9.13 Paper Presentation

Yibin Xiong

September 13, 2021

Table of Contents

1. Top 10 Algorithms in Data Mining
2. Soft Gradient Boosting Machine
3. XGBoost: A Scalable Tree Boosting System
4. Learning from Incomplete and Inaccurate Supervision
5. ML-KNN: A Lazy Learning Approach to Multi-label Learning
6. Towards Convergence Rate Analysis of Random Forests for Classification

C4.5

- An algorithm that grows an initial decision tree (and prune it later if necessary)
- Idea:
  Given a set of samples S, build a decision tree classifier *recursively*:

  i) If all samples belong to the same class or $|S|$ is small, the tree is just a leaf labeled with the most frequent class in S

  ii) Otherwise, choose a test (criterion for split) based on a single attribute with 2 or more outcomes. Split S into $S_1, S_2, \ldots$ according to the attribute value of the samples.
  Do the same procedures for $S_1, S_2, \ldots$.

C4.5

- An algorithm that grows an initial decision tree (and prune it later if necessary)
- Idea:
  Given a set of samples S, build a decision tree classifier *recursively*:

  i) If all samples belong to the same class or $|S|$ is small, the tree is just a leaf labeled with the most frequent class in S

  ii) Otherwise, choose a test (criterion for split) based on a single attribute with 2 or more outcomes. Split S into $S_1, S_2, \ldots$ according to the attribute value of the samples.
  Do the same procedures for $S_1, S_2, \ldots$.
- Note that the attribute for split can be numerical
  e.g. $S_1 = \{x : x_j > h\}$ and $S_2 = \{x : x_j \leq h\}$

How do we choose the split rules

Given a set of possible split criteria, we evaluate their **information gain** (ID3) or the default **gain ratio** (C4.5).

How do we choose the split rules

Given a set of possible split criteria, we evaluate their **information gain** (ID3) or the default **gain ratio** (C4.5).

Preliminaries: entropy and conditional entropy

- Entropy measures the *uncertainty* of a distribution.

$$\begin{aligned}\mathcal{H}(S) &= -E_{x\sim S}[\log_2 p(x)] \\ &= -\sum_{k=1}^{K} \frac{|C_k|}{|S|} \log_2 \frac{|C_k|}{|S|}\end{aligned}$$

- Conditional entropy
  Let categorical feature A has m possible values.

$$\mathcal{H}(S|A) = \sum_{i=1}^{m} \frac{|S_i|}{|S|} \mathcal{H}(S_i)$$

How do we choose the split rules

- Information Gain:

$$IG(S, A) = \mathcal{H}(S) - \mathcal{H}(S|A)$$

  - Measures how much *uncertainty* decreases when we know about A
  - Tend to favor attributes that have more possible values

How do we choose the split rules

- Information Gain:

$$IG(S, A) = \mathcal{H}(S) - \mathcal{H}(S|A)$$

  - Measures how much *uncertainty* decreases when we know about A
  - Tend to favor attributes that have more possible values

- Gain Ratio:

$$\text{Gain Ratio}(S, A) = \frac{IG(S, A)}{H_A(S)}$$

  where $H_A(S) = -\sum_{i=1}^{m} \frac{|D_i|}{|D|} \log_2 \frac{|S_i|}{|S|}$ (like entropy w.r.t. attribute A but not class label C)
  - solve the issue above, but could favor attributes with less possible values

How do we choose the split rules

- Information Gain:

$$IG(S, A) = \mathcal{H}(S) - \mathcal{H}(S|A)$$

  - Measures how much *uncertainty* decreases when we know about A
  - Tend to favor attributes that have more possible values

- Gain Ratio:

$$\text{Gain Ratio}(S, A) = \frac{IG(S, A)}{H_A(S)}$$

  where $H_A(S) = -\sum_{i=1}^{m} \frac{|D_i|}{|D|} \log_2 \frac{|S_i|}{|S|}$ (like entropy w.r.t. attribute A but not class label C)
  - solve the issue above, but could favor attributes with less possible values

- Heuristic approach: first choose a few candidates based on Gain Ratio, then find the best one based on IG

Pruning

- Avoid overfitting
- Pessimistic estimate of the error rate: upper limit of the binomial probability when E events have been observed in N trials
- Carried out from leaves to the root
- For a subtree, C4.5 adds the estimated error of the branches and compare this with the estimated error of a leaf. If a leaf has smaller error rate, then the subtree is pruned.

