Tree Models

- Decision Tree
  - Terminology
    - root node, decision node, leaf node, sub-tree, pruning: avoid overfitting by cutting down some nodes which are not significant
      - Pruning
        - Pre-pruning: stop growing tree earlier
        - Post-pruning: once tree is built to its depth, we start pruning based on importance.
    - Entropy
      - Definition: the uncertainty in dataset or measure of disorder. In decision tree, it measures the impurity of a node.
        - Impurity: degree of randomness
        - *lower entropy, higher purity, lower impurity* is the node, which making decision easier to finish.
      - Formula: $E(S) = -p_{(+)}logp_{(+)} - p_{(-)}logp_{(-)} = \sum_i -p_i logp_i$
        - $p_{(+)}$: propability of positive class
        - S: subset of training set
    - Information Gain
      - Definition: it measures how much the parent entropy has decreased after splitting some features in decision tree.
        - Generally, it tells the reduction of uncertainty given some feature and how important a factor is.-->use to discriminate between classes
        - select feature which has highest IG
      - Formula: $IG = E(parent) - \bar{E}(children) = E(X) - E(Y|X)$
    - Gini Index
      - Formula: $Gini = \sum_{i \neq j} p(i)p(j) = 1 - \sum_{i=0}^{i=k} P_i^2$
      - Definition: the probability of a random choosen sample in a node would be incorrectly labeled based on the samples of the node.
      - Intuition: the smaller Gini coefficient is, the more purity is the node.
    - Splitting point
      - Issue: too many nodes and branches will lead to overfitting problems.
      - Solution
        - Hyperparameter tuning
          - max_depth

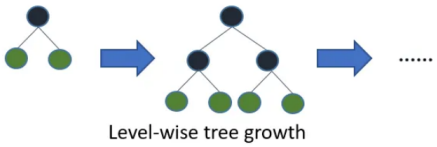
                - the bigger parmeter is, the more complex model is, the less training error. But we cannot set it too small as well, which might lead to underfitting problem-->use **GridSearchCV** method
            - min_samples_split
                - if a node has <N samples using this parameter, then we should stop splitting this node. The default value=2
            - min_samples_leaf
                - the bigger this parmeter we set, the more likely to be overfitting.
                - minimum number of samples required to be at a leaf node, default value=1
            - max_features
    - Application: can be used as regression and classification tasks.
    - Coding
        - Visualization
            - `from sklearn.tree import export_graphviz`
        - Decision Tree Creation
            - `from sklearn.tree import DecisionTreeClassifier`
- Random Forest
    - Definition: ensamble model with many decision trees using bootstramping
        - Ensamble methods
            - **1. Bagging**
                - Bootstramp: create a different training subset randomly from sample training data(row sampling) *with replacement*
                - Aggregation: output is based on *majority voting* after combining the results of all models.
                - eg: random forest
            - **2. Boosting**
                - combines weak learners into stronger learners by creating sequential models
                - eg: Adaboost, Xgboost
    - Feature Importance: describe how important features are for the model and dataset
        - Coding
            - Method1: permutation importance
                - `from sklearn.inspection import permutation_importance`
                - `perm_importance = permutation_importance(rf, X_test, y_test)`
            - Method2: built-in feature importance

        - After building the random forest, `rf.feature_importances_`
        - `plt.barh(boston.feature_names, rf.feature_importances_)`
    - Method3: SHAP Values: estimate how does each feature contributes to the prediction.
        - `pip install shap`
        - `explainer = shap.TreeExplainer(rf)`
        - `shap_values = explainer.shap_values(X_test)`
        - `shap.summary_plot(shap_values, X_test, plot_type="bar")`
- Hyparameter Tuning
    - n_estimators: numbers of trees of the algorithm before averaging the results
    - max_features: maximum number of features RF considers when splitting the nodes
    - mini_sample_leaf
    - criterion: how to split the node in each tree(entropy/gini impurity/ log loss)
    - max_leaf_nodes
    - n_jobs: how many processors is allowed to used(=-1, no limit)
    - random_state: control randomness of the sample
    - oob_score: out of bag,
        - a kind of cross-validation method. It declares how many proportion of training data *not used to training*, instead used to evaluate the performance.
        - computed as the number of correctly predicted rows from out of bag sample
- Coding:
    - `from sklearn.ensemble import RandomForestRegressor/Classifier`
- Gradient Boosting Decision Tree(GBDT)
    - Procedures
        - 1. build base model for prediction, get a γ for minimizing loss function
        - 2. calculate pseudo residuals ($y_i - \gamma$)
            - Notation:
                - y_i: observed values
                - γ: predicted values
        - 3. Build a model on pseudo residuals and make predictions, for minimizing residuals to improve model accuracy
            - $h_m(x)$: residuals, the loss
                - m: the numbers of decision tree
        - 4. find out the output values of each leaf of decision tree
            - simply take average of all the numbers of a leaf, cuz one leaf sometimes has >1 residuals
                - Formula: $\gamma_m = arg\ min_\gamma \sum_{i=1}^{n}(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$

