Introduction to Boosting

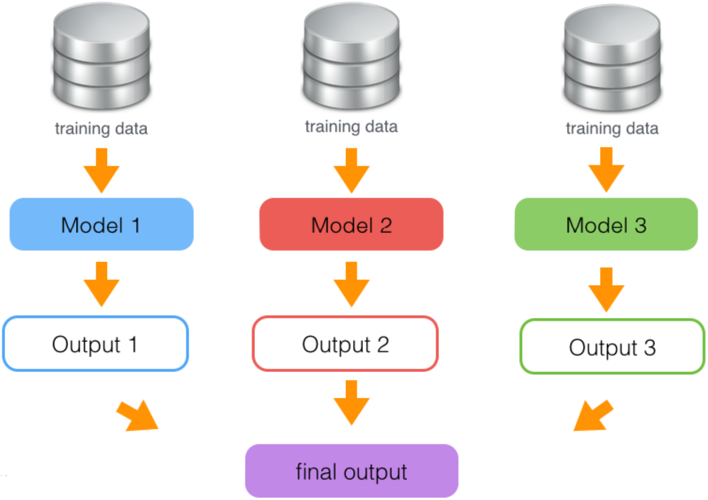
Predrag Tadić

School of Electrical Engineering
University of Belgrade

MLA@MATF, November 14, 2018

Outline

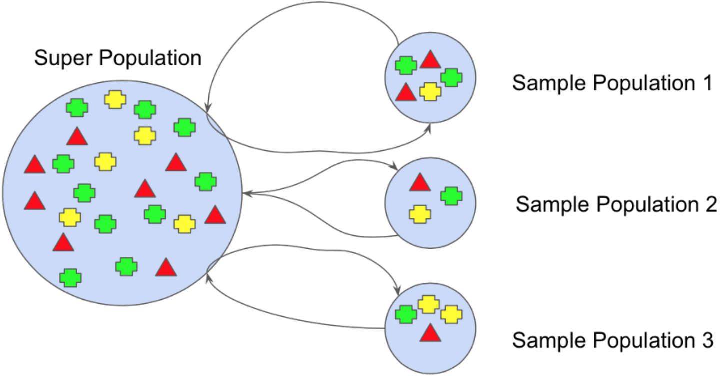
Ensemble (committee)



[dataversioncontrol.com]

Bootstraping

- Sampling N out of N with replacement, M times.
- 30% of examples are not chosen in each sample.



[hackernoon.com]

Weak learner, strong learner

Weak learner simple classifier, slightly better than guessing

Strong learner can achieve arbitrary accuracy with enough data

[Kidsday staff artist / Maggie Flaherty, Merrick]

Weak learner, strong learner

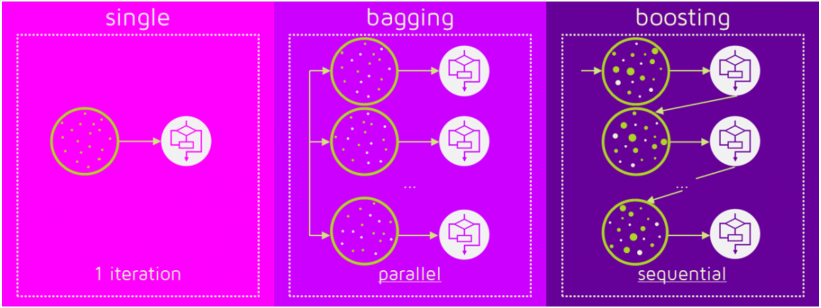
In the PAC framework

- Notation

$\{\mathbf{x}_i, y_i\}_{i=1}^N$	training set
P	distribution of training set
$f(\mathbf{x}) = y$	true hypothesis
$h(\mathbf{x}) = \hat{y}$	learned hypothesis
$\Pr_P[h(\mathbf{x}) \neq f(\mathbf{x})]$	generalization error

