UVA CS 6316: Machine Learning

Lecture 18c: More and Extra about Boosting

Dr. Yanjun Qi

University of Virginia
Department of Computer Science

Boosting:

- Learners are ordered: Each learner tries to reduce error (residual) on “hard” examples (those misclassified by earlier learners).

- ADABOOST: weight hard samples more;

- GRADIENT BOOST: use residual to train later models. Reduces bias and possibly variance compared to base learners.

- Gradient-boosted decision trees (GBDT) often gives state-of-the-art performance on simple classification tasks, e.g. XGBOOST.

- Neural networks are used fairly often with bagging, but rarely with boosting.

- Decision trees work well in both bagging and boosting.

Boosting

- Sequential algorithm where at each step, a weak learner is trained based on the results of the previous learner.
- Two main types:
  - **Adaptive Boosting**: Reweight datapoints based on performance of last weak learner. Focuses on points where previous learner had trouble. Example: AdaBoost.
  - **Gradient Boosting**: Train new learner on residuals of overall model. Constitutes gradient boosting because approximating the residual and adding to the previous result is essentially a form of gradient descent. Example: XGBoost.

credit: Camilo Fosco

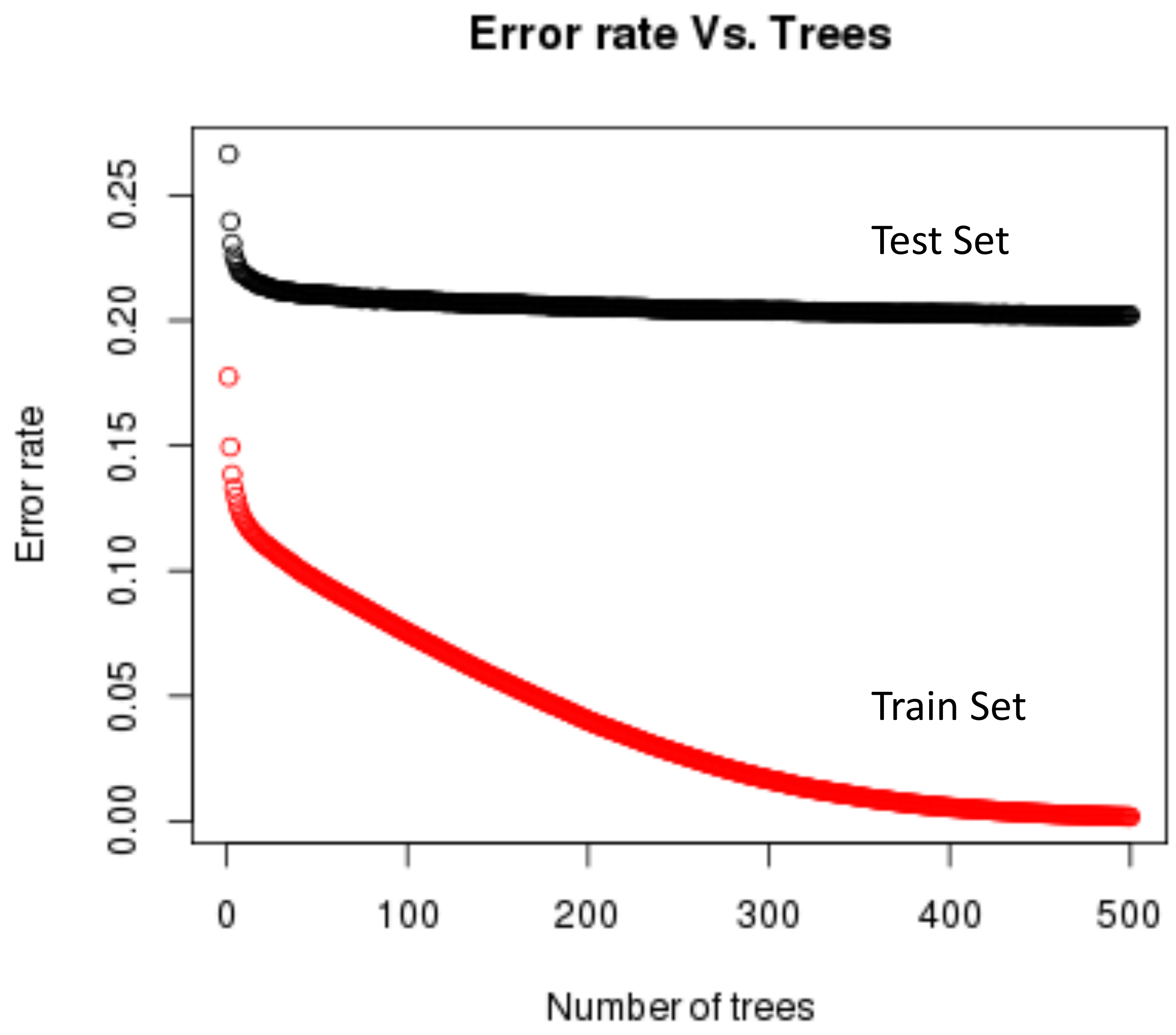
Gradient Boosting

XGBoost

- XGBoost is a very efficient Gradient Boosting Decision Tree implementation with some interesting features:
- **Regularization:** Can use L1 or L2 regularization.
- **Handling sparse data:** Incorporates a sparsity-aware split finding algorithm to handle different types of sparsity patterns in the data.
- **Weighted quantile sketch:** Uses distributed weighted quantile sketch algorithm to effectively handle weighted data.
- **Block structure for parallel learning:** Makes use of multiple cores on the CPU, possible because of a block structure in its system design. Block structure enables the data layout to be reused.
- **Cache awareness:** Allocates internal buffers in each thread, where the gradient statistics can be stored.
- **Out-of-core computing:** Optimizes the available disk space and maximizes its usage when handling huge datasets that do not fit into memory.

credit: Camilo Fosco

XGBoost (an example performance figure)



Gradient Boosting

