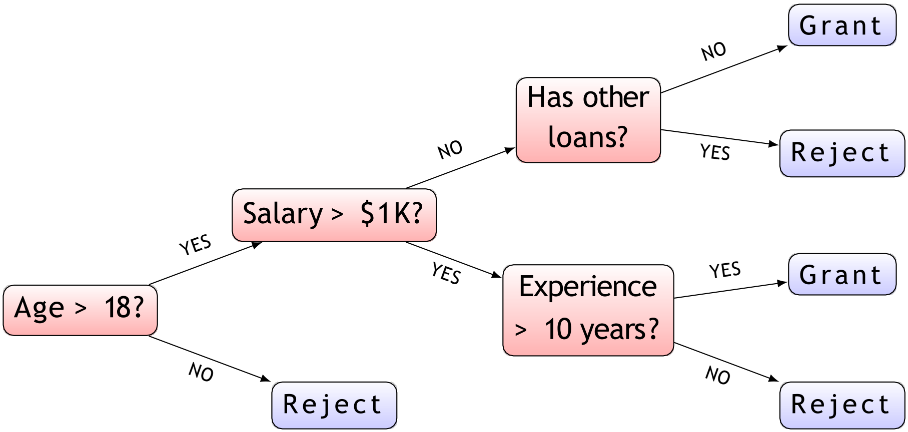
Decision Trees and Ensemble Algorithms

Machine Learning Techqiques (for High Energy Physics)

Adriano Di Florio

12 May 2023

Decision making at a bank

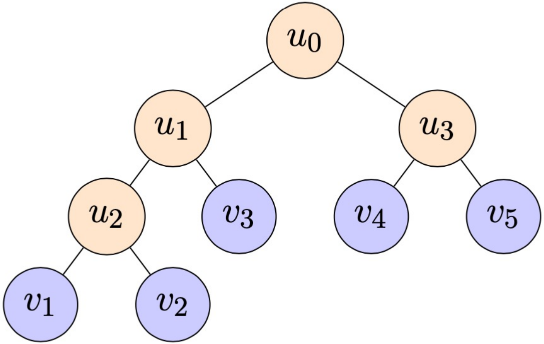


Decision tree formalism

- Decision tree is a binary tree V
- **Internal nodes** $u \in V$: predicates

$$\beta_u : \mathbb{X} \to \{0, 1\}$$

- **Leafs** $v \in V$: predictions x
- Algorithm h(x) starts at $u = u_0$
  - Compute $b = \beta_u(x)$
  - If $b = 0$, $u \leftarrow \mathrm{LeftChild}(u)$
  - If $b = 1$, $u \leftarrow \mathrm{RightChild}(u)$
  - If u is a leaf, return b

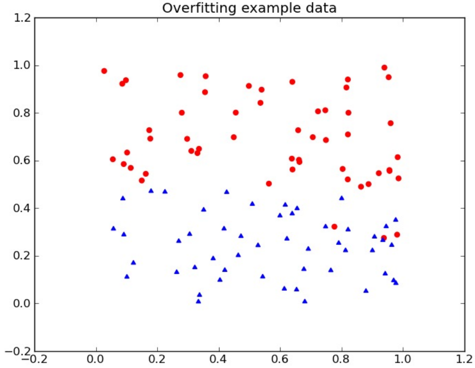


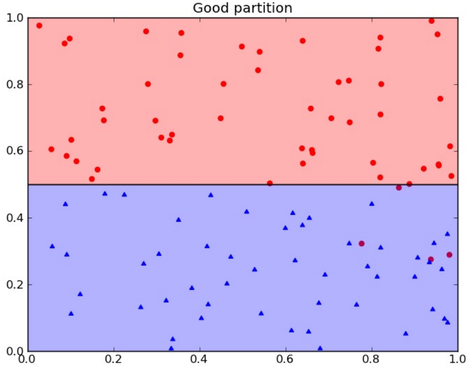
- In practice: $\beta_u(\mathbf{x}; j, t) = [\mathbf{x}_j < t]$

Greedy tree learning for binary classification

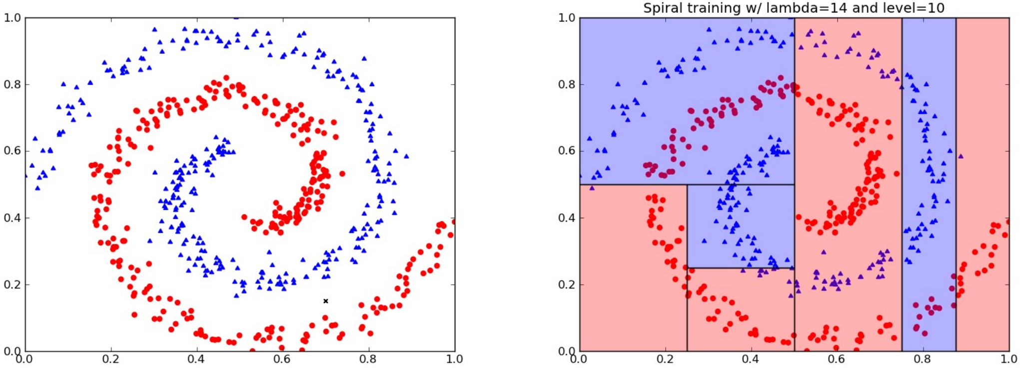
- Input: training set $X^\ell = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}$
  1. Greedily split X^ℓ into R_1 and R_2:
     $$R_1(j,t) = \{\mathbf{x} \in X^\ell | \mathbf{x}_j < t\}, \qquad R_2(j,t) = \{\mathbf{x} \in X^\ell | \mathbf{x}_j > t\}$$
     optimizing a given loss: $Q(X^\ell, j, t) \to \min\limits_{(j,t)}$
  2. Create internal node u corresponding to the predicate $[x_j < t]$
  3. If a stopping criterion is satisfied for u,
     declare it a leaf, setting some $c_u \in Y$ as leaf prediction
  4. If not, repeat 1–2 for $R_1(j, t)$ and $R_2(j, t)$
- Output: a decision tree V