CART: Splitting

- Uses *binary* splits $\Rightarrow$ speed up the process for each split (but may have more splits)
- Gini measurement of impurity of a node t

$$G(t) = 1 - p(t)^2 - (1 - p(t))^2$$

where $p(t)$ is the relative frequency of class 1 in the node
  - Faster computation because no log evaluations
  - The lower, the better

CART: Splitting

- Uses *binary* splits $\Rightarrow$ speed up the process for each split (but may have more splits)
- Gini measurement of impurity of a node t

$$G(t) = 1 - p(t)^2 - (1 - p(t))^2$$

where $p(t)$ is the relative frequency of class 1 in the node
  - Faster computation because no log evaluations
  - The lower, the better

- Gain generated by a split is

$$I(P) = G(P) - qG(L) - (1 - q)G(R)$$

where q is the fraction of samples that goes to the left child node

Pruning

- Cost function:

$$C_a(T) := C(T) + a|T|$$

where $|T|$ is the number of leaves of tree T

Pruning

◦ Cost function:

$$C_a(T) := C(T) + a|T|$$

where $|T|$ is the number of leaves of tree T

For any internal node t:
i) consider it as a single leaf, the cost is $C_a(t) = C(t) + a$
ii) consider is as the root of its children nodes, the cost is $C_a(T_t) = C(T_t) + a|T_t|$

Compare $C_a(t)$ and $C_a(T_t)$ to see whether to prune

Pruning

◦ Cost function:

$$C_a(T) := C(T) + a|T|$$

where $|T|$ is the number of leaves of tree T

For any internal node t:
i) consider it as a single leaf, the cost is $C_a(t) = C(t) + a$
ii) consider is as the root of its children nodes, the cost is $C_a(T_t) = C(T_t) + a|T_t|$

Compare $C_a(t)$ and $C_a(T_t)$ to see whether to prune

◦ We generate a sequence of candidate subtrees and cross-validate their performance to choose the best one

Pruning

◦ Cost function:

$$C_a(T) := C(T) + a|T|$$

where $|T|$ is the number of leaves of tree T

For any internal node t:
i) consider it as a single leaf, the cost is $C_a(t) = C(t) + a$
ii) consider is as the root of its children nodes, the cost is $C_a(T_t) = C(T_t) + a|T_t|$

Compare $C_a(t)$ and $C_a(T_t)$ to see whether to prune

◦ We generate a sequence of candidate subtrees and cross-validate their performance to choose the best one
◦ The hyperparameter a represents the tradeoff between cost and complexity. The larger a is, the simpler the model but probably higher cost.

Missing Values

- Use surrogate tests
- Treat the attribute with missing values as *target*
- design decision trees(surrogate tests) that uses other relevant attributes without missing values to predict the target

CART vs C4.5

- Tests in CART are binary, while C4.5 allows more than 2 outcomes
- CART uses Gini-index, while C4.5 uses gain ratio and IG
- Different pruning approach
- CART handles missing values for an attribute of a sample, but C4.5 does not
- CART evaluates tree performance on validation data, while C4.5 uses training data

Table of Contents

1. Top 10 Algorithms in Data Mining
2. Soft Gradient Boosting Machine
3. XGBoost: A Scalable Tree Boosting System
4. Learning from Incomplete and Inaccurate Supervision
5. ML-KNN: A Lazy Learning Approach to Multi-label Learning
6. Towards Convergence Rate Analysis of Random Forests for Classification

Motivation

- It is difficult for GBM to handle streaming data (when new data are added) because GBM needs all the data when training