        - 5. Update the predictions of previous model
            - $F_m(x) = F_{m-1}(x) + \nu_m h_m(x)$
                - ν: learning rate$\in [0, 1]$, how fast the model learns
                    - lower ν, more robust the model is, the more trees we need to train the model(relate paramter: *n_estimator*)
    - Coding
        - `from sklearn.ensemble import GradientBoostingClassifier/Regressor` (the difference is loss function)
            - Objective: minimize loss function by adding weak learners using gredient descent
            - eg: for regression: MSE, for classificiation tasks: log-likelihood
    - Other Notes
        - Pruning: it stop splitting node until the loss function is negative, using greedy algorithm.
- XGBoost(Extreme Gradient Boosting)
    - A implementation of GBDT
    - Advantages
        - Regularization: add L1 & L2 regularization boosting technique
        - Parallel Processing: support implementation on Hadoop, blazing faster, data is divided into several blocks for using all cores of CPU
        - High flexibility: use custom optimization objective function and evaluation criteria
            - objective function:$L(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$(loss function+regularization)
                - the selective objective function:
                    - binary: logistic
                    - multi: softmax or softprob
                    - regression: linear
            - Evaluation matrix
                - rmse, mae, error, merror, mlogloss, auc
        - Handling missing data automatically, by assigning them to default direction then finding the best imputation value to minimize training losses.
        - Tree Pruning: XGBoost splits up to *max_depth* specified and starts pruning from backward until the loss < threshold.
        - Build-in CV
        - Sparsity Aware: DMatrix, an internal data structure of XGBoost, optimizes memory efficiency and training speed
    - Important Parameters

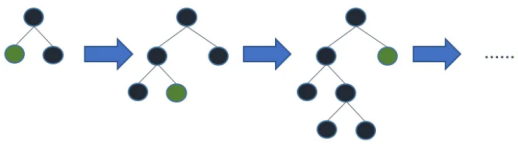
        - booster: select type of model to run at each iteration
            - gbtree
            - gblinear
        - silent: 0 or 1, the running message can be printed(1) or not(0)
        - nthread: defined for parallel programming, shows # of cores the system enter
        - eta: weight shrinkage for achieving robust, usually $(0.01, 0.2)$, can be treated as learning rate
        - gamma: minimum loss reduction required to make split, usually set $(0, 0.5)$
        - max_delta_step: tree's weight estimatation constrains
        - subsample: fraction of observations to be random samples for each tree.
            - fraction of observations to be random samples for each tree.
            - lower subsample, less likely to be overfitting
            - suggested value range $(0.5, 0.8)$
        - colsample_bytree
            - fraction of columns to be selected as random samples for each tree.
            - suggested value range $(0.5, 0.8)$
        - lambda: L2 regularition term on weight, suggested value range $[1, 10]$
        - alpha: L1 regularithion term on weight, suggested value range $[1, 10]$
    - Coding:
        - `pip install xgboost`
        - `import xgboost as xgb`
        - `from xgboost import XGBClassifier/XGBRegressor`
        - cross-validation: `xgb.cv()`
        - feature importance: `xgb.plot_importance(model)`
- LightGBM
    - Gradient One-Side Sampling(GOSS)单边梯度采样
        - reduce the samples needed and keep accuracy high, putting more focus on under-trained instances
        - instances with small graidents are well trained, large gradients under trained.
            - Hence, we only select samples with a% largest gradients(normally **top 20%**)
            - we select b% samples of remaining part randomly. (normally we choose **random 10%**)
    - Exclusive Feature Bundling(EFB)互斥特征捆绑
        - features with high dimensions are parse sometimes
        - the two features must mutually exclusive, never take 0 value at the same time, can be bundled into one single feature

        - if conflicts(fraction of exclusive features have overlapping non-zero values)> threshold, create new bundle
        - merge features: we can simply recognize and extract original features from merged features, by mathmatical operations of bins range.
    - Time complexity decreases, cuz # of bundles<< # of features
- Histogram Decision Tree Algorithm
    - continuous feature values are divided into series of bins with a discrete intervals, for *selecting the best split*.
    - Each feature gains a histogram graph
    - reduce the losses, memory needs, and computation time complexity
- Support categorical feature and parallel processing
- Leaf-wise algorithm



