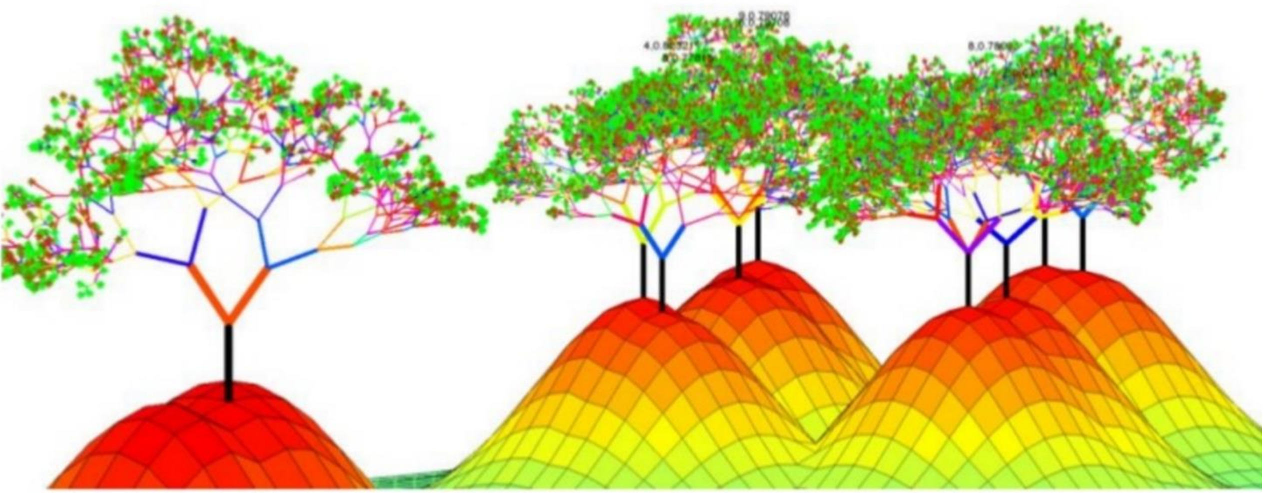
XGBoost

Efficient boosting with tree models

Martin Jullum

Big Insight lunch, Jan 31, 2018



XGBoost = eXtreme Gradient Boosting

- A machine learning library built around an efficient implementation of boosting for tree models (like GBM)
  - Developed by Tianqi Chen (Uni. Washington) i 2014
- Core library in C++, with interfaces for many languages/platforms
  - C++, Python, R, Julia, Java, etc.
  - Distributed version for Hadoop + Spark
- Engineering goal: “Push the limit of computational resources for boosted tree algorithms”
  - Parallelizable, cheap on memory, scales to large data sets
- Very powerful and flexible – lots of (hyper)parameters
- Huge succsess
  - «Winning practically every prediction competiton on Kaggle»