Greedy tree learning for binary classification

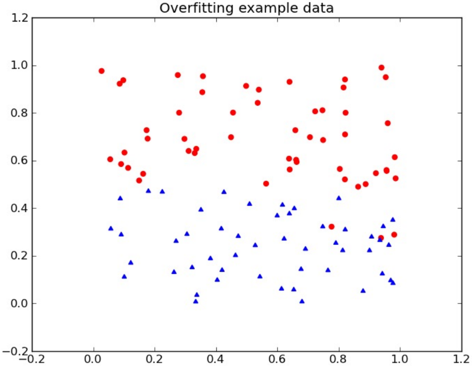


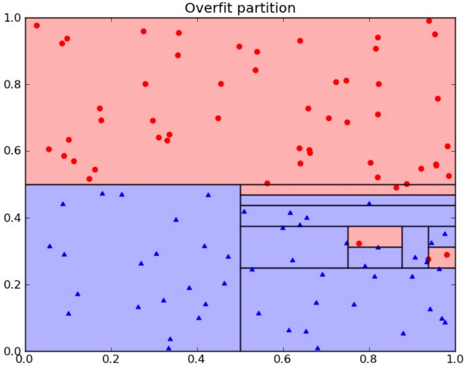


Greedy tree learning for binary classification



With decision trees, overfitting is extra-easy!





Design choices for learning a decision tree classifier

- Type of predicate in internal nodes
- The loss function $Q(X^\ell, j, t)$
- The stopping criterion
- Hacks: missing values, pruning, etc.

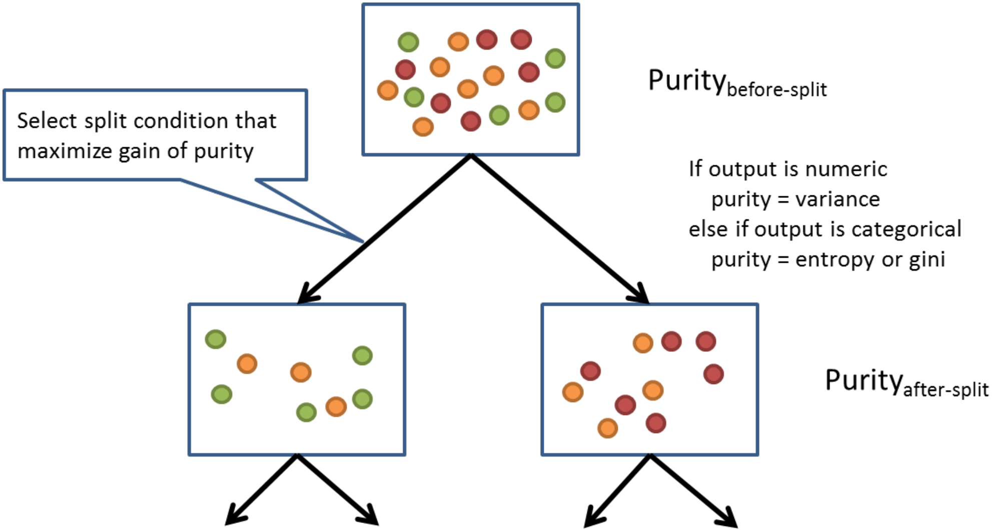
The loss function $Q(X^\ell, j, t)$

- R_m: the subset of X^ℓ at step m
- With the current split, let $R_l \subseteq R_m$ go left and $R_l \subseteq R_m$ go right
- Choose predicate to optimize

$$Q(R_m, j, t) = H(R_m) - \frac{|R_l|}{|R_m|} H(R_l) - \frac{|R_r|}{|R_m|} H(R_r) \to \max$$

- $H(R)$: impurity criterion
- Generally $H(R) = \min_{c \in \mathbb{Y}} \frac{1}{|R|} \sum_{(\mathbf{x}_i, y_i) \in R} L(y_i, c)$

The idea: maximize purity



Picture credit: https://dzone.com/refcardz/machine-learning-predictive

Examples of information criteria

Regression:

- $H(R) = \min_{c \in \mathbb{Y}} |R|^{-1} \sum_{(\mathbf{x}_i, y_i) \in R} (y_i - c)^2$
- Sum of squared residuals minimized by $c = |R|^{-1} \sum_{(\mathbf{x}_j, y_j) \in R} y_j$
- Impurity ≡ variance of the target

Classification:

