootstrap samples by taking n observations with replacement from this population in each sample.
- For any statistic, S(Z), we can estimate its distribution by the empirical distribution of its bootstrap replications $S(\mathbf{Z}^{*b})$ for b=1, …, B.

Bagging

- Fit a model from each bootstrap sample

- Take average of the predictors from the B fitted models (or majority for classification cases)

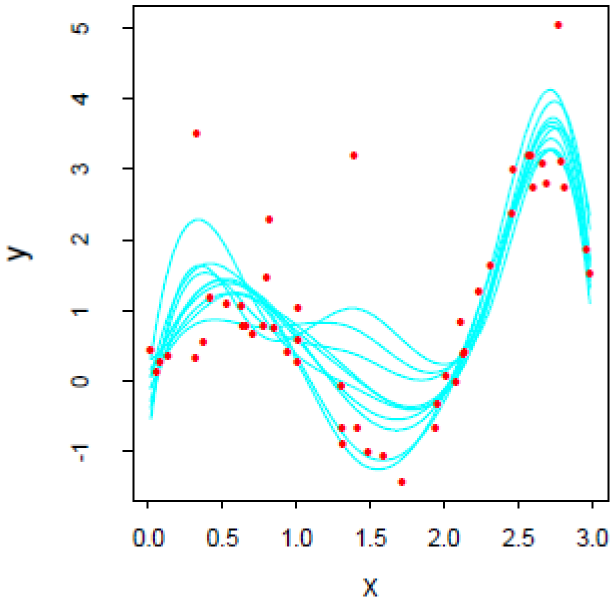
- For each bootstrap sample, $\mathbf{Z}^{*b}$, we fit our model (Tree), giving prediction $\hat{f}^{*b}(x)$. Then the bagging estimate is

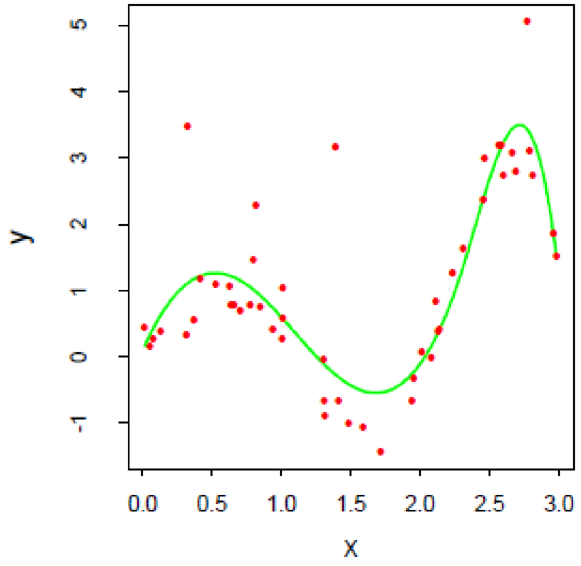
$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x) \quad \text{or} \quad (arg\,\max_{k} \hat{f}_{bag}(x))$$

Bagging

- **Bagging is not useful for improving linear models**

Bagging对线性模型没有效果，因为Bagging本身是一些模型的线性组合，线性模型的线性组合还是线性的。
总要对非线性的做，用的较多的就是决策树。

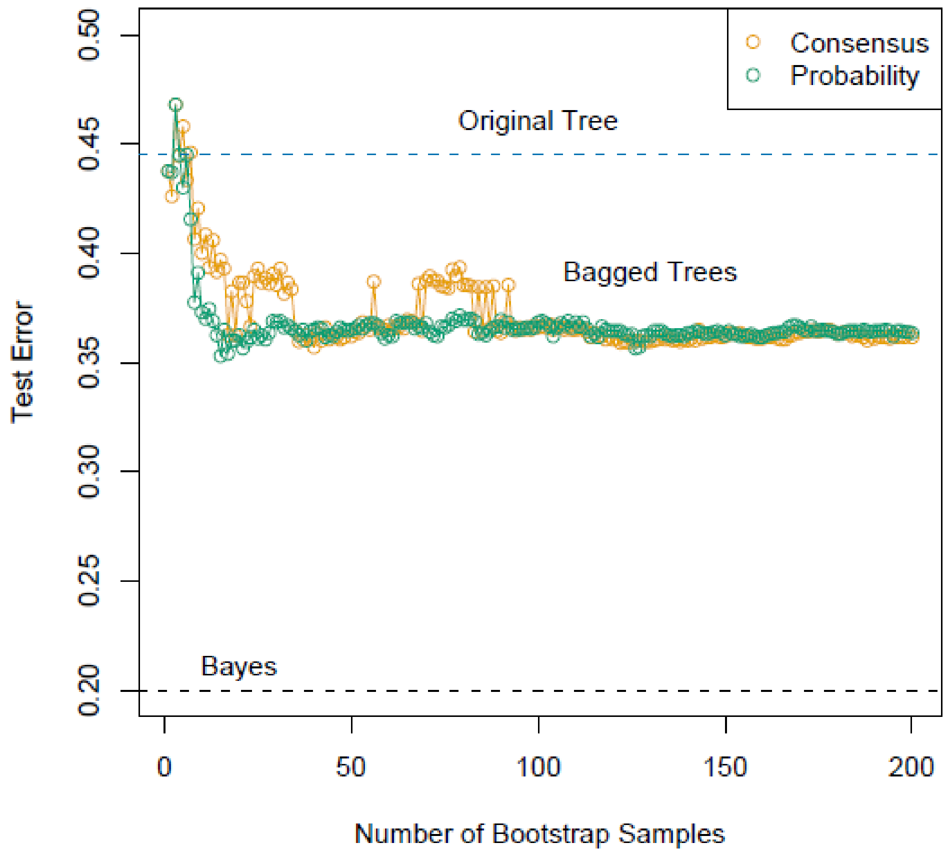




- **The bagged estimate will differ from the original estimate only when the latter is nonlinear or adaptive function of the data.**

Bagging

- **Bagging can dramatically improve prediction of unstable procedures like tree (because averaging reduce variance and leaves bias unchanged).**



Features of Bagging

1. Double-edged sword

- **For classification, bagging a good classifier can make it better, but bagging a bad classifier can make it worse.**
- **Usually, CART is a good classifier.**

2. Dependence

Bagging的方差Var(\hat f_bag)=[B\sigma^2+B(B-1)\rho \sigma^2]/B^2=

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

\rho 增大，方差增大；B增大，方差减小（B是子模型个数，\sigma^2是单个子模型的方差，\rho是子模型间的相关系数）

- **Dependence of bagging trees limits the benefits of averaging.**
- **Random forest will improve from this respect.**

随机森林让每棵树长得不太一样，这样\rho就小了，从而增强模型的稳定性

Features of Bagging

3. Lost simple structure

- **When we bag a model, any simple structure in the model is lost.**
- **A bagged tree is no longer a tree and loses the advantage of interpretatioin in the tree.**

Application Keys of Bagging

- Bagging is useful for nonliear/adaptive/unstable models.

- Tuning parameter: B

- R: bagging (in adabag package)

  这个不支持回归

  - mfinal
  - control

Bagging

Target		Keys	Scale	Missing Data
Class	Regre			
√	X	(1) #trees (2) Trees construction (minsplit, maxdepth, complexity, …)	X	√

2. Random Forests

A random forest (Breiman, 2001) consists of many decision trees and outputs the class on a majority vote.

- One starts with a data set. From that, one draws B (say, 100) bootstrap samples, constructing a classification rule for each (say, CART);

- The random forest consists of the trees formed from the bootstrap samples (no prune of the trees is done); randomly pick some input variables and take the best splits among them;

- To classify a new observation, one uses each of the trees in the forest.

Process of Random Forests

- Before each split, select m of the input variables at random as candidates for splitting.

- After B such trees $\{T(x;\Theta_b)\}_1^B$ are grown, the random forest (regression) predictor is

$$\hat{f}_{\mathrm{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T(x;\Theta_b)$$

Random Forests

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

- The idea in random forest is to improve the variance reduction of bagging by **reducing the correlation** between the trees, without increasing σ^2 too much.

- This is achieved in the tree-growing process through **random selection of the input variables.**

The Parameter: m

- Intuitively, reducing m will reduce the correlation between any pair of trees in the ensemble, and hence reduce the variance of the average.

- Recommendations of the inventors:
  - For classification, the default value for m is $\lfloor \sqrt{p} \rfloor$ and the minimum leaf node size is one.
  - For regression, the default value for m is $\lfloor p/3 \rfloor$ and the minimum leaf node size is five.

Out of Bag Samples (OOB)

- An important feature of random forest is its use of out-of-bag (OOB) samples:

不用额外做交叉验证，直接用袋外样本就能验证

When evaluating：

For each observation zi=(xi,yi), construct its random forest predictor by averaging only those trees corresponding to bootstrap samples in which zi did not appear.

- An OOB error estimate is almost identical to that obtained by N-fold cross-validation. Once the OOB error stabilizes, the training can be terminated.

Variable Importance

两个重要性指标，见代码

- Varibale importance measures the **relative** importance of predictor variables. It is an important **interpretation** technique.

- At each split in each tree, the **improvement in the split-criterioin** is the importance measure attibuted to the splitting variable, and is accumulated over all the trees in the forest separately for each variable.

- The **OOB samples** can be used in the variable importance measure and they tend to spread the importances **more uniformly**.

Application Keys of Random Forests

- Pros: Accuracy

- Cons:
    - Low speed
    - Hard to interpret
    - Overfitting

    这里对单棵树不再做剪枝

- Tuning parameter: B, m

Application Keys of Random Forests

randomForest() (in randomForest package)

- ntree
- mtry
- na.action
- importance
- proximity
- nodesize
- maxnodes

RF

Target		Keys	Scale	Missing Data
Class	Regre			
√	√	**(1) #trees** **(2) #variable candidates** **(3) Trees construction** **(nodesize, maxnodes)**	X	X

3. Boosting

- **Boosting was originally designed for classification problems by Schapire (1990), but it can be extended to regression.**

罗伯特·夏皮尔是普林斯顿大学计算机科学学院的教授和研究员。他的主要研究领域是机器学习的理论和应用研究。他促进了 Boosting Meta 算法应用于机器学习领域的发展。1996 年，他和 Yoav Freunduoqo 发明了 AdaBoost 算法，并因此获得 2003 年“歌德尔奖”——计算理论界的最高奖项。

Boosting

Boosting中较大的一类方法是adaptive（自适应）的，但adaptive不能做到并行，要上一个算完再改进，所以速度上较慢

- **Motivation: combine the outputs of many “weak” learners to produce a powerful “committee”**

- **The idea is to improve certain weak classification rules by iteratively optimizing them on the training data.**

Adaboost – Adaptive Boosting

- AdaBoost: Most popular boosting algorithm. Introduced by Freund and Schapire, 1997

