From Gradient Boosting to XGBoost to LambdaMART: An Overview

Liam Huang*

December 18, 2016

*liamhuang0205@gmail.com

1 Notations

- $\vec{x} \in \mathbb{R}^d$, a d-dimesion feature vector; $y \in \mathbb{R}$, ground truth.
- $(\vec{x}, y)$, a sample point; $S = \left\{(\vec{x}_i, y_i)\right\}_{i=1}^{N}$, a sample set with N sample points.
- $F : \mathbb{R}^d \mapsto \mathbb{R}$, a model, or a function; denote $\hat{y} = F(\vec{x})$, or $\hat{y}_i = F(\vec{x}_i)$ for a specific point in the set.
- $l : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$, the loss function, measures the gap between predict and ground truth.
- $L(F) = \sum_{i=1}^{N} l(y_i, \hat{y}_i)$: the global loss on the set.
- $\Omega(F)$: the regularization, measures the complexity of a specific model.

2 Target and Loss

Finding a good enough F^* to predict.

When we are talking about "good", we are actually talking about a standard: loss function $l : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$.

$$l(y, \hat{y}) = l(y, F(\vec{x})).$$

Target transforms:

$$F^* = \arg\min_F E_{y,\vec{x}}\Big[l(y, F(\vec{x})\Big] = \arg\min_F L(F).$$

Suppose F has a fix structure, with undetermined params,

$$F = F(\vec{x}; \vec{P}).$$

Target transforms:

$$\begin{cases} \vec{P}^* = \arg\min_{\vec{P}} L(F(\vec{x}; \vec{P})), \\ F^* = F(\vec{x}; \vec{P}^*). \end{cases}$$

3 Boosting comes

Brickwall ahead:

- with iron fist: rashly break it;
- without: *bypass* it.

Introduce: Addition Model, *break a difficult problem down to series of simple problems*.

$$F = F_M(\vec{x}; \vec{P}_M) = \sum_{m=1}^{M} f_m(\vec{x}; \vec{p}_m),$$

Learner = sum of base-learners.

3.1 The m-th iteration

Fact: F_{m-1} is not good enough.

$$L\Big(F_{m-1}(\vec{x}; \vec{P}_{m-1})\Big) \text{ needs to be descented.}$$

Hence, f_m models the residual error between F_{m-1} and the ground truth.

Target transforms:

$$f_m = \arg\min_f L(F_{m-1} + f).$$

3.2 Another brickwall comes

f is an element in functional space, hard to search.

Introduce: the double jump.

1. Which direction should the model, F, go, to reduce loss?
2. What is the appropriate size to go, in the direction?

3.3 The Gradient

Gradient: the direction that function *increases* fastest.

$$\vec{g} = \frac{\partial L(F)}{\partial F}.$$

That is, for a small change $f = \Delta F$,

$$L(F + f) \approx L(F) + \vec{g} \cdot f = L(F) + \frac{\partial L(F)}{\partial F} \cdot f$$

goes fastest.

We need the opposite direction!

3.4 The Line Search

Here, we have

- the Global Loss: $L(F)$, and
- the steepest-descent direction: $-\vec{g} = -\frac{\partial L(F)}{\partial F}$.

We need a step-size, say ρ, s.t.

$$\rho^* = \arg\min_{\rho} \Big[L(F - \rho \cdot \vec{g}) \Big].$$

3.5 For the very m-th iteration, the Procedure

Suppose f has the general from $f = h(\vec{x}; \vec{a})$, while $\vec{a}$ is the undetermined params.