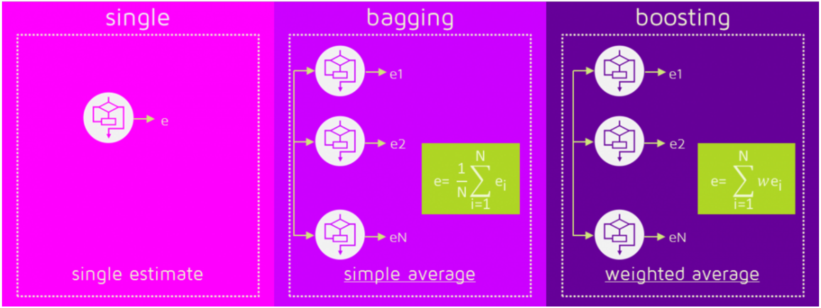
- Strong learner (SL)
  - for any $P, f, \delta, \epsilon \geq 0$
  - for large enough N
  - outputs a classifier with $\Pr_P[h(\mathbf{x}) \neq f(\mathbf{x})] \leq \epsilon$
  - with probability at least $1 - \delta$
- Weak learner (WL)
  - for any P, f, δ and some $0 \leq \epsilon < 1/2$
  - for large enough N
  - outputs a classifier with $\Pr_P[h(\mathbf{x}) \neq f(\mathbf{x})] \leq \epsilon$
  - with probability at least $1 - \delta$

Bagging & Boosting: training



[quantdare.com]

Bagging & Boosting: decision



[quantdare.com]

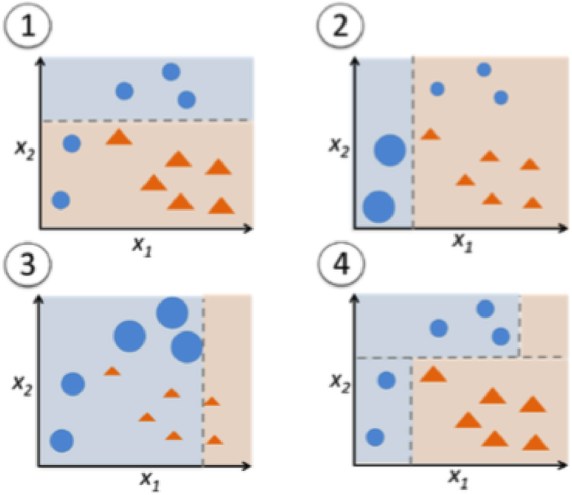
History

- 1989 Does weak learnability imply strong learnability [KV94]?
- 1990 3 weak learners on 3 modified distributions [Sch90]
- 1995 Boosting by majority [Fre95]
- 1996 AdaBoost [FS96]
- 2001 Gradient Boosting [Fri01]
- 2016 XGBoost [CG16]

First boosting algorithm [Sch90]

- Requires a continuous stream of labeled data.
- Learns 3 hypothesis on 3 modified distributions.
- Outputs their majority vote.
- Algorithm:
    1. Randomly choose first first N samples.
       Use them to learn h_1.
    2. Choose next batch so that $N/2$ samples are misclassified by h_1.
       Use it to learn h_2.
    3. Choose next batch of N samples so that h_1 and h_2 disagree.
       Use it to learn h_3.
    4. Apply recursively.

AdaBoost



[sebastianraschka.com]

AdaBoost

Preliminaries

$h_l(\mathbf{x})$	l-th WL, $h_l(\mathbf{x}) = \pm 1$ (e.g. stump or perceptron)
α_l	voting weight of l-th WL
$\omega_{l,i}$	weight of i-th example in l-th iteration, $\sum_{i=1}^{N} \omega_{l,i} = 1$

- Hypothesis (strong learner) after k iterations

$$H_k(\mathbf{x}) = \frac{1}{2} \sum_{l=1}^{k} \alpha_l h_l(\mathbf{x})$$

- In iteration k, min exponential loss w.r.t. α_k and $h_k(\mathbf{x})$ only

$$\begin{aligned} E_k &= \sum_{i=1}^{N} \exp\left[-y_i H_k(\mathbf{x}_i)\right] \\ &= \sum_{i=1}^{N} \underbrace{\exp\left[-y_i H_{k-1}(\mathbf{x}_i)\right]}_{\omega_{k,i}} \exp\left[-\frac{1}{2} y_i \alpha_k h_k(\mathbf{x}_i)\right] \end{aligned}$$