Motivation

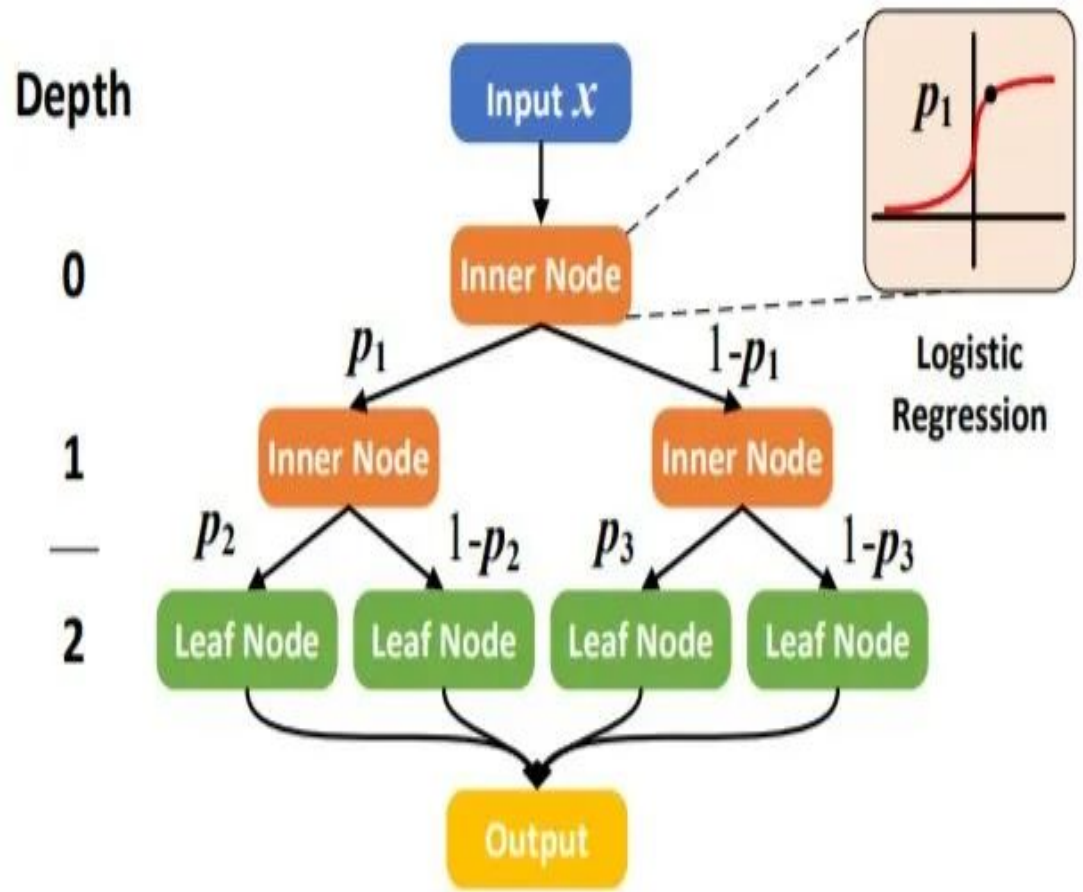
- It is difficult for GBM to handle streaming data (when new data are added) because GBM needs all the data when training
- Differential programming models, which has better representation learning ability, requires the learning modules to be *differentiable*.
- The base learners of GBM (i.e. CART) is NOT differentiable