Algorithm 1 Gradient and Line Search

```
procedure Gradient and Line Search(S = {(x⃗, y)}, N = |S|, m)
    for i : 1 → N do                                   ▷ Get Gradients for each sample point.
        g_i ← ∂L(F_{m-1}(x⃗_i)) / ∂F_{m-1}(x⃗_i)
    end for
    g⃗ ← {g_1, g_2, ..., g_N}
    a⃗* ← arg min_a⃗ Σ_{i=1}^{N} [l(−g_i, h(x⃗_i; a⃗))]
    ρ* ← arg min_ρ [L(F_{m-1} + ρ · h(x⃗; a⃗*))]
    return f ← ρ* · h(x⃗; a⃗*)
end procedure
```

$$g_i \leftarrow \frac{\partial L\left(F_{m-1}(\vec{x}_i)\right)}{\partial F_{m-1}(\vec{x}_i)}$$

$$\vec{a}^* \leftarrow \arg\min_{\vec{a}} \sum_{i=1}^{N} \left[l\left(-g_i, h(\vec{x}_i; \vec{a})\right)\right]$$

$$\rho^* \leftarrow \arg\min_{\rho} \left[L\left(F_{m-1} + \rho \cdot h(\vec{x}; \vec{a}^*)\right)\right]$$

3.6 Gradient Boosting, the Gradient

Algorithm 2 Gradient Boosting

procedure Gradient Boosting($S = \{(\vec{x}, y)\}$, $N = |S|$, M, η)
    $F_0(\vec{x}) \leftarrow \arg\min_\rho L(\rho)$ ▷ Initialization.
    for $m : 1 \rightarrow M$ **do** ▷ Get base-learners, and update the model.
        $f_m \leftarrow$ Gradient and Line Search(S, N, m) ▷ Algorithm 1.
        $F_m \leftarrow F_{m-1} + \eta \cdot f_m$ ▷ Update the model.
    end for
    return $F^* \leftarrow F_M$
end procedure

4 Gradient Boosting Decision Tree

The term "tree" here, means the Classification and Regression Tree (CART).

Specific structure of base-leaner:

- slices the feature space into J disjoint parts, and
- gives sample points in each part an output score.

$$\begin{aligned} f &= \rho \cdot h(\vec{x}; \vec{a}) \\ &= \rho \cdot h\Big(\vec{x}; \{o_j, R_j\}_{j=1}^{J}\Big) \end{aligned}$$

If we treat $\vec{a} = \{o_j, R_j\}_{j=1}^{J}$, then algorithm 2 could be used directly. However, fact comes

$$\rho \cdot h(\vec{x}; \{o_j, R_j\}_{j=1}^{J}) = h(\vec{x}; \{\rho \cdot o_j, R_j\}_{j=1}^{J}).$$

Modify the original algorithm:

- determines $\{R_j\}_{j=1}^{J}$ with $o_j = \mathrm{avg}_{\vec{x} \in R_j}\, g_i$, in the first search; and
- determines $\{w_j = \rho_j \cdot o_j\}_{j=1}^{J}$ in the line search.

$$f = \sum_{j=1}^{J} w_j \cdot I(\vec{x} \in R_j).$$

Algorithm 3 CART Search

procedure CART Search($S = \{(\vec{x}, y)\}$, $N = |S|$, m)
  for $i : 1 \to N$ **do** ▷ Get Gradients for each sample point.
    $g_i \leftarrow \frac{\partial L\left(F_{m-1}(\vec{x}_i)\right)}{\partial F_{m-1}(\vec{x}_i)}$
  end for
  $\{R_j^*\}_{j=1}^{J^*} \leftarrow \arg\min_{\{R_j\}_{j=1}^{J}} \sum_{i=1}^{N} \left[l\left(-g_i, \sum_{j=1}^{J} \mathrm{avg}_{\vec{x} \in R_j} g_i \cdot I(\vec{x}_i \in R_j)\right)\right]$ ▷ Learn the structure of CART.
  $\{w_j^*\}_{j=1}^{J^*} \leftarrow \arg\min_{\{w_j\}_{j=1}^{J^*}} \left[L(F_{m-1} - \sum_{j=1}^{J^*} w_j \cdot I(\vec{x} \in R_j^*))\right]$
  return $f \leftarrow \sum_{j=1}^{J^*} w_j^* \cdot I(\vec{x} \in R_j^*)$
end procedure