Freund

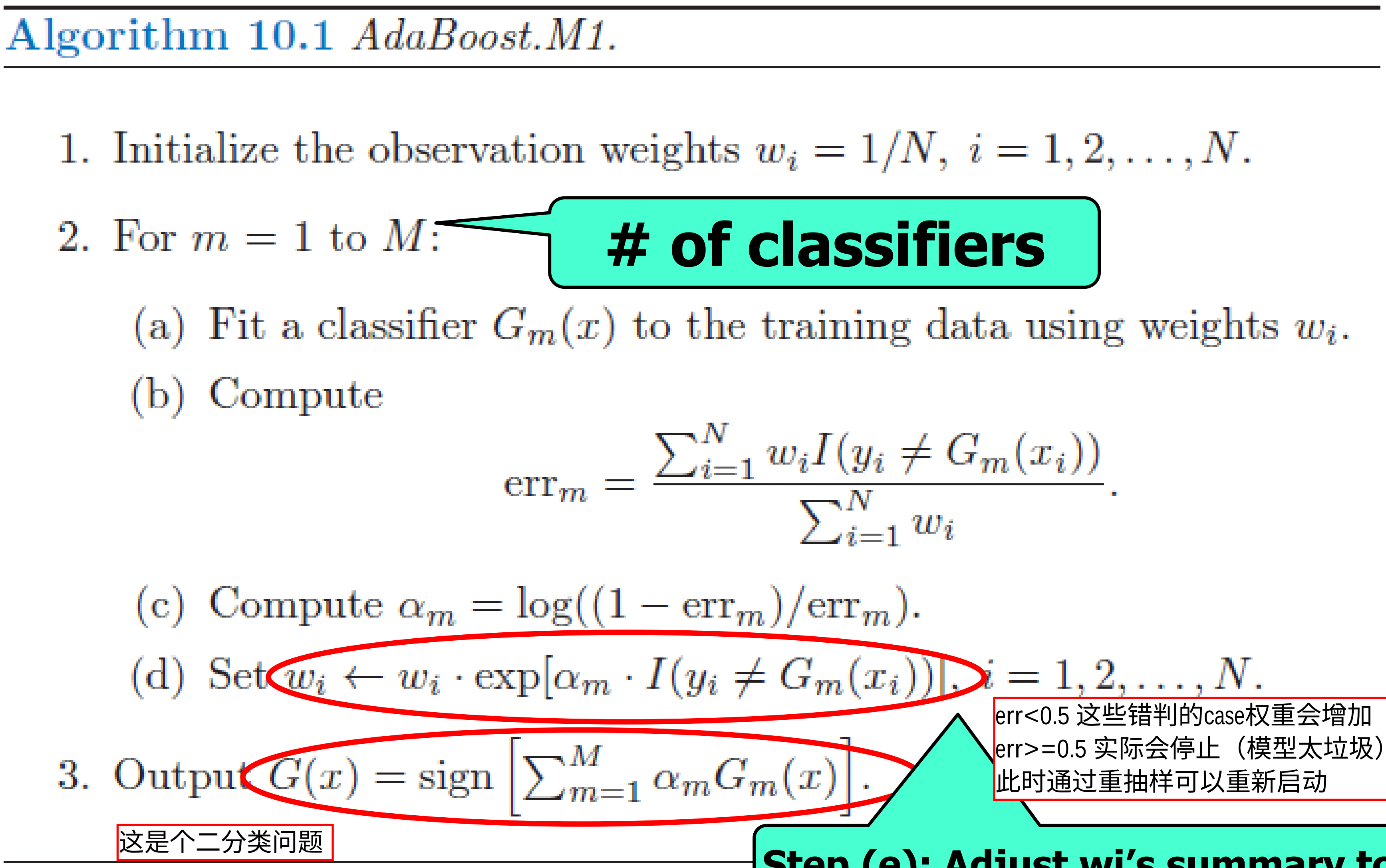
Adaboost – Adaptive Boosting

- Idea: Re-weight instead of re-sampling as in bootstrap or bagging

代码中也可以做到重抽样，但boosting与bagging最大的区别是重加权；代码只能做分类不能做回归

- What’s so good about Adaboost?
  - Improve classification accuracy
  - Can be used with many different classifiers
  - Simple to implement
  - Not prone to overfitting
    (though it is sensitive to noisy data and outliers)

Discrete Adaboost (Friedman's wording)



Algorithm 10.1 *AdaBoost.M1.*

```
1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to $M$:
    (a) Fit a classifier $G_m(x)$ to the training data using weights $w_i$.
    (b) Compute
        $$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
    (c) Compute $\alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$.
    (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.
```

# of classifiers

err<0.5 这些错判的case权重会增加
err>=0.5 实际会停止（模型太垃圾）
此时通过重抽样可以重新启动

这是个二分类问题

Step (e): Adjust wi's summary to 1

Principle of Adaboost Algorithm

Questions:

- **Why are we recursively updating $G_t(x)$? And how?**
- **Why is $\alpha_t = \ln\left(\frac{1-err_t}{err_t}\right)$?**

Principle of Adaboost Algorithm

Exponential loss function:

$$L_{exp}(G) = \exp\{-f(x)G(x)\}$$

where f(x) is the true model, G(x) is the fitted model.

The exponential loss function is continuous, differentiable, and works similarly with 0/1 loss function.

Its average loss is

$$E_{x\sim D}[L_{exp}(G)]$$

Principle of Adaboost Algorithm

Denote

$$H_t(x) = \sum_{m=1}^{t} \beta_m G_m(x).$$

Suppose we already have $H_{t-1}(x)$. Then, we should find $G_t(x)$ to obtain $H_t(x)$.

$$\begin{aligned} G_t(x) &= \arg\min_{h(x)} E_{x\sim D}[L_{exp}(H_{t-1}(x) + \beta_t h(x))] \\ &= \arg\min_{h(x)} E_{x\sim D}[\exp\{-f(x)(H_{t-1}(x) + \beta_t h(x))\}] \\ &= \arg\min_{h(x)} E_{x\sim D}[\exp\{-f(x)H_{t-1}(x)\} \cdot \exp\{-f(x)\beta_t h(x)\}]. \end{aligned}$$

Principle of Adaboost Algorithm

Using Taylor expansion and with $f(x) = \pm 1$, $h(x) = \pm 1$,

$$\begin{aligned}\exp\{-f(x)\beta_t h(x)\} &\approx 1 - f(x)\beta_t h(x) + \frac{f^2(x)\beta_t^2 h^2(x)}{2} \\ &= 1 - f(x)\beta_t h(x) + \frac{1}{2}\beta_t^2.\end{aligned}$$

Then, we have

$$G_t(x) \approx \arg\min_{h(x)} E_{x\sim D}\left[\exp\{-f(x)H_{t-1}(x)\} \cdot \left(1 - f(x)\beta_t h(x) + \frac{1}{2}\beta_t^2\right)\right]$$

$$= \arg\max_{h(x)} E_{x\sim D}[\exp\{-f(x)H_{t-1}(x)\} \cdot f(x)h(x)],$$

or

$$G_t(x) \approx \arg\max_{h(x)} E_{x\sim D}\left[\frac{\exp\{-f(x)H_{t-1}(x)\}}{E_{x\sim D}[\exp\{-f(x)H_{t-1}(x)\}]} \cdot f(x)h(x)\right].$$

Principle of Adaboost Algorithm

Construct a new distribution $D_t(x)$ **as**

$$D_t(x) = \frac{D(x)\exp\{-f(x)H_{t-1}(x)\}}{E_{x\sim D}[\exp\{-f(x)H_{t-1}(x)\}]}.$$

Then,

$$G_t(x) \approx \arg\max_{h(x)} E_{x\sim D_t}[f(x)h(x)].$$

Principle of Adaboost Algorithm

Denoting $I(\cdot)$ as an indicator function, we have

$$G_t(x) \approx \arg\max_{h(x)} E_{x\sim D_t}[1 - 2I_{f\neq h}(x)],$$

or

$$G_t(x) \approx \arg\min_{h(x)} E_{x\sim D_t}[I_{f\neq h}(x)].$$

It indicates that $G_t(x)$ should minimize the classification error under the distribution D_t.

Principle of Adaboost Algorithm

$$\begin{aligned}
D_{t+1}(x) &= \frac{D(x)\exp\{-f(x)H_t(x)\}}{E_{x\sim D}[\exp\{-f(x)H_t(x)\}]} \\
&= D_t(x)\exp\{f(x)\beta_t G_t(x)\}\frac{E_{x\sim D}[\exp\{-f(x)H_{t-1}(x)\}]}{E_{x\sim D}[\exp\{-f(x)H_t(x)\}]} \\
&\propto D_t(x)\exp\{f(x)\beta_t G_t(x)\} \\
&= D_t(x)\cdot\begin{cases}\exp\{-\beta_t\}, & f(x)=G_t(x)\\ \exp\{\beta_t\}, & f(x)\neq G_t(x)\end{cases} \\
&\propto D_t(x)\cdot\begin{cases}1, & f(x)=G_t(x)\\ \exp\{2\beta_t\}, & f(x)\neq G_t(x)\end{cases}
\end{aligned}$$

$D_t(x)$ has the recursive form, and thus $G_t(x)$ and $H_t(x)$ can be determined recursively.

Principle of Adaboost Algorithm

After obtaining $G_t(x)$, we should find its weight β_t or α_t.

$$\begin{aligned}
\beta_t(x) &= \arg\min_\beta E_{x\sim D}[L_{exp}(H_{t-1}(x)+\beta G_t(x))] \\
&= \arg\min_\beta E_{x\sim D_t}[L_{exp}(\beta G_t(x))] \\
&= \arg\min_\beta E_{x\sim D_t}[\exp\{-f(x)\beta G_t(x)\}] \\
&= \arg\min_\beta E_{x\sim D_t}[\exp\{-\beta\} I_{f=G_t}(x) + \exp\{\beta\} I_{f\neq G_t}(x)] \\
&= \arg\min_\beta[\exp\{-\beta\}(1-err_t) + \exp\{\beta\} err_t] \\
&= \frac{1}{2}\ln\left(\frac{1-err_t}{err_t}\right).
\end{aligned}$$

Principle of Adaboost Algorithm

Letting $\alpha_t = 2\beta_t$**, we have**

$$\alpha_t = 2\beta_t = \ln\left(\frac{1 - err_t}{err_t}\right).$$

Weights in AdaBoost

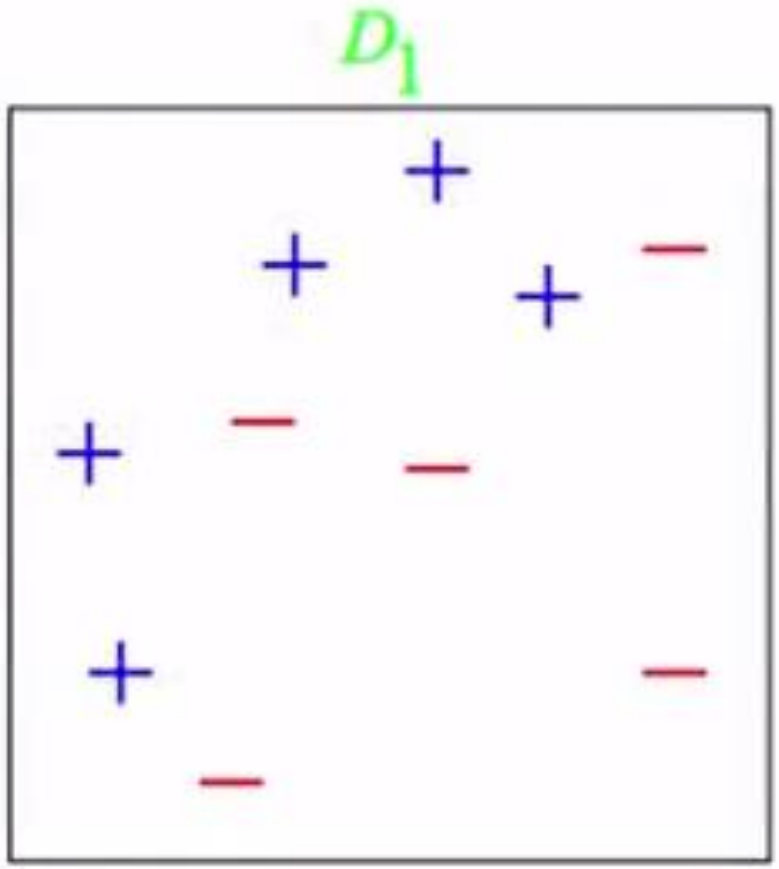
- Observations

At step m, those observations that were **misclassified** by the classifier $G_{m-1}(x)$ induced at the previous step have their **weights increased**, whereas the weights are decreased for those that were classified correctly.

- Classifiers

Give **higher influence** to the **more accurate classifiers** in the sequence.