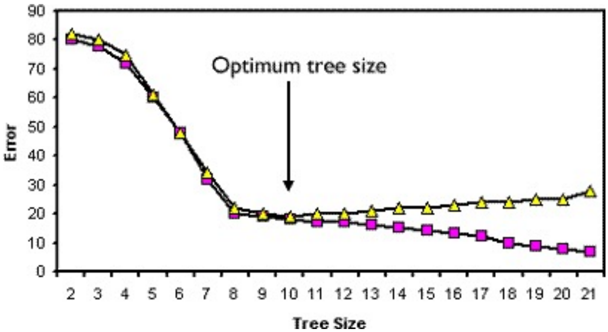
- Let $p_k = |R|^{-1} \sum_{(\mathbf{x}_i, y_i) \in R} [y_i = k]$ (share of *y*'s equal to *k*)
- Miss rate: $H(R) = \min_{c \in \mathbb{Y}} |R|^{-1} \sum_{(\mathbf{x}_i, y_i) \in R} [y_i \neq c]$

Minimizing miss rate	$1 - p_{k_*}$,
Gini index	$\sum_{k=1}^{K} p_k (1 - p_k)$,
Cross-entropy	$- \sum_{k=1}^{K} p_k \log p_k$

Stopping rules for decision tree learning

- Significantly impacts learning performance
- Multiple choices available:
  - Maximum tree depth
  - Minimum number of objects in leaf
  - Maximum number of leafs in tree
  - Stop if all objects fall into same leaf
  - Constrain quality improvement
    - (stop when improvement gains drop below s%)
- Typically selected via exhaustive search and cross-validation

Decision tree pruning



- ➢ Learn a large tree (effectively overfit the training set)
- ➢ Detect overfitting via K -fold cross-validation
- ➢ Optimize structure by removing least important nodes

Decision Tree in a nutshell

- Intuitive algorithm
- Suitable for classification and regression
- Interpretable
- Does not require much preprocessing (e.g. feature normalisation)
- Prone to overfitting (unstable)

Bagging and Random Forests

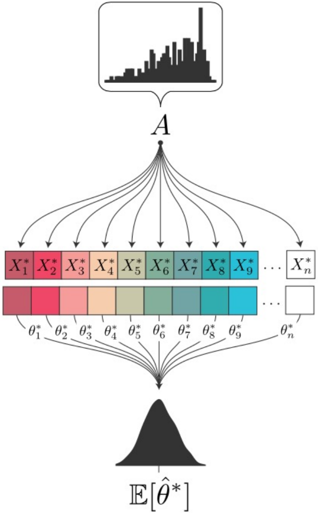
The bootstrapping procedure

Input: a sample $X^\ell = \{(x_i, y_i)\}_{i=1}^{\ell}$

Bootstrapping: generate new samples X_1^m of $(x_i; y_i)$ drawn from X^l uniformly at random with replacement (replication possible!)

Ensemble learning idea:

1. Generate N bootstrapped samples $X_1^m, \ldots, X_N^m$
2. Learn N hypotheses $h_1, \ldots, h_N$
3. Average predictions to obtain $$h(x) = \frac{1}{N} \sum_{i=1}^{N} h_i(x)$$
4. Profit!



Picture credit: http://www.drbunsen.org/bootstrap-in-picture

Bagging: bootstrap aggregation

- Input: a sample

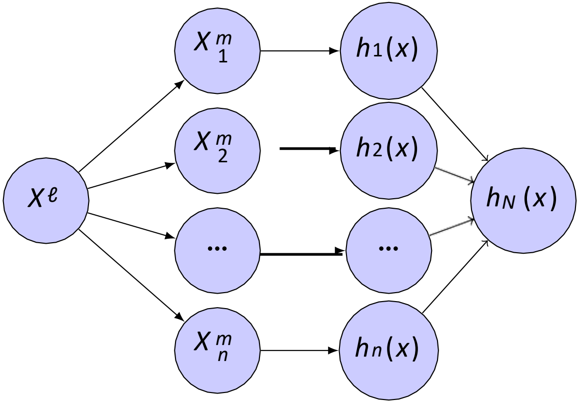
$$X^\ell = \{(x_i, y_i)\}_{i=1}^{\ell}$$

- Weak learners via bootstrapping

$$\tilde{\mu}(X^\ell) = \mu(\tilde{X}^\ell)$$

- Ensemble average

$$h_N(x) = \frac{1}{N}\sum_{i=1}^{N} h_i(x) = $$
$$= \frac{1}{N}\sum_{i=1}^{N} \tilde{\mu}(X^\ell)(x)$$



The Random Forest algorithm

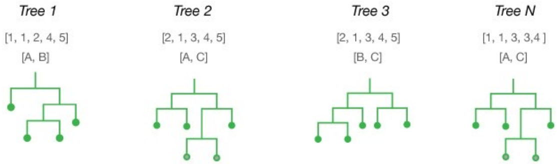
- Bagging over (weak) decision trees
- Reduce error via averaging over instances and features
- Input: a samp$X^\ell = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}$ where $x_i \in R^d$, $y_i \in Y$
- The algorithm iterates for $i = 1, \ldots, N$:
  1. Pick ρ random features out of d
  2. Bootstrap a sample $X_i^m = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}$ where $x_i \in R^d$, $y_i \in Y$
  3. Learn a decision tree $h_i(x)$ using bootstrapped X_i^m
  4. Stop when leafs in h_i contain less that $n_{\min}$ instances