Level-wise tree growth

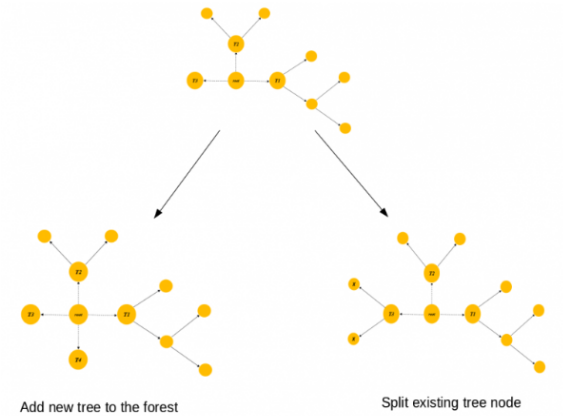
source



Leaf-wise tree growth

    - grow the tree from bottom to top
    - automatically choose the best distribution based on loss reduction
    - drawback: overfitting, easy to create a tree with large depth-->solution: add a constain on max_depth
- Hangling missing values: set `use_missing=false` , deal missing values with NA. set `zero_as_missing=true` , use zero
- Parameter Tuning:
    - feature_fraction: setting xx% of features at each tree node, dealing with overfitting
    - bagging_fraction: the fraction of data to be used for each iteration, for speeding up and avoiding overfitting. suggested range $[0.5, 0.8]$
    - bagging_freq: frequency of bagging. 0: bagging is disabled.
    - early_stopping_round: training stops if a certain parameter fails to improve.
    - lambda: specify regularization. suggested range $[0, 1]$
    - min_gain_to_split: the minimum gain to make a split.
    - max_cat_group: set the maximum numbers of category group. if the category groups amout is so large, finding the split point will be overfitting. By default, set=64.

        - task: train or predict
        - objective: logloss, l2, lambdarank, cross_entropy, rank_xendcg, mape, gamma, binary, multiclass, regression_l1
        - boosting: types of algorithm we want to apply. gbdt, rf, dart, goss
        - num_boost_round: # of boosting iterations, >100 usually
        - learning_rate
        - num_leaves: # of leaves in the whole tree. suggested range $[31, 63]$
        - max_bin: maximum numbers of bin for feature values, for feaster the speed
        - save_binary: speed up data loading for future learning, save dataset to a binary file
        - categorical_feature: the index of categorical features. ie: 0,1,2,3...
        - ignore_column: ingnore the specific columns
    - Coding
        - `pip install lightgbm`
        - `from lightgbm import LGBMClassifier`
- Regularized Greedy Forest(RGF)
    - Weight Optimization in each nodes
        - the loss function and interval of weight optimization can be specified by parameters
        - each time we optimize weights, k leaf nodes are added working well.
            - larger k, faster training speed.
    - Search for Optimal Structure Change to obtain new forest minimizing loss function



        - Split an existing leaf node and start a new tree, only these 2 kinds of operations are improved in search process, for improving computational efficiency
        - search entire tree is too expensive, hence, the search is limited to the most created trees, defaultly t=1.
        - when all weights of leaf nodes are **fixed**, the search is finished, by repeatedly evaluating maximum loss reduction of all possible sructure changes.
    - Regularization
        - Several variants of L2 regularization for growing forest and weight correction

        - 1. only implied in leaf nodes
            - penalty term: $G(F) = \lambda \sum_v \alpha_v^2/2$
        - 2. min-penalty regularizers, used in each tree

$$\lambda \cdot \min_{\{\beta_v\}} \left\{ \sum_v \gamma^{d_v} \beta_v^2/2 \ : \ \text{some conditions on } \{\beta_v\} \right\}$$

        - 3. min-penalty regularizers with sum-to-zero slibing constraints
    - Parameters Tuning
        - max_leaf： up to this specific value, the training process stop
        - loss_function: we can choose from `LS/Expo/Log` , stands for square loss/exponential loss/logistic loss respectively.
        - algorithm: we can choose from `RGF/RGF Opt/RGF Sib`
        - reg_depth: range: <1, a larger value penalizes deeper nodes more severely
        - l2: control the degree of L2 regularization, selected value can be 1, 0.1, 0.01, 1e-10...
        - sl2: λ override parameter of regularization in growing forest
            - if specify, for the weight correlation process we use λ and forest growing process we use λ_g
            - if omitted, no override, only λ used in training.
        - normalize: if set to be true to normalize training targets, training target average=0
        - learning_rate: the smaller the better
        - test_interval: wholly update weights for all leafs on all trees, at the time of the specified interval(k) and training finish
            - 每次新增k个叶节点，仿真结束训练，对模型进行测试和权重更新。
    - Coding
        - `from rgf.sklearn import RGFRegressor`
- CatBoost(真滴难理解)
    - Application: used in recommendation system
    - Advantages
        - 1. Ordered Boosting:
            - Effect: reduce the gradient bias and prediction shift problem
                - Biased pointwise gradient estimates
                    - Reason: in each gradient step we use the same data points, which leads to *distribution shift* in feature space domain.
            - It uses decision trees as the base model and gradient descent algorithm, *a new training dataset* is generated in each step of boosting.
            - The leaf value for the tree with ith sample=average gradients computed previously
        - 2. 🌟Handle Categorical Feature(categorical → numerical)

