Gradient Boosting and variants

Gonzalo Martínez-Muñoz

Machine learning

- The goal is to find the function F that minimizes the expected loss

$$\hat{F} = \arg\min_{F} \mathbb{E}_{x,y}[L(y, F(x))]$$

by minimizing its data-based estimate

$$\hat{F} = \operatorname{argmin}_{F} \sum_{i=1}^{N} L(y_i, F(\mathbf{x}_i))$$

over the training data $\mathrm{D} = \{(\mathbf{x}_i, y_i)\}_1^N$

Gradient boosting

- Function F is built as an additive model

$$F_T(\mathbf{x}) = \sum_{t=0}^{T} \rho_t h_t(\mathbf{x})$$

sequentially and stagewise as

$$F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \rho_t h_t(\mathbf{x}),$$

with

$$F_0 = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \gamma) \quad \gamma \in \mathbb{R}$$

https://statweb.stanford.edu/~jhf/ftp/trebst.pdf

Gradient boosting algorithm

1. Initialize with a constant value

$$F_0 = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \gamma) \quad \gamma \in \mathbb{R}$$

2. for i=1 to T
   I. Compute pseudo-residuals for each instance

$$\mathrm{r}_{ti} = -\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\bigg|_{F(\mathbf{x})=F_{t-1}(\mathbf{x})}$$

   II. Fit a regressor h(x) on dataset $\{(\mathbf{x}_i, r_{ti})\}_1^N$ ⟸ On MSE

   III. Compute multiplier with a Newton-Raphson step on

$$f(\rho_t) = \sum_{i=1}^{N} L(y_i, F_{t-1}(\mathbf{x}_i) + \rho_t h_t(\mathbf{x}_i))$$

$$\rho_t \approx -f'(\rho_t = 0)/f''(\rho_t = 0)$$

   IV. Update model

$$F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \rho_t h_t(\mathbf{x})$$

For regression

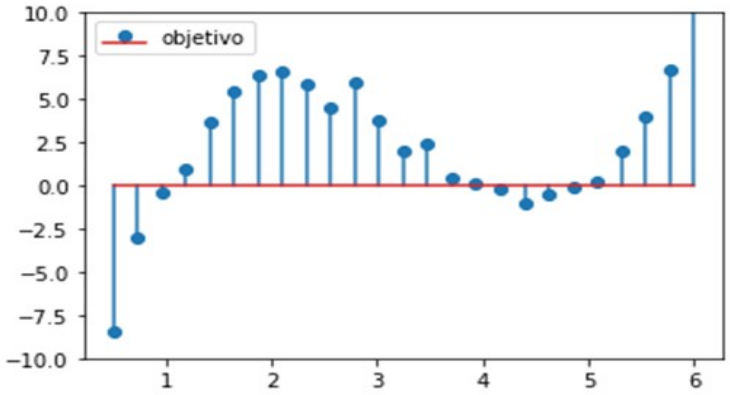
- With square loss: $L(y, F(\mathbf{x})) = (y - F(\mathbf{x}))^2/2$

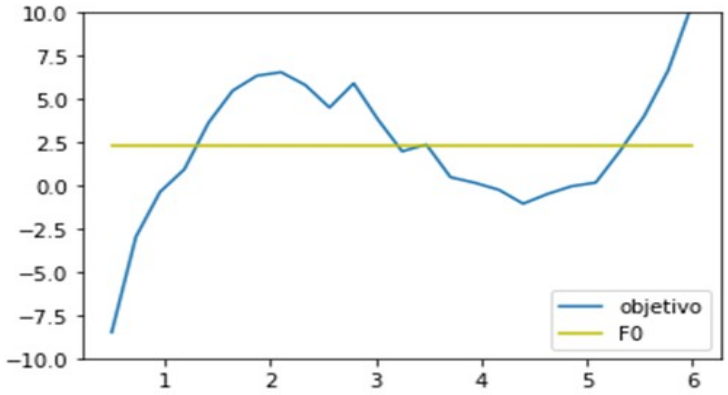
- Init:
$$F_0 = \frac{1}{N}\sum_{i=1}^{N} y_i$$