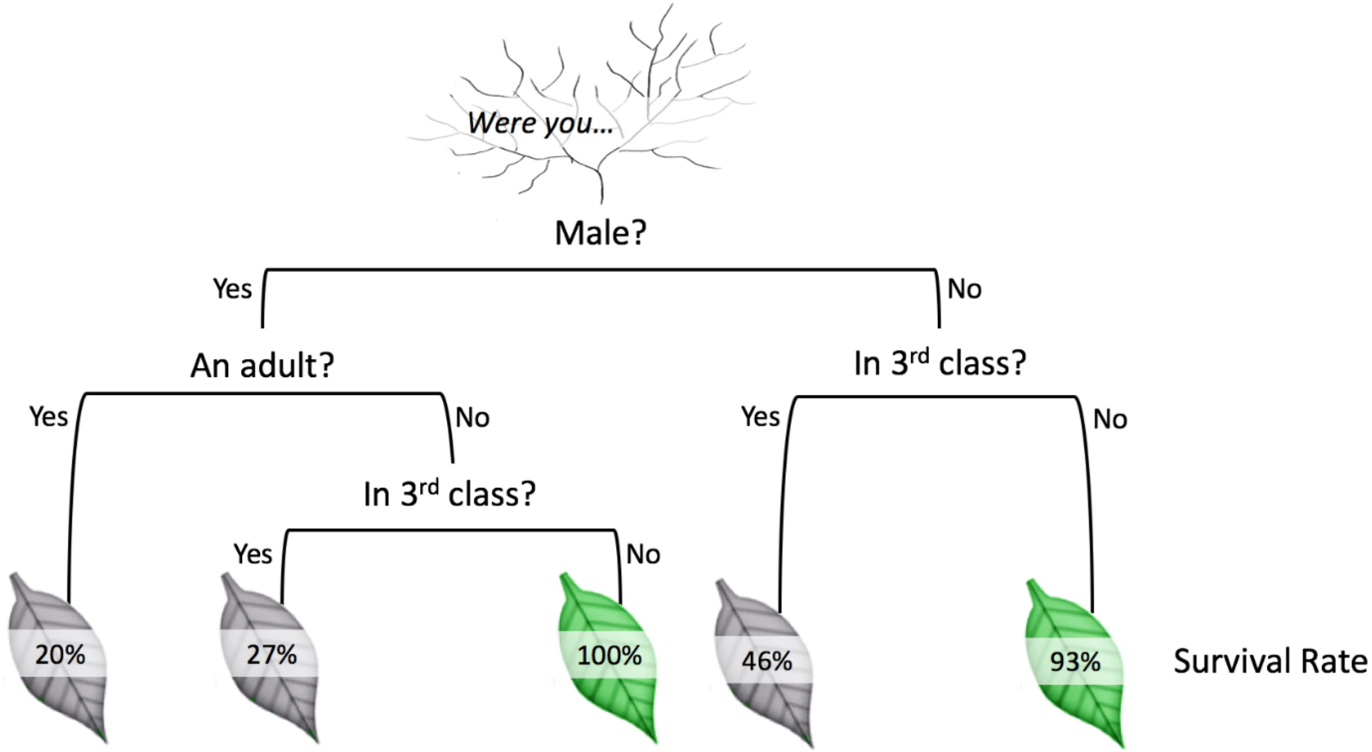
Problem setup

- Assume with have training data set of size n
  - Response: y_i
  - Covariates: $x_i = (x_{i1}, \dots, x_{ip})^{\top}, i = 1, \dots n$

- Want to train a model f on these data such that $f(x_i)$ approximates y_i as well as possible (on a seperate test data set!) in terms of a loss function $L(y, f)$

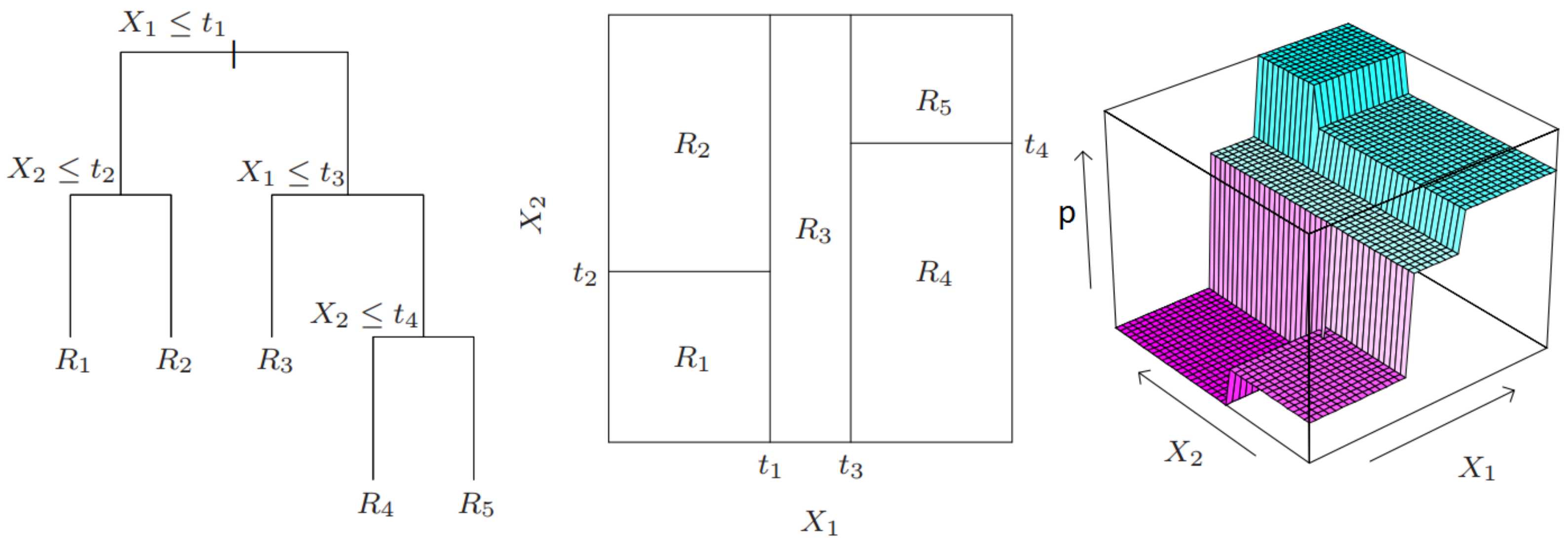
Tree models (I)

