Gradient Boosted Trees with XGBoost and scikit-learn

Jay Urbain, PhD

Credits:

Greedy Function Approximation: A Gradient Boosting Machine, by Friedman

Michael Kerns, Hypothesis Boosting, 1988.

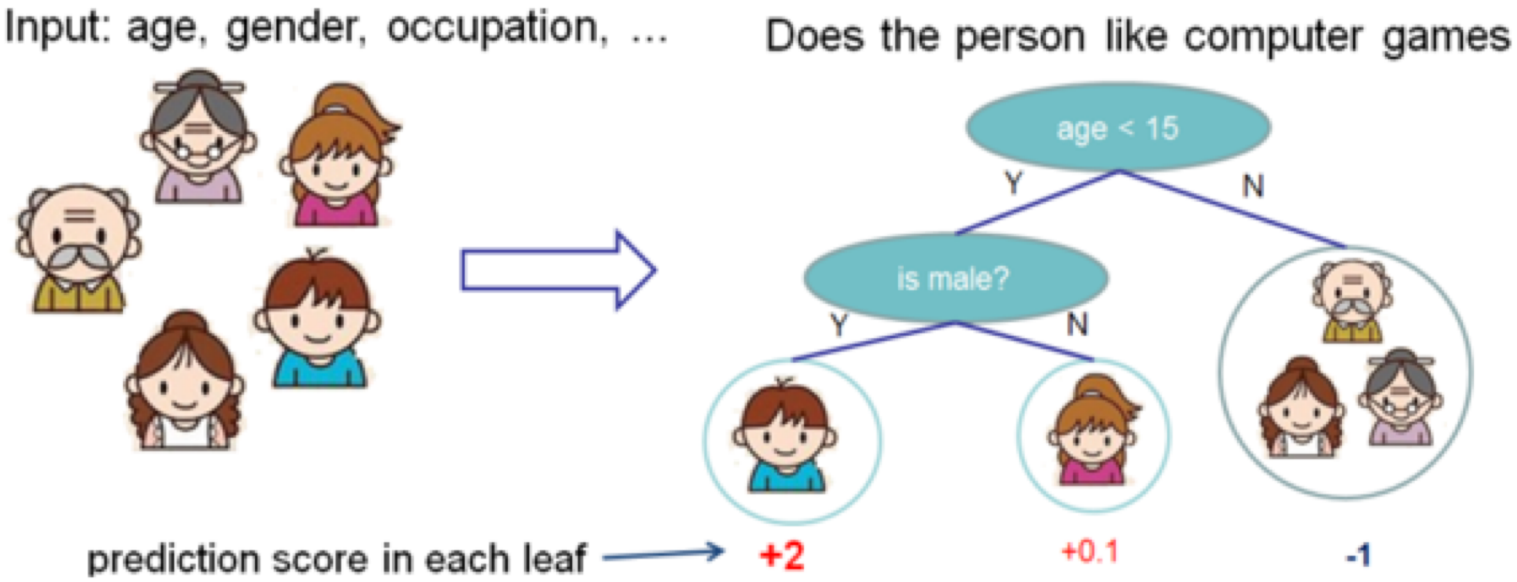
Jason Brownlee, XGBoost with Python

https://xgboost.readthedocs.io/en/latest/tutorials/model.html

https://en.wikipedia.org/wiki/Gradient_boosting

See additional references on last slide.