$x_i \in \{A, B, C\}$

$X^\ell = \{(x_i, y_i)\}_{i=1}^{5}$

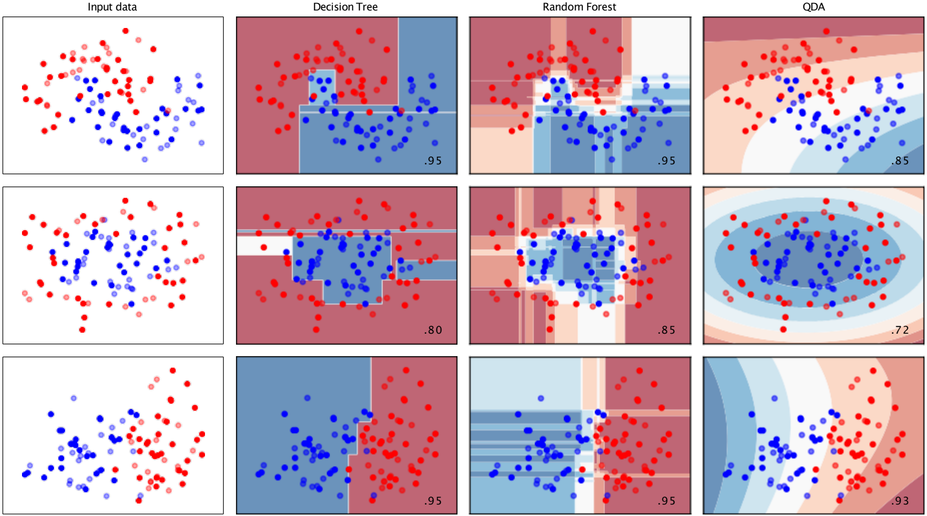
Bootstrap X_i^m, $i \in \{1, 2, 3, 4\}$

Learn $\text{Tree}_i(x)$ using$_i$ X^m



Picture credit: http://www.thefactmachine.com/random-forests

Random Forest: synthetic examplesx

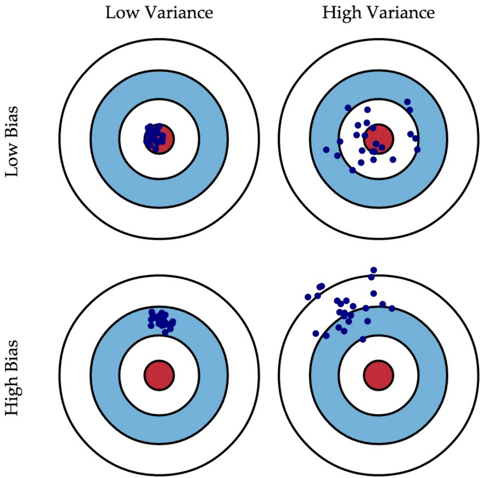


Remember: bias-variance decomposition

$$Q(\mu) = \underbrace{\mathbb{E}_{x,y}\left[\left(y - \mathbb{E}[y \mid x]\right)^2\right]}_{\text{noise}} +$$

$$+ \underbrace{\mathbb{E}_x\left[\left(\mathbb{E}_{X^\ell}\left[\mu(X^\ell)\right] - \mathbb{E}[y \mid x]\right)^2\right]}_{\text{bias}} +$$

$$+ \underbrace{\mathbb{E}_x\left[\mathbb{E}_{X^\ell}\left[\left(\mu(X^\ell) - \mathbb{E}_{X^\ell}\left[\mu(X^\ell)\right]\right)^2\right]\right]}_{\text{variance}}$$



Bagging: bias

- Bias: not made any worse by bagging multiple hypotheses

$$\mathbb{E}_{x,y}\left[\left(\mathbb{E}_{X^\ell}\left[\frac{1}{N}\sum_{n=1}^{N}\tilde{\mu}(X^\ell)(x)\right] - \mathbb{E}[y\,|\,x]\right)^2\right] =$$

$$= \mathbb{E}_{x,y}\left[\left(\frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_X^\ell[\tilde{\mu}(X^\ell)(x)] - \mathbb{E}[y\,|\,x]\right)^2\right] =$$

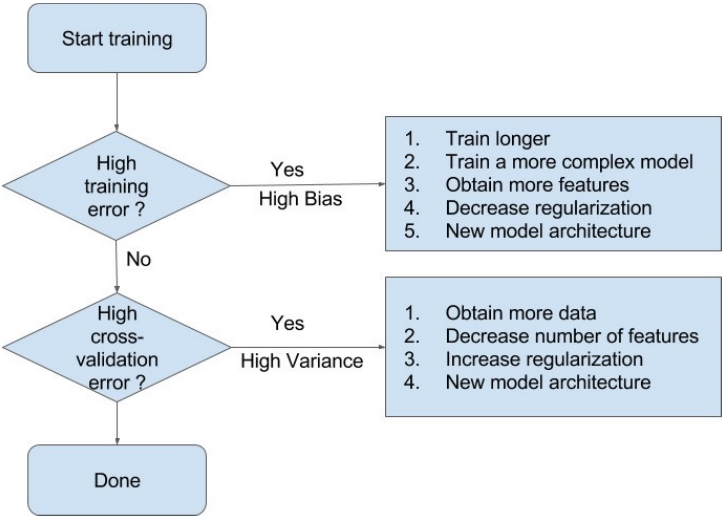
$$= \mathbb{E}_{x,y}\left[\left(\mathbb{E}_{X^\ell}\left[\tilde{\mu}(X^\ell)(x)\right] - \mathbb{E}[y\,|\,x]\right)^2\right]$$

