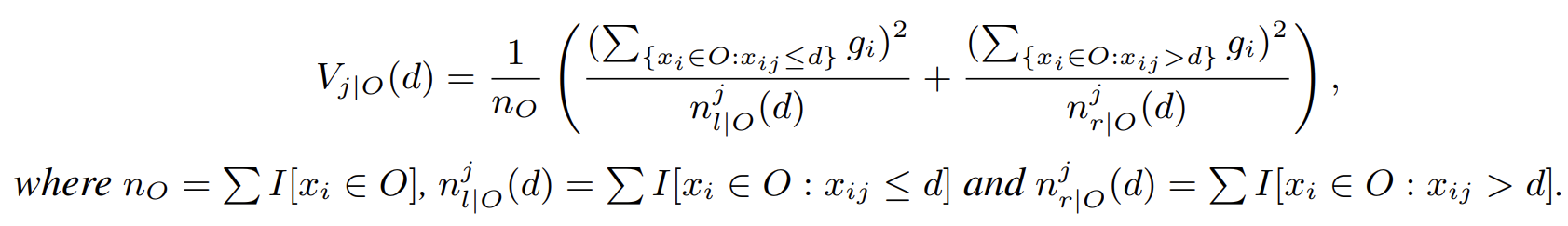
**BY PAPER:**

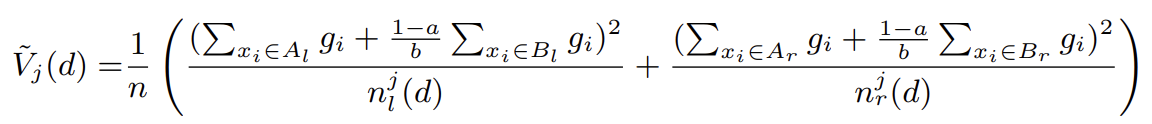
**1. Theoretical Analysis**

GBDT uses decision trees to learn a function from the input space X s to the gradient space G [1]. Suppose that we have a training set with n i.i.d. instances {x1, · · · , xn}, where each xi is a vector with dimension s in space X s . In each iteration of gradient boosting, the negative gradients of the loss function with respect to the output of the model are denoted as {g1, · · · , gn}. The decision tree model splits each node at the most informative feature (with the largest information gain). For GBDT, the information gain is usually measured by the variance after splitting,

**Definition 1** Let O be the training dataset on a fixed node of the decision tree. The variance gain of splitting feature j at point d for this node is defined as



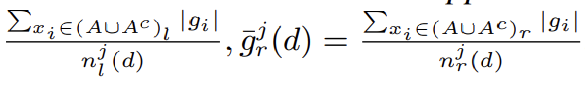
For feature j, the decision tree algorithm selects d∗j = argmaxdVj (d) and calculates the largest gain Vj (d∗j ). Then, the data are split according feature j∗ at point dj ∗ into the left and right child nodes. In our proposed GOSS method, first, we rank the training instances according to their absolute values of their gradients in the descending order; second, we keep the top-a × 100% instances with the larger gradients and get an instance subset A; then, for the remaining set Ac consisting (1 − a) × 100% instances with smaller gradients, we further randomly sample a subset B with size b × |Ac |; finally, we split the instances according to the estimated variance gain V˜j (d) over the subset A ∪ B, i.e.



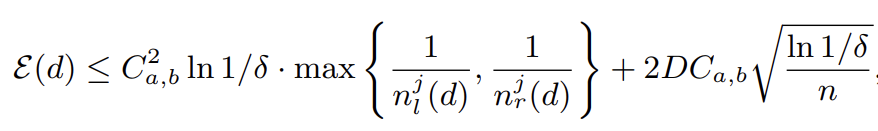
where Al = {xi ∈ A : xij ≤ d},Ar = {xi ∈ A : xij > d},Bl = {xi ∈ B : xij ≤ d},Br = {xi ∈ B : xij > d}, and the coefficient 1−a b is used to normalize the sum of the gradients over B back to the size of Ac

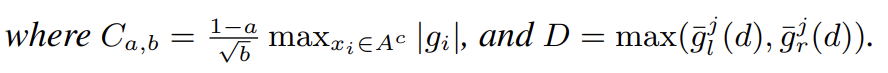
Thus, in GOSS, we use the estimated V˜ j (d) over a smaller instance subset, instead of the accurate Vj (d) over all the instances to determine the split point, and the computation cost can be largely reduced. More importantly, the following theorem indicates that GOSS will not lose much training accuracy and will outperform random sampling. Due to space restrictions, we leave the proof of the theorem to the supplementary materials.