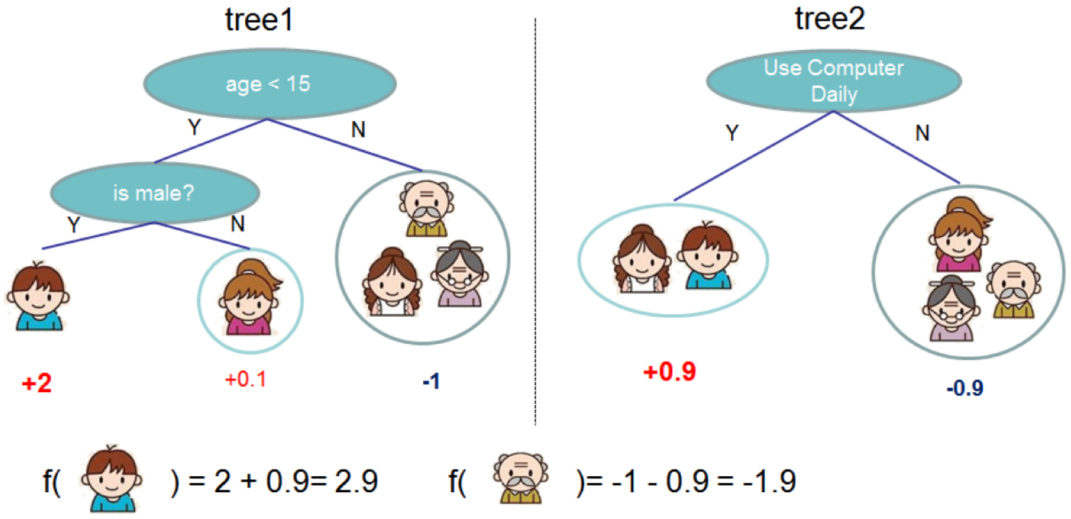
Decision Tree



- Classify the members of a family into different leaves, and assign them the score on the corresponding leaf.

Decision Tree Ensembles

- Usually, a single tree is not strong enough to be used in practice.
- Drawback to using a single decision/regression tree – **it fails to include predictive power from multiple, overlapping regions of the feature space**. , i.e., **interactions between features.**
- Instead, use ensemble model, which sums the prediction of multiple trees together.
- If you look at the example, an important fact is that the two trees try to *complement* each other.



Tree Ensemble methods

- Widely used methods: GBM (Gradient Boosted Machines) and random forests.
- Almost half of data mining competitions at Kaggle are won by using some variants of tree ensemble methods.
- Invariant to scaling of inputs (tree splits only use ordering), so you do not need to do careful feature normalization.
- Learn higher order interaction between features.
- Can be scalable.
- Used in Industry.

Model and Parameters for Tree Ensembles

- 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 of a regression tree that is a function that maps the attributes to the score.
- Parameters: *structure of each tree*, and the *score in the leaf*.
- Or simply use functions as parameters.

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

- Key difference: Instead of learning weights in $\boldsymbol{R^d}$, we are learning ***functions***(trees).

Decision Tree Ensembles

- Mathematically, we can write the ensemble model in the following form:

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

- ***K*** is the number of trees, ***f*** is a function in the functional space ***F***.
- The objective function to be optimized is given by:

$$\text{obj}(\theta) = \sum_{i}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

- What is the *model* used in random forests? **Tree ensembles**.
- So *random forests* and *boosted trees* are really the same models.
- **The difference arises from how we train them.**

Boosting

Michael Kearns articulated the goal as the *Hypothesis Boosting Problem*:

... an efficient algorithm for converting relatively poor hypotheses into very good hypotheses

– Thoughts on Hypothesis Boosting, 1988.

Gradient Boosting

- Gradient boosting is one of the most powerful techniques for building predictive models.
- The idea of boosting came out of the idea of whether a weak learner can be modified to become better.
- A weak hypothesis or weak learner is defined as one whose performance is at least slightly better than random chance.

Hypothesis boosting: filtering observations

- ***leave those observations that the weak learner can handle and focus on developing new weak learners to handle the remaining difficult observations.***

AdaBoost

- The first realization of boosting that saw great success in applications.
- The weak learners in AdaBoost are decision trees with a single split, called ***decision stumps***.
- AdaBoost works by weighting the observations, putting *more weight* on difficult to classify instances and less on those already handled well.
- New weak learners are added sequentially that focus their training on the more difficult patterns.
- ***This means that samples that are difficult to classify receive increasingly larger weights until the algorithm identifies a model that correctly classifies these samples.***
- Predictions are made by majority vote of the weak learners' predictions, weighted by their individual accuracy.