Algorithm 4 Gradient Boosting Decision Tree

procedure Gradient Boosting($S = \{(\vec{x}, y)\}$, $N = |S|$, M, η)
  $F_0(\vec{x}) \leftarrow \arg\min_\rho L(\rho)$ ▷ Initialization.
  for $m : 1 \to M$ **do** ▷ Get base-learners, and update the model.
    $f_m \leftarrow$ CART Search(S, N, m) ▷ Algorithm 3.
    $F_m \leftarrow F_{m-1} + \eta \cdot f_m$ ▷ Update the model.
  end for
  return $F^* \leftarrow F_M$
end procedure

5 XGBoost, what's the special?

Engineering problems:

- Hate to compute $l(\cdot, \cdot)$ so many times.

  Every structure and every feature, a round of N loss function will be calculated.

- Search space of $\{R_j\}_{j=1}^{J}$ is tremendous.

Max Depth	3	4	5	6
Possibilities	26	677	458,330	210,066,388,901

  Asymptotic: $A(k) = A^2(k-1) + 1 \quad (k > 1)$, $O\left(2^{2^k}\right)$.

- How to prevent from overfitting?

Who is the apostle?

5.1 Recall: the Taylor Expansion

It takes a infinate sum as the approximate to an infinitely differentiable function.

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots .$$

The second order Taylor Expansion:

$$f(x+\Delta x) \approx f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2.$$

Apply it to the global loss, for the m-th iteration:

$$L(F_m) \approx \sum_{i=1}^{N}\Big[l(y_i, F_{m-1}(\vec{x}_i)) + g_i f_m(\vec{x}_i) + \frac{1}{2} h_i f_m^2(\vec{x}_i)\Big], \qquad \begin{cases} g_i = \frac{\partial l(y_i, F_{m-1}(\vec{x}_i))}{\partial F_{m-1}(\vec{x}_i)}, \\ h_i = \frac{\partial^2 l(y_i, F_{m-1}(\vec{x}_i))}{(\partial F_{m-1}(\vec{x}_i))^2}, \end{cases}$$

$$= \sum_{i=1}^{N}\Big[g_i f_m(\vec{x}_i) + \frac{1}{2} h_i f_m^2(\vec{x}_i)\Big] + \text{Constant}.$$

We've just kicked the loss function out.

5.2 Mathematics Transformation

For a specific structure of CART $\{R_j\}_{j=1}^{J}$, introduce

$$\begin{aligned} I_j &= \{i \mid \vec{x}_i \in R_j\}, \\ G_j &= \sum_{i \in I_j} g_i, \\ H_j &= \sum_{i \in I_j} h_i. \end{aligned}$$

Revisit the Global Loss

$$\begin{aligned} L(F_m) &\approx \sum_{i=1}^{N} \Big[g_i f_m(\vec{x}_i) + \frac{1}{2} h_i f_m^2(\vec{x}_i) \Big] + \text{Constant} \\ &= \sum_{j=1}^{J} \Big[G_j w_j + \frac{1}{2} H_j w_j^2 \Big] + \text{Constant} \end{aligned}$$

For $H > 0$,

$$\arg\min_x \Big[Gx + \frac{1}{2} Hx^2 \Big] = -\frac{G}{H}, \quad \min_x \Big[Gx + \frac{1}{2} Hx^2 \Big] = -\frac{G^2}{2H}.$$

Here comes the magic,

$$L(F) = \min_x \Big[\sum_{j=1}^{J} \Big[G_j w_j + \frac{1}{2} H_j w_j^2\Big]\Big] = -\frac{1}{2} \sum_{j=1}^{J} \Big[\frac{G_j^2}{H_j}\Big],$$

$$w_j^* = \arg\min_x \Big[G_i x + \frac{1}{2} H_i x^2\Big] = -\frac{G_i}{H_i}.$$

Two advantages, for specific structure:

- get $\{w_j\}_{j=1}^{J}$ directly, no optimization any longer, and
- get the general form of global loss.