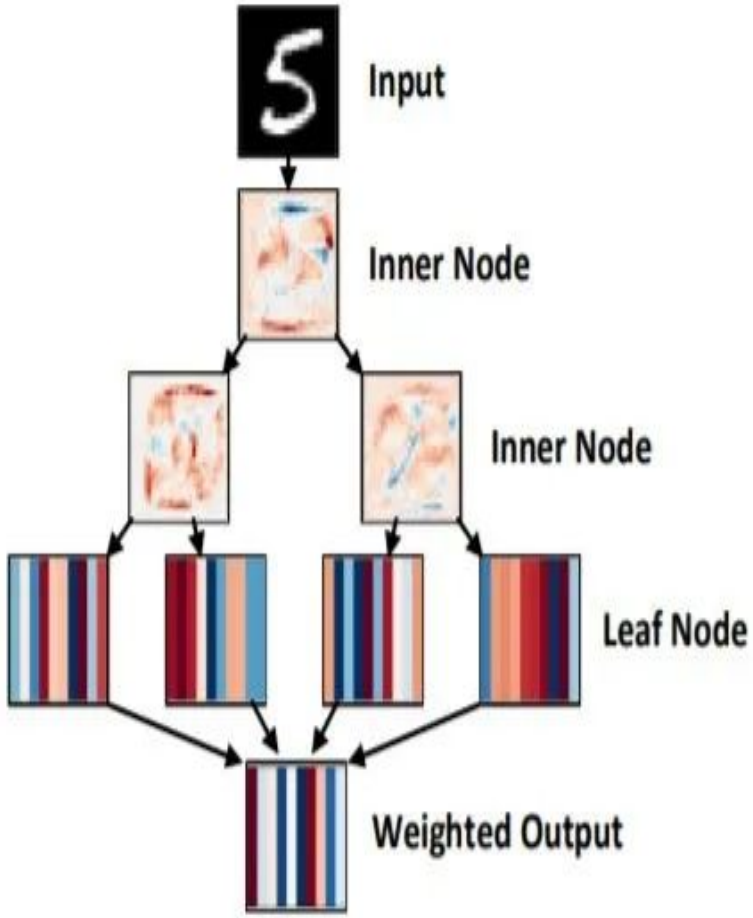
Motivation

- It is difficult for GBM to handle streaming data (when new data are added) because GBM needs all the data when training
- Differential programming models, which has better representation learning ability, requires the learning modules to be *differentiable*.
- The base learners of GBM (i.e. CART) is NOT differentiable
- Build a differentiable system that behaves like GBM, thus retain both advantages
- Other advantages of the soft version

Base Learner: Soft Decision Trees

Soft means assignment by *probability* into all children nodes.





Composition of logistic functions $\Rightarrow$ *Differentiable*

Soft GBM

Algorithm 2: Training sGBM

Input: Training batches $\mathcal{B} = \{B_1, B_2, \cdots, B_{|\mathcal{B}|}\}$, number of base learner M, current sGBM parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}_m\}_{m=1}^{M}$

Output: Updated sGBM parameters $\boldsymbol{\theta}$

for $b = 1$ *to* $|\mathcal{B}|$ **do**

  $o_0^i \leftarrow 0 \quad \text{for } \mathbf{x}^i \in B_b$; // Initialize

  for $m = 1$ *to* M **do**

    $o_m^i \leftarrow h_m(\mathbf{x}^i; \boldsymbol{\theta}_m) \quad \text{for } \mathbf{x}^i \in B_b$; // Data forward

    $r_m^i \leftarrow -\frac{\partial l(\sum_{j=0}^{m-1} o_j^i, y^i)}{\partial \sum_{j=0}^{m-1} o_j^i} \quad \text{for } \mathbf{x}^i \in B_b$; // Residual

    $l_m \leftarrow \sum_{\mathbf{x}^i \in B_b} \left\| r_m^i - o_m^i \right\|_2^2$; // Local learner loss

  end

  $\mathcal{L} \leftarrow \sum_{i=1}^{M} l_m$; // Global sGBM loss

  Update $\boldsymbol{\theta}$ w.r.t $\mathcal{L}$ using gradient descent ; // Update sGBM

end

return $\boldsymbol{\theta}$;

ensemble effect

Advantages over (Hard) GBM

- Parallelizable rather than sequential, much faster
- Parameterized and differentiable $\Rightarrow$ adaptable to changes in the environment
- Hard GBM is not efficient to tackle *multi-output* regression problems because each tree can only have one dimension for output
- More suitable for distill and twice learning

  "When the soft targets have high entropy, they provide much more information per training case than hard targets and much less variance in the gradient between training cases..."

Table of Contents

1. Top 10 Algorithms in Data Mining
2. Soft Gradient Boosting Machine
3. XGBoost: A Scalable Tree Boosting System
4. Learning from Incomplete and Inaccurate Supervision
5. ML-KNN: A Lazy Learning Approach to Multi-label Learning
6. Towards Convergence Rate Analysis of Random Forests for Classification

Work

A highly **scalable** implementation of gradient tree boosting

Accuracy:

- Weighted Quantile Sketch
- Sparsity-aware Split Finding

Efficiency:

- Column Block for Parallel Learning
- Cache-aware Access
- Blocks for Out-of-core Computation

Gradient Tree Boosting

We define the loss function with regularization as

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

where $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \|w\|^2$. T is the number of leaves and w is the vector of leaf values.

Gradient Tree Boosting

We define the loss function with regularization as

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

where $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \|w\|^2$. T is the number of leaves and w is the vector of leaf values.

Using **2nd order Taylor expansion** at the point $\hat{y}_i^{(t-1)}$, we get

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

Collect 1st and 2nd order gradient statistics from samples to estimate the loss, and minimize the estimated loss

Algorithm

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do** k represents a feature that the split uses
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do** j represents the index of
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$ an example
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

Algorithm

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ ***to*** m **do** // propose different tree structures

  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.

  Proposal can be done per tree (global), or per split(local).

end

for $k = 1$ ***to*** m **do**

  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$

  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$

end

Follow same step as in previous section to find max score only among proposed splits.