Generalization of AdaBoost as Gradient Boosting

- AdaBoost and related algorithms were recast in a **statistical framework** first by Breiman and then by Friedman as **Gradient Boosting Machines**.
- Cast boosting as a **numerical optimization problem** where the objective is to minimize the loss of the model by adding weak learners using gradient descent.
- This class of algorithms were described as a **stage-wise additive model**.
- One new weak learner is added at a time and existing weak learners in the model are frozen and left unchanged.

How Gradient Boosting Works

Gradient boosting involves three elements:

1. A **loss function** to be optimized.
2. A **weak learner** to make predictions.
3. An **additive model** to add weak learners to minimize the loss function.

Loss Function

- The loss function depends on the type of problem being solved.
- It must be differentiable, but many standard loss functions are supported and you can define your own.
- For example, for regression you would typically use a **squared error**, and for classification you would use **cross-entropy** loss.
- A benefit of the gradient boosting framework is that a new boosting algorithm does not have to be derived for each loss function.

Training Loss *L*

- The training loss measures how *predictive* our model is with respect to the training data.

$$\text{obj}(\theta) = L(\theta) + \Omega(\theta)$$

- Regression: *mean squared error*:

$$L(\theta) = \sum_i (y_i - \hat{y}_i)^2$$

- Logistic regression:

$$L(\theta) = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Weak Learner

- Decision trees are used as the weak learner in gradient boosting.
- Regression trees output real values for splits and whose output can be added together, allowing subsequent model outputs to be added and correct the residuals in the predictions.
- Trees are constructed in a greedy manner, choosing the best split points based on purity scores like Gini or squared error to minimize the loss.

Weak Learner

- Initially, such as in the case of AdaBoost, very short decision trees were used that only had a single split, called a decision stump.
- Larger trees can be used generally with 4-to-8 levels.
- It is common to constrain the weak learners in specific ways, such as a maximum number of layers, nodes, splits or leaf nodes. This is to ensure that the learners remain weak, but can still be constructed in a greedy manner.

Additive Model

- Trees are added one at a time, and existing trees in the model are not changed.
- A gradient descent procedure is used to minimize the loss when adding trees.
- Instead of updating parameters, a tree is added to the model that reduces the loss (functional space vs. parameter space).
- Done by parameterizing the tree, then modify the parameters of the tree to move in the right direction by (reducing the residual loss).
- Generally this approach is called ***functional gradient descent*** or gradient descent with functions.

Additive Model

- The output for the new tree is then added to the output of the existing sequence of trees in an effort to correct or improve the final output of the model.
- A fixed number of trees are added or training stops once loss reaches an acceptable level or no longer improves on an external validation dataset.

XGBoost – e**X**treme **G**radient **B**oosting

- XGBoost is one of the most popular machine learning algorithms.
- Works for regression and classification.
- XGBoost is the "state-of-the-art” machine learning algorithm to deal with structured data (tabular).
- Use Deep Learning for perceptual tasks such as computer vision or NLP.

XGBoost benefits

- **Speed and performance** :
  - Originally written in C++, often faster than other ensemble classifiers.
- **Core algorithm is parallelizable** :
  - Core XGBoost algorithm is parallelizable. So it can harness the power of multi-core computers, and GPU's.
- **Consistently outperforms other algorithm methods** :
  - It has shown better performance on a variety of machine learning benchmark datasets.
- **Wide variety of tuning parameters** :
  - XGBoost internally has parameters for cross-validation, regularization, user-defined objective functions, missing values, tree parameters, scikit-learn compatible API etc.