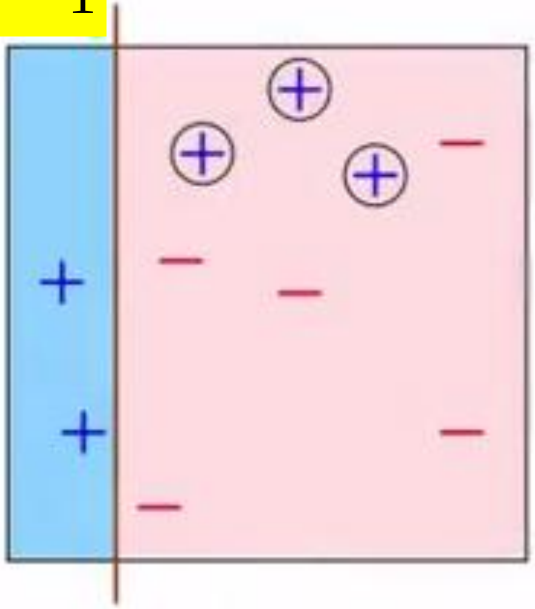
A Simple Example

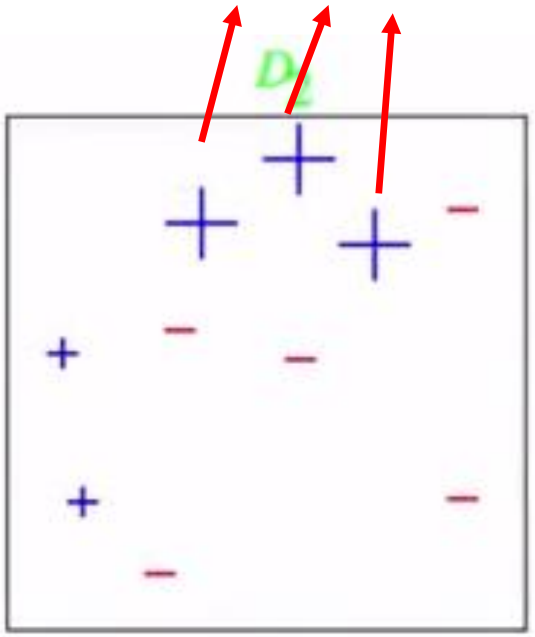


A Simple Example

Round 1

G_1





$$w_i = 0.17$$

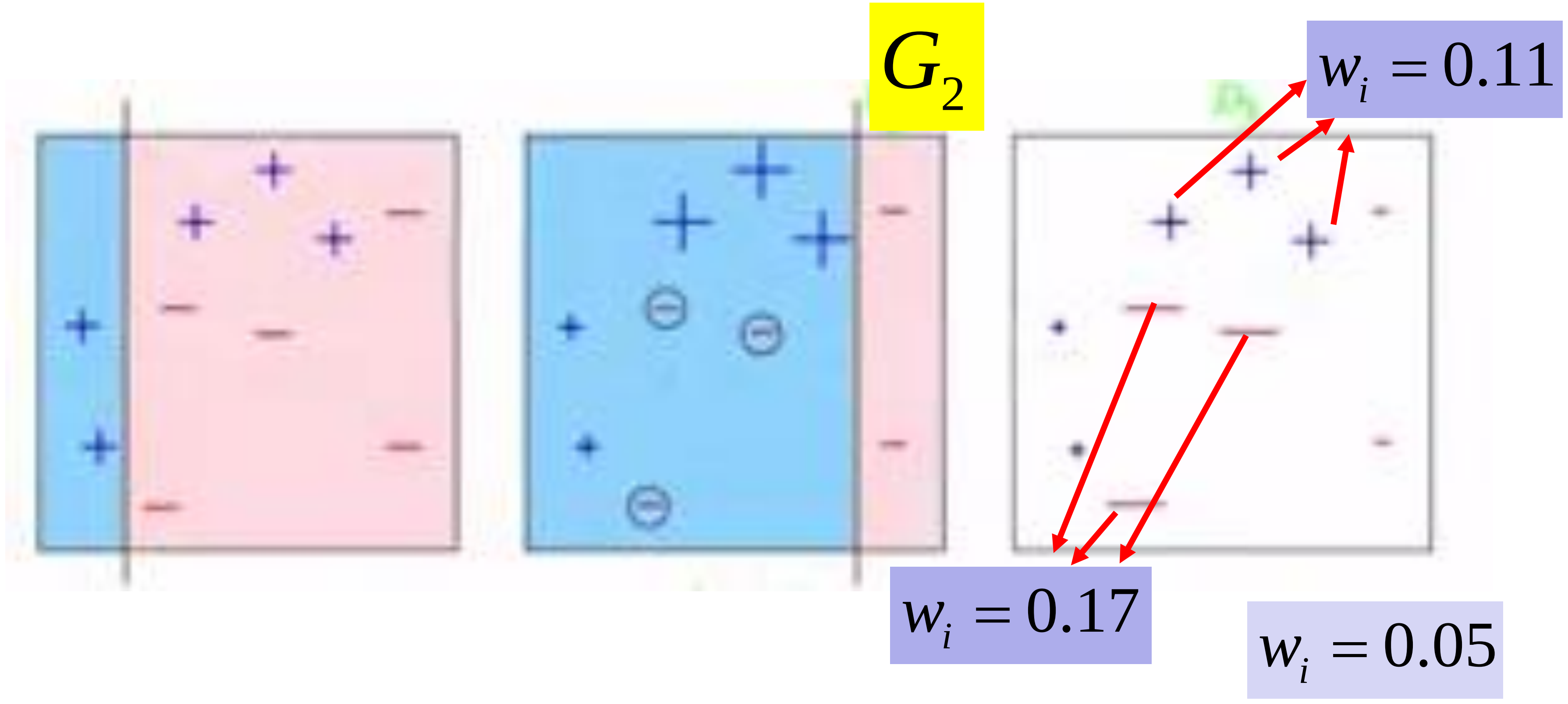
$$err_1 = \frac{0.1 \times 3}{1} = 0.3$$

$$\alpha_1 = \log \frac{0.7}{0.3} \approx 0.85$$

错判case权重：0.1*7/3
没错判的还是0.1
然后保证求和为1

$$w_i = 0.07$$

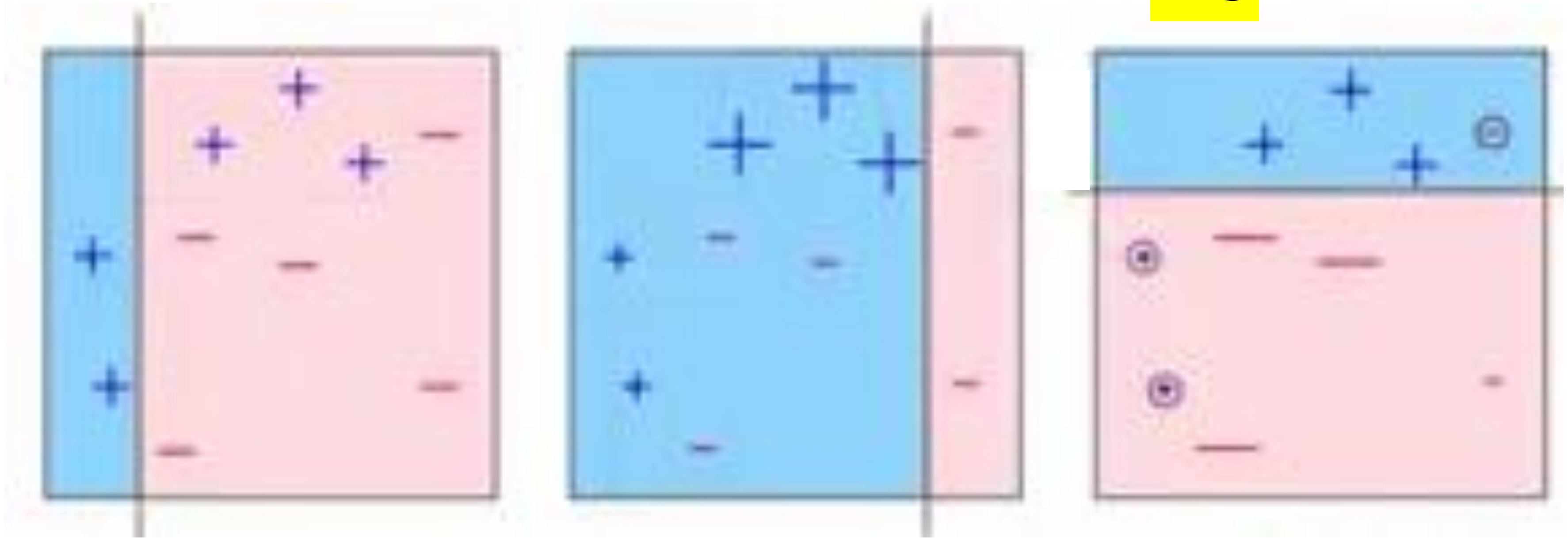
Round 2



$$err_2 = \frac{0.07 \times 3}{1} = 0.21 \qquad \alpha_2 = \log \frac{0.21}{0.79} \approx 1.30$$

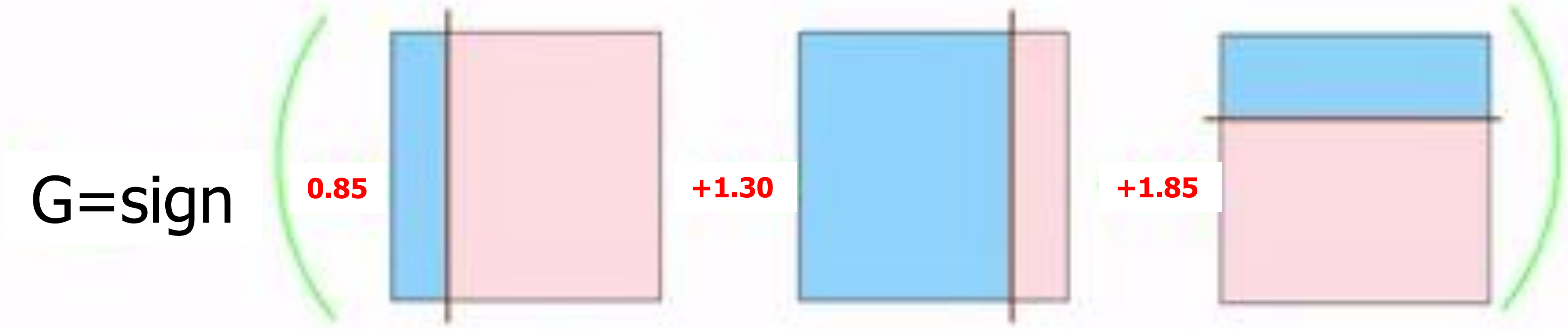
Round 3

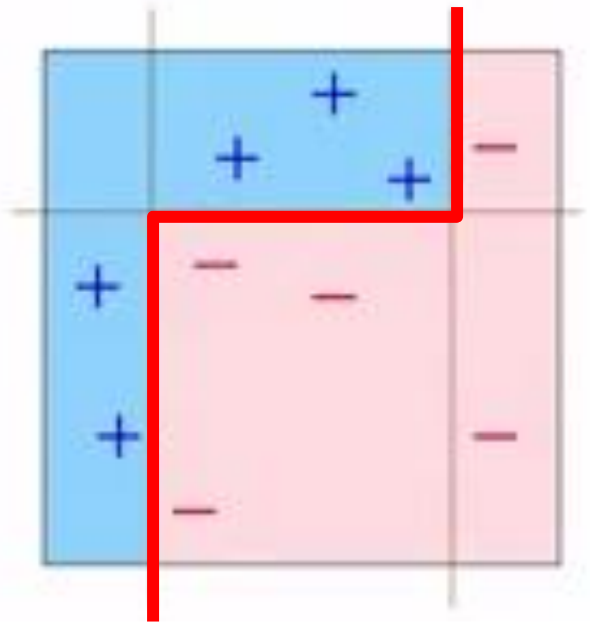
G_3



$$err_3 = \frac{0.05 \times 3}{1} \approx 0.14 \qquad \alpha_2 = \log \frac{0.86}{0.14} \approx 1.85$$