AdaBoost

Training

- Initialization: $\omega_{1,1} = \cdots = \omega_{1,N} = 1/N$
- For $k = 1, \ldots, K$ (until convergence)
  1. Train weak learner
  $$\text{choose } h_k \text{ to minimize } J_k = \sum_{i=1}^{N} \omega_{k,i} \mathbb{1}\{h_k(\mathbf{x}_i) \neq y_i\}$$
  2. Compute its voting weight
  $$\epsilon_k = \sum_{i=1}^{N} \omega_{k,i} \mathbb{1}\left\{h_k(\mathbf{x}_i) \neq y_i\right\} \qquad \text{(weighted error)}$$
  $$\alpha_k = \ln \frac{1 - \epsilon_k}{\epsilon_k} \qquad \text{(voting weight)}$$
  3. Update sample weights for next iteration
  $$\omega_{k+1,i} \propto \omega_{k,i} e^{\alpha_k \mathbb{1}\{h_k(\mathbf{x}_i) \neq y_i\}}, \qquad \sum_{i=1}^{N} \omega_{k+1,i} = 1$$

AdaBoost

Convergence

- Loss is an upper limit on training error

$$\hat{\epsilon}_k \triangleq \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left\{H_k\left(\mathbf{x}_i\right) y_i < 0\right\} \le \frac{E_k}{N}$$

- If weighted error is $\le \frac{1}{2} - \delta$ for each WL

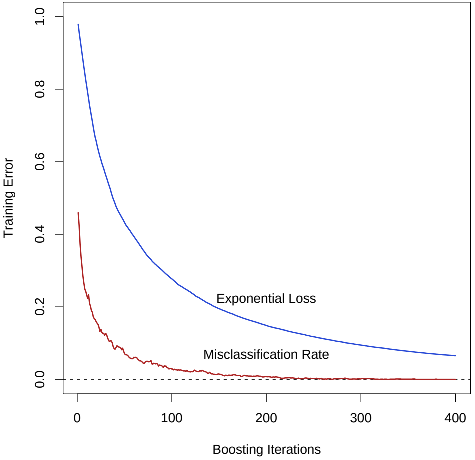
$$E_k \le \sqrt{1-4\delta^2} E_{k-1} \le \left(1-4\delta^2\right)^{k/2} N \qquad (E_0 \le N)$$

- Both the loss and the training error are always decreasing!
- Zero training error after finite number of iterations

$$\hat{\epsilon}_k = 0 \quad \text{for} \quad k \ge -2\frac{\ln N}{\ln(1-4\delta^2)}$$

AdaBoost

Convergence



[HTF09]

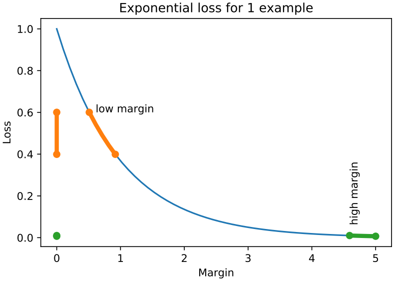
AdaBoost I

Margins & Overfitting

- Margin in boosting iteration k for example i

$$\gamma_{k,i} \triangleq y_i H_k(\mathbf{x}_i)$$

- Assume zero training error: $\gamma_{k,i} > 0, \forall i$
- Exponential loss $E_k = \sum_{i=1}^{N} e^{-\gamma_{k,i}}$ can still be reduced!
- Loss reduces more sharply for examples with smaller $\gamma_{k,i}$



AdaBoost II

Margins & Overfitting

- AdaBoost tends to increase worst-case margin $\min_i \gamma_{k,i}$
- How does AdaBoost avoid overfitting?
  - Stagewise addition of new learners makes learning slow
  - Impact of change is localized as iterations procees
  - Worst-case margin is pushed up (?)

AdaBoost

Why exponential loss?