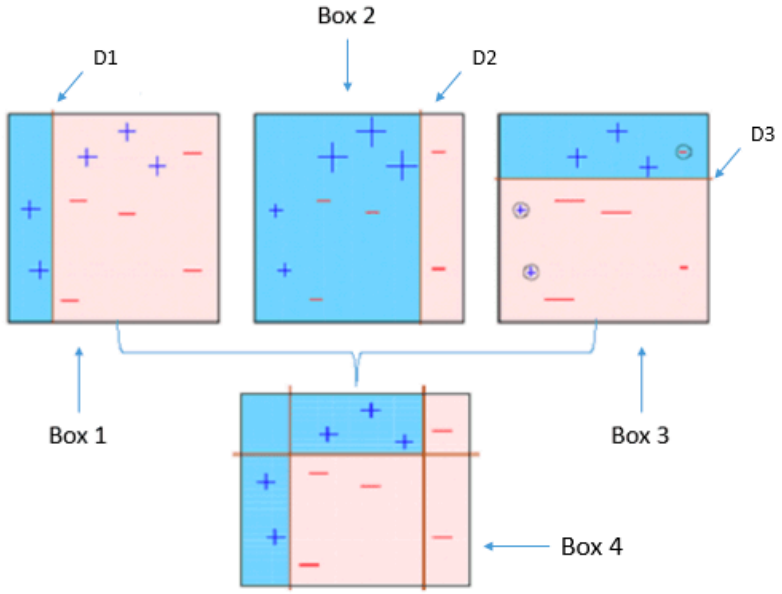
Boosting: Three classifiers trying to classify **+** and **-** classes as homogeneously as possible.

1. Box 1: The first classifier (usually a decision stump) creates a vertical line (split) at D1.

2. Box 2: The second classifier gives more weight to the three **+** misclassified points and creates a vertical line at D2.

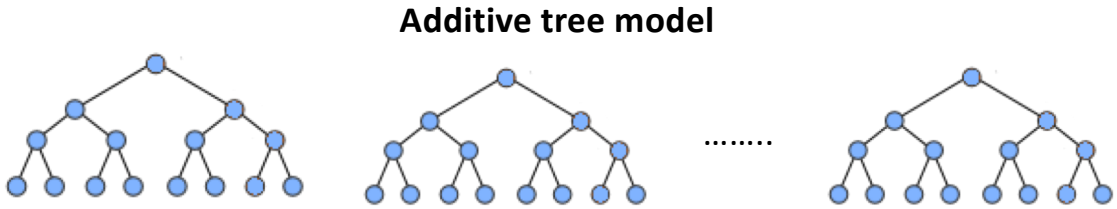
3. Box 3: The third classifier gives more weight to the three **-** misclassified points and creates a horizontal line at D3.

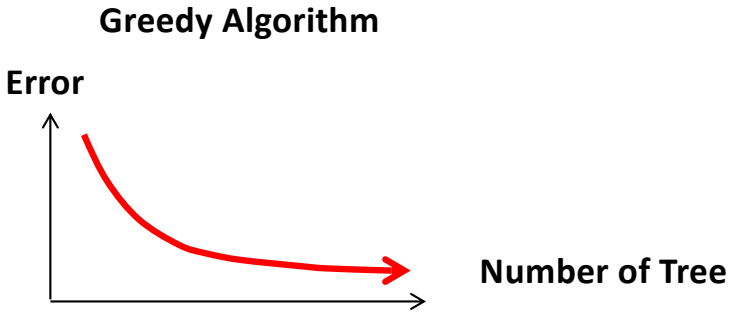
4. Box 4: Weighted combination of the weak classifiers (Box 1,2 and 3).