**Theorem 2** We denote the approximation error in GOSS as E(d) = |V˜j (d) − Vj (d)| and g¯j l P (d) =

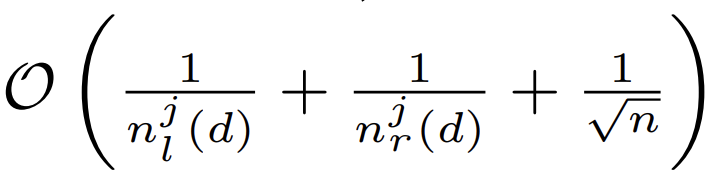


With probability at least 1 − δ, we have





According to the theorem, we have the following discussions: (1) The asymptotic approximation ratio of GOSS is



If the split is not too unbalanced (i.e., njl (d) ≥ O( √ n) and njr(d) ≥ O( √ n)), the approximation error will be dominated by the second term of Ineq. Which decreases to 0 in O( √ n) with n → ∞. That means when number of data is large, the approximation is quite accurate. Random sampling is a special case of GOSS with a = 0. In many cases, GOSS could outperform random sampling, under the condition C0,β > Ca,β−a, which is equivalent to αa/√β > √1−a β−a with αa = maxxi∈A∪Ac |gi|/ maxxi∈Ac |gi|.

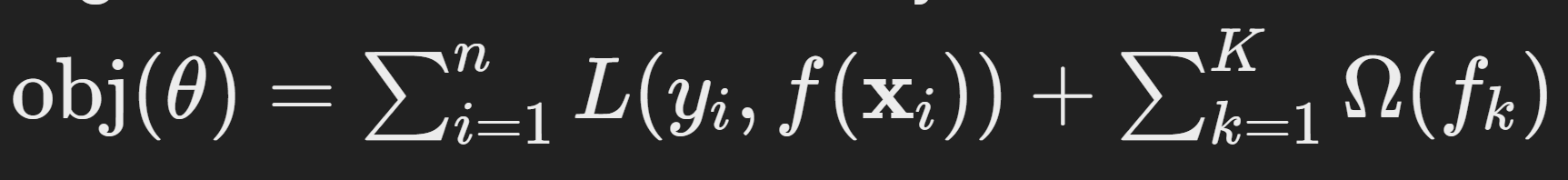
Next, we analyze the generalization performance in GOSS. We consider the generalization error in GOSS EGOSSgen (d) = |V˜j (d) − V∗(d)|, which is the gap between the variance gain calculated by the sampled training instances in GOSS and the true variance gain for the underlying distribution. We have E GOSS gen (d) ≤ |V˜j (d) − Vj (d)| + |Vj (d) − V∗(d)| ∆= EGOSS(d) + Egen(d). Thus, the generalization error with GOSS will be close to that calculated by using the full data instances if the GOSS approximation is accurate. On the other hand, sampling will increase the diversity of the base learners, which potentially help to improve the generalization performance

LightGBM (Light Gradient Boosting Machine) is a popular gradient boosting framework that uses tree-based learning algorithms. The core idea behind gradient boosting is to sequentially add weak learners (usually decision trees) to the ensemble, with each subsequent learner focusing on the mistakes made by the previous ones.

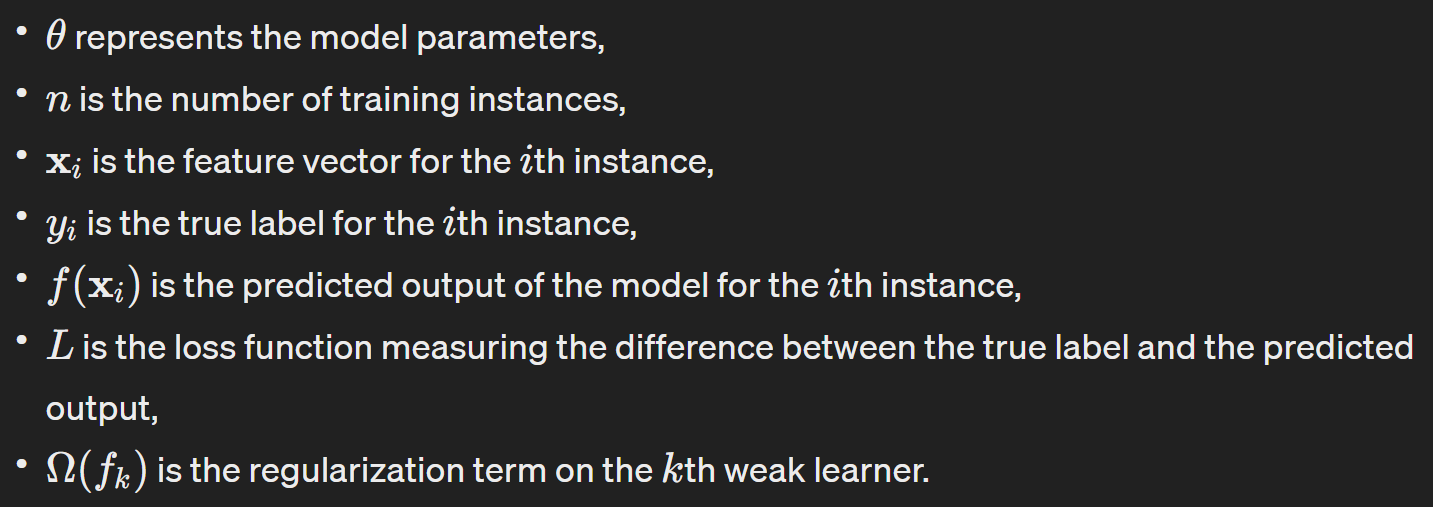
BY CHAT GPT:

**1. Objective Function:**

LightGBM aims to minimize a differentiable objective function, typically a loss function with regularization terms. The objective function to be minimized can be represented as:

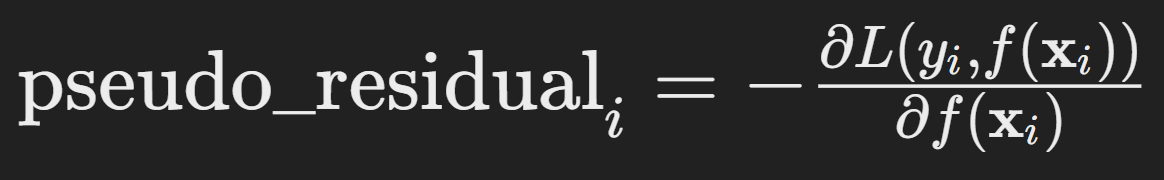


where:



**2. Gradient Boosting Iterations:**

At each iteration, LightGBM fits a new weak learner to the negative gradient of the loss function with respect to the ensemble's current prediction. For a regression problem with the mean squared error (MSE) loss, this can be expressed as:



where pseudo\_residual\_i is the negative gradient (pseudo-residual) for the ith instance.

**3. Constructing Trees:**

LightGBM constructs decision trees using a leaf-wise approach. In this approach, the algorithm grows the tree by splitting the leaf that leads to the maximum reduction in the loss function. The process involves selecting the leaf with the maximum delta loss and expanding the tree in that direction. This is in contrast to traditional depth-wise approaches where the tree grows level by level.a

**4. Regularization:**

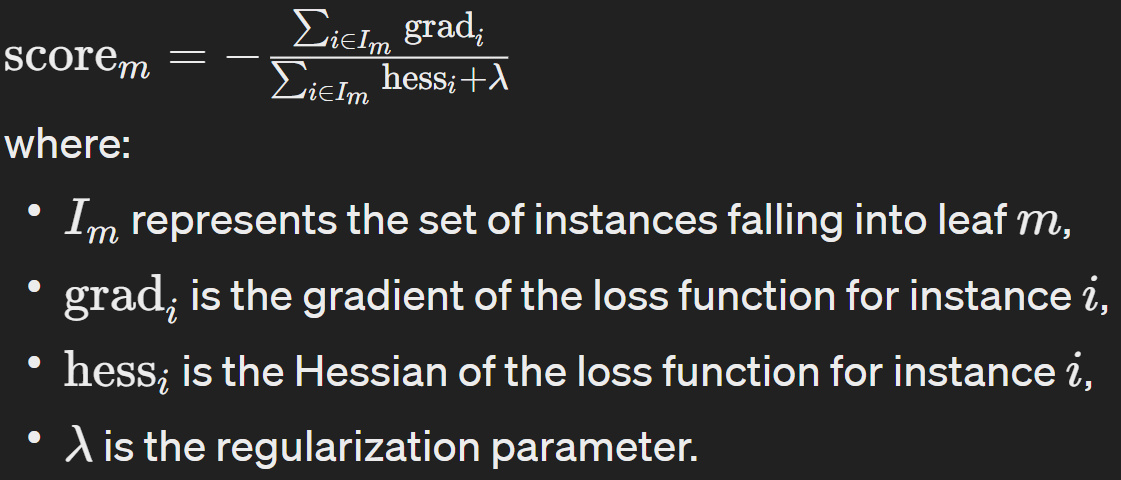
LightGBM employs regularization techniques to prevent overfitting and improve generalization. Two common types of regularization used are:

* **L1 Regularization (Lasso):** Adds an L1 penalty term to the loss function, encouraging sparsity in feature selection.
* **L2 Regularization (Ridge):** Adds an L2 penalty term to the loss function, penalizing large coefficients.

The regularization terms are typically added to the objective function and controlled by hyperparameters.

**5. Leaf-wise Splitting:**

After the tree structure is determined, LightGBM updates the leaf values by computing the optimal scores for each leaf. This process involves finding the optimal score for each leaf node by considering the sum of gradients and Hessians of all instances falling into that leaf. Mathematically, the optimal score for a leaf node *m* is given by:



**6. Learning Rates and Early Stopping:**

LightGBM incorporates learning rate decay and early stopping techniques to optimize the training process.

* **Learning Rate Decay:** LightGBM reduces the learning rate over time to fine-tune the model parameters gradually. This prevents overshooting and helps the algorithm converge to a stable solution.
* **Early Stopping:** LightGBM monitors the performance on a validation set during training. If the performance does not improve for a certain number of iterations, training is stopped early to prevent overfitting and save computational resources.