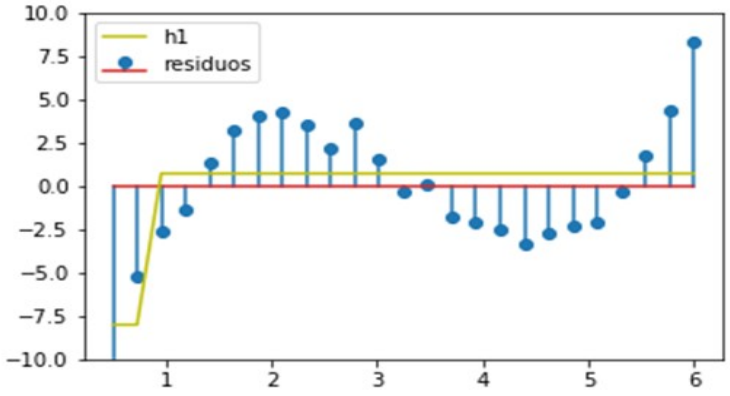
- Residuals:
$$\mathrm{r}_{ti} = y_i - F_{t-1}(\mathbf{x}_i)$$

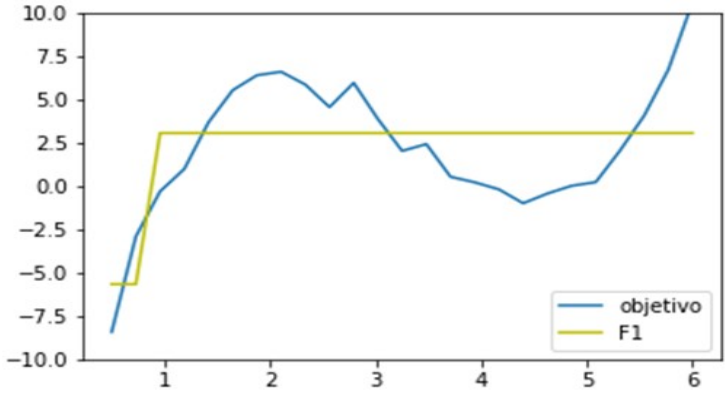
- Multiplier:
$$\rho_t = 1$$

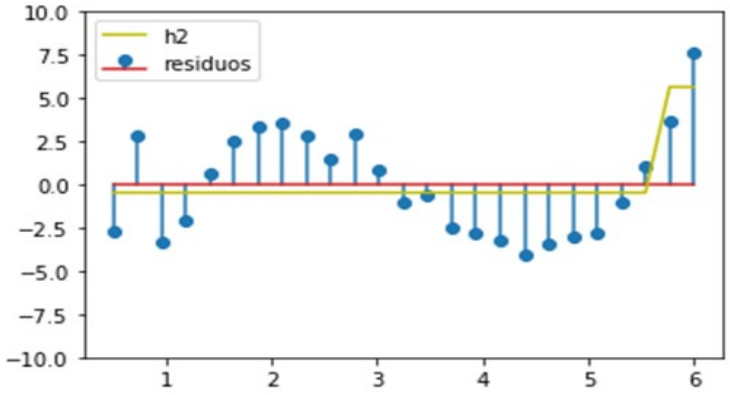
For regression example

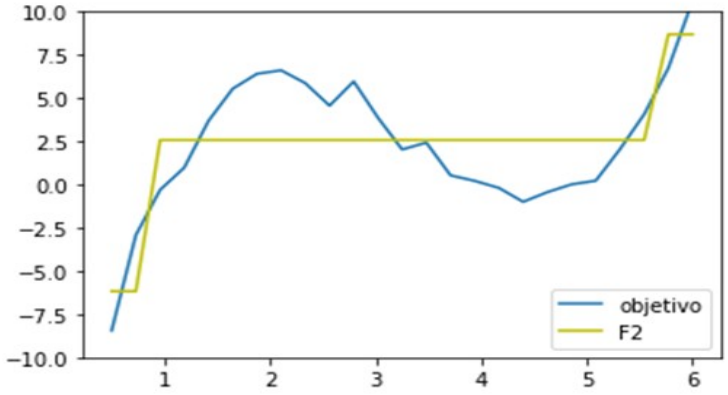












Decision trees as base models

- A decision tree can be seen as an additive model

$$h(\mathbf{x}) = \sum_{j=1}^{J} b_j \mathbb{I}(\mathbf{x} \in R_j)$$

where $\{R_j\}_i^J$ are disjoint regions that cover the space, and we can improve the quality of the fit by computing a different multiplier per node

