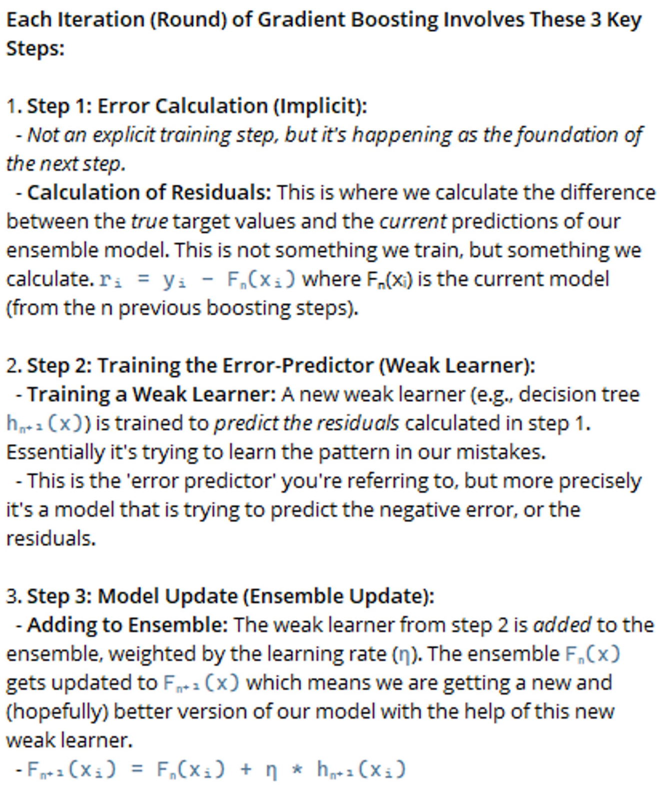
Ensemble models

23 декабря 2024 г. 9:19

1. The core idea is to combine outputs from weak, simple models

2. **Stacking technic:**
3. Simple voting. The major prediction wins. 2 level system: base-models and meta-model. Meta-model learns how to better combine outputs from weak learners.
4. But we use same training set here with different algos.
5. **Bagging - Bootstrap Aggregating:**
6. Taking different training sets, learning same model, avraging prediction.
7. Big drawback for trees. We could have had another variable as most informative on whole dataset and more or less the same tree for subsets. Hence we will be falling to local optima.
8. **Random Subspace Method:**
9. Separating dataset not horizontally, but vertically. So we take random set of features on all dataset and try to implement tree on them.
10. **Bagging+RSM=Random Forest**
11. Both horizontal and vertical subsetting. So we take subsets and take random set of features on each subset.
12. Then averaging predictions
13. **Boosting**
14. On each step we are training model and calculating error and accounting it in next step. So next model will predict error of previous model. This error usually multiplied by some hyper param lambda.
15. $$F(x) = f_1(x) + \gamma f_2(x) + \gamma f_3(x) \dots$$
    $$\underbrace{R_1(X)}_{\uparrow R_2} = F(x) - f_1(x)$$
    142
16. **Gradient boosting**

17. 

Each Iteration (Round) of Gradient Boosting Involves These 3 Key Steps:

1. **Step 1: Error Calculation (Implicit):**
 - *Not an explicit training step, but it's happening as the foundation of the next step.*
 - **Calculation of Residuals:** This is where we calculate the difference between the *true* target values and the *current* predictions of our ensemble model. This is not something we train, but something we calculate. $r_i = y_i - F_n(x_i)$ where $F_n(x_i)$ is the current model (from the n previous boosting steps).

2. **Step 2: Training the Error-Predictor (Weak Learner):**
 - **Training a Weak Learner:** A new weak learner (e.g., decision tree $h_{n+1}(x)$) is trained to *predict the residuals* calculated in step 1. Essentially it's trying to learn the pattern in our mistakes.
 - This is the 'error predictor' you're referring to, but more precisely it's a model that is trying to predict the negative error, or the residuals.

3. **Step 3: Model Update (Ensemble Update):**
 - **Adding to Ensemble:** The weak learner from step 2 is *added* to the ensemble, weighted by the learning rate (η). The ensemble $F_n(x)$ gets updated to $F_{n+1}(x)$ which means we are getting a new and (hopefully) better version of our model with the help of this new weak learner.
 - $F_{n+1}(x_i) = F_n(x_i) + \eta * h_{n+1}(x_i)$

ChatGPT4 | Midjourney, [23.12.2024 11:46]
Okay, let's break down each of those XGBoost parameters and explain what they control:

ChatGPT4 | Midjourney, [23.12.2024 11:46]
```
params = {
   'max_depth':    8,
   'max_leaves':   2**8, #which is 256
   'device': 'cuda',
   'tree_method':  'hist',
   'objective':    'binary:logistic',
   'grow_policy':  'lossguide',
   'eval_metric':  'logloss',
   'subsample':    0.8
}
```