- Expected exponential loss is minimized for

$$H^*(\mathbf{x}) = \arg\min_{H(\mathbf{x})} \mathrm{E}_{Y\,|\,\mathbf{x}}\, e^{-YH(\mathbf{x})}$$

- For binary classification with $Y = \pm 1$

$$\mathrm{E}_{Y\,|\,\mathbf{x}}\, e^{-YH(\mathbf{x})} = \Pr(Y = 1\,|\,\mathbf{x}) e^{-H(\mathbf{x})} + \Pr(Y = -1\,|\,\mathbf{x}) e^{H(\mathbf{x})}$$

- Differentiating w.r.t $H(\mathbf{x})$ and setting to zero gives

$$H^*(\mathbf{x}) = \frac{1}{2} \ln \frac{\Pr(Y = 1\,|\,\mathbf{x})}{\Pr(Y = -1\,|\,\mathbf{x})}$$

- Now, assume $Y \sim \text{Bernoulli}(\phi(\mathbf{x}))$ with

$$\phi(\mathbf{x}) = \frac{1}{1 + e^{-H(\mathbf{x})}}$$

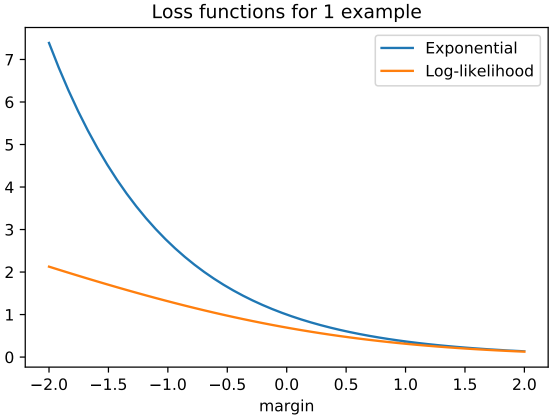
- Negative log-likelihood loss is given by

$$-l\left(H(\mathbf{x})\right) = -\ln\left(1 + e^{-YH(\mathbf{x})}\right)$$

- Population minimizer is the same as for exponential loss

$$\underset{H(\mathbf{x})}{\arg\min}\, \mathrm{E}_{Y\,|\,\mathbf{x}}\, e^{-YH(\mathbf{x})} = \underset{H(\mathbf{x})}{\arg\max}\, \mathrm{E}_{Y\,|\,\mathbf{x}}\, l\left(H(\mathbf{x})\right)$$

- Equivalence does not hold for finite data sets!



- Exponential loss puts more emphasis on misclassified examples
- Log-likelihood loss is more robust if
  - Bayes error rate is high
  - there are mislabeled data

Real AdaBoost [FHT00]

- Initialization: $\omega_1^{(1)} = \cdots = \omega_1^{(N)} = 1/N$
- For $k = 1, \ldots, K$ (until convergence)
  1. Fit classifier to target
     $$p_k(\mathbf{x}) = \hat{P}_\omega(Y = 1 \,|\, \mathbf{x})$$
  2. k-th weak learner outputs
     $$h_k(\mathbf{x}) = \frac{1}{2} \ln \frac{p_k(\mathbf{x})}{1 - p_k(\mathbf{x})}$$
  3. Update and re-normalize the weights
     $$\omega_{k+1,i} \propto \omega_{k,i} \exp\left[-y_i h_k(\mathbf{x}_i)\right], \qquad \sum\nolimits_{i=1}^{N} \omega_{k+1,i} = 1$$
- Ensemble output is
  $$H_K(\mathbf{x}) = \operatorname{sign}\left(\sum\nolimits_{k=1}^{K} h_k(\mathbf{x})\right)$$

LogitBoost [FHT00]

- Additive logistic regression models.
- Newton optimization of the Bernoulli log-likelihood.
- Start with $H(\mathbf{x}) = 0$, $\omega_{1:N} = 1/N$ and $p(\mathbf{x}_i) = 1/2$
- At iteration k, compute the weights and "working responses"

$$\omega_i = p(\mathbf{x}_i)\,(1 - p(\mathbf{x}_i))\,, \quad z_i = \min\left\{\frac{\mathbb{1}\{y_i = 1\} - p(\mathbf{x}_i)}{\omega_i}, z_{\max}\right\}$$

- Find $h_k(\mathbf{x})$ via weighted least-squares

$$h_k(\mathbf{x}) = \underset{h(\mathbf{x})}{\arg\min} \sum_{i=1}^{N} \omega_i \left[z_i - h(\mathbf{x}_i)\right]^2$$