- Conceptually possibly the simplest statistical model existing!
  - The function is evaluated by a series of conditional IF-ELSE rules
  - As a tree: Start at the root and work your way through the branches depending on your covariate values, ending up at the leaves



Trained tree model for survival rate on Titanic

Tree models (II)



3 visualizations of the same tree model

- May be written as a weighted sum of indicator values

$$f(x) = \sum_{j=1}^{T} \theta_j \, 1_{\{x \in R_j\}}$$

Training a tree model

- Computationally intractable to find the best partitioning w.r.t. general $L(y, f)$, so need a greedy algorithm, which iteratively grows the tree
- Algorithm:
  - For each leaf node $N_j, j = 1, \ldots,$ in the current tree DO:
    - For each covariate x_j, find the split point corresponding to new potential regions R_{1j}, R_{2j} minimizing the split loss

$$\sum_{i \in N_j} \left[L\left(y_i, \hat{y}_{R_{1j}}\right) + L\left(y_i, \hat{y}_{R_{2j}}\right) \right]$$

where $\hat{y}_{R_{kj}} = \operatorname{argmin}_c \sum_{i \in R_{kj}} L(y_i, c)$.

  - Choose the leaf node, covariate and split point with smallest split loss
  - Perform the split if loss reduction is large enough in terms of e.g. previous loss reductions, depth, number of nodes, etc.
  - REPEAT