Final





A simple AdaBoost Example

- **Training Data:**

Y	1	-1	1
X	-1	0	1

- **Weak Learner:**

这里并不是adaptive，
子模型已经事先确定了，
但是计算时还是按顺序算

  - $h_1(x) \equiv 1$
  - $h_2(x) = \begin{cases} 1 & if\ x < -0.5 \\ -1 & if\ x > -0.5 \end{cases}$
  - $h_3(x) = \begin{cases} -1 & if\ x < 0.5 \\ 1 & if\ x > 0.5 \end{cases}$

A simple AdaBoost Example

- **Please fill in:**

Loop (m)	$W_i \leftarrow w_i exp(c_m I(y_i \neq f_m(X_i)))$			$f_m(x)$	$Err_m = E_w(I(y \neq f_m(X))$	$\alpha_m = log(\frac{1-Err_m}{Err_m})$
	w_1	w_2	w_3			
1	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$h_1(x)$		
2				$h_2(x)$		
3				$h_3(x)$		

Y	1	-1	1
X	-1	0	1

- $h_1(x) \equiv 1$
- $h_2(x) = \begin{cases} 1 & if\ x < -0.5 \\ -1 & if\ x > -0.5 \end{cases}$
- $h_3(x) = \begin{cases} -1 & if\ x < 0.5 \\ 1 & if\ x > 0.5 \end{cases}$

A simple AdaBoost Example

Y	1	-1	1
X	-1	0	1

- $h_1(x) \equiv 1$
- $h_2(x) = \begin{cases} 1 & if\ x < -0.5 \\ -1 & if\ x > -0.5 \end{cases}$
- $h_3(x) = \begin{cases} -1 & if\ x < 0.5 \\ 1 & if\ x > 0.5 \end{cases}$

Loop (m)	$W_i \leftarrow w_i exp(c_m I(y_i \neq f_m(X_i)))$			$f_m(x)$	$Err_m = E_w(I(y \neq f_m(X))$	$\alpha_m = log(\frac{1-Err_m}{Err_m})$
	w_1	w_2	w_3			
1	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$h_1(x)$	$\frac{1}{3}$	$\log 2$
2	$\frac{1}{3}\ \left(\frac{1}{4}\right)$	$\frac{2}{3}\ \left(\frac{1}{2}\right)$	$\frac{1}{3}\ \left(\frac{1}{4}\right)$	$h_2(x)$	$\frac{1}{4}$	$\log 3$
3	$\frac{1}{4}\ \left(\frac{1}{6}\right)$	$\frac{1}{2}\ \left(\frac{1}{3}\right)$	$\frac{3}{4}\ \left(\frac{1}{2}\right)$	$h_3(x)$	$\frac{1}{6}$	$\log 5$

A simple AdaBoost Example

Y	1	-1	1
X	-1	0	1

- $h_1(x) \equiv 1$
- $h_2(x) = \begin{cases} 1 & if\ x < -0.5 \\ -1 & if\ x > -0.5 \end{cases}$
- $h_3(x) = \begin{cases} -1 & if\ x < 0.5 \\ 1 & if\ x > 0.5 \end{cases}$

- **At M=3, the classifier is**

$$f(x) = sign\big((\log 2)h_1(x) + (\log 3)h_2(x) + (\log 5)h_3(x)\big)$$

$$= \begin{cases} sign(log2 + log\ 3\ - log\ 5) = 1 & if\ x < -0.5 \\ sign(log2 - log\ 3\ - log\ 5) = -1 & if\ -0.5 < x < 0.5 \\ sign(log2 - log\ 3 + log\ 5) = 1 & if\ x > 0.5 \end{cases}$$

K-class AdaBoost

For K-class problem, we may construct K separate models Gk(x), k=1,2,...,K.

Details:

Step 1: **Code y_{ik} as 1 if observation i is in class k and -1 otherwise. Using Algorithm 10.1 (AdaBoost.M1.), construct a classification tree $G_k(x)$.**

Step 2: **Combine the independent K models $G_k(x)$, k=1,2,...,K together to produce a K-class classifier.**

Variable Importance

- For each class k (k=1,2,…,K), we may have a variable importance measure V_{lk} for the predictor X_l. V_{lk} may also be set as **improvement in the split-criterioin.**

- The overall importance of X_l is obtained by averaging over all of the classes

$$V_l = \frac{1}{K}\sum_{k=1}^{K} V_{lk}$$

Application Keys of Boosting

- Adaboost with trees is the “best off-the-shelf classifier in the world” (Breiman, 1996)
- Tuning parameter: M

Application Keys of Boosting

- boosting() (in adabag package)
  - boos 重抽样设置
  - mfinal
  - coeflearn 有三种系数算法，其中两个类似
  - control

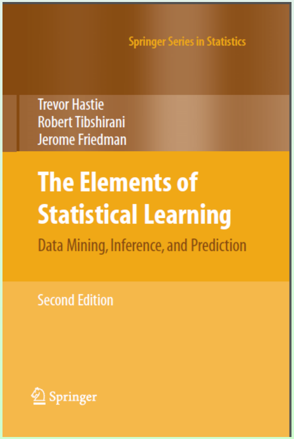
- boosting.cv() (in adabag package)

Adaboost

Target		Keys	Scale	Missing Data
Class	Regre			
√	X	(1) #trees (2) Trees construction (minsplit, maxdepth, complexity, …)	X	X

GBM/GBDT

GBM / GBDT


Springer Series in Statistics

Trevor Hastie
Robert Tibshirani
Jerome Friedman

The Elements of Statistical Learning

Data Mining, Inference, and Prediction

Second Edition

Springer

GBM/GBDT

Gradient Boosting Model (GBM) or Gradient Boosting Decision Tree (GBDT) is based on an additive model:

$$f_m(x) = f_{m-1}(x) + h_m(x).$$

Consider the squared-error loss:

$$L = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(x_i))^2.$$

$\triangle f(x)$ **(i.e.,** $h_m(x)$ **at Step** m**) should be in the local direction for which** L **is most rapidly decreasing at** $f_{m-1}(x)$**.**

GBM/GBDT

Define the gradient:

$$\begin{aligned} g_{mi} &= \left.\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right|_{f(x_i)=f_{m-1}(x_i)} \\ &= -(y_i - f(x_i))|_{f(x_i)=f_{m-1}(x_i)} \\ &= -(y_i - f_{m-1}(x_i)). \end{aligned}$$

Then, $-g_m$ is the required local direction.

GBM/GBDT

At Step m,

$$\begin{aligned} L &= \tfrac{1}{2}\sum_{i=1}^{n}(y_i - f_m(x_i))^2 \\ &= \tfrac{1}{2}\sum_{i=1}^{n}(y_i - f_{m-1}(x_i) - h_m(x_i))^2 . \end{aligned}$$

Denote by $\varepsilon_{mi} = y_i - f_{m-1}(x_i)$, the residuals produced by f_{m-1}. Then, the loss becomes

$$\frac{1}{2}\sum_{i=1}^{n}(\varepsilon_{mi} - h_m(x_i))^2 .$$

It means that the trees constructed at Step m should fit the residuals produced by f_{m-1}.

GBM/GBDT

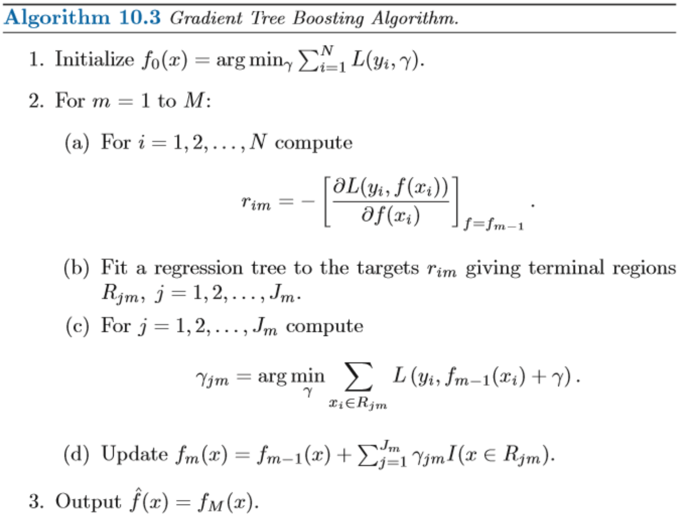
Recall at Step m,

- $h_m(x)$ **should be in the local direction specified by the gradient** $-g_m = y - f_{m-1}(x)$**.**
- $h_m(x)$ **should be a decision tree with the dependent variable** $\varepsilon_m = y - f_{m-1}(x)$**.**

Thus, ideally $h_m(x_i) = -g_{mi} = \varepsilon_{mi}$. However, the decision tree may not attain these ideal values.

GBM/GBDT

gbm



Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

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

UNIVERSITY OF WASHINGTON

Introduction to Boosted Trees

Tianqi Chen
Oct. 22 2014

Outline

- **Review of key concepts of supervised learning**
- Regression Tree and Ensemble (What are we Learning)
- Gradient Boosting (How do we Learn)
- Summary

Elements in Supervised Learning

- Notations: $x_i \in \mathbf{R}^d$ i-th training example
- **Model**: how to make prediction $\hat{y}_i$ given x_i
  - Linear model: $\hat{y}_i = \sum_j w_j x_{ij}$ (include linear/logistic regression)
  - The prediction score $\hat{y}_i$ can have different interpretations depending on the task
    - Linear regression: $\hat{y}_i$ is the predicted score
    - Logistic regression: $1/(1 + exp(-\hat{y}_i))$ is predicted the probability of the instance being positive
    - Others… for example in ranking $\hat{y}_i$ can be the rank score
- **Parameters**: the things we need to learn from data
  - Linear model: $\Theta = \{w_j | j = 1, \cdots, d\}$

Elements continued: Objective Function

- Objective function that is everywhere

$$Obj(\Theta) = \underbrace{L(\Theta)}_{\text{Training Loss measures how well model fit on training data}} + \underbrace{\Omega(\Theta)}_{\text{Regularization, measures complexity of model}}$$

Training Loss measures how well model fit on training data

Regularization, measures complexity of model

- Loss on training data: $L = \sum_{i=1}^{n} l(y_i, \hat{y}_i)$
  - Square loss: $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
  - Logistic loss: $l(y_i, \hat{y}_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})$
- Regularization: how complicated the model is?
  - L2 norm: $\Omega(w) = \lambda \|w\|^2$
  - L1 norm (lasso): $\Omega(w) = \lambda \|w\|_1$

Putting known knowledge into context

- Ridge regression: $\sum_{i=1}^{n}(y_i - w^T x_i)^2 + \lambda\|w\|^2$
  - Linear model, square loss, L2 regularization
- Lasso: $\sum_{i=1}^{n}(y_i - w^T x_i)^2 + \lambda\|w\|_1$
  - Linear model, square loss, L1 regularization
- Logistic regression:

$$\sum_{i=1}^{n}[y_i \ln(1 + e^{-w^T x_i}) + (1 - y_i)\ln(1 + e^{w^T x_i})] + \lambda\|w\|^2$$

  - Linear model, logistic loss, L2 regularization
- The conceptual separation between model, parameter, objective also gives you **engineering benefits**.
  - Think of how you can implement SGD for both ridge regression and logistic regression