- Update strong learner and probabilities

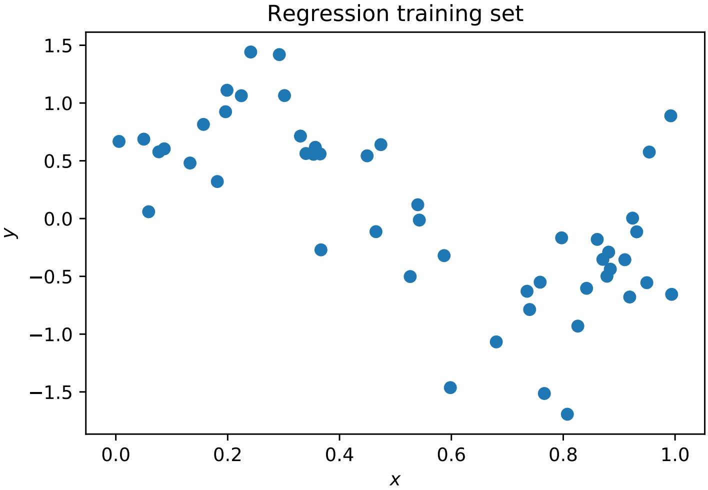
$$H(\mathbf{x}) \leftarrow H(\mathbf{x}) + \frac{1}{2} h_k(\mathbf{x}), \quad p(\mathbf{x}) \leftarrow \frac{e^{H(\mathbf{x})}}{e^{-H(\mathbf{x})} + e^{H(\mathbf{x})}}$$

Other AdaBoost modifications

- Gentle AdaBoost [FHT00]
  - Real AdaBoost + Newton steps
  - weighted least-squares regression instead of Pr estimates
  - more stable: no computation of log-ratios
- LPBoost [DBST02]
  - maximizes margin between classes
  - learning is formulated as a linear programming problem
  - totally corrective: weights of all past WLs are updated
- Brown Boost [Fre01]
  - "gives up" on repeatedly misclassified examples
  - robust to misslabeled datasets
- Many many more [FF12]