Table of Contents

1. Top 10 Algorithms in Data Mining
2. Soft Gradient Boosting Machine
3. XGBoost: A Scalable Tree Boosting System
4. Learning from Incomplete and Inaccurate Supervision
5. ML-KNN: A Lazy Learning Approach to Multi-label Learning
6. Towards Convergence Rate Analysis of Random Forests for Classification

Motivation

- real-life data is:
  - incomplete (not every instance is labeled)
  - inaccurate (e.g. with one-sided noise)
    One-sided noise: positively labeled samples are true positive, but negatively labeled ones can be actually positive or negative
- We can handle these 2 problems individually, BUT it is difficult to handle them *simultaneously*.
- semi-supervised learning (SSL) requires accurate supervision and noisy labeled learning (NLL) requires sufficient labeled data.

Idea

Let g be the learned decision function.

- Try to find an empirical risk $\hat{R}(g)$ that is an *unbiased* estimator of the expected risk under the true distribution $\mathcal{D}$. Then try to minimize it.
- For SSL, we find $\hat{R}_{PU}$ by following du Plessis et al.'s procedures
- For NLL, we use a *large* amount of unlabeled data to help estimate $\sigma_+(x)$ and $\sigma_-(x)$, which are used to find the unbiased empirical risk $\hat{R}_{oIS}(g)$.
- We define the LIoIS risk as a *weighted* combination of these 2 empirical risks. Find $\hat{g}_{LIoIS} = \underset{g}{\mathit{argmin}}\ \hat{R}_{LIoIS}(g)$

Normal: Complete and Accurate Supervision

Let l be a non-negative Lipschitz continuous loss function.

The expected risk is

$$\begin{aligned} R(g) &= E_{(x,y)\sim\mathcal{D}}[l(g(x), y)] \\ &= \theta_P E_P[l(g(x), +1)] + \theta_N E_N[l(g(x), -1)] \end{aligned}$$

where $\theta_P = P(x|y = +1)$ and $\theta_N = P(x|y = -1)$ are class priors.

The empirical risk(Monte Carlo estimate) is

$$\hat{R}(g) = \frac{\theta_P}{n_P} \sum_{i=1}^{n_P} l(g(x_i), +1) + \frac{\theta_N}{n_N} \sum_{j=1}^{n_N} l(g(x_j), +1)$$

Accurate but Incomplete

Assume that l satisfies $l(g(x), +1) + l(g(x), -1) = 1$.
We regard the unlabeled data as negative. Then

$$\begin{aligned} E_U[l(g(x), -1)] &= \theta_P E_P[l(g(x), -1)] + \theta_N E_N[l(g(x), -1)] \\ &= \theta_P(1 - E_P[l(g(x), +1)]) + \theta_N E_N[l(g(x), -1)] \\ &= \theta_P - \theta_P E_P[l(g(x), +1)] + \theta_N E_N[l(g(x), -1)] \end{aligned}$$

$$\Rightarrow \ \theta_N E_N[l(g(x), -1)] = E_U[l(g(x), -1)] - \theta_P + \theta_P E_P[l(g(x), +1)]$$

Plug this into $R(g)$

$$R(g) = 2\theta_P E_P[l(g(x), +1)] + E_U[l(g(x), -1)] - \theta_P$$

With this, we can estimate it by

$$\hat{R}_{PU}(g) = 2\frac{\theta_P}{n_P}\sum_{i=1}^{n_P} l(g(x_i), +1) + \frac{1}{n_U}\sum_{k=1}^{n_U} l(g(x_k), -1)$$

(why do we drop θ_P at the end? Because a constant does not affect min?)

One-side Inaccurate Supervision(oIS)

Let $\tilde{P}$ be the clean positive data and $\tilde{N}$ be the noisy negative data.

- We want to rewrite the expected risk under the **true distribution** $\mathcal{D}$ in terms of expectations under the one-sided inaccurate distribution we have.

$$R_{oIS}(g) = \theta_{\tilde{P}}\ E_{\tilde{P}}[\sigma_+(x)l(g(x),+1)] + \theta_{\tilde{N}}\ E_{\tilde{N}}[\sigma_-(x)l(g(x),-1)]$$

  where $\sigma_+(x) = \frac{1}{P(\hat{y}=+1|x,y=+1)}$, $\sigma_-(x) = P(y=-1|x,\hat{y}=-1)$ are the *important weights*.