Is every $H_j > 0$?

5.3 Degradation: Greedy Search for Split

Search space is tremendous, we need a degradation:
search each split point greedily — max positive gain in each split.

Gain: the reduction of global loss, after split.

- Before: $-\frac{1}{2}\frac{(G_L+G_R)^2}{H_L+H_R}$.
- After: $-\frac{1}{2}\left[\frac{G_L^2}{H_L}+\frac{G_R^2}{H_R}\right]$.
- Gain: $\frac{1}{2}\left[\frac{G_L^2}{H_L}+\frac{G_R^2}{H_R}-\frac{(G_L+G_R)^2}{H_L+H_R}\right]$.

Algorithm 5 Split Finding

procedure Split Finding($S = \{(\vec{x}, y)\}$, $N = |S|$, G, H, $\vec{g}$, $\vec{h}$)
    $L \leftarrow$ empty list
    for $k : 1 \rightarrow d$ **do** ▷ Search the best split for each feature.
        Sort S by feature k; get Z_k split points.
        $M_k \leftarrow (0, 0)$
        $G_L \leftarrow 0, \quad H_L \leftarrow 0$
        $G_R \leftarrow G, \quad H_R \leftarrow H$
        for $z : 1 \rightarrow Z_k$ **do** ▷ Attempt each candidate split point, calculate the gain.
            $G_L \leftarrow G_L + g_z, \quad H_L \leftarrow H_L + h_z$
            $G_R \leftarrow G_R - g_z, \quad H_R \leftarrow H_R - h_z$
            $C \leftarrow \left[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G^2}{H}\right]$
            if $C > M_k[0]$ **then** ▷ Update best Split Point for current feature.
                $M_k \leftarrow (C, z)$
            end if
        end for
        if $M_k[0] > 0$ **then**
            Append (k, M_k) to L
        end if
    end for
    if L **then**
        return $\max_{(k, M_k)}[L]$
    else
        return None
    end if
end procedure

5.4 Regularization

Tree growing could be overfitting, need a limitation to restrict growing.

Regularization: describe complexity of a CART.

$$\omega(f) = \gamma J + \frac{1}{2}\lambda \sum_{j=1}^{J} w_j^2,$$

$$\Omega(F) = \sum_{m=1}^{M} \Big[\omega(f_m)\Big].$$

Objective Function and value on leaf:

$$\text{Obj} = -\frac{1}{2}\sum_{j=1}^{J} \frac{G_j^2}{H_j + \lambda} + \gamma J,$$

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

Algorithm 6 Split Finding with Regularization

```
procedure Split Finding with Regularization(S = {(x⃗, y)}, N = |S|, G, H, g⃗, h⃗)
    L ← empty list
    for k : 1 → d do                                   ▷ Search the best split for specific feature.
        Sort S by feature k; get Z_k split points.
        M_k ← (0, 0)
        G_L ← 0,    H_L ← 0
        G_R ← G,    H_R ← H
        for z : 1 → Z_k do                             ▷ Attempt each candidate split point, calculate the gain.
            G_L ← G_L + g_z,    H_L ← H_L + h_z
            G_R ← G_R − g_z,    H_R ← H_R − h_z
            C ← [ G_L^2/(H_L+λ) + G_R^2/(H_R+λ) − G^2/(H+λ) − γ ]
            if C > M_k[0] then                         ▷ Update best Split Point for current feature.
                M_k ← (C, z)
            end if
        end for
        if M_k[0] > 0 then
            Append (k, M_k) to L
        end if
    end for
    if L then
        return max_{(k,M_k)}[L]
    else
        return None
    end if
end procedure
```