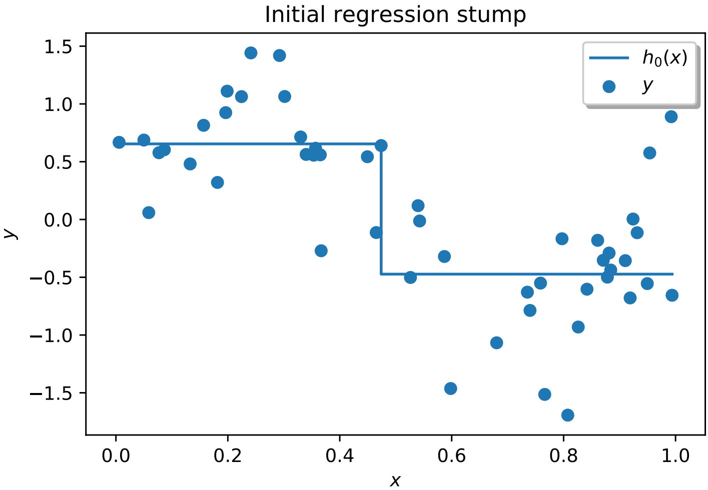
Gradient Boosting I

Toy example: sinusoidal regression



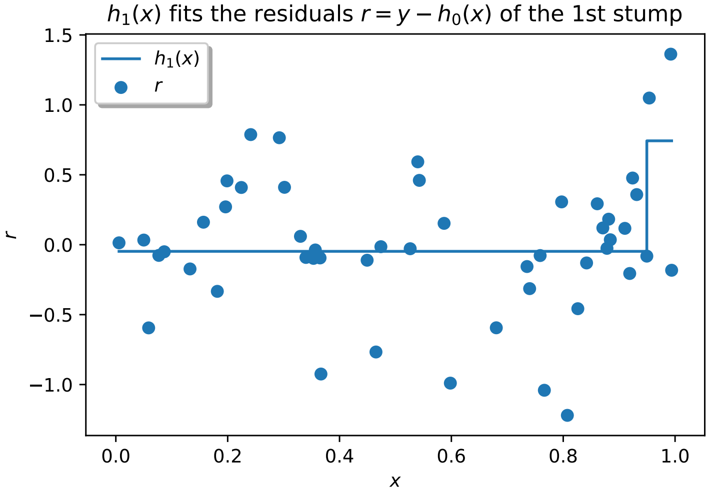
Gradient Boosting II

Toy example: sinusoidal regression



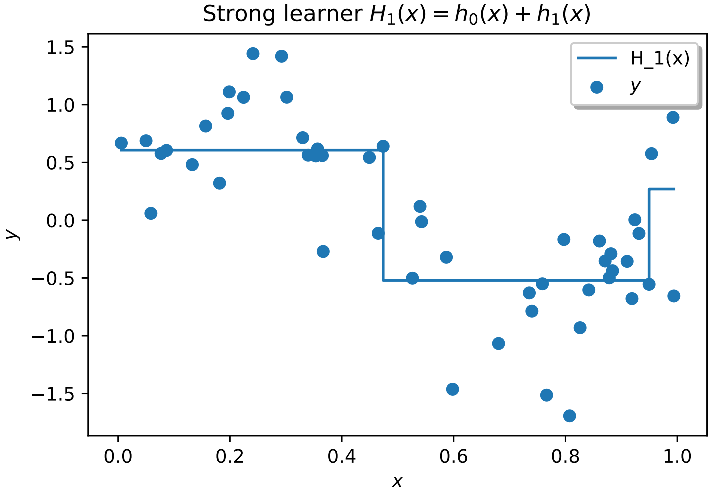
Gradient Boosting III

Toy example: sinusoidal regression



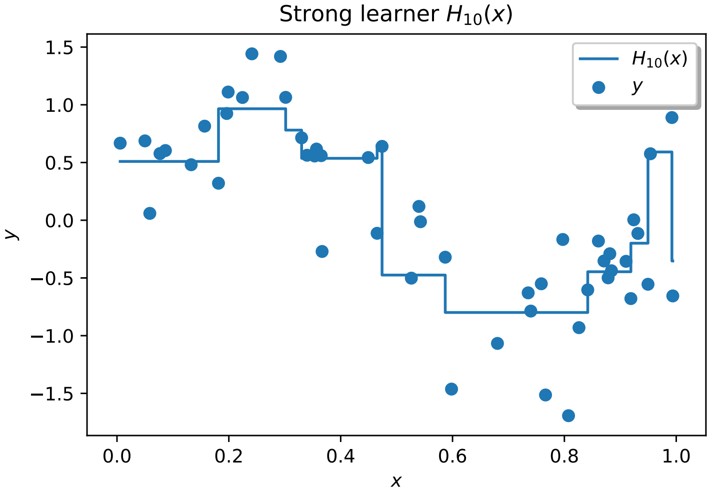
Gradient Boosting IV

Toy example: sinusoidal regression



Gradient Boosting V

Toy example: sinusoidal regression



Why does residual fitting work?

- Typical ML task: find $H(\mathbf{x})$ to minimize loss $L(y, H(\mathbf{x}))$.
- Generally unfeasible. Let's try a stagewise additive approach.
- Start with some simple $H(\mathbf{x}) = h_0(\mathbf{x})$ (e.g. regression stump).
- Add $h_1(\mathbf{x})$ to minimize resulting loss:

$$h_1^*(\mathbf{x}) = \arg\min_{h(\mathbf{x})} L\left[y, H(\mathbf{x}) + h(\mathbf{x})\right]$$

- Gradient tells us where to go! Ideally,

$$\begin{aligned} g(\mathbf{x}) &\triangleq \left[\frac{\partial L(y,h)}{\partial h}\right]_{h=H(\mathbf{x})} \\ h_1(\mathbf{x}) &= -g(\mathbf{x}) && \text{(optimal direction)} \\ \alpha_1 &= \arg\min_{\alpha} L\left[y, H(\mathbf{x}) + \alpha h_1(\mathbf{x})\right] && \text{(optimal step size)} \end{aligned}$$

- But loss is evaluated on $\{y_i, \mathbf{x}_i\}_{i=1}^N$ and setting

$$h_1(\mathbf{x}_i) = -g(\mathbf{x}_i) \quad \text{simultaneously for each } i$$

  is too hard (and would amount to overfitting, anyway)
- Approximate solution: try to fit the negative gradient

$$\text{train } h_1(\mathbf{x}) \text{ to minimize } \sum_{i=1}^{N} [-g(\mathbf{x}_i) - h_1(\mathbf{x}_i)]^2$$

  i.e. do a regression with negative gradient as target.
- For our sinusoidal regression toy example