- The corresponding empirical risk is

$$\hat{R}_{oIS}(g) = \frac{\theta_{\tilde{P}}}{n_{\tilde{P}}}\sum_{i=1}^{n_{\tilde{P}}}\sigma_+(x_i)l(g(x_i),+1) + \frac{\theta_{\tilde{N}}}{n_{\tilde{N}}}\sum_{j=1}^{\tilde{N}}\sigma_-(x_j)l(g(x_j),-1)$$

- Theoretical Foundation: excess risk $R(\hat{g}_{oIS}) - R(g)$ is bounded

Estimating $\sigma_+(x)$ and $\sigma_-(x)$ by Incomplete Supervision

To evaluate $R_{olS}(g)$, we need to estimate $\sigma_+(x)$ and $\sigma_-(x)$ from the samples.

- Ratio matching method:

$$\sigma_+(x) = \frac{P(x, y = +1)}{P(x, \hat{y} = +1)} = \frac{\theta_P P(x|y = +1)}{\theta_{\tilde{P}} P(x|\hat{y} = +1)} := \frac{\theta_P}{\theta_{\tilde{P}}} \sigma_{+r}(x)$$

- First we estimate θ_P and $\theta_{\tilde{P}}$
  Use $\hat{g}_{PU}$ to generate pseudo labels for unlabeled data. By **law of large numbers**,

$$\hat{\theta}_P = \frac{n_{y_{PU}=+1}}{n_{\tilde{P}} + n_{\tilde{N}}}, \quad \hat{\theta}_{\tilde{P}} = \frac{n_{\hat{y}=+1}}{n_{\tilde{P}} + n_{\tilde{N}}}$$

- Find an approximated $\hat{\sigma}_{+r}$ by minimizing the **Bregman divergence** between the approximated and true remaining density

Combined: Learning from Incomplete and one-sided Inaccurate Supervision (LIoIS)

- Minimize a weighted combination of oIS risk and PU risk.

$$\hat{R}_{LIoIS}(g) = \gamma \hat{R}_{PU} + (1-\gamma)\hat{R}_{oIS}$$

- Find the minimizer $\hat{g}_{LIoIS}$

Table of Contents

1. Top 10 Algorithms in Data Mining
2. Soft Gradient Boosting Machine
3. XGBoost: A Scalable Tree Boosting System
4. Learning from Incomplete and Inaccurate Supervision
5. ML-KNN: A Lazy Learning Approach to Multi-label Learning
6. Towards Convergence Rate Analysis of Random Forests for Classification

Basics

- Task: Given a test example, predict a set of labels whose size is unknown *a priori*

Basics

- Task: Given a test example, predict a set of labels whose size is unknown *a priori*
- $\mathcal{X}$ is the domain and $\mathcal{Y} = \{1, 2, \ldots Q\}$ is the set of all possible classes.
- Learn a function $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ that outputs a *score* of each possible class
  If class y_i is in the ground-truth label set of x_i, then it should have a relatively high score.
- Rank function $rank_f$ is developed based on the score function f.(Used in some evaluation criteria)
  True labels have smaller ranks.
- A possible class y_i is labeled True if $f(y_i) > t$ for some threshold t.

Maximum A Posteriori(MAP)

Given an instance x and its label set $Y \subseteq \mathcal{Y}$, consider its k-nearest neighbors.
Count:

$$\vec{C}_x(l) = \sum_{a \in N(x)} \vec{y}_a(l),\ l \in \mathcal{Y} \tag{6}$$

where $\vec{C}_x(l)$ counts the number of neighbors of x belonging to the l-th class.

MAP:

$$\vec{y}_t(l) = \arg\max_{b \in \{0,1\}} P(H_b^l | E^l_{\vec{C}_t(l)}),\ l \in \mathcal{Y} \tag{7}$$

Using the Bayesian rule, Eq.(7) can be rewritten as:

$$\begin{aligned} \vec{y}_t(l) &= \arg\max_{b \in \{0,1\}} \frac{P(H_b^l) P(E^l_{\vec{C}_t(l)} | H_b^l)}{P(E^l_{\vec{C}_t(l)})} \\ &= \arg\max_{b \in \{0,1\}} P(H_b^l) P(E^l_{\vec{C}_t(l)} | H_b^l) \end{aligned} \tag{8}$$

Algorithm

s is a smoothing parameter

$[\vec{y}_t, \vec{r}_t]$=ML-KNN(T, K, t, s)