Objective and Bias Variance Trade-off

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

Training Loss measures how well model fit on training data

Regularization, measures complexity of model

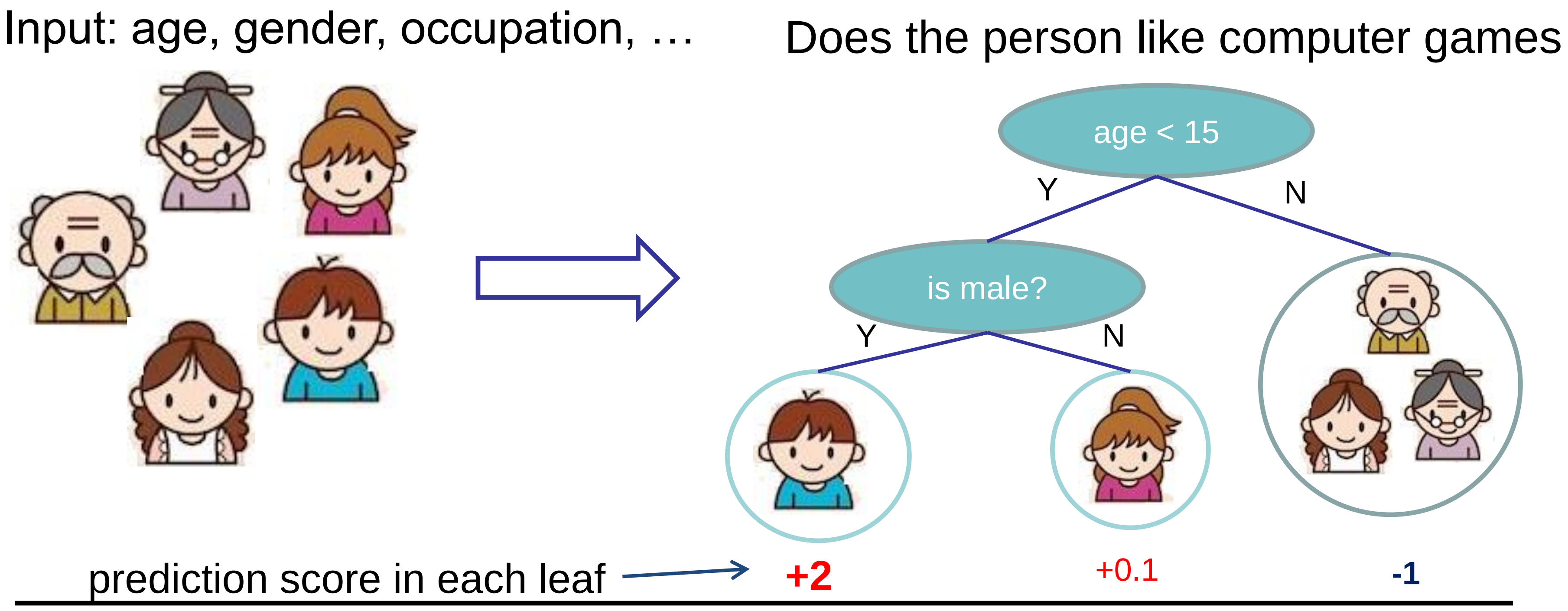
- Why do we want to contain two component in the objective?
- Optimizing training loss encourages **predictive** models
  - Fitting well in training data at least get you close to training data which is hopefully close to the underlying distribution
- Optimizing regularization encourages **simple** models
  - Simpler models tends to have smaller variance in future predictions, making prediction **stable**

Outline

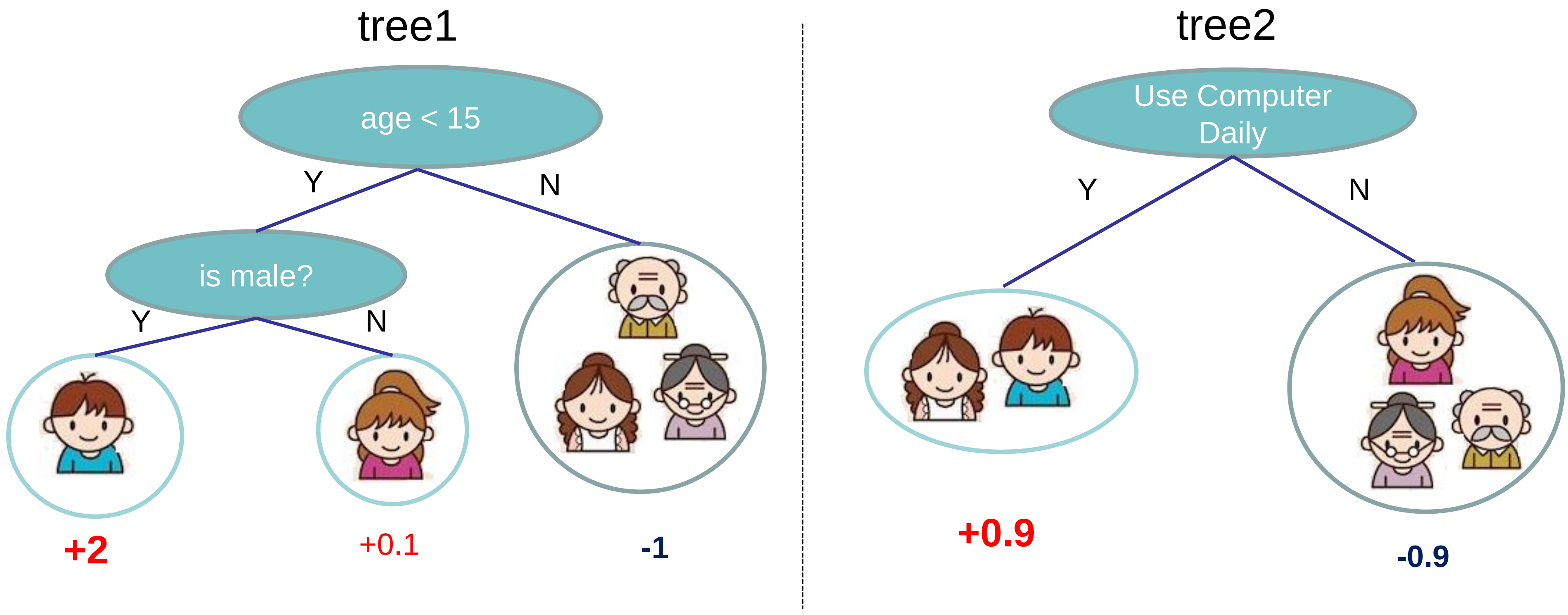
- Review of key concepts of supervised learning
- **Regression Tree and Ensemble (What are we Learning)**
- Gradient Boosting (How do we Learn)
- Summary

Regression Tree (CART)

- regression tree (also known as classification and regression tree):
  - Decision rules same as in decision tree
  - Contains one score in each leaf value


Input: age, gender, occupation, …
Does the person like computer games
age < 15
Y
N
is male?
Y
N
prediction score in each leaf
+2
+0.1
-1

Regression Tree Ensemble



f( ) = 2 + 0.9= 2.9   f( )= -1 - 0.9= -1.9

Prediction of is sum of scores predicted by each of the tree

Tree Ensemble methods

- Very widely used, look for GBM, random forest…
  - Almost half of data mining competition are won by using some variants of tree ensemble methods
- Invariant to scaling of inputs, so you do not need to do careful features normalization.
- Learn higher order interaction between features.
- Can be scalable, and are used in Industry

Put into context: Model and Parameters

- Model: assuming we have K trees

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}$$

Space of functions containing all Regression trees

Think: regression tree is a function that maps the attributes to the score

- Parameters
  - Including structure of each tree, and the score in the leaf
  - Or simply use function as parameters

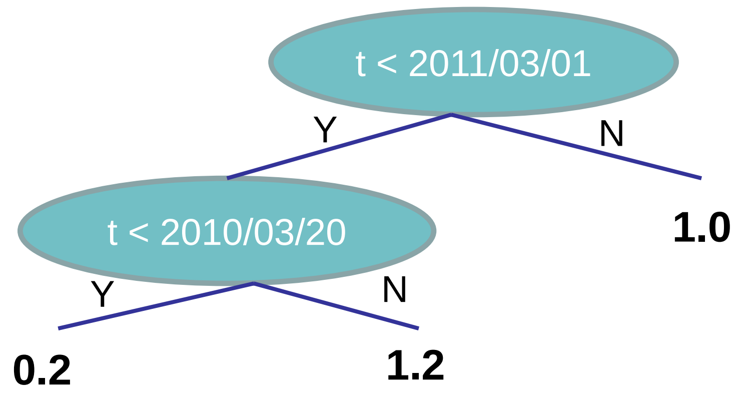
$$\Theta = \{f_1, f_2, \cdots, f_K\}$$

  - Instead learning weights in $\mathbf{R}^d$, we are learning functions(trees)

Learning a tree on single variable

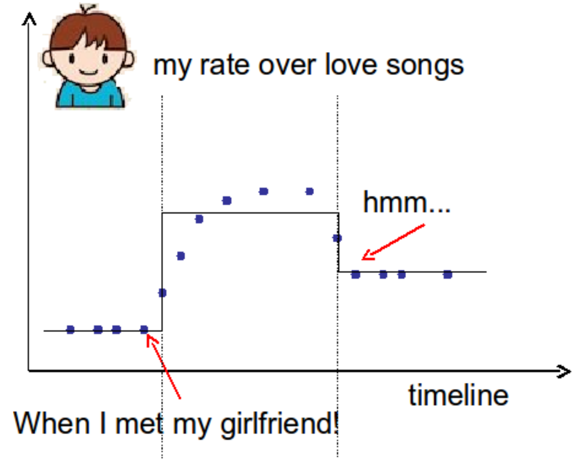
- How can we learn functions?
- Define objective (loss, regularization), and optimize it!!
- Example:
  - Consider regression tree on single input t (time)
  - I want to predict whether I like romantic music at time t

The model is regression tree that splits on time



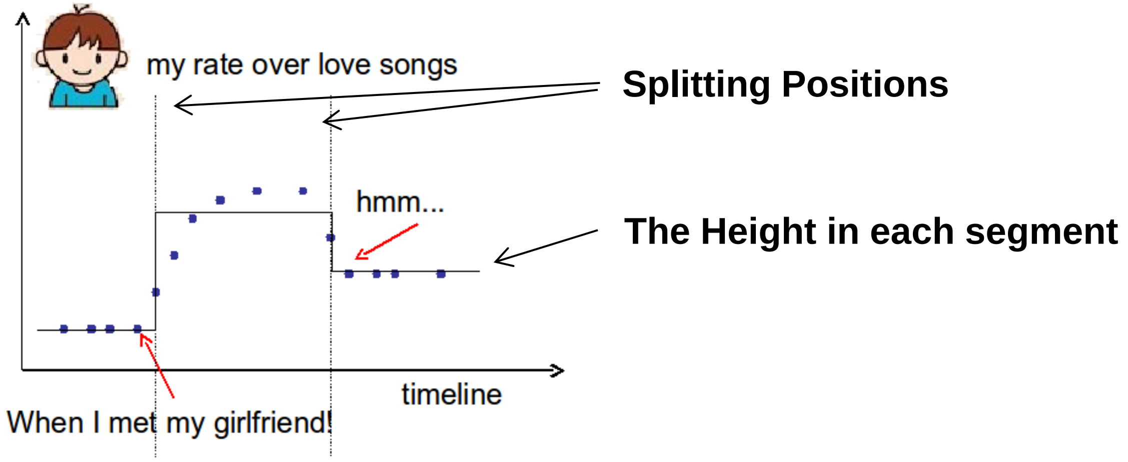
Equivalently

Piecewise step function over time



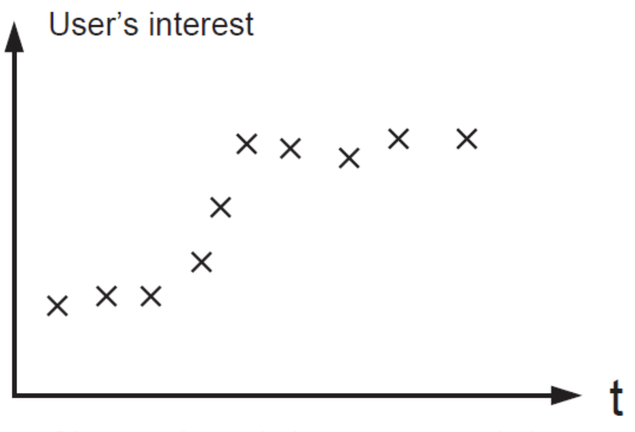
Learning a step function

- Things we need to learn

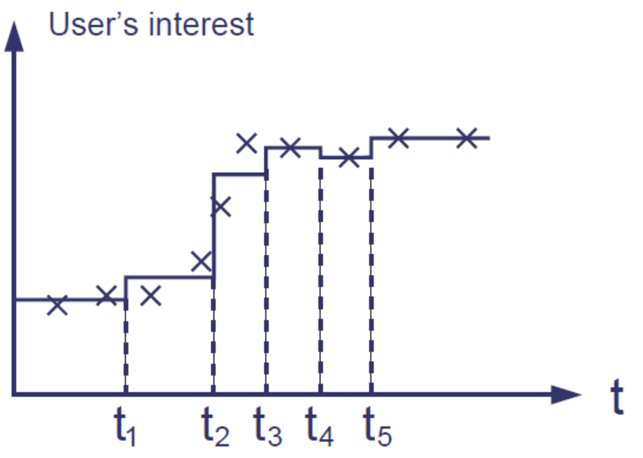


- Objective for single variable regression tree(step functions)
  - Training Loss: How will the function fit on the points?
  - Regularization: How do we define complexity of the function?
    - Number of splitting points, l2 norm of the height in each segment?

