C++ DP-GBDT Side-channel Analysis

Contents

1 Severity List

General

entity	secrecy
X	✓
X_cols_size	×
X_rows_size	×
y	✓
y_rows_size	×

parameter	secret
nb_trees	×
learning_rate	×
privacy_budget	×
task	×
max_depth	×
min_samples_split	×
balance_partition	×
gradient_filtering	×
leaf_clipping	×
scale_y	×
use_decay	×
l2_threshold	×
l2_lambda	×
cat_idx	×
num_idx	×

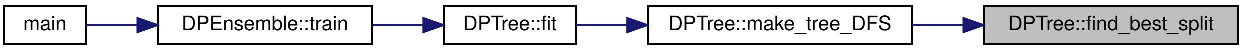
While building a single tree

entity	secrecy	
X_subset	✓	
X_subset_cols_size	×	
X_subset_rows_size	✓	
y_subset	✓	
y_subset_rows_size	✓	
gradients	✓	
gradients_size	✓	

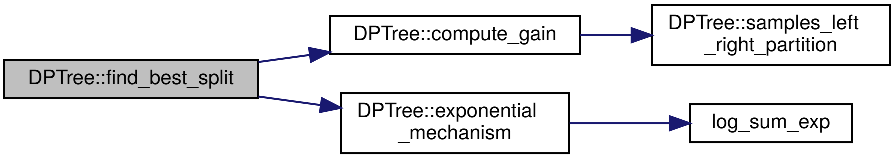
2 Building a single tree

2.1 `find_best_split`

Caller graph



Call graph



Arguments / used variables

variable	secret
X	✓
gradients	✓
curr_depth	?
tree_budget	×
params.use_decay	×
params.Δg	×
params.max_depth	×

Algorithm 1: find_best_split

Function `find_best_split(`X`, gradients, curr_depth)`
  `// determine node privacy budget`
  if `params.use_decay` **then**
    if `curr_depth == 0` **then**
      `node_budget` $= \frac{\texttt{tree\_budget}}{2*(2^{\texttt{max\_depth}+1}+2^{\texttt{curr\_depth}+1})}$
    else
      `node_budget` $= \frac{\texttt{tree\_budget}}{2*2^{\texttt{curr\_depth}+1}}$
  else
    `node_budget` $= \frac{\texttt{tree\_budget}}{2*\texttt{max\_depth}}$
  `// iterate over all possible splits`
  for `feature_index : features` **do**
    for `feature_value : X[feature_index]` **do**
      if `"already encountered feature_value"` **then**
        continue
      `gain` = *compute_gain(X, gradients, feature_index, feature_value)*
      if `gain < 0` **then**
        continue
      `gain` $= \frac{\texttt{node\_budget}*\texttt{gain}}{2*\Delta g}$
      `candidates.insert(Candidate(feature_index, feature_value, gain))`
  `// choose a split using the exponential mechanism`
  `index` = *exponential_mechanism(candidates)*
  `// construct the node`
  `TreeNode *node = new TreeNode(candidates[index])`
  return `node`

2.1.1 Side channel leakage

leakage in `compute_gain` and `exponential_mechanism`

From branches/loops

- params.use_decay
- curr_depth == 0
- number of features (columns of X)
- number of rows in X resp. length of gradients
- number of unique feature values of a feature
- number of splits that don't give any gain
- number of split candidates

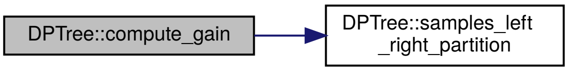
Potential arithmetic leakage ?

- Not sure about this in SGX though
- edge cases of variables appearing in formulas → `tree_budget` and `curr_depth` and Δg and `gain`

2.2 `compute_gain`

Caller graph

Call graph



Arguments / used variables

variable	secret	
X	✓	
gradients	✓	
feature_index	×	
feature_value	×	
params.l2_lambda	×	

Algorithm 2: compute_gain

```
Function compute_gain(X, gradients, feature_index, feature_value)
    // // partition into lhs/rhs
    lhs, rhs = samples_left_right_partition(X, feature_index, feature_value)
    lhs_size = lhs.size()
    rhs_size = rhs.size()
    // return on useless split
    if lhs_size == 0 || rhs_size == 0 then
        return -1
    // sums of lhs/rhs gains
    lhs_gain = sum(gradients[lhs])
    rhs_gain = sum(gradients[rhs])
    lhs_gain = lhs_gain^2 / (lhs_size + params.l2_lambda)
    rhs_gain = rhs_gain^2 / (rhs_size + params.l2_lambda)
    total_gain = lhs_gain + rhs_gain
    total_gain = max(total_gain, 0)
    return total_gain
```

2.2.1 Side channel leakage

leakage in `samples_left_right_partition`

From branches/loops/function calls

- size (#rows) of X/gradients
- lhs/rhs size
- whether it's a useless split
- memory access pattern of left/right gradients
- max function might leak whether `total_gain` < 0

Potential arithmetic leakage ?

- edge cases of variables appearing in formulas $\rightarrow$ `lhs_gain` and `lhs_size`, rhs respectively.