Bagging: variance

- Variance: N times lower for uncorrelated hypotheses, yet not as much an improvement for highly correlated

$$
\begin{aligned}
&\mathbb{E}_{x,y}\left[\mathbb{E}_{X^\ell}\left[\left(\frac{1}{N}\sum_{n=1}^{N}\tilde{\mu}(X^\ell)(x) - \mathbb{E}_{X^\ell}\left[\frac{1}{N}\sum_{n=1}^{N}\tilde{\mu}(X^\ell)(x)\right]\right)^2\right]\right] = \\
&= \frac{1}{N}\mathbb{E}_{x,y}\left[\mathbb{E}_{X^\ell}\left[\left(\tilde{\mu}(X^\ell)(x) - \mathbb{E}_{X^\ell}\left[\tilde{\mu}(X)(x)\right]\right)^2\right]\right] + \\
&+ \frac{N(N-1)}{N^2}\mathbb{E}_{x,y}\left[\mathbb{E}_{X^\ell}\left[\left(\tilde{\mu}(X^\ell)(x) - \mathbb{E}_{X^\ell}\left[\tilde{\mu}(X^\ell)(x)\right]\right)\times\right.\right. \\
&\left.\left.\times\left(\tilde{\mu}(X^\ell)(x) - \mathbb{E}_{X^\ell}\left[\tilde{\mu}(X^\ell)(x)\right]\right)\right]\right]
\end{aligned}
$$

Bias-variance fix strategy



Picture credit: http://www.learnopencv.com/bias-variance-tradeoff-in-machine-learni3n1g

Checkpoint

- Bootstrapping: a general statistical technique for computing sample functionals (and their variance)
- Bagging: meta-learner over arbitrary weak algorithms via bootstrap aggregation
- The Random Forest algorithm: bagging over decision trees
- Bias/Variance trade-off - important concept that allows to understand and improve model performance

Reweighting and AdaBoost

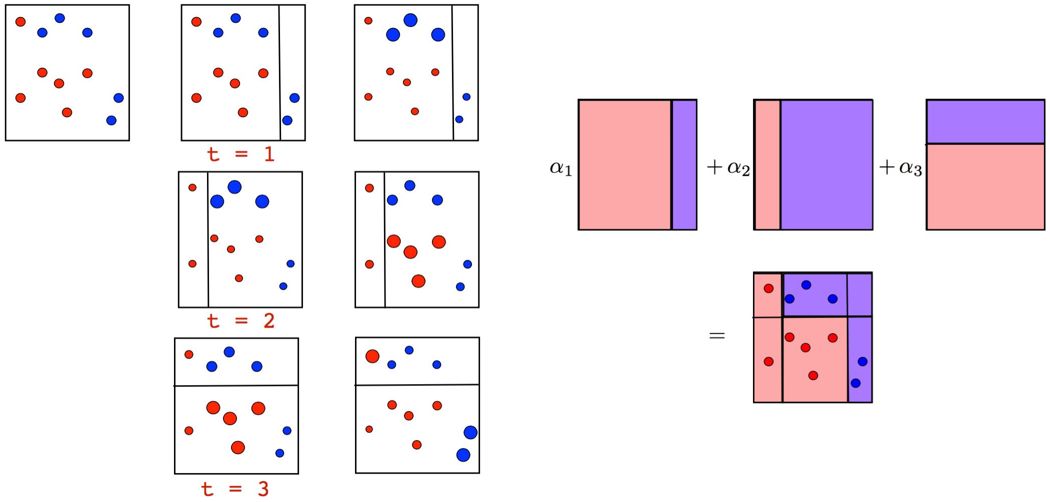
Adaptive boosting for classification

- Training set $X^\ell = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}, y_i \in \{-1, +1\}$
- Search for solution in the form of a weighted $a_N(\mathbf{x}) = \sum_{n=1}^{N} \gamma_n h_n(\mathbf{x})$ with weak learners $h_n \in \mathrm{H}$
- At step N, extend a with $h_N : a_N(\mathbf{x}_i) = a_{N-1}(\mathbf{x}_i) + \gamma_N h_N(\mathbf{x}_i)$.

  How do we choose h_N and its weight γ_N ?
- Measure fit quality using exponential loss $Q(a_N, X^\ell) = \sum_{i=1}^{\ell} \exp\{-y_i a_N(\mathbf{x}_i)\}$
- Derivation yields $h_N = \arg\min_h \sum_{i=1}^{\ell} (h(\mathbf{x}_i) - w_i y_i)^2$

  with weights $w_i = \exp\{-y_i a_{N-1}(\mathbf{x}_i)\}$
- Weak learner weight: minimize $\gamma_N = \arg\min_\gamma Q(a_{N-1}(\mathbf{x}_i) + \gamma h_N(\mathbf{x}_i), X^\ell)$

Adaptive boosting for classification



Picture credit: https://habrahabr.ru/company/ods/blog/327250

Adaptive boosting for classification

[Video: AdaBoost in Action]

https://www.youtube.com/watch?v=k4G2VCuOMMg

Gradient Boosting

Gradient boosting: motivation

- With $a_{N-1}(x)$ already built, how to find the next γ_N and h_N if

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(\mathbf{x}_i) + \gamma h(\mathbf{x}_i)) \to \min_{\gamma, h}$$

- Recall: functions decrease in the direction of negative gradient
- View $L(y, z)$ as a function of z $(= a_N(x_i))$, execute gradient descent on z
- Search for such $s_1, \ldots, s_\ell$ that

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(\mathbf{x}_i) + s_i) \to \min_{s_1, \ldots, s_\ell}$$