- Task is to estimate target continuous function F(x). We measure goodness of estimation with loss function $L(y, F(x))$.
- Gradient boosting assumes that:
- $F(x) = \alpha_0 + \alpha_1 h_1(x) + \cdots + \alpha_M h_M(x)$
- Basic Gradient boosting workflow:
  1. Initialize $F_0(x) = \alpha_0$
  2. Estimate α_m and $h_{m(x)}$ such that:

$$L(y, F_{m-1}(x) + \alpha_m h_m(x)) < L(y, F_{(m-1)}(x))$$

  3. Update $F_m(x) = F_{m-1}(x) + \alpha_m h_m(x)$
  4. Repeat from 2, M times.

credit: Camilo Fosco

Gradient Boosting

$$L(y, F_{m-1}(x) + \alpha_m h_m(x)) < L(y, F_{(m-1)}(x))$$

If we can find a vector r_m that we can plug in here to make this equation true, we can train a basic learner $h_m(x)$ to predict r_m from x!

We are basically searching for a vector that points to the direction that reduces our loss… does that sound familiar?

Gradient descent!

credit: Camilo Fosco

Gradient Boosting

- By solving a simple 1D optimization problem, we could also find the optimal α_m for each step, by computing:
- $\alpha_m = argmin_\gamma L(y, F_{m-1}(x) + \gamma h_m(x))$
- This gives us an updated Gradient Boosting algorithm:
    1. Initialize $F_0(x) = \alpha_0$
    2. Compute negative gradient per observation: $r_{m_i} = -\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}$
    3. Train base learner $h_m(x)$ on the gradients
    4. Compute α_m with line search strategy
    5. Update $F_m(x) = F_{m-1}(x) + \alpha_m h_m(x)$
    6. Repeat from 2, M times.

credit: Camilo Fosco

Gradient Boosting

- Where do the residuals come in?

- If we consider Mean Squared Error as our loss function, the per-observation gradient is:

- $$\frac{\partial L(y_i, F_m(x_i))}{\partial F_m(x_i)} = \frac{\partial\left(\frac{1}{2n}\sum_i (y_i - F_m(x_i))^2\right)}{\partial F_m(x_i)} = \frac{\partial\left(\frac{1}{2}(y_i - F_m(x_i))^2\right)}{\partial F_m(x_i)} = y_i - F_m(x_i)$$

- The derivation we found before works with any loss function.

credit: Camilo Fosco

Gradient Tree Boosting

- When dealing with decision trees, we can take the concept further by selecting a specific α_m for each of the tree's regions. The output of a tree is:
- $h_m(x) = \sum^{J_m} b_{jm} 1_{R_{jm}}(x)$
- The model update rule becomes:
- $F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J_m} \alpha_{jm} \mathbf{1}_{R_{jm}(x)}$
- $\alpha_{jm} = argmin_\gamma \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$