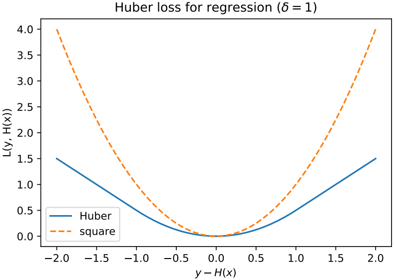
$$L[y, H(\mathbf{x})] = \frac{1}{2}[y - H(\mathbf{x})]^2$$
$$-g(\mathbf{x}) = y - H(\mathbf{x})$$

  This is why residual fitting works!

Typical loss functions

- Huber loss is less sensitive to outliers

$$L\left[y, H(\mathbf{x})\right] = \begin{cases} \left(y - H(\mathbf{x})\right)^2 /2, & |y - H(\mathbf{x})| \leq \delta \\ \delta \left(|y - H(\mathbf{x})| - \delta\right) & \end{cases}$$



- What about classification? Cross-entropy loss.

Gradient tree boosting

0. Start with $H_0(\mathbf{x}) = \arg\min_\chi \sum_{i=1}^N L(y_i, \chi) = \text{const.}$
1. For $k = 1, \ldots, K$ (until convergence)
   a) Compute "pseudo-residuals" $r_{k,i} = -g(\mathbf{x}_i)$
   b) Fit a regression tree on $\{\mathbf{x}_i, r_{k,i}\}$. This partitions input space into regions $R_{k,1}$, ..., R_{k,J_k}
   c) Compute best output for each region

$$\chi_{k,j} = \arg\min_\chi \sum_{\mathbf{x}_i \in R_{k,j}} L\left[y_i, H_{k-1}(\mathbf{x}_i) + \chi\right]$$

   d) Update strong learner

$$H_k(\mathbf{x}) = H_{k-1}(\mathbf{x}) + \sum_{j=1}^{J_k} \chi_{k,j} \mathbb{1}\{\mathbf{x} \in R_{k,j}\}$$

2. Output $H_K(\mathbf{x})$ as final model.

Gradient tree boosting for classification

- Similar as for regression.
- $M-1$ trees for M classes, outputting $f_{1:M-1}(\mathbf{x})$

$$
\begin{aligned}
p_m(\mathbf{x}) &= \hat{P}(Y = m \mid \mathbf{x}) \\
&= \begin{cases} \dfrac{e^{f_m(\mathbf{x})}}{1 + \sum_{l=1}^{M-1} e^{f_l(\mathbf{x})}}, & m = 1, \ldots, M-1 \\ 1 - \sum_{l=1}^{M-1} p_l(\mathbf{x}), & m = M \end{cases}
\end{aligned}
$$

- Cross-entropy (deviance) loss

$$
\begin{aligned}
L(y, \mathbf{p}(\mathbf{x})) &= -\ln p_y(\mathbf{x}) \\
-\frac{\partial L(y, \mathbf{p}(\mathbf{x}))}{\partial f_i(\mathbf{x})} &= \mathbb{1}\{y = i\} - p_i(\mathbf{x})
\end{aligned}
$$

Gradient tree boosting hyper-parameters

- Size of trees
  - controls amount of interactions between inputs
  - "experience indicates $4 \leq J \leq 8$" [HTF09]
- Number of iterations K
  - large K leads to over-fitting
  - chosen through early stopping
- Shrinkage

$$H_k(\mathbf{x}) = H_{k-1}(\mathbf{x}) + \nu \sum_{j=1}^{J} \chi_{k,j} \mathbb{1}\{\mathbf{x} \in R_{k,j}\}$$

  - smaller ν = less overfitting, but requires larger K
  - set $\nu < 0.1$ and choose K via early stopping [Fri01]
- Subsampling ("stochastic gradient boosting")
  - sample w/o replacement a fraction of η training examples
  - grow k-th tree using this sample
  - poor performance without shrinkage