Choose $s_i = - \left. \frac{\partial L(y_i, z)}{\partial z} \right|_{z = a_{N-1}(\mathbf{x}_i)}$, approximate s_i's by h_N

The Gradient Boosting Machine [Friedman, 2001]

- Input:
  - Training set $X^\ell = \{(\mathbf{x}_i, y_i)\}_{i=1}^\ell$
  - Number of boosting iterations N
  - Loss function $Q(y, z)$ with its gradient $\frac{\partial Q}{\partial z}$
  - A family H = $\{h(\mathrm{x})\}$ of weak learners and their associated learning procedures
  - Additional hyperparameters of weak learners (tree depth, etc.)
- Initialize GBM $h_0(\mathrm{x})$: with simple rules (zero, most popular class, etc.)
- Execute boosting iterations $t = 1, \ldots, N$ (see next slide)
- Compose the final GBM learner $a_N(\mathbf{x}) = \sum_{t=0}^N \gamma_i h_i(\mathbf{x})$

The Gradient Boosting Machine [Friedman, 2001]

At every iteration:

1. Compute pseudo-residuals:
2. Learn $h_N(\mathbf{x}_i)$ by regressing onto $s_1, \ldots, s_\ell$:

$$h_N(x) = \arg\min_{h \in \mathbb{H}} \sum_{i=1}^{\ell} (h(\mathbf{x}_i) - s_i)^2$$

3. Find the optimal γ_N using plain gradient descent:

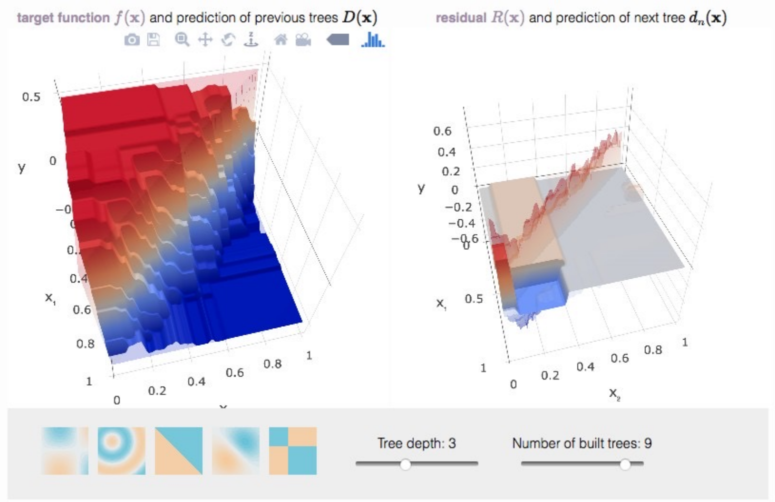
$$\gamma_N = \arg\min_{\gamma \in \mathbb{R}} \sum_{i=1}^{\ell} Q(y_i, a_{N-1}(\mathbf{x}_i) + \gamma h_N(\mathbf{x}_i))$$

4. Update the GBM by $a_N(\mathbf{x}_i) \leftarrow a_{N-1}(\mathbf{x}) + \gamma_N h_N(\mathbf{x})$

GBM: an interactive demo

Credits: Andrey Ustyuzhanin

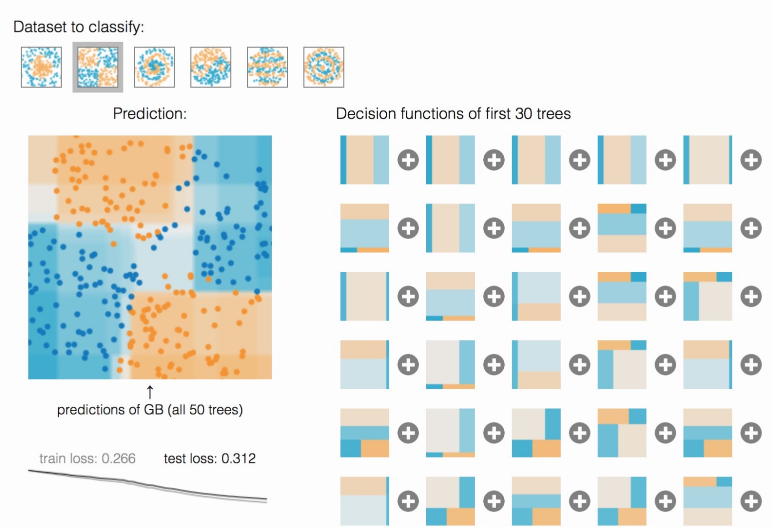
http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html



GBM: an interactive demo

Credits: Andrey Ustyuzhanin

http://arogozhnikov.github.io/2016/07/05/gradient_boosting_playground.html



GBM: regulization via shrinkage

- For too simple weak learners, the negative gradient is approximated badly $\Rightarrow$ random walk in space of samples
- For too complex weak learners, a few boosting steps may be enough for overfitting
- Shrinkage: make shorter steps using a learning rate $\eta \in (0, 1]$

$$a_N(\mathbf{x}_i) \leftarrow a_{N-1}(\mathbf{x}) + \eta \gamma_N h_N(\mathbf{x})$$

(effectively distrust gradient direction estimated via weak learners)