- low-cardinality features: use **one-hot encoding** (with 0,1 indicator) directly, then use histogram-based approach for split searching. set `simple_ctr`
- high cardinality features: divide features into several categories by **Target Statistics**, then use *one-hot encoding* techique. set `combinations_ctr`
  - Target Statistics:
    - **Greedy Target Based Statistics**(an improvement of Greedy TS)
      - $x_{i,k} = \frac{\sum_{j=1}^{p-1}[x_{j,k}=x_{i,k}*Y_j]+a*p}{\sum_{j=1}^{p-1}[x_{j,k}=x_{i,k}]+a}$
        - p: the prior value added, to reduce noisy data, the target average value from previous training datasets
        - a: weight coefficient of prior>0
        - j: index for permutation methods
        - k: feature index
        - $[\cdot]$: =1 if $x_{j,k} = x_{i,k}$(numerical feature value=label value), =0 otherwise.
      - use the *average value of labal* to decide whether split or not
    - Others: Holdout, Leave-one-out
  - Application in CatBoost:
    - 1. calculate some summary statistics based on categories and labels, add some hyperparameters, generate *new numerical features*
    - 2. use several different permutations to generate trees
    - 3. use Greedy algorithm generates feature combinations
    - 4. use One-hot encoding to deal with low-dimension feature
  - Cons:
    - avoid overfitting problems
    - keep training dataset sufficient
    - Reduce the listed problems:
      - Target leakage: in prediction process, we don't know target value
      - Train/Test data shift: the distribution of training and testing datasets are not the same sometimes. Mean encoding in training set mislead model prediction.
- 3. Symmetric Trees(oblivious trees):
  - help decrease prediction time, low latency requirements, faster scorer
    - Reason: nodes in the same level of tree obtain the same splitting criterion. To calculate leaf index, binary vector is built $\sum_{i=0}^{d-1} 2^i * B(x, f(t,i))$
      - Notation
        - d: tree depth; i: depth index
        - t: tree index

                - f: binary feature
                - $f(t, i)$: number of binary tree
                - B: a vector stores all binary feature values
            - this special structure can be treated as regularization technique for preventing overfitting and utilizing efficient CPU.
    - Parameter Tuning https://blog.csdn.net/xiangxiang613/article/details/106234234
    - Coding
        - `pip install catboost`
        - `from catboost import CatBoostClassifier /CatBoostRegressor`
        - `model = CatBoostClassifier()`
        - `model.fit(X_train, y_train, plot=True, eval_set=(X_test, y_test))`
- Optimization Algorithms for Hyperparameter Tuning
    - RandomSearchCV
        - draw a random value during each iteration from the range of specified values for each hyperparameter and evaluate the model, pick the hyperparameter configuration with best score at final.
        - parameters:
            - n_iter: # of parameter combinations are sampled
    - GridSearchCV
        - pass on a parameter's dictionary with some specified values to the function and compare the cross-validation score to find the best parameter value from different hyperparameter values combination
        - parameters:
            - estimator: select the model type we use
            - param_grid: the parameter re-set dictionary
            - scoring: evaluate performance metric of CV model
    - Other: HalvingGridSearchCV, HalvingRandomSearchCV, ParameterGrid, ParameterSampler...
- References:
    - Decision Tree Algorithm--A complete guide. https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/
    - Understanding Random Forest Algorithms with Example. https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/
    - Shap. https://github.com/slundberg/shap
    - Gradient Boosting Algorithm: a complete guide for beginners. https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/
    - xgboost. https://xgboost.readthedocs.io/en/stable/

- XGBoost: A Deep Dive into Boosting. https://medium.com/sfu-cspmp/xgboost-a-deep-dive-into-boosting-f06c9c41349
- LightGBM library. https://lightgbm.readthedocs.io/en/v3.3.2/Parameters.html
- 深入理解LightGBM. https://zhuanlan.zhihu.com/p/99069186
- LightGBM: A high-efficient graident decision tree. https://heartbeat.comet.ml/lightgbm-a-highly-efficient-gradient-boosting-decision-tree-53f62276de50?gi=5c70ad960f81
- CatBoost原理. https://zhuanlan.zhihu.com/p/108110195
- Paper: CatBoost: gradient boosting with categorical features support.http://learningsys.org/nips17/assets/papers/paper_11.pdf
- An introductory guide to regularized greedy forests with a case study. https://www.analyticsvidhya.com/blog/2018/02/introductory-guide-regularized-greedy-forests-rgf-python/

以上内容整理于 幕布文档