XGBoost

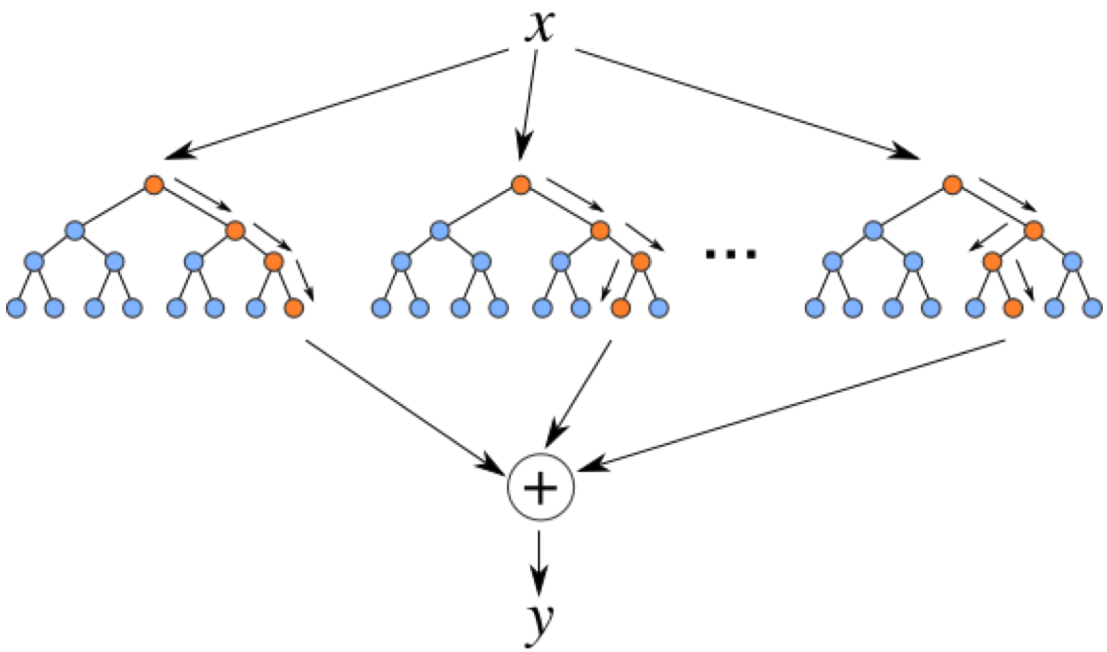
- Additive tree model: add new trees that complement the already-built ones
- Response is the optimal linear combination of all decision trees
- Popular in Kaggle competitions for efficiency and accuracy





XGBoost

- Additive tree model: add new trees that complement the already-built ones
- Response is the optimal linear combination of all decision trees
- Popular in Kaggle competitions for efficiency and accuracy



Tree Boosting - Details

- How should we learn the trees?
  - *define an objective function and optimize it.*
- Let the following be the objective function (needs to contain training loss and regularization):

$$Obj(\Theta) = \overset{n}{L}(\Theta) + \overset{t}{\Omega}(\Theta)$$

Number of nodes in tree, tree depth, L2 norm of leaf weights, etc.

Training Loss measures how well model fit on training data

Regularization, measures complexity of model

- Learn the functions f_i each containing the structure of a tree and the leaf scores.
- Learning tree structure is much harder than traditional optimization problems where you can simply take the gradient and adjust parameters.
- It is intractable to learn all the trees at once.

How we learn: Additive Training (Boosting)

- 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).
- Instead, we use an additive strategy:
  - Fix what we have learned, and add one new tree at a time.

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

New function

Model at training round t

Keep functions added in previous round

Gradient Boosting Algorithm

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
    1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

    2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
    3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

    4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

residuals $y - F(x)$ for a given model are the negative gradients (with respect to $F(x)$) of the squared error loss function $\frac{1}{2}(y - F(x))^2$

Gradient Tree Boosting

Friedman proposes to modify this algorithm so that it chooses a separate optimal value γ_{jm} for each of the tree's regions, instead of a single γ_m for the whole tree. He calls the modified algorithm "TreeBoost". The coefficients b_{jm} from the tree-fitting procedure can be then simply discarded and the model update rule becomes:

$$F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}_{R_{jm}}(x), \quad \gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma).$$

References

- Greedy function approximation a gradient boosting machine. *J.H. Friedman*
  - *The 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