$$F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \sum_{j=1}^{J} \rho_{tj} \mathbb{I}(\mathbf{x} \in R_j)$$

For regression

- With absolute loss:

$$L(y, F(\mathbf{x})) = |y - F(\mathbf{x})|$$

- Init:

$$F_0 = median\{y_i\}_1^N$$

- Residuals:

$$\mathrm{r}_{ti} = sign(y_i - F_{t-1}(\mathbf{x}_i))$$

- Multiplier (leaf-node output values):

$$\rho_{tj} = median_{\mathbf{x}_i \in R_j}\{y_i - F_{t-1}(\mathbf{x}_i)\}_1^N$$

Gradient boosting variants

- Shrinkage or learning rate: the update rule changes to

$$F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \nu \rho_t h_t(\mathbf{x}) \quad 0 < \nu \leq 1$$

- Control complexity of the trees: # of leaves, depth, etc.

- Stochastic gradient boosting: bootstrap sampling

- Other randomizations: column sampling by level or by tree, subbagging…

For classification

- With loss:

$$L(y, F(\mathbf{x})) = \log(1 + \exp(-2yF)), \quad y \in \{-1, 1\}$$

- Init:

$$F_0 = \frac{1}{2} \log \frac{1 + \overline{y}}{1 - \overline{y}}$$

- Residuals:

$$\mathrm{r}_{ti} = 2y_i / (1 + \exp(2y_i F_{t-1}(\mathbf{x}_i)))$$

- Multiplier:

$$\rho_{tj} = \sum_{\mathbf{x}_i \in R_j} r_{ti} / \sum_{\mathbf{x}_i \in R_j} |r_{ti}|(2 - |r_{ti}|)$$

XGBoost

- An gradient boosting ensemble optimized for speed
- Includes a penalty term for the complexity of the trees in the loss function.

$$\mathbb{E}_{x,y}[L(y, F(x))] + \sum_{m=1}^{M} \Omega(f_m)$$

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

T: Number of leaves

w: outputs of leaves

XGBoost: package

- Loss function with a penalty term for the complexity of the trees
- As Gradient boosting:
  - Shrinkage, Row subsampling, Limits on depth, # of leaves, # of instances, etc.
- As random forest: Column subsampling
- For speed:
  - Binning of data for faster split computation: local and global
  - Data stored in blocks in memory to avoid continuous sorting of the attributes for splitting

XGBoost: package

- Sparsity-aware algorithm for finding the best split: removes zeros from split gain computations.
- Monotonic constrains: force the output to be monotonic wrt to any set of inputs.
- Feature interaction constraints: limits the inputs that can be combined from root to leaves.

LightGBM

- Library especially focused on creating a computationally efficient algorithm
- Based on precomputation of histograms of features, as XGBoost
- New features proposed:
  - Gradient-based One-Side Sampling (GOSS): novel subsampling technique.
  - Exclusive Feature Bundling (EFB) for sparse features.

LightGBM: GOSS

- Gradient-based One-Side Sampling: Each h is trained on a training set with:
  - The top fraction of instances with largest gradients (a)
  - A random sample fraction (b) retrieved from instances with lowest gradients.
  - Low gradient instances are up-weighted by (1-a)/b to compensate for the change of the original distrib
- As in AdaBoost, this technique aims at incrementing the importance of instances with higher uncertainty

LightGBM: EFB

- Exclusive Feature Bundling:
  - Bundles sparse features into a single feature
  - This can be done without losing any information when those features do not have non-zero values simultaneously.
- It mainly increments speed
- This is more a feature preprocessing technique

LightGBM: package

- From Microsoft
- It has...everything:
  - Randomization, optimization, GPU training, parallel execution
- In fact it has over 50 possible hyper-parameters!!