%Computing the prior probabilities $P(H_b^l)$

(1) for $l \in \mathcal{Y}$ do

(2) $P(H_1^l) = (s + \sum_{i=1}^{m} \vec{y}_{x_i}(l)) / (s \times 2 + m)$; $P(H_0^l) = 1 - P(H_1^l)$;

average with smoothing

%Computing the posterior probabilities $P(E_j^l | H_b^l)$

(3) Identify $N(x_i)$, $i \in \{1, 2, \ldots, m\}$;

(4) for $l \in \mathcal{Y}$ do

(5) for $j \in \{0, 1, \ldots, \mathrm{K}\}$ do

(6) $c[j] = 0$; $c'[j] = 0$; one for true, the other for false

(7) for $i \in \{1, 2, \ldots, m\}$ do

(8) $\delta = \vec{C}_{x_i}(l) = \sum_{a \in N(x_i)} \vec{y}_a(l)$;

(9) if $(\vec{y}_{x_i}(l) == 1)$ then $c[\delta] = c[\delta] + 1$;

(10) else $c'[\delta] = c'[\delta] + 1$;

(11) for $j \in \{0, 1, \ldots, \mathrm{K}\}$ do

(12) $P(E_j^l | H_1^l) = (s + c[j]) / (s \times (\mathrm{K} + 1) + \sum_{p=0}^{\mathrm{k}} c[p])$;

(13) $P(E_j^l | H_0^l) = (s + c'[j]) / (s \times (\mathrm{K} + 1) + \sum_{p=0}^{\mathrm{k}} c'[p])$;

%Computing $\vec{y}_t$ and $\vec{r}_t$

(14) Identify $N(t)$;

(15) for $l \in \mathcal{Y}$ do

(16) $\vec{C}_t(l) = \sum_{a \in N(t)} \vec{y}_a(l)$;

(17) $\vec{y}_t(l) = \arg\max_{b \in \{0,1\}} P(H_b^l) P(E^l_{\vec{C}_t(l)} | H_b^l)$;

(18) $\vec{r}_t(l) = P(H_1^l | E^l_{\vec{C}_t(l)}) = (P(H_1^l) P(E^l_{\vec{C}_t(l)} | H_1^l)) / P(E^l_{\vec{C}_t(l)})$

$= (P(H_1^l) P(E^l_{\vec{C}_t(l)} | H_1^l)) / (\sum_{b \in \{0,1\}} P(H_b^l) P(E^l_{\vec{C}_t(l)} | H_b^l))$;

Fig. 1. Pseudo code of ML-KNN.

Table of Contents

1. Top 10 Algorithms in Data Mining
2. Soft Gradient Boosting Machine
3. XGBoost: A Scalable Tree Boosting System
4. Learning from Incomplete and Inaccurate Supervision
5. ML-KNN: A Lazy Learning Approach to Multi-label Learning
6. Towards Convergence Rate Analysis of Random Forests for Classification

Work: Establishing Theoretical Foundation

- Prove the consistency of random forest classifiers
- Derive the finite-sample convergence rate of pure random forests

$$\mathcal{O}(n^{\frac{-1}{(8d+2)}})$$

- midpoint split(rather than uniform split) gives us better convergence rate

$$\mathcal{O}(n^{\frac{-1}{3.87d+2}})$$

- Introduce another simplified variant of random forests, with a nearly optimal convergence rate

$$\mathcal{O}(n^{\frac{-1}{d+2}} \ (\ln n)^{\frac{1}{d+2}})$$

References

Xindong Wu et al. "Top 10 algorithms in data mining". In: *Knowledge and information systems* 14.1 (2008), pp. 1–37

Ji Feng et al. "Soft gradient boosting machine". In: *arXiv preprint arXiv:2006.04059* (2020)

Tianqi Chen et al. "Xgboost: extreme gradient boosting". In: *R package version 0.4-2* 1.4 (2015), pp. 1–4

Zhen-Yu Zhang et al. "Learning from incomplete and inaccurate supervision". In: *IEEE Transactions on Knowledge and Data Engineering* (2021)

Min-Ling Zhang and Zhi-Hua Zhou. "ML-KNN: A lazy learning approach to multi-label learning". In: *Pattern recognition* 40.7 (2007), pp. 2038–2048

Wei Gao and Zhi-Hua Zhou. "Towards Convergence Rate Analysis of Random Forests for Classification". In: *Advances in Neural Information Processing Systems* 33 (2020)