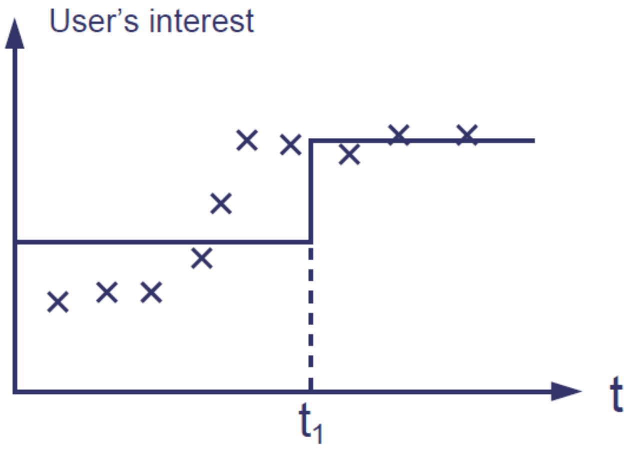
Learning step function (visually)



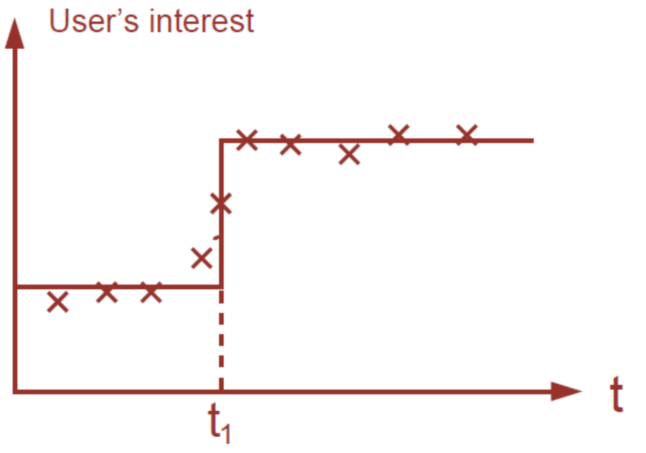
Observed user's interest on topic k against time t



☒ Too many splits, $\Omega(f)$ is high



☒ Wrong split point, $L(f)$ is high



☑ Good balance of $\Omega(f)$ and $L(f)$

Coming back: Objective for Tree Ensemble

- Model: assuming we have K trees

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}$$

- Objective

$$Obj = \underbrace{\sum_{i=1}^{n} l(y_i, \hat{y}_i)}_{\text{Training loss}} + \underbrace{\sum_{k=1}^{K} \Omega(f_k)}_{\text{Complexity of the Trees}}$$

- Possible ways to define Ω ?
  - Number of nodes in the tree, depth
  - L2 norm of the leaf weights
  - … detailed later

Objective vs Heuristic

- When you talk about (decision) trees, it is usually heuristics
  - Split by information gain
  - Prune the tree
  - Maximum depth
  - Smooth the leaf values
- Most heuristics maps well to objectives, taking the formal (objective) view let us know what we are learning
  - Information gain -> training loss
  - Pruning -> regularization defined by #nodes
  - Max depth -> constraint on the function space
  - Smoothing leaf values -> L2 regularization on leaf weights

Regression Tree is not just for regression!

- Regression tree ensemble defines how you make the prediction score, it can be used for
  - Classification, Regression, Ranking….
  - ….
- It all depends on how you define the objective function!
- So far we have learned:
  - Using Square loss $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
    - Will results in common gradient boosted machine
  - Using Logistic loss $l(y_i, \hat{y}_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})$
    - Will results in LogitBoost

Outline

- Review of key concepts of supervised learning
- Regression Tree and Ensemble (What are we Learning)
- **Gradient Boosting (How do we Learn)**
- Summary

Take Home Message for this section

- Bias-variance tradeoff is everywhere
- The loss + regularization objective pattern applies for regression tree learning (function learning)

- We want **predictive** and **simple** functions

- This defines what we want to learn (objective, model).
- But how do we learn it?
  - Next section

So How do we Learn?

- Objective: $\sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), f_k \in \mathcal{F}$
- We can not use methods such as SGD, to find f (since they are trees, instead of just numerical vectors)
- Solution: **Additive Training (Boosting)**
  - Start from constant prediction, add a new function each time

$$
\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}
$$

New function

Model at training round t

Keep functions added in previous round

Additive Training

- How do we decide which f to add?
  - Optimize the objective!!
- The prediction at round t is $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$

This is what we need to decide in round t

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^t \Omega(f_i) \\ &= \sum_{i=1}^n l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant \end{aligned}$$

Goal: find f_t to minimize this

- Consider square loss

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

This is usually called residual from previous round

Taylor Expansion Approximation of Loss

- Goal $Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant$
  - Seems still complicated except for the case of square loss
- Take Taylor expansion of the objective
  - Recall $f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2$
  - Define $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

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

- *If you are not comfortable with this, think of square loss*

$$g_i = \partial_{\hat{y}^{(t-1)}} (\hat{y}^{(t-1)} - y_i)^2 = 2(\hat{y}^{(t-1)} - y_i) \quad h_i = \partial^2_{\hat{y}^{(t-1)}} (y_i - \hat{y}^{(t-1)})^2 = 2$$

- Compare what we get to previous slide

Our New Goal

- Objective, with constants removed

$$\sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t)$$

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

- Why spending s much efforts to derive the objective, why not just grow trees …
  - Theoretical benefit: know what we are learning, convergence
  - **Engineering** benefit, recall the elements of supervised learning
    - g_i and h_i comes from definition of loss function
    - The learning of function only depend on the objective via g_i and h_i
    - Think of how you can separate modules of your code when you are asked to implement boosted tree for both square loss and logistic loss

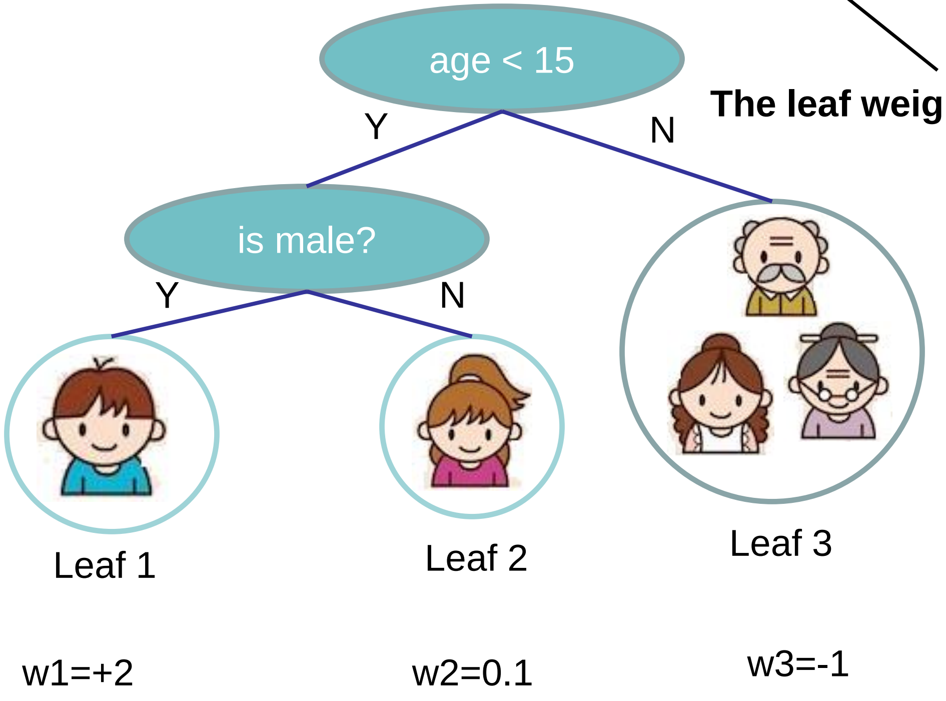
Refine the definition of tree

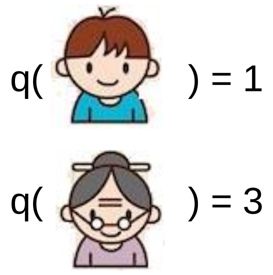
- We define tree by a vector of scores in leafs, and a leaf index mapping function that maps an instance to a leaf

$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q : \mathbf{R}^d \to \{1, 2, \cdots, T\}$$

The structure of the tree

The leaf weight of the tree



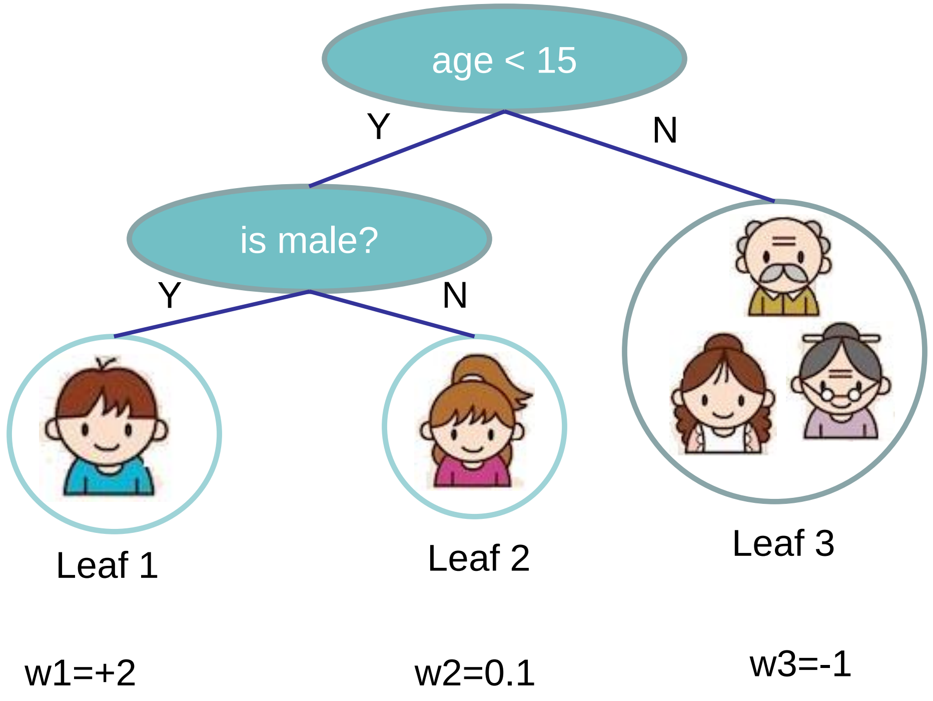


Define Complexity of a Tree (cont')

- Define complexity as (this is not the only possible definition)

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Number of leaves **L2 norm of leaf scores**



$$\Omega = \gamma 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

Revisit the Objectives

- Define the instance set in leaf j as $I_j = \{i | q(x_i) = j\}$
- Regroup the objective by each leaf

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

- This is sum of T independent quadratic functions

The Structure Score

- Two facts about single variable quadratic function

$$argmin_x \; Gx + \frac{1}{2}Hx^2 = -\frac{G}{H}, \;\; H > 0 \qquad \min_x \; Gx + \frac{1}{2}Hx^2 = -\frac{1}{2}\frac{G^2}{H}$$

- Let us define $G_j = \sum_{i \in I_j} g_i \;\; H_j = \sum_{i \in I_j} h_i$

$$\begin{aligned} Obj^{(t)} &= \sum_{j=1}^{T}\left[(\sum_{i \in I_j} g_i)w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda)w_j^2\right] + \gamma T \\ &= \sum_{j=1}^{T}\left[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2\right] + \gamma T \end{aligned}$$

- Assume the structure of tree (q(x)) is fixed, the optimal weight in each leaf, and the resulting objective value are

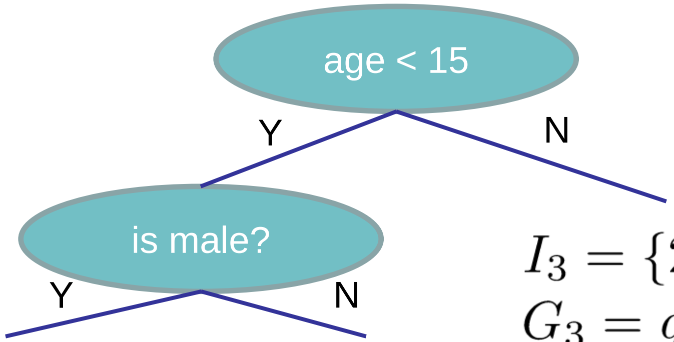
$$w_j^* = -\frac{G_j}{H_j + \lambda} \qquad Obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

$$\uparrow$$

This measures how good a tree structure is!