Properties of tree models

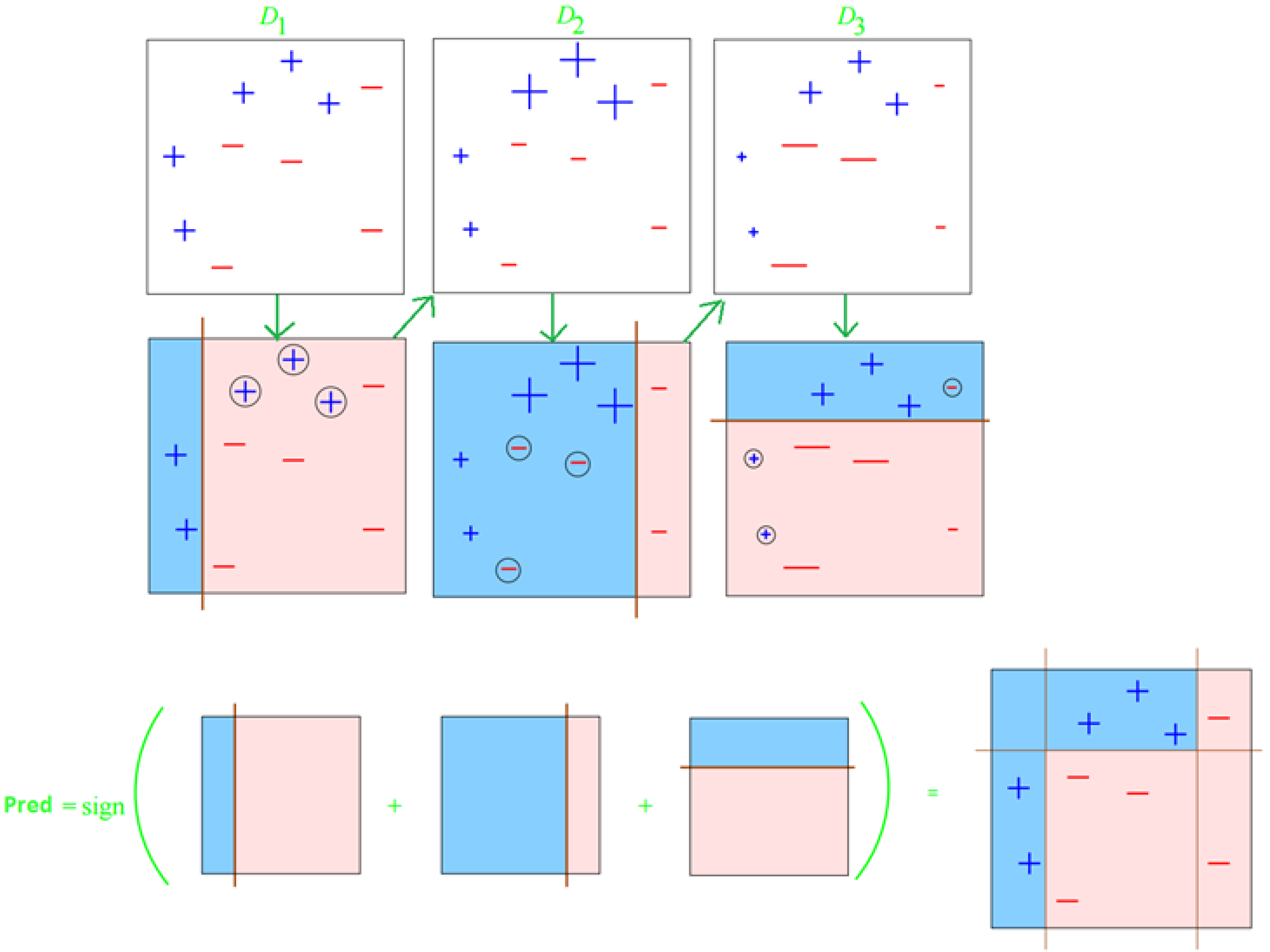
- Benefits
  - Models non-linearities and interactions directly
  - Invariant under monotone transformations of the covariates
  - Easy to train – scales well to large data sets
  - Naturally combines continuous and categorical data
  - Easy to explain and interpret
  - Can handle missing data
  - Robust to outliers in the covariates
- Drawbacks
  - Limited predictive power
  - High variance
  - Somewhat arbitrary handling of overfitting/regularization
  - Lack smoothness

Boosting: The principle

- Kearns (1988) asked whether weak classifiers could be combined into a strong classifier
- Freund and Schapire (1997): YES, with AdaBoost
- AdaBoost idea
  - Iteratively fit simple models $f_m(x)$ (weak learners) trying to correct «mistakes» of previous models
  - Combine them additively into a model ensamble with good predictive performance (strong learner)