ChatGPT4 | Midjourney, [23.12.2024 11:46]
1. max_depth: 8

• Meaning: This parameter sets the maximum depth of each decision tree within the boosting ensemble. A decision tree's depth is the longest path from its root node to a leaf node.
• Impact:
 • Lower Values (e.g., 3-5): Lead to simpler trees that are less prone to overfitting, but may not capture complex relationships.
 • Higher Values (e.g., 8 or more): Allow trees to grow deeper, capturing more complex patterns, but at the risk of overfitting. Deeper trees may also take longer to train.
 • 8: In your case, you are allowing decision trees up to 8 levels.
• Typical Range: 3-10 (often depending on the dataset complexity)

2. max_leaves: 256

• Meaning: This parameter sets the maximum number of leaf nodes allowed in each tree. A leaf node is the final node in a tree where the prediction occurs.
• Impact:
 • Lower Values: Limit the complexity of individual trees.
 • Higher Values: Allow more complex trees.
 • 256: In your case, you are allowing trees to have up to 256 terminal nodes.
• Relation to max_depth: max_leaves can control complexity more directly than max_depth. It's common to tune one or the other but not both. If grow_policy is set to lossguide, then max_leaves is used, and max_depth is ignored.
• Note: In XGBoost with tree_method='hist', there is an interaction between max_leaves and max_bin

3. device: 'cuda'

• Meaning: Specifies the device used for training the model (i.e., whether training is done on the CPU or the GPU).
• Impact:
 • 'cuda': Leverages the GPU for faster training (if a CUDA-enabled NVIDIA GPU is available). It is recommended for faster training if the GPU is available.
 • 'cpu': Uses the CPU for training.
• Note: 'cuda' is a very common setting for XGBoost on modern machines, and will use the GPU, provided the necessary

dependencies are installed and the device is compatible with CUDA.

4. tree_method: 'hist'

• Meaning: Defines the algorithm used to construct the trees. 'hist' stands for histogram based tree building.
• Impact:
  • 'hist': A very efficient method to build trees, especially for large datasets. It is faster and uses less memory.
    * It approximates the best split by creating histograms, which speeds up the split finding process.
  • Other options include: 'exact' (slower but precise), 'approx' (faster but can lead to approximations).
• Note: 'hist' is the recommended tree method for most cases, especially when you have larger dataset and you want a good balance between speed and accuracy.

5. objective: 'binary:logistic'

• Meaning: Specifies the loss function used for training the model. It determines what the model tries to optimize during training.
• Impact:
  • 'binary:logistic': Indicates that the task is a binary classification problem (where the target variable has only two classes, e.g., 0 or 1), and the loss function being optimized is the logistic loss (log loss). This function outputs the probability of an instance belonging to the positive class (class 1).
• Other Options: XGBoost supports many other objectives, such as 'multi:softmax' (for multi-class classification), 'reg:squarederror' (for regression), etc.

6. grow_policy: 'lossguide'

• Meaning: Specifies how trees should be grown in the boosting process.
• Impact:
  • 'lossguide': It's a strategy that guides the tree growth based on the expected reduction in loss. It prioritizes expanding nodes that lead to the highest reduction in the training loss. This typically leads to better model performance than the depthwise growth policy, which expands nodes based

ChatGPT4 | Midjourney, [23.12.2024 11:46]
• Meaning: Defines the metric used to evaluate the performance of the model during training. This metric is used to monitor progress, not to optimize the model.
• Impact:
    * 'logloss' (Logarithmic Loss): A common metric for binary classification that penalizes probabilities that are far away from the actual labels. It's closely related to the objective function ('binary:logistic').
• Other Options: You can use other metrics like 'auc', 'error', 'rmse' (depending on the problem).
• Note: The metric you choose to evaluate doesn't have to be the same as the objective. eval_metric is used to monitor training performance and is used by early stopping.

8. subsample: 0.8

• Meaning: Specifies the fraction of training data to randomly sample before growing each tree.
• Impact:
    * 0.8: Means that 80% of the training data is randomly selected to grow each new tree in the ensemble.
    * Values < 1: Helps to prevent overfitting (by making each tree less reliant on any specific subset of the training data) and can also improve training speed.
• Typical Range: 0.5-1

In summary:

These parameters represent a configuration for an XGBoost model aimed at binary classification, using a GPU for training, and employing a histogram based tree method. They carefully control tree growth to balance complexity and performance using several regularization methods (subsampling), and monitor the training process using log loss.

Important Note:

• Hyperparameter Tuning: These parameters should typically be tuned (e.g. using cross-validation) to find the optimal combination for a specific dataset. The provided values might perform well as a starting point, but are unlikely to be optimal without experimentation.
• XGBoost Documentation: Refer to the official XGBoost documentation for a complete understanding of all parameters and their interactions.

18. d