Tree Based Models

Saturday, 4 June 2022 5:35 PM

References:
https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db

https://github.com/AnshulSaini17/Income_evaluation/blob/main/Income_Evalutation.ipynb

https://neptune.ai/blog/gradient-boosted-decision-trees-guide
https://neptune.ai/blog/how-to-organize-your-lightgbm-ml-model-development-process-ex

57m •

LightGBM is a gradient boosting based on tree-model that boast faster time development and high efficiency. 🚀

There are a few reasons why LightGBM was considerably faster, including:

① Histogram-based Binning: LightGBM uses histograms for binning features, which makes the training process faster as the number of calculations required to determine feature splits is reduced.

② Exclusive Feature Bundling (EFB): LightGBM groups similar features together and processes them in a single split. It reduce the number of splits and makes the training process faster.

③ Gradient-based One-Side Sampling (GOSS): LightGBM uses GOSS, which would keep data with the larger gradient in the information gain and randomly drop data with a smaller gradient.

④ Parallel and GPU Processing: LightGBM supports parallel and GPU processing on large datasets, allowing for faster training.

⑤ Early Stopping: LightGBM uses early stopping to stop building trees when the improvement in loss is slight, resulting in a faster training process.

Code Blocks:

b ==> hands on GBM

xamples-of-best-practices

A)

```
lgb_train = lgb.Dataset(X_train, y_train)
lgb_eval = lgb.Dataset(X_test, y_test, reference=lgb_train)
gbm = lgb.train(model_params,
                lgb_train,
                num_boost_round=20,
                valid_sets=lgb_eval,
                early_stopping_rounds=5)
probas = gbm.predict(X_test)
# then i'm using these probabilities to find the best precision / recall tradeoff using the very
# algorithm between the native api method and the sklearn wrapper method
```

------------ below is wrapper one ------------------------

```
model = lgb.LGBMClassifier(n_estimators=20, **model_params)
model.fit(X_train, y_train, eval_set=[(X_test, y_test)], early_stopping_rounds=5)
probas = model.predict_proba(X_test)
```

A.1)

To get both train and val accuracy :

https://github.com/microsoft/LightGBM/issues/3312

These both are

```
importlightgbmfromsklearnimportmetricsfit=lightgbm.Dataset(X_fit, y_fit)
val=lightgbm.Dataset(X_val, y_val)
model=lightgbm.train(
    params={
        'learning_rate': .01,
        'objective': 'binary',
        'metric': 'binary_logloss',
    },
    train_set=fit,
    num_boost_round=10_000,
    valid_sets=(fit, val),
    valid_names=('fit', 'val'),
    early_stopping_rounds=20,
    verbose_eval=100,
)
y_pred=model.predict(X_test)
print()
print(f'ROC AUC: {metrics.roc_auc_score(y_test, y_pred):.5f}')
print(f'Log loss: {metrics.log_loss(y_test, y_pred):.5f}')
```

A.2)

same

same we can use any 1 of them to get results.

```
    evals_result = {}
    model = lightgbm.train(
        param_grid,
        train_data,
        num_boost_round= trial.suggest_int("num_boost_round", 10, 1000),  #
sense)
        feval=callback_auprc,
        valid_sets=(train_data,test_data),
        valid_names=('train','val'),
        verbose_eval=100,
        evals_result=evals_result,
        early_stopping_rounds=10,
        #callbacks=[lightgbm.record_evaluation(evals_result)],
    )
```

A.3)

0)

LightGBM provides direct support of categories as long as they are integer encoded prior to

Methods	Note	Need to change these parameters	Advantage	Disadvantage
Lgbm gbdt	it is the default type of boosting	because gbdt is the default parameter for lgbm you do not have to change the value of the rest of the parameters for it (still tuning is a must!)	stable and reliable	over-specialization, time-consuming, memory-consuming
Lgbm dart	try to solve over-specialization problem in gbdt	drop_seed: random seed to choose dropping modelsUniform_dro:set this to true, if you want to use uniform dropxgboost_dart_mode: set this to true, if you want to use xgboost dart modeskip_drop: the probability of skipping the dropout procedure during a boosting iterationmax_dropdrop_rate: dropout rate: a fraction of previous trees to drop during the dropout	better accuracy	too many settings
Lgbm goss	goss provides a new sampling method for GBDT by separating those instances with larger gradients	top_rate: the retain ratio of large gradient dataother_rate: the retain ratio of small gradient data	converge faster	overfitting when dataset is small

1) XGBoost vs LGBM:

In Xgboost, you have to manually create dummy variable/ label encoding for categorical fea
Catboost/Lightgbm can do it own their own, you just need to define categorical features nar
Training time is pretty high for larger dataset, if you compare against catboost/lightgbm.

default 100 constraint >= 0 (although 0 of course makes no

the training.

tures before feeding them into the models.
mes or indexes.

LightGBM is different from other gradient boosting frameworks because it uses a leaf-w
converge faster than depth-wise growth algorithms. However, they're more prone to overfit

3)light GBM:

But the irritating part that took my day to resolve is that lgbm wont accept categorical data in object format or string format you have to convert that to **categorical** type. So

```
for feature in obj_feat:
    data[feature] = pd.Series(data[feature], dtype="category")
```

The problem is that lightgbm can handle only features, that are of `category` type, not `object`. Here the list of all possible categorical features is extracted. Such features are encoded into integers in the code. But nothing happens to `object`s and thus `lightgbm` complains, when it finds that not all features have been transformed into numbers.

So the solution is to do

```
for c in categorical_feats:
    train[c] = train[c].astype('category')
```

LightGBM is a histogram-based algorithm which places continuous values into discrete bins,

Key