$$f^{(M)}(x) = \sum_{m=1}^{M} f_m(x)$$

Example Adaboost



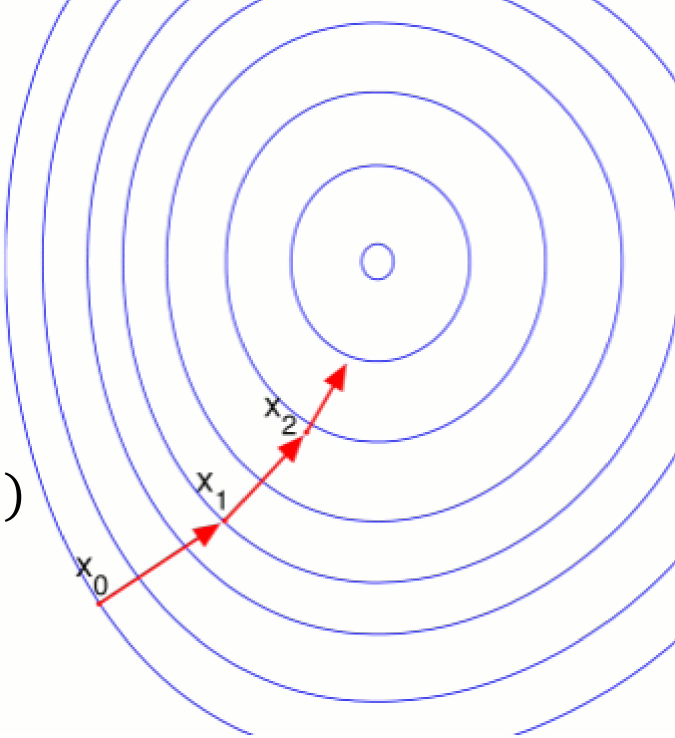
Adaboost as FSAM

- Friedman et al. (2000): Adaboost is equivalent to Forward Stagewise Additive Modelling (FSAM) with the loss function: $L(y, f(x)) = \exp(-yf(x))$
- FSAM: For $m = 1, \ldots, M$, find model f_m by minimizing the empirical risk

$$f_m = \underset{h \in \Phi}{argmin} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f^{(m-1)}(x_i) + h(x_i))$$

  - $f^{(m-1)}(x) = \sum_{j=1}^{m-1} f_j(x)$, with $\mathrm{f}^{(0)}(x) = 0$
  - For some model class Φ
  - A very general procedure, but hard to do for general loss function

Gradient boosting



- Gradient descent
  - Iterative procedure for finding minimum of (multivariate) function $s(z)$
  - Iteratively take steps along the negative gradient: $z_m = z_{m-1} - \rho_m s'(z_{m-1})$
- Notation: Let $s_i(z) = \mathrm{L}(y_i, z)$
- Gradient boosting (Friedman (2001))
  - Take a gradient descent step towards the minimum of s_i, at $\mathrm{z} = f^{(m-1)}(x_i)$ for all i
  - Class restriction solved by using the function closest in L2 to the negative gradient: $f_{m,0} = \underset{h \in \Phi}{argmin} \sum_{i=1}^{n} [-s_i'(f^{(m-1)}(x_i)) - h(x_i)]^2$
  - The step length is found by $\rho_m = \underset{\rho}{argmin} \sum_{i=1}^{n} L\left(y_i, f^{(m-1)}(x_i) + \rho f_{m,0}(x_i)\right),$
  - Finally: $f_m(x) = \gamma \rho_m \, f_{m,0}(x)$, for some pre-set learning rate $\gamma \in (0,1]$
  - This is the most common boosting method, e.g. gbm package in R