5.5 GBDT in XGBoost

Algorithm 7 CART Search in XGBoost

procedure CART Search in XGBoost($S = \{(\vec{x}, y)\}$, $N = |S|$, m)
  for $i : 0 \rightarrow N$ **do** ▷ Get Gradients for each sample point.
    $g_i \leftarrow \frac{\partial L\left(F_{m-1}(\vec{x}_i)\right)}{\partial F_{m-1}(\vec{x}_i)}$
    $h_i \leftarrow \frac{\partial^2 L\left(F_{m-1}(\vec{x}_i)\right)}{(\partial F_{m-1}(\vec{x}_i))^2}$
  end for
  $\vec{g} \leftarrow [g_1, g_2, \ldots, g_N]$
  $\vec{h} \leftarrow [h_1, h_2, \ldots, h_N]$
  Grow tree by Split Finding with Regularization($S, N, G, H, \vec{g}, \vec{h}$) ▷ Algorithm 6.
  return $f \leftarrow$ tree
end procedure

Algorithm 8 GBDT in XGBoost

procedure GBDT in XGBoost($S = \{(\vec{x}, y)\}$, $N = |S|$, M, η)
  $F_0(\vec{x}) \leftarrow \arg\min_\rho L(\rho)$ ▷ Initialization.
  for $m : 0 \rightarrow M$ **do** ▷ Get base-learners, and update the model.
    $f_m \leftarrow$ CART Search in XGBoost(S, N, m) ▷ Algorithm 7.
    $F_m \leftarrow F_{m-1} + \eta \cdot f_m$ ▷ Update the model.
  end for
  return $F^* \leftarrow F_M$
end procedure

6 Implementation Details in XGBoost

Code Snippet 1: Initialization, **InitModel** - learner.cc

```
...
// mparam.base_score, bias of each base-learner
gbm_.reset(GradientBooster::Create(name_gbm_, cache_, mparam.base_score));
...
```

Code Snippet 2: Main Training Logic, **CLITrain** - cli_main.cc, Algorithm 8

```
...
for (int i = 0; i < param.num_round; ++i) {
  learner->UpdateOneIter(i, dtrain.get());
}
...
```

Code Snippet 3: **UpdateOneIter** - learner.cc, Algorithm 7

```
void UpdateOneIter(int iter, DMatrix* train) override {
  ...
  // get predict scores from last iteration.
  this->PredictRaw(train, &preds_);
  // get gradient and hessian
  obj_->GetGradient(preds_, train->info(), iter, &gpair_);
  // boost one iteration
  gbm_->DoBoost(train, &gpair_, obj_.get());
}
```

Recall, in LambdaMART:

$$\lambda_{ij} \stackrel{\text{def}}{=} -\frac{\exp[-x]}{1+\exp[-x]} \cdot \Delta|\text{NDCG}|$$

$$h_{ij} \stackrel{\text{def}}{=} \frac{\mathrm{d}\lambda}{\mathrm{d}x} = \frac{\exp[-x]}{(1+\exp[-x])^2} \cdot \Delta|\text{NDCG}|$$

Code Snippet 4: Get Lambda and Hessian, **GetGradient** - rank_obj.cc

```
for (size_t i = 0; i < pairs.size(); ++i) {
  // the sample in the pair with higher score
  const ListEntry &pos = lst[pairs[i].pos_index];
  // the sample in the pair with lower score
  const ListEntry &neg = lst[pairs[i].neg_index];
  // the ΔNDCG when swap pos and neg in the list
  const float w = pairs[i].weight; constexpr float eps = 1e-16f;
  // $\frac{1}{1+\exp[-x]}$
  float p = common::Sigmoid(pos.pred - neg.pred);
  // $g \stackrel{\text{def}}{=} -\frac{\exp[-x]}{1+\exp[-x]}$
  float g = p - 1.0f;
  // $h \stackrel{\text{def}}{=} \frac{d\lambda}{dx} = \frac{\exp[-x]}{(1+\exp[-x])^2}$
  float h = std::max(p * (1.0f - p), eps);
  // accumulate gradient and hessian in both pid, and nid
  gpair[pos.rindex].grad += g * w;
  gpair[pos.rindex].hess += 2.0f * w * h;
  gpair[neg.rindex].grad -= g * w;
  gpair[neg.rindex].hess += 2.0f * w * h;
}
```