GBM: shrinkage animated

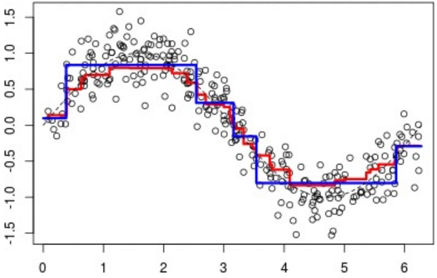


Figure: High shrinkage

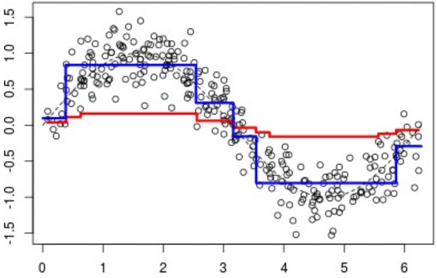
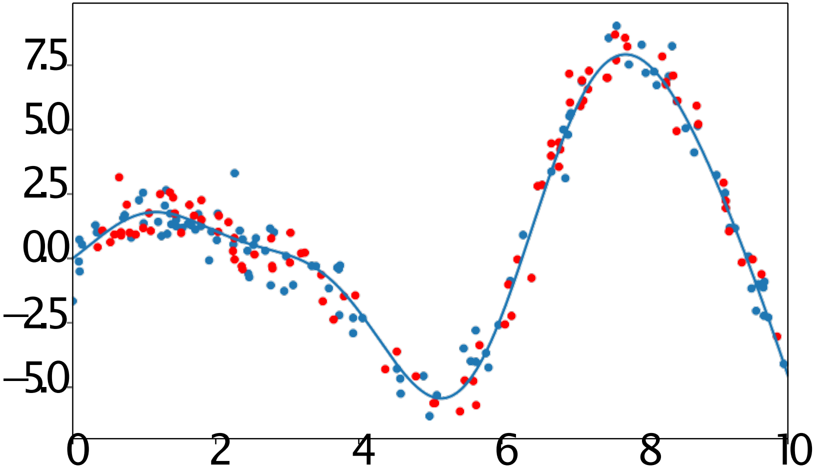
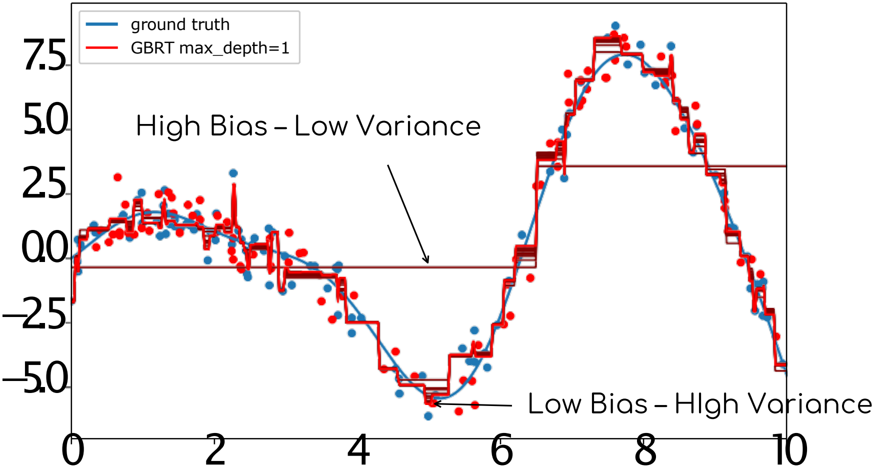