2. order approximation

- Approximate $\mathrm{L}\left(y_i, f^{(m-1)}(x_i) + h(x_i)\right)$, using a 2. order Taylor approximation of $s_i(z)$ around $z = f^{(m-1)}(x_i)$
- $s^*\left(f^{(m-1)}(x_i) + h(x_i)\right) = s_i\left(f^{(m-1)}(x_i)\right) + s_i'\left(f^{(m-1)}(x_i)\right)h(x_i) + \frac{1}{2}s_i''\left(f^{(m-1)}(x_i)\right)h(x_i)^2$
- Inserting this into the FSAM solution gives
- $$f_{m,0} = \underset{h\in\Phi}{argmin} \sum_{i=1}^{n}[s^*(f^{(m-1)}(x_i) + h(x_i))\ ]$$
  $$= \underset{h\in\Phi}{argmin} \sum_{i=1}^{n}[s_i'\left(f^{(m-1)}(x_i)\right)h(x_i) + s_i'{}'\left(f^{(m-1)}(x_i)\right)h(x_i)]$$
  $$= \underset{h\in\Phi}{argmin} \sum_{i=1}^{n}\frac{1}{2}s'(f^{(m-1)}(x_i))[-\frac{s'\left(f^{(m-1)}(x_i)\right)}{s''\left(f^{(m-1)}(x_i)\right)} - h(x_i)]^2$$
- Just a weighted least squares problem w.r.t. $h \in \Phi$
- Finally: $f_m(x) = \gamma\, f_{m,0}(x)$, for some pre-set learning rate γ
- This is the method used by XGBoost, with Φ being tree models (originally proposed through LogitBoost)

XGBoost – methodological improvements

- Tree boosting inherits most benefits and fixes the drawbacks of individual tree models
- 2. order approx. to FSAM – more precise than regular gradient boosting
- Introduced regularization directly in the tree growing procedure
  - Actually tries to minimize, $\mathrm{L}\left(y_i, f^{(m-1)}(x_i) + h(x_i)\right) + \Omega(h)$,
  - $\Omega(h) = \gamma \mathrm{T} + \frac{1}{2}\lambda \sum_j^T w_j^2 + \alpha \sum_j^T |w_j|$, for w_j the leaft values of tree of depth T
  - Also other regularization parameters available
- Subsampling of both rows and columns of covariate matrix available for better generalization properties

XGBoost – technical improvements

- Very fast and cheap on memory
  - Store data in internal sparsity aware format – memory friendly
  - The tree learning algorithm utilizes the sparse structure
  - Parallelizes tree learning per covariate
  - Example: n=2*10^6, p=200, Y={0,1}, depth=6, **150sec** with 16 threads, a few GB of RAM consumption.
- Allows the user to view the performance of the current model during training
- Can automatically stop boosting when performance on spearate validation set decreases
- User can set custom loss function and evaluation metric for stopping
- Implemented direct handling of missing values – learning a default direction for NA

Some recent community contributions

- DART (Dropout Additive Regression Trees) (Feb 2016)
  - Drop given proportion of trained trees when learning a new tree
  - More randomization -> link to random forest
- Histogram approach (Jan 2017)
  - Discretize continuous covariates into default bins for faster training, 4-10 times faster
- GPU version (Aug 2017)
  - 2-4 times faster than histogram approach on CPU
- Covariate contribution per prediction supported natively by SHAP (Oct 2017)

Remarks

- Competitors
  - LightGBM (Microsoft)
    - Very similar, not as mature and feature rich
    - Slightly faster than XGBoost – much faster when it was published
  - CatBoost (Yandex, “Russian Google”)
    - Also similar, but handles categorical variables directly
    - Benchmarks show better results, but is much slower
- CRAN verison of xgboost is outdated (do NOT use!)
  - Install from private repo (simple) or directly from Github (advanced) http://xgboost.readthedocs.io/en/latest/build.html#r-package-installation
- Can be called from caret, h2o R-packages + scikit-learn in Python
- I have still not seen an example where Random Forest outperforms XGBoost

Key resources

- Didrik Nielsen, Master thesis NTNU, 2016: https://brage.bibsys.no/xmlui/handle/11250/2433761
- Chen & Guestrin (2016), XGBoost: A Scalable Tree Boosting System: https://arxiv.org/abs/1603.02754
- Hastie et al. (2009), Elements of Statistical Learning, Ch 9.2 + 10
- XGBoost Github: https://github.com/dmlc/xgboost
- XGBoost documentation: http://xgboost.readthedocs.io
- Slides from Meetup in LA with Tianqi Chen: http://datascience.la/xgboost-workshop-and-meetup-talk-with-tianqi-chen/