Project Proposal

- Deadline: Oct 30 (Sunday), 11:59pm PST.
- Page limit: 2 pages (single column).
- Overview
- Related work (describe what has been done in the literature)
- Main idea and Technical approach
- Experiments (how to conduct experiments, what’s the evaluation plan)
- Expect Result
Outline

- Decision Tree
- Random Forest
- Gradient Boosted Decision Tree
Decision trees are among the most widely used non-linear methods.

Fast prediction: $O(p)$ (the average depth of the tree, usually less than 10)

Small model size $O(2^p)$ (each node only needs two variables).
Splitting the node

- ID3, CART, ...  

  Split the node to maximize the entropy

- Let $S$ be the set of data points in a node and $c = 1, \cdots, C$ are the labels:

  
  \[
  \text{Entropy} : H(S) = - \sum_{c=1}^{C} p(c) \log p(c),
  \]

  where $p(c)$ is the proportion of the data belong to class $c$.

  Entropy=0 if all samples are in the same class

  Entropy is large if $p(1) = \cdots = p(C)$

- The “information gain” of the split $S = S_1 \cup \cdots \cup S_T$:

  \[
  H(S) - \sum_{t} \frac{|S_t|}{|S|} H(S_t)
  \]
Regression Tree

- Commonly used in Gradient Boosted Decision Tree (will see later)

Objective function:

\[
\min_{F} \frac{1}{n} \sum_{i=1}^{n} (y_i - F(x_i))^2 + \text{(Regularization)}
\]

The quality of partition \( S = S_1 \cup S_2 \) can be computed by the objective function:

\[
\sum_{i \in S_1} (y_i - y^{(1)})^2 + \sum_{i \in S_2} (y_i - y^{(2)})^2,
\]

where \( y^{(1)} = \frac{1}{|S_1|} \sum_{i \in S_1} y_i, \quad y^{(2)} = \frac{1}{|S_2|} \sum_{i \in S_2} y_i \)
Splitting the node

- Test all the features \( \{1, \cdots, d\} \) and all the potential cutting values, and find the \((\text{feature}, \text{value})\) pair that maximize information gain.
- Assume the samples are sorted with respect to feature \( i \):
  \[
  (x_{\pi(1)}, d, y_{\pi(1)}), (x_{\pi(2)}, d, y_{\pi(2)}), \cdots, (x_{\pi(n)}, d, y_{\pi(n)})
  \]
- Search through cut values according to the sorted list:
  \[
  \frac{x_{\pi(1)} + x_{\pi(2)}}{2}, \frac{x_{\pi(2)} + x_{\pi(3)}}{2}, \cdots, \frac{x_{\pi(n-1)} + x_{\pi(n)}}{2}
  \]
- Maintain the "count" when scanning from left to right:
  \[
  \text{number of class } i \text{ on left/right for all } i
  \]
  Can be maintained in constant time
- \( O(\tilde{n}d) \) for splitting each node (\( \tilde{n} : \text{number of samples in the current node} \))
Parameters

- Maximum depth: (usually ~ 10)
- Minimum number of nodes in each node: (10, 50, 100)
- Regularization
Parallel Decision Tree

- Naive approach: each thread (machine) computes the split of a node
  Poor performance due to imbalance work load
- Better approach: splitting all the nodes in the same level together
  Feature parallelism
Random Forest
Random Forest

- Random Forest (Bootstrap ensemble for decision trees):
  - Create $T$ trees
  - Learn each tree using a subsampled dataset $S_i$ and subsampled feature set $D_i$
  - Prediction: Average the results from all the $T$ trees

- Benefit:
  - Avoid over-fitting
  - Improve stability and accuracy

- Good software available:
  - R: “randomForest” package
  - Python: Scikit Learn
Random Forest

Embarrassingly parallel
Gradient Boosted Decision Tree
Boosted Decision Tree

- Goal: minimizing a loss function $\ell(y, F(x))$ using boosting method.
- Gradient boosting considers estimating $F$ in an additive form:

$$F^* = \arg\min_F \sum_{i=1}^{n} \ell(y_i, F(x_i)) \quad \text{with} \quad F(x) = \sum_{m=1}^{T} f_m(x)$$

- Direct loss minimization: at each stage $m$, find the best function to minimize objective function:
  - solve $\theta_m = \arg\min_{\theta} \sum_{i=1}^{N} \ell(y_i, F_{m-1}(x_i) + f_m(x_i, \theta))$
  - update $F_m(x) \leftarrow F_{m-1}(x) + f_m(x, \theta_m)$
- $F_m(x) = \sum_{j=1}^{m} f_j(x, \theta_j)$ is the prediction of $x$ after $m$ iterations.
Boosted Decision Tree

- Goal: minimizing a loss function $\ell(y, F(x))$ using boosting method.
- Gradient boosting considers estimating $F$ in an additive form:

$$F^* = \arg\min_F \sum_{i=1}^{n} \ell(y_i, F(x_i)) \quad \text{with} \quad F(x) = \sum_{m=1}^{T} f_m(x)$$