Code Snippet 5: **DoBoost** - gbtree.cc

```
void DoBoost(DMatrix* p_fmat,
             std::vector<bst_gpair>* in_gpair,
             ObjFunction* obj) override {
  const std::vector<bst_gpair>& gpair = *in_gpair;
  std::vector<std::vector<std::unique_ptr<RegTree> > > new_trees;
  // grow a CART
  BoostNewTrees(gpair, p_fmat, 0, &ret);
  new_trees.push_back(std::move(ret));
  this->CommitModel(std::move(new_trees[0]), 0);
}
```

Code Snippet 6: **BoostNewTrees** - gbtree.cc

```
inline void BoostNewTrees(const std::vector<bst_gpair> &gpair, DMatrix *p_fmat,
  int bst_group, std::vector<std::unique_ptr<RegTree> >* ret) {
  this->InitUpdater();
  std::vector<RegTree*> new_trees;
  ret->clear();
  // create the trees
  ...
  new_trees.push_back(ptr.get());
  ret->push_back(std::move(ptr));
  // update the trees
  for (auto& up : updaters) {
    up->Update(gpair, p_fmat, new_trees);
  }
}
```

Code Snippet 7: **Update** - updater_colmaker.cc

```
virtual void Update(const std::vector<bst_gpair>& gpair, DMatrix* p_fmat,
    RegTree* p_tree) {
  this->InitData(gpair, *p_fmat, *p_tree);
  // root node
  this->InitNewNode(qexpand_, gpair, *p_fmat, *p_tree);
  for (int depth = 0; depth < param.max_depth; ++depth) {
    this->FindSplit(depth, qexpand_, gpair, p_fmat, p_tree);
    this->ResetPosition(qexpand_, p_fmat, *p_tree);
    this->UpdateQueueExpand(*p_tree, &qexpand_);
    this->InitNewNode(qexpand_, gpair, *p_fmat, *p_tree);
    // if nothing left to be expand, break
    if (qexpand_.size() == 0) break;
  }
  // set all the rest expanding nodes to leaf
  for (size_t i = 0; i < qexpand_.size(); ++i) {
    const int nid = qexpand_[i];
    (*p_tree)[nid].set_leaf(snode[nid].weight * param.learning_rate);
  }
  // remember auxiliary statistics in the tree node
  for (int nid = 0; nid < p_tree->param.num_nodes; ++nid) {
    p_tree->stat(nid).loss_chg = snode[nid].best.loss_chg;
    p_tree->stat(nid).base_weight = snode[nid].weight;
    p_tree->stat(nid).sum_hess = static_cast<float>(snode[nid].stats.sum_hess);
    snode[nid].stats.SetLeafVec(param, p_tree->leafvec(nid));
  }
}
```

Code Snippet 8: **FindSplit** - updater_colmaker.cc

```cpp
inline void FindSplit(int depth, const std::vector<int> &qexpand,
    const std::vector<bst_gpair> &gpair, DMatrix *p_fmat, RegTree *p_tree) {
  std::vector<bst_uint> feat_set = feat_index;
  // sample feature, if needed
  ...
  // Algorithm 6, search through features
  dmlc::DataIter<ColBatch>* iter = p_fmat->ColIterator(feat_set);
  while (iter->Next()) {
    this->UpdateSolution(iter->Value(), gpair, *p_fmat);
  }
  // synchronize in dist-calc
  ...
  // get the best result, we can synchronize the solution
  for (size_t i = 0; i < qexpand.size(); ++i) {
    // update the RegTree for each expand point
    ...
  }
}
```