J_m: Disjoint regions partitioned by the tree

R_{jm}: Number of leaves

credit: Camilo Fosco

XGBoost

- Three main forms of gradient boosting are supported:
- **Gradient Boosting** algorithm, as we defined above.
- **Stochastic Gradient Boosting** with sub-sampling at the row, column and column per split levels.
  - Random procedure where we subsample observations and features
- **Regularized Gradient Boosting** with both L1 and L2 regularization.
  - add a regularization term to the loss function that we are optimizing:

$$L_R\big(y, F(x)\big) = L\big(y, F(x)\big) + \Omega(F)$$

Where $\Omega(F) = \gamma T + \frac{1}{2}\lambda\|w\|^2$

T: Number of leaves

W: Leaf weights: prediction of each leaf

credit: Camilo Fosco

XGBoost

- Remember, we still want to find the tree structure that minimizes our loss, which means best score structure. Doing this for all possible tree structures is unfeasible.
- A greedy algorithm that starts from a single leaf and iteratively adds branches to the tree is used instead.

credit: Camilo Fosco

XGBoost

- XGBoost adds multiple other important advancements that make it state of the art in several industrial applications.
- In practice:

- Can take a while to run if you don’t set the n_jobs parameter correctly
- Defining the eta parameter (analogous to learning rate) and max_depth is crucial to obtain good performance.
- Alpha parameter controls L1 regularization, can be increased on high dimensionality problems to increase run time.

credit: Camilo Fosco

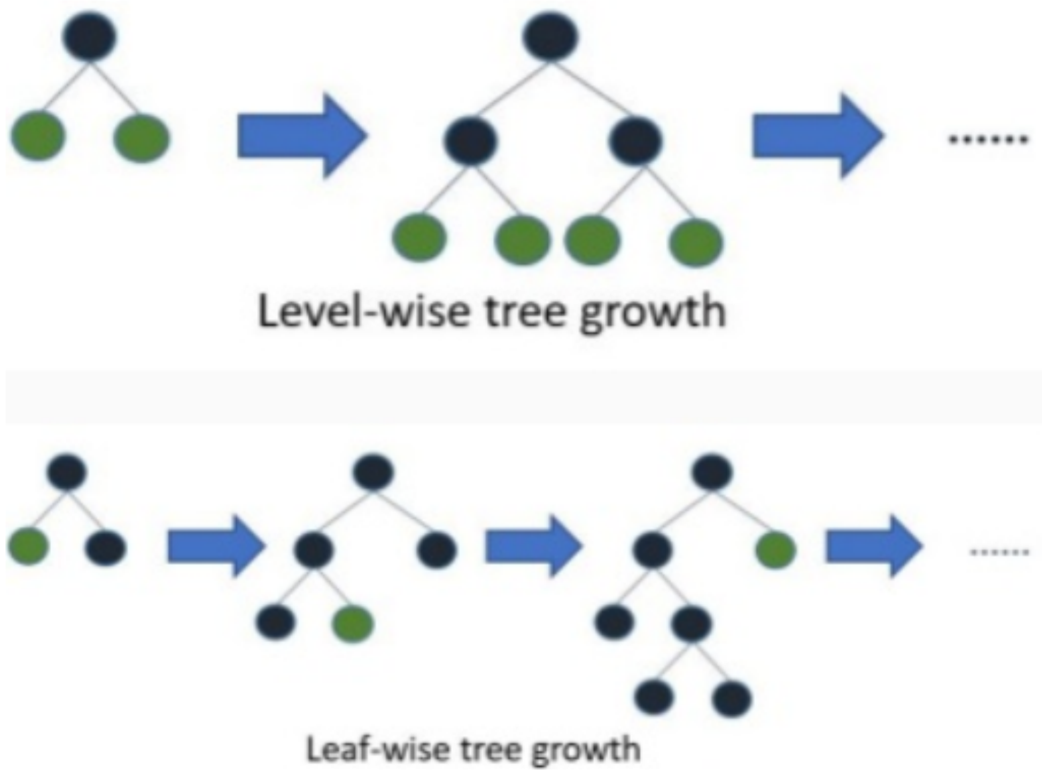
XGBoost

- General approach to parameter tuning:
- Cross-validate **learning rate**.
- Determine the **optimum number of trees for this learning rate**. XGBoost can perform cross-validation at each boosting iteration for this, with the “cv” function.
- **Tune tree-specific parameters** (max_depth, min_child_weight, gamma, subsample, colsample_bytree) for chosen learning rate and number of trees.
- Tune **regularization parameters** (lambda, alpha).

credit: Camilo Fosco

LGBM

- Stands for Light Gradient Boosted Machines. It is a library for training GBMs developed by Microsoft, and it competes with XGBoost.
- Extremely efficient implementation.
- Usually much faster than XGBoost with low hit on accuracy.
- Main contributions are two novel techniques to speed up split analysis: Gradient based one-side sampling and Exclusive Feature Building.
- Leaf-wise tree growth vs level-wise tree growth of XGBoost.



credit: Camilo Fosco