Gradient Boosting Trees

Maria Rigaki

30-11-2018
Ensemble Methods

- Combine the predictions of several base estimators in order to improve generalization and robustness

- **Bagging** or averaging methods build several estimators independently and average their predictions. Ex: Random Forests

- **Boosting** methods build estimators sequentially. Combining several weak estimators to produce an ensemble. Ex: AdaBoost, Gradient Tree Boosting, etc

**Bagging** methods reduce the variance (on average).
**Boosting** methods try to reduce the bias.
Gradient Boosting

High level idea

- Fit an additive model (ensemble) in a forward stage-wise manner.
- In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.
- ”shortcomings” are identified by gradients.
- Gradients tell us how to improve the model.
A simple Boosting algorithm

Dataset: $D = \{(x_1, y_1), (x_2, y_2) \ldots (x_n, y_n)\}$

Task: Fit a model $F(X)$ to minimize square loss $L = (Y - F(X))^2$

1. Initialize $F_0(X) = \frac{1}{N} \sum y_i$
2. for $m = 1$ to $M$:
3. let $r_{m-1} = Y - F_{m-1}(X)$ be the residual vector
4. train a regression tree $h_m(X)$ on $r_{m-1}$
5. Update $F_m(X) = F_{m-1}(X) + h_m(X)$
6. end
Figure 1: Intuition behind Gradient Boosting (From explained.ai)
Simple Boosting demo
What about the Gradient part?

- It turns out that when using the square loss, the residual is equal to the negative gradient.
- In essence, when we update $F$ we use the negative gradient.
- Gradient descent on $F$ (not on the model parameters).
Final Formulation

Additive model of the form:

\[ F(X) = \sum_m \gamma_m h_m(X) \]

where the new tree \( h_m \) tries to minimize the loss \( L \)

\[ h_m = \arg\min \sum_i L(y_i, F_{m-1}(x_i) + h_{m-1}(x_i)) \]

and the update rule is:

\[ F_m(X) = F_{m-1}(X) - \gamma_m \sum_i \nabla_F L(y_i, F_{m-1}(x_i)) \]
Loss Functions (regression)

- Changing from residuals to gradients allows us to change the loss functions.
- Square loss is mostly used but it emphasizes the outliers.
- Absolute loss and Huber loss are also used when robustness to outliers is required.
- Other options are Least Absolute Deviation and Quantile.
Challenges

- Models can overfit

- **Regularization** is achieved using shrinkage or subsampling

- **Shrinkage** is reducing the impact of each added learner.

- **Subsampling** is a combination of boosting and bagging.
Shortcomings

▷ Scalability
▷ What if the data do not fit in the memory?
▷ Can it be used in more than one CPUs or machines?
XGBoost (Chen & Guestrin, 2016)

- Scalable gradient boosting trees
- Very popular algorithm in ML competitions
- It can be used for regression, ranking and classification
- Parallel, Distributed computing and Out of core computing
- Cache aware access
Algorithmic improvements

Tree building
How to find the best split points?
How to choose the feature to split?

Approximate algorithm
Most algorithms use an *exact greedy* approach that requires sorting.
XGBoost proposes candidate splitting points according to percentiles of feature distribution.

Sparsity Aware split finding
Parameter Tuning

- General parameters (number of threads)
- Boosting parameters (stepsize, regularization, tree parameters, etc)
- Task parameters (objective, evaluation metric)
LightGBM (Ke et al., 2017)

- Open source algorithm developed by Microsoft
- Gains in popularity and has won ML competitions
- Speed and Memory Usage optimizations
- Sparsity Optimization
- Accuracy optimizations
- Parallel Learning (feature, data and voting parallelization)
Algorithmic improvements

**Gradient-based One-Side Sampling**
Exclude a significant proportion of data instances with small gradients, and only use the rest to estimate the information gain.

**Exclusive Feature Bundling**
Bundle mutually exclusive features (i.e., they rarely take nonzero values simultaneously), to reduce the number of features without hurting the accuracy.
System improvements

- Data, Feature and Voting parallelization
- Network communication
- GPU support
Parameter Tuning

- Learning Controls (tree related parameters, bagging regularization)
- IO (verbosity, outputs, binarization)
- Objectives
- Metrics
- Network (num_machines, connectivity, etc)
- GPU
Demo time

Ember dataset (Anderson & Roth, 2018)

- A collection of pre-processed Windows binary files
- Features extracted from 1.1M binaries from 2017
- 900K training samples (300K malicious, 300K benign, 300K unlabeled)
- 200K test samples (100K malicious, 100K benign)
Features

- File information: size, imported and exported functions
- Raw bytes histograms, Byte entropy histograms
- Header information
- Strings, etc
- 2351 model features
Results I

- Using a subset of the data: 150K training samples and 50K test samples.
- Training set was 1/3 malicious, 2/3 benign.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AUC</th>
<th>FPR</th>
<th>FNR</th>
<th>Training (sec)</th>
<th>Prediction (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGBoost</td>
<td>0.944</td>
<td>0.037</td>
<td>0.07</td>
<td>136</td>
<td>1.62</td>
</tr>
<tr>
<td>LightGBM</td>
<td>0.966</td>
<td>0.019</td>
<td>0.05</td>
<td>75</td>
<td>0.87</td>
</tr>
<tr>
<td>RF</td>
<td>0.959</td>
<td>0.010</td>
<td>0.07</td>
<td>272</td>
<td>1.43</td>
</tr>
</tbody>
</table>

Table 1: Results on smaller dataset
Results II

- Same settings as before, only with the full dataset: 600K training samples, 200K test samples.
- Balanced dataset.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AUC</th>
<th>FPR</th>
<th>FNR</th>
<th>Training (sec)</th>
<th>Prediction (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightGBM</td>
<td>0.986</td>
<td>0.011</td>
<td>0.01</td>
<td>283</td>
<td>3.43</td>
</tr>
<tr>
<td>RF</td>
<td><strong>0.989</strong></td>
<td>0.011</td>
<td>0.01</td>
<td>2258</td>
<td>8.67</td>
</tr>
</tbody>
</table>

*Table 2: Results on the full dataset*
Conclusions

- LightGBM performed surprisingly well with no tuning.
- XGBoost required some settings even with the smaller dataset.
- Random Forests performed really well but the training time required was significantly longer.
- The above do NOT mean that XGBoost is a worse library!
- Both Gradient Boosting libraries have a large number of parameters.
https://explained.ai/gradient-boosting/index.html


https://xgboost.readthedocs.io/en/latest/

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