Figure: Low shrinkage

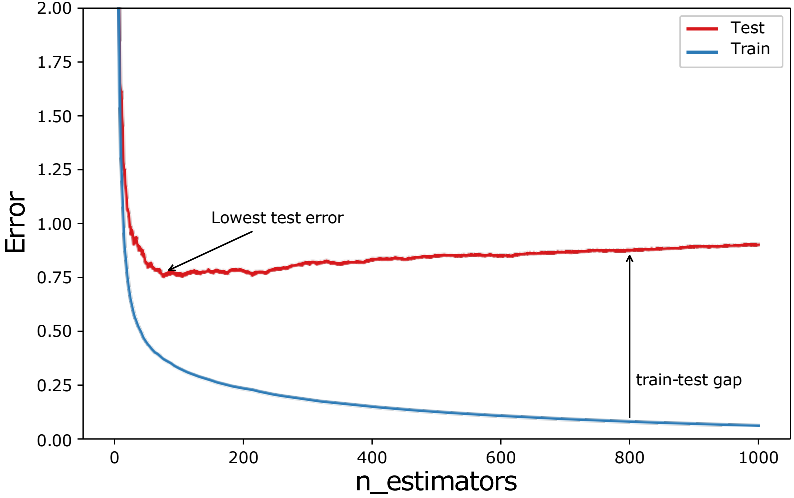
GBM: regularization approaches



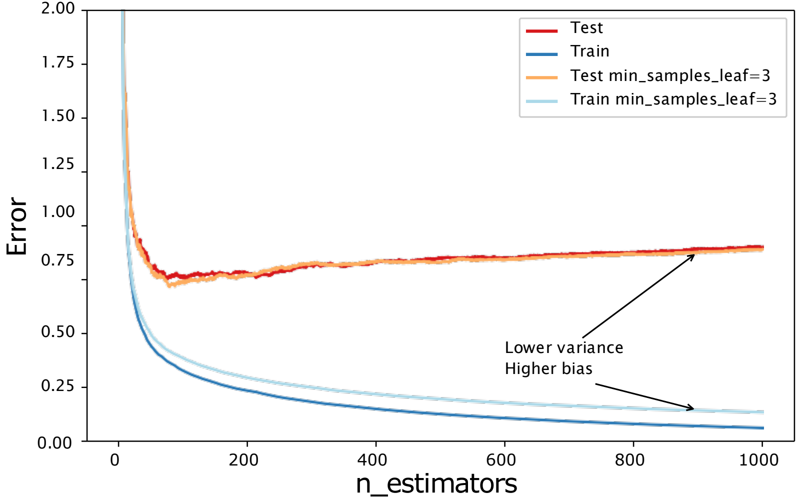
GBM: regularization approaches



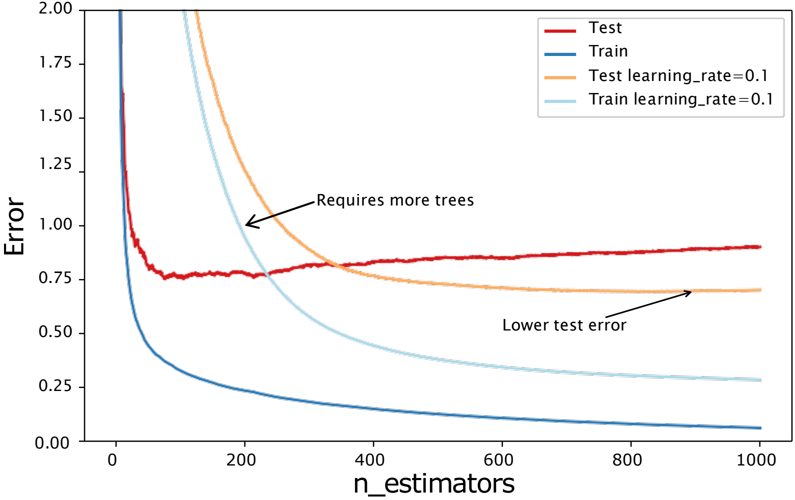
GBM: regularization approaches



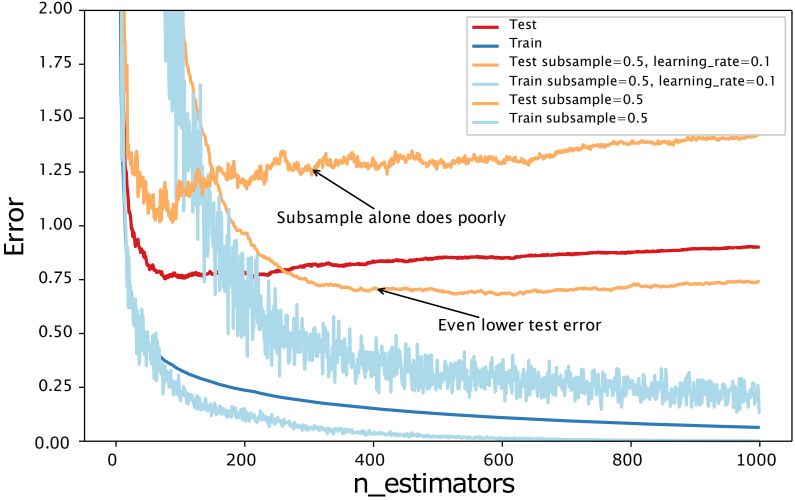
GBM: regularization approaches



GBM: regularization approaches



GBM: regularization approaches



XGBoost Algorithm

Extreme Gradient Boosting

1. Approximate the descent direction constructed using second order derivatives

$$\sum_{i=1}^{\ell}\left(-s_i h(\mathbf{x}_i)+\frac{1}{2}t_i h^2(\mathbf{x}_i)\right)\to\min_h,\qquad t_i=\left.\frac{\partial^2}{\partial z^2}L(y_i,z)\right|_{a_{N-1}(\mathbf{x}_i)}$$

2. Penalize large leaf counts J and large leaf coefficient norm $\|b\|_2^2=\sum_{j=1}^{J}b_j^2$

$$\sum_{i=1}^{\ell}\left(-s_i h(x_i)+\frac{1}{2}t_i h^2(x_i)\right)+\gamma J+\frac{\lambda}{2}\sum_{j=1}^{J}b_j^2\to\min_h$$

where $b(\mathbf{x})=\sum_{j=1}^{J}b_j[\mathbf{x}\in R_j]$

Extreme Gradient Boosting

3. Choose split $[\mathbf{x}_j < t]$ at node R to maximize

$$Q = H(R) - H(R_\ell) - H(R_r) \to \max,$$

where the impurity criterion

$$H(R) = -\frac{1}{2}\left(\sum_{(t_i,s_i)\in R} s_j\right)^2 \Bigg/ \left(\sum_{(t_i,s_i)\in R} t_j + \lambda\right) + \gamma$$

4. The stopping rule: declare the node a leaf if even the best split gives negative Q

Summary

- Boosting: a general meta-algorithm aimed at composing a strong hypothesis from multiple weak hypotheses
- Boosting can be applied for arbitrary losses, arbitrary problems (regression, classification) and over arbitrary weak learners
- The Gradient Boosting Machine: a general approach to boosting adding weak learners that approximate gradient of the loss function
- AdaBoost: gradient boosting with an exponential loss function resulting in reweighting training instances when adding weak learners
- XGBoost: gradient boosting with second order optimization, penalized loss and particular choice of impurity criterion