The Structure Score Calculation

Instance index		gradient statistics
1		g1, h1
2		g2, h2
3		g3, h3
4		g4, h4
5		g5, h5



$$I_1 = \{1\} \quad G_1 = g_1 \quad H_1 = h_1$$

$$I_2 = \{4\} \quad G_2 = g_4 \quad H_4 = h_4$$

$$I_3 = \{2, 3, 5\} \quad G_3 = g_2 + g_3 + g_5 \quad H_3 = h_2 + h_3 + h_5$$

$$Obj = -\sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

Searching Algorithm for Single Tree

- Enumerate the possible tree structures q
- Calculate the structure score for the q, using the scoring eq.

$$Obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$$

- Find the best tree structure, and use the optimal leaf weight

$$w_j^* = -\frac{G_j}{H_j+\lambda}$$

- But… there can be infinite possible tree structures..

Greedy Learning of the Tree

- In practice, we grow the tree greedily
  - Start from tree with depth 0
  - For each leaf node of the tree, try to add a split. The change of objective after adding the split is

The complexity cost by introducing additional leaf

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

the score of left child

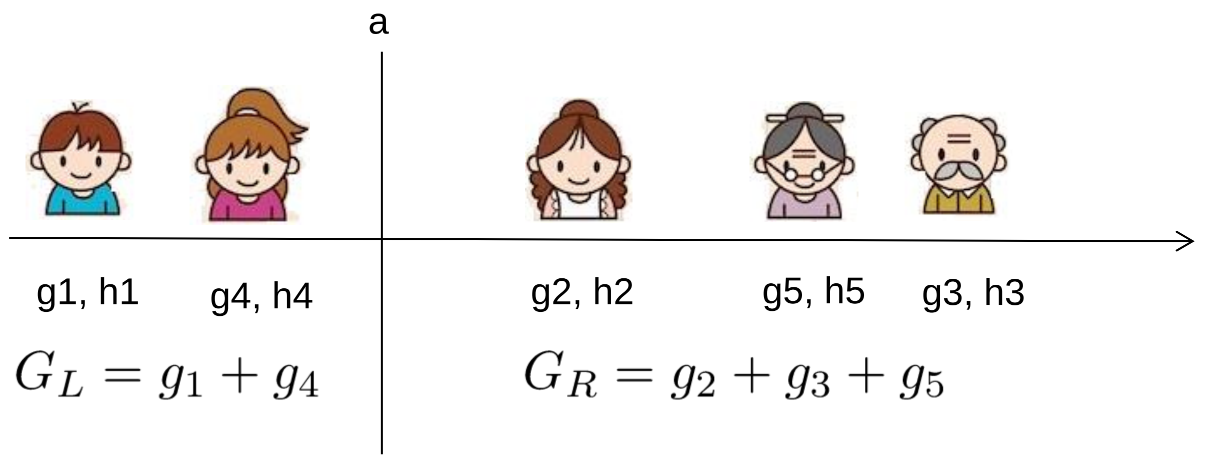
the score of if we do not split

the score of right child

  - Remaining question: how do we find the best split?

Efficient Finding of the Best Split

- What is the gain of a split rule $x_j < a$? Say x_j is age



- All we need is sum of g and h in each side, and calculate

$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

- Left to right linear scan over sorted instance is enough to decide the best split along the feature

An Algorithm for Split Finding

- For each node, enumerate over all features
  - For each feature, sorted the instances by feature value
  - Use a linear scan to decide the best split along that feature
  - Take the best split solution along all the features

- Time Complexity growing a tree of depth K
  - It is O(n d K log n): or each level, need O(n log n) time to sort
    There are d features, and we need to do it for K level
  - This can be further optimized (e.g. use approximation or caching the sorted features)
  - Can scale to very large dataset

What about Categorical Variables?

- Some tree learning algorithm handles categorical variable and continuous variable separately
  - We can easily use the scoring formula we derived to score split based on categorical variables.
- Actually it is not necessary to handle categorical separately.
  - We can encode the categorical variables into numerical vector using one-hot encoding. Allocate a #categorical length vector

$$z_j = \begin{cases} 1 & \text{if } x \text{ is in category } j \\ 0 & otherwise \end{cases}$$

  - The vector will be sparse if there are lots of categories, the learning algorithm is preferred to handle sparse data

Pruning and Regularization

- Recall the gain of split, it can be negative!

$$Gain = \underbrace{\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}} - \gamma$$

  - When **the training loss reduction** is smaller than **regularization**
  - Trade-off between simplicity and predictivness
- Pre-stopping
  - Stop split if the best split have negative gain
  - But maybe a split can benefit future splits..
- Post-Prunning
  - Grow a tree to maximum depth, recursively prune all the leaf splits with negative gain

Recap: Boosted Tree Algorithm

- Add a new tree in each iteration
- Beginning of each iteration, calculate

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

- Use the statistics to greedily grow a tree $f_t(x)$

$$Obj = -\frac{1}{2}\sum_{j=1}^{T} \frac{G_j^2}{H_j+\lambda} + \gamma T$$

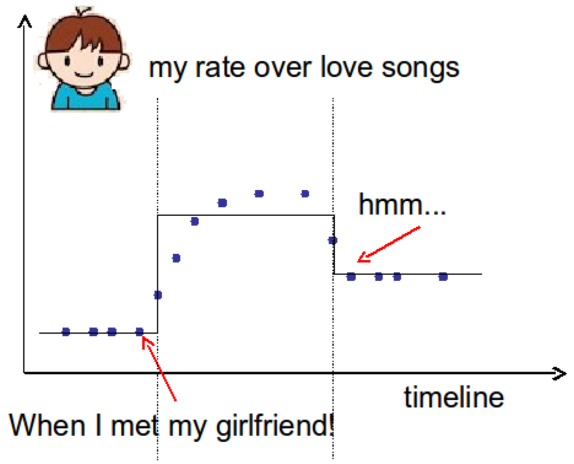
- Add $f_t(x)$ to the model $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$
  - Usually, instead we do $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$
  - ϵ is called step-size or shrinkage, usually set around 0.1
  - This means we do not do full optimization in each step and reserve chance for future rounds, it helps prevent overfitting

Outline

- Review of key concepts of supervised learning
- Regression Tree and Ensemble (What are we Learning)
- Gradient Boosting (How do we Learn)
- **Summary**

Questions to check if you really get it

- How can we build a boosted tree classifier to do weighted regression problem, such that each instance have a importance weight?
- Back to the time series problem, if I want to learn step functions over time. Is there other ways to learn the time splits, other than the top down split approach?



Questions to check if you really get it

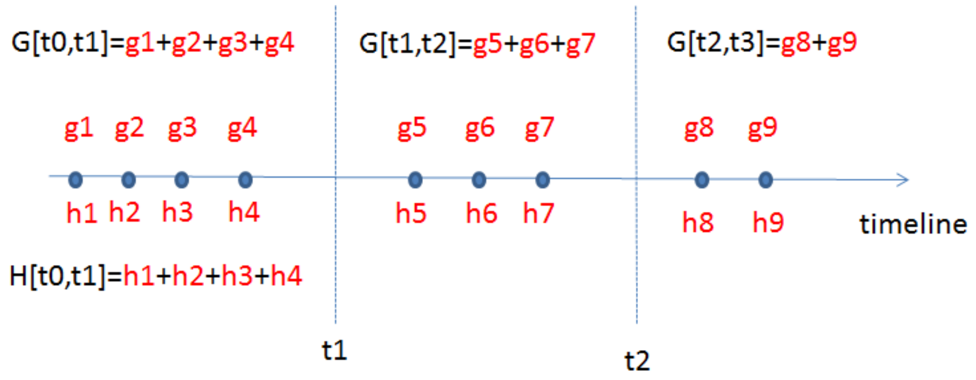
- How can we build a boosted tree classifier to do weighted regression problem, such that each instance have a importance weight?
  - Define objective, calculate g_i, h_i , feed it to the old tree learning algorithm we have for un-weighted version

$$l(y_i, \hat{y}_i) = \frac{1}{2} a_i (\hat{y}_i - y_i)^2 \qquad g_i = a_i(\hat{y}_i - y_i) \quad h_i = a_i$$

  - Again think of separation of model and objective, how does the theory can help better organizing the machine learning toolkit

Questions to check if you really get it

- Time series problem



- All that is important is the structure score of the splits

$$Obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$$

  - Top-down greedy, same as trees
  - Bottom-up greedy, start from individual points as each group, greedily merge neighbors
  - Dynamic programming, can find optimal solution for this case

Summary

- The separation between model, objective, parameters can be helpful for us to understand and customize learning models
- The bias-variance trade-off applies everywhere, including learning in functional space

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

- We can be formal about what we learn and how we learn. Clear understanding of theory can be used to guide cleaner implementation.

Reference

- Greedy function approximation a gradient boosting machine. *J.H. Friedman*
  - *First paper about gradient boosting*
- *Stochastic Gradient Boosting. J.H. Friedman*
  - *Introducing bagging trick to gradient boosting*
- *Elements of Statistical Learning. T. Hastie, R. Tibshirani and J.H. Friedman*
  - *Contains a chapter about gradient boosted boosting*
- Additive logistic regression a statistical view of boosting. *J.H. Friedman T. Hastie R. Tibshirani*
  - *Uses second-order statistics for tree splitting, which is closer to the view presented in this slide*
- Learning Nonlinear Functions Using Regularized Greedy Forest. *R. Johnson and T. Zhang*
  - *Proposes to do fully corrective step, as well as regularizing the tree complexity. The regularizing trick is closed related to the view present in this slide*

- Software implementing the model described in this slide: https://github.com/tqchen/xgboost

Implementation of XGBoost

1、防止过拟合：正则化、收缩步长

2、考虑了训练数据为稀疏值的情况，可以为缺失值或者指定的值指定分支的默认方向，这能大大提升算法的效率

3、在寻找最佳分割点时，采用近似“最佳”：仅从几个百分位数中寻找“最佳”的分割点

4、并行：将特征列排序后以块的形式存储在内存中，在迭代中可以重复使用；虽然boosting算法迭代必须串行，但是在处理每个特征列时可以做到并行

5、考虑了当数据量比较大、内存不够时怎么有效的使用磁盘，主要是结合多线程、数据压缩、分片的方法

Application Keys of XGBoost

- xgboost() (in xgboost package)
  - nrounds
  - objective
  - max_depth、min_child_weight
  - eta、gamma、lambda、alpha
  - colsample_bytree、subsample

XGBoost

Target		Keys	Scale	Missing Data
Class	Regre			
√	√	(1) Models (2) Computation	X	√