Package: https://lightgbm.readthedocs.io/en/latest/

Paper:
https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf

CatBoost

- A GB library that aims at reducing the prediction shift that occurs during training
- This distribution shift is the depart of

$$F(x_i)|x_i$$

- with x_i being a training instance, with respect to the true

$$F(x)|x$$

- This occurs because in GB the same instances are used both to estimate the gradients and to compute the models that minimize those gradients

CatBoost

- To solve this the idea is to build i = 1, ... , N base models per each of the T boosting iteration.
- The i-th model of the t-th iteration is trained on the first i instances
  - This model is used to estimate the gradient of the (i+1)-th instance for the next iteration
  - The process is repeated with different random permutations of the data.
- This can be very slow...

CatBoost

- The implementation of CatBoost is optimized such that a single tree structure is build per iteration that handles all models.
- To do so: symmetric trees (or decision tables) are used as base models.
  - These trees are grown by extending level-wise all leaf nodes using the same split condition.
- Iteration cost O(#perm · (N + #candidate_splits))

CatBoost

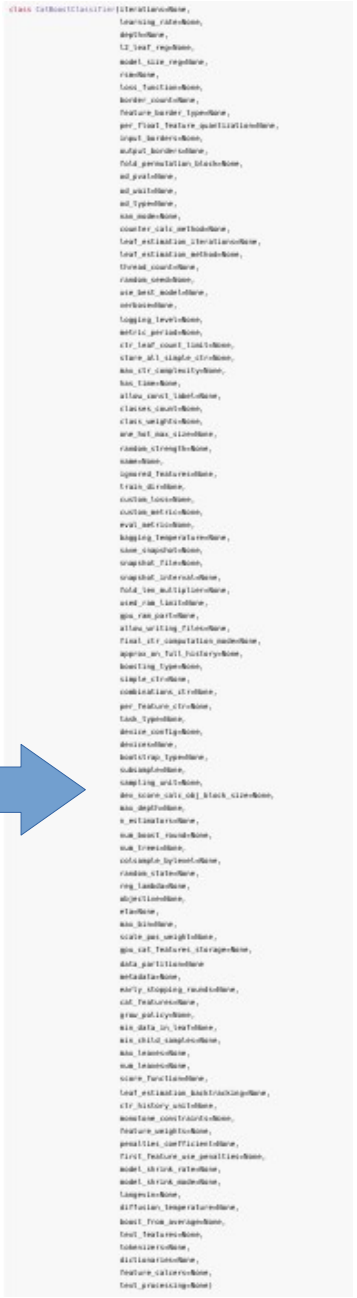
- In CatBoost, categorical features are substituted by a numeric values that measure the expected target value for each category:

$$P(\text{Target} \mid \text{CatFeature})$$

- Ideally the P(Target | Feature) should be calculated on a separate dataset.
  - As it is not possible: a similar procedure to the tree building process is followed
  - The information of instances < i are used to compute the feature value of instance i averaged aver several permutations

CatBoost: package

- Desarrolado en Yandex
- It has...everything:
  - Randomization, optimization, GPU training, parallel execution
- It has 100 hyper-parameters!!



Package: https://catboost.ai/docs/

Paper:
https://papers.nips.cc/paper/7898-catboost-unbiased-boosting-with-categorical-features.pdf

Considerations about GB

- Obtain really good performance. One of the best, if not the best, ML method for tabular data
- In general, it is combined with simple (not very deep) decision trees
- It is important to tune the hyperparameters to get a good performance. Specially: learning rate (or Shrinkage) and complexity of trees.

Comparison

A comparative analysis of gradient boosting algorithms, Candice Bentéjac, Anna Csörgő & Gonzalo Martínez-Muñoz, Artificial Intelligence Review (2020)

DOI:

https://doi.org/10.1007/s10462-020-09896-5