XGBoost

- Fast implementation of gradient boosted trees.
- Reduces search space of possible splits using the distribution of features across all examples in each leaf.
- Additional regularization—objective in iteration k is

$$\underbrace{\sum_{i=1}^{N} L\left[y_i, H_{k-1}(\mathbf{x}_i) + h_k(\mathbf{x}_i)\right]}_{\text{loss}} + \underbrace{\gamma T_k + \frac{\lambda}{2}\sum_{j=1}^{T_k}\omega_{k,j}^2 + \alpha\sum_{j=1}^{T_k}|\omega_{k,j}|}_{\text{regularization}}$$

T_k	number of leafs in k-th tree
$\omega_{k,j}$	output value (weight) in j-th leaf

- Uses 2nd order Taylor expansion of the objective
- Resources:
  - Tianqi Chens paper [CG16] and slides (2014, 2016)
  - web xgboost.ai, github repo dmlc/xbgoost

Some success stories

- Fruend & Schapire won the 2003 Gödel Prize for AdaBoost.
- Viola-Jones object detection framework [VJ01]
  - 1st framework with competitive detection rates in real-time
  - AdaBoost with Haar features
- Many more successful AdaBoost applications in [FF12]
- Yahoo [CZ08], Yandex (slides): gradient boosting for ranking
- XGBoost
  - Higgs Machine Learning Challenge [CH15]
  - "Dominates structured or tabular datasets on classification and regression predictive modeling" [machinelearningmastery.com]
  - List of ML competition winning solutions
  - Very popular on Kaggle

Implementations

- AdaBoost
  - available in C++, Matlab, Python, R
  - see wikipedia entry
- Gradient Boosting
  - Python/sklearn
  - R (as Generalized Boosting Model)
- XGBoost
  - Available for C++, Java, Python, R, Julia on Windows/Mac/Linux
  - Support integration with scikit-learn
  - Can be integrated into Spark, Hadoop, Flink
  - see wikipedia entry and github repo

Concluding remarks

- Pros of gradient boosted trees
  - naturally handles data of mixed types
  - can handle missing values
  - computationally scalable
  - able to deal with irrelevant inputs
  - feature importance assessment
  - interpretability
- Cons w.r.t. deep nets
  - lower predictive power
  - cannot extract features

When in doubt, use xgboost [Kaggle winner]

References I

Tianqi Chen and Carlos Guestrin.
XBGoost: a scalable tree boosting system.
In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

Tianqi Chen and Tong He.
Higgs boson discovery with boosted trees.
In *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, pages 69–80, 2015.

David Cossock and Tong Zhang.
Statistical analysis of bayes optimal subset ranking.
IEEE Transactions on Information Theory, 54(11):5140–5154, 2008.

References II

Ayhan Demiriz, Kristin P Bennett, and John Shawe-Taylor.
Linear programming boosting via column generation.
Machine Learning, 46(1-3):225–254, 2002.

Artur Ferreira and Mário Figueiredo.
Boosting algorithms: A review of methods, theory, and applications.
In *Ensemble machine learning*, pages 35–85. Springer, 2012.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani.
Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors).
The annals of statistics, 28(2):337–407, 2000.

Yoav Freund.
Boosting a weak learning algorithm by majority.
Information and computation, 121(2):256–285, 1995.

References III

Yoav Freund.
An adaptive version of the boost by majority algorithm.
Machine learning, 43(3):293–318, 2001.

Jerome H Friedman.
Greedy function approximation: a gradient boosting machine.
Annals of statistics, pages 1189–1232, 2001.

Yoav Freund and Robert E Schapire.
Experiments with a new boosting algorithm.
In *Proceedings of the 13th ICML*, pages 148–156, 1996.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman.
The elements of statistical learning: data mining, inference, and prediction.
Springer, 2009.

References IV

Michael Kearns and Leslie Valiant.
Cryptographic limitations on learning boolean formulae and finite automata.
Journal of the ACM (JACM), 41(1):67–95, 1994.

Robert E Schapire.
The strength of weak learnability.
Machine learning, 5(2):197–227, 1990.

Paul Viola and Michael Jones.
Rapid object detection using a boosted cascade of simple features.
In *Computer Vision and Pattern Recognition (CVPR), 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.