LightGBM

LightGBM

LightGBM: A Highly Efficient Gradient Boosting Decision Tree

Guolin Ke[1], **Qi Meng**[2], **Thomas Finley**[3], **Taifeng Wang**[1],
Wei Chen[1], **Weidong Ma**[1], **Qiwei Ye**[1], **Tie-Yan Liu**[1]
[1]Microsoft Research [2]Peking University [3] Microsoft Redmond
[1]{guolin.ke, taifengw, wche, weima, qiwye, tie-yan.liu}@microsoft.com;
[2]qimeng13@pku.edu.cn; [3]tfinely@microsoft.com;

Abstract

Gradient Boosting Decision Tree (GBDT) is a popular machine learning algorithm, and has quite a few effective implementations such as XGBoost and pGBRT. Although many engineering optimizations have been adopted in these implementations, the efficiency and scalability are still unsatisfactory when the feature dimension is high and data size is large. 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, we propose two novel techniques: *Gradient-based One-Side Sampling* (GOSS) and *Exclusive Feature Bundling* (EFB). With GOSS, we exclude a significant proportion of data instances with small gradients, and only use the rest to estimate the information gain. We prove that, since the data instances with larger gradients play a more important role in the computation of information gain, GOSS can obtain quite accurate estimation of the information gain with a much smaller data size. With EFB, we bundle mutually exclusive features (i.e., they rarely take nonzero values simultaneously), to reduce the number of features. We prove that finding the optimal bundling of exclusive features is NP-hard, but a greedy algorithm can achieve quite good approximation ratio (and thus can effectively reduce the number of features without hurting the accuracy of split point determination by much). We call our new GBDT implementation with GOSS and EFB *LightGBM*. Our experiments on multiple public datasets show that, LightGBM speeds up the training process of conventional GBDT by up to over 20 times while achieving almost the same accuracy.

1 Introduction

Gradient boosting decision tree (GBDT) [1] is a widely-used machine learning algorithm, due to its efficiency, accuracy, and interpretability. GBDT achieves state-of-the-art performances in many machine learning tasks, such as multi-class classification [2], click prediction [3], and learning to rank [4]. In recent years, with the emergence of big data (in terms of both the number of features and the number of instances), GBDT is facing new challenges, especially in the tradeoff between accuracy and efficiency. Conventional implementations of GBDT need to, for every feature, scan all the data instances to estimate the information gain of all the possible split points. Therefore, their computational complexities will be proportional to both the number of features and the number of instances. This makes these implementations very time consuming when handling big data.

LightGBM

Performance:

LightGBM speeds up the training process of conventional GBDT by up to over 20 times while achieving almost the same accuracy.

Main techniques:

- **Gradient-based One-Side Sampling (GOSS)**
- **Exclusive Feature Bundling (EFB)**

GOSS

Gradient-based One-Side Sampling (GOSS)

- **Motivation:** To reduce the size of the training data, a common approach is to down sample the data instances.

- **Previous methods:**
  - (1) Random sampling - SGB
  - (2) Sampling according to weights in Adaboost

- **Problems:**
  - (1) Random sampling is inefficient.
  - (2) There are no weights under the framework of GBDT.

GOSS

Basic idea of GOSS: sampling according to gradients

- Keep all the instances with large gradients.
- Perform random sampling on the instances with small gradients.
- Adjust the weights of the sampled small gradients.

GOSS

Review of GBDT:

For GBDT, the information gain is usually measured by the variance after splitting, i.e., the variance gain:

$$V(d) = \frac{1}{n}\left(\frac{(\Sigma_{x_i \leq d}\ g_i)^2}{n_l(d)} + \frac{(\Sigma_{x_i > d}\ g_i)^2}{n_r(d)}\right)$$

where $n_l(d) = \sum I[x_i \leq d]$, $n_r(d) = \sum I[x_i > d]$ **and**

$n = n_l + n_r$.

Note that larger g_i's contribute more to the variance gain.

GOSS

Review of GBDT:

For a regression tree, the objective is usually measured by mean squared error (i.e., the variance):

$$V_0 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2.$$

For a split, the mean squared error becomes:

$$\begin{aligned} V_{split} &= \frac{n_l}{n}V_l + \frac{n_r}{n}V_r \\ &= \frac{n_l}{n}\cdot\frac{1}{n_l}\Sigma_{x_i\le d}(y_i - \bar{y}_l)^2 + \frac{n_r}{n}\cdot\frac{1}{n_r}\Sigma_{x_i>d}(y_i - \bar{y}_r)^2 \\ &= \frac{1}{n}\left(\Sigma_{x_i\le d}y_i^2 - n_l(\bar{y}_l)^2 + \Sigma_{x_i>d}y_i^2 - n_r(\bar{y}_r)^2\right) \\ &= \frac{1}{n}\left(\sum_{i=1}^{n} y_i^2 - n_l(\bar{y}_l)^2 - n_r(\bar{y}_r)^2\right). \end{aligned}$$

GOSS

The gain of the split is:

$$\begin{aligned} V_0 - V_{split} &= \frac{1}{n}\left(n_l(\bar{y}_l)^2 + n_r(\bar{y}_r)^2 - n(\bar{y})^2\right) \\ &\propto \frac{1}{n}\left(n_l(\bar{y}_l)^2 + n_r(\bar{y}_r)^2\right). \end{aligned}$$

Recall in GBDT, at Step m, the dependent variable is ε_m, and $\varepsilon_m = -g_m$. Therefore, the variance gain is

$$V_0 - V_{split} = \frac{1}{n}\left(n_l(\bar{g}_l)^2 + n_r(\bar{g}_r)^2\right).$$

GOSS

The GOSS method:

- **(1) Keep the top a $\times 100\%$ instances with the larger gradients (instance subset A)**
- **(2) Randomly sample b $\times 100\%$ instances from the remaining $(1-a)$ $\times 100\%$ instances (subset B)**
- **(3) Estimate the variance gain based on sets A and B:**

$$\hat{V}(d) = \frac{1}{n}\left(\frac{\left(\Sigma_{x_i \in A_l}\, g_i + \frac{1-a}{b}\Sigma_{x_i \in B_l}\, g_i\right)^2}{n_l(d)} + \frac{\left(\Sigma_{x_i \in A_r}\, g_i + \frac{1-a}{b}\Sigma_{x_i \in B_r}\, g_i\right)^2}{n_r(d)}\right)$$

GOSS

Performances:

- **Accuracy:**
  - **(1) It is proven that $|\hat{V}(d) - V(d)|$ is bounded with some probability.**
  - **(2) Sampling will increase the diversity of the base learners, which potentially helps to improve the generalization performance.**
  - **(3) The LightGBM vs SGB experiments show that GOSS has better performance than stochastic sampling when using the same sampling ratio.**

GOSS

Performances:

- **Speed:** **The LightGBM vs EFB-only experiments show that GOSS may bring nearly $2\times$ speed-up with using $10\% - 20\%$ data.**

Note that the speed-up brought by GOSS is not linearly correlated with the percentage of sampled data, since it retains some computation on the full data, such as the predictions and gradients.

EFB

Exclusive Feature Bundling (EFB)

- **Motivation:** To reduce the dimensionality on mutually exclusive features, i.e., the features which never take nonzero values simultaneously.
- **Previous methods:** Ignoring the features with zero values-XGboost with the pre-sorted algorithm.
- **Problems:**
  - LightGBM uses histogram-based algorithm, which does not have efficient sparse optimization solutions.
  - Reduce the optimal bundling problem to the graph coloring problem which, however, is NP-hard.

EFB

EBF:

- **Ideas:**
  - (1) Use a greedy algorithm.
  - (2) Allow a small fraction of conflicts.
- **Steps (processed only once before training):**
  - Order by the count of nonzero values, which is similar to ordering by degrees since more nonzero values usually lead to higher probability of conflicts.
  - Check each feature in the ordered list, and either assign it to an existing bundle with a small conflict, or create a new bundle.
  - Merge the features in the same bundle by letting exclusive features reside in different bins.

EFB

Performances:

- **EFB merges many sparse features (both the one-hot coding features and implicitly exclusive features) into much fewer features.**
- **EFB is very effective to leverage sparse property in the histogram-based algorithm, and it can bring a significant speed-up for GBDT training process.**

LightGBM

Features of LightGBM:

(https://lightgbm.readthedocs.io/en/latest/Features.html)

- **Optimization in speed and memory usage (histogram-based algorithms):**
  - Reduced cost of calculating the gain for each split
  - Use histogram subtraction for further speedup
  - Reduce memory usage
  - Reduce communication cost for parallel learning
  - Sparse optimization

LightGBM

- Optimization in accuracy:
  - Leaf-wise (best-first) tree growth
  - Optimal split for categorical features
- Optimization in Network Communication (state-of-art algorithms):
- Optimization in Parallel Learning
  - Data parallel
  - Feature parallel
  - Voting parallel (two-stage voting)

LightGBM

Advantages of LightGBM:

(https://github.com/Microsoft/LightGBM)

- **Fast training speed and high efficiency**
- **Lower memory usage**
- **Better accuracy**
- **Parallel learning supported (innately distributed)**
- **Capability of handling large-scaling data**
- **Support categorical features directly**

4. Stacking

- **Bagging, random forest, boosting**

  --Usually combine similar classifiers

- **Stacking, Bayes model averaging, general ensemble methods**

  --May combine different models

Stacking

- **Stacking was invented by Wolpert (1992) and studied by Breiman (1996), among others.**

Stacking

- Stacking is an adaption of cross-validating (CV) to model averaging as the models are weighted by coefficients derived from CV
- In contrast to bagging, stacking puts weights of varying sizes on models rather than pooling over repeated evaluations of the same model class
- Stacking can be thought of as a version of Bayes model average where the estimated weights correspond to priors that downweight complex or ill-fitting models

Basic Idea of Stacking

- Suppose there are K models in which each model has some parameters that must be estimated. When plugging in estimators, write $\widehat{f}_k(\mathbf{x}) = f_k(\mathbf{x}|\widehat{\theta}_k)$ for the model used to get prediction. Then the stacking prediction at x is

$$\widehat{f}_{stack}(\mathbf{x}) = \sum_{k=1}^{K} \widehat{w}_k \widehat{f}_k(\mathbf{x})$$

- Weights: let $\widehat{f}_k^{(-i)}(\mathbf{x})$ be the prediction at x using model k with the ith observation removed. Then

$$(\widehat{w}_1, \ldots, \widehat{w}_K) = arg\,\text{min} \sum_{i=1}^{n} [y_i - \sum_{k=1}^{K} w_k \widehat{f}_k^{(-i)}(\mathbf{x}_i)]^2$$

(low weights for models with poor LOO accuracy)

Application Keys of Stacking

- Tuning parameters: K
- caretStack() (in caretEnsemble package)
  - trControl
  - methodList
  - tuneList
  - method

Stacking

Target		Keys	Scale	Missing Data
Class	Regre			
√ (Binary Only)	√	(1) Models (2) Stacking method	X	X

Remarks of Ensemble Methods

The methods are diverse and invite freewheeling applications. One could:

- Simple blending:

For a classification problem, build an odd number models, predict with each model, and predict the class by majority vote.

Remarks of Ensemble Methods

Netflix Recommendations: Beyond the 5 stars

We looked at the two underlying algorithms with the best performance in the ensemble: *Matrix Factorization* (which the community generally called SVD,*Singular Value Decomposition*) and *Restricted Boltzmann Machines* (RBM). SVD by itself provided a 0.8914 RMSE, while RBM alone provided a competitive but slightly worse 0.8990 RMSE. A linear blend of these two reduced the error to 0.88.

https://www.techdirt.com/blog/innovation/articles/20120409/03412518422
http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html

Remarks of Ensemble Methods

Use at your own risk!

- Bag a boosted classifier
- Boost a bagged classifier
- First **stack** classifiers of diverse forms, say trees and KNNs, and then **boost** the **bagged** version
- First take a **Bayes Model Average** of **stacked** trees and then **boost** the result.