- Direct loss minimization: at each stage $m$, find the best function to minimize objective function:
  - solve $\theta_m = \arg\min_{\theta} \sum_{i=1}^{N} \ell(y_i, F_{m-1}(x_i) + f_m(x_i, \theta))$
  - update $F_m(x) \leftarrow F_{m-1}(x) + f_m(x, \theta_m)$

- $F_m(x) = \sum_{j=1}^{m} f_j(x, \theta_j)$ is the prediction of $x$ after $m$ iterations.
- Two problems:
  - Hard to implement for general loss
  - Tend to overfit training data
Gradient Boosted Decision Tree (GBDT)

- Approximate the current loss function by a quadratic approximation:

\[
\sum_{i=1}^{n} \ell_i(\hat{y}_i + f_m(x_i)) \approx \sum_{i=1}^{n} \left( \ell_i(\hat{y}_i) + g_i f_m(x_i) + \frac{1}{2} h_i f_m(x_i)^2 \right)
\]

\[
= \sum_{i=1}^{n} \frac{h_i}{2} \| f_m(x_i) - g_i / h_i \|^2 + \text{constant}
\]

where \( g_i = \partial_{\hat{y}_i} \ell_i(\hat{y}_i) \) is gradient,

\( h_i = \partial^2_{\hat{y}_i} \ell_i(\hat{y}_i) \) is second order derivative

Gradient Boosted Decision Tree

- Finding $f_m(x, \theta_m)$ by minimizing the loss function:

$$\arg\min_{f_m} \sum_{i=1}^{N} [f_m(x_i, \theta) - g_i/h_i]^2 + R(f_m)$$

- Reduce the training of any loss function to regression tree (just need to compute $g_i$ for different functions)
- $h_i = \alpha$ (fixed step size) for original GBDT.
- XGboost shows computing second order derivative yields better performance
Gradient Boosted Decision Tree

- Finding $f_m(x, \theta_m)$ by minimizing the loss function:

$$\argmin_{f_m} \sum_{i=1}^{N} \left[ f_m(x_i, \theta) - g_i / h_i \right]^2 + R(f_m)$$

- Reduce the training of any loss function to regression tree (just need to compute $g_i$ for different functions)
- $h_i = \alpha$ (fixed step size) for original GBDT.
- XGboost shows computing second order derivative yields better performance

**Algorithm:**

- Computing the current gradient for each $\hat{y}_i$.
- Building a base learner (decision tree) to fit the gradient.
- Updating current prediction $\hat{y}_i = F_m(x_i)$ for all $i$. 
Gradient Boosted Decision Trees (GBDT)

Key idea:
- Each base learner is a decision tree
- Each regression tree approximates the functional gradient \( \frac{\partial \ell}{\partial F} \)

\[
(x, g_1) \\
\downarrow \\
f_1(x)
\]
Gradient Boosted Decision Trees (GBDT)

**Key idea:**
- Each base learner is a decision tree
- Each regression tree approximates the functional gradient \( \frac{\partial \ell}{\partial F} \)

\[
F_{m-1}(x_i) = \sum_{j=1}^{m-1} f_j(x_i) \quad g_m(x_i) = \left. \frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)} \right|_{F(x_i)=F_{m-1}(x_i)}
\]
Gradient Boosted Decision Trees (GBDT)

- **Key idea:**
  - Each base learner is a decision tree
  - Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial F}$

$$F_{m-1}(x_i) = \sum_{j=1}^{m-1} f_j(x_i) \quad \text{and} \quad g_m(x_i) = \left. \frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)} \right|_{F(x_i) = F_{m-1}(x_i)}$$
Gradient Boosted Decision Trees (GBDT)

- **Key idea:**
  - Each base learner is a decision tree
  - Each regression tree approximates the functional gradient \( \frac{\partial \ell}{\partial F} \)

\[
F_{m-1}(x_i) = \sum_{j=1}^{m-1} f_j(x_i) \\
g_m(x_i) = \frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)} \bigg|_{F(x_i) = F_{m-1}(x_i)}
\]
**Gradient Boosted Decision Trees (GBDT)**

**Key idea:**
- Each base learner is a decision tree
- Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial f}$

**Final prediction**
$$F(x_i) = \sum_{j=1}^{T} f_j(x_i)$$
Learning to rank: given samples $x_1, \ldots, x_n$ and a set of pairwise comparisons $\Omega = (i, j, y_{ij})$, minimize the ranking loss

$$\min_f \sum_{(i,j) \in \Omega} \max(1 - y_{ij}(f(x_i) - f(x_j)), 0).$$

$y_{ij} = \{+1, -1\}$ is the pairwise comparison results.

Hard to directly construct a tree to minimize ranking loss

Easy to compute the gradient of the objective function given the current $f$

$\Rightarrow$ GBDT becomes the best algorithm for learning to rank
Parallelism

- Parallelize the construction of decision tree.
- Feature-parallelism: each machine/core computes the best splits for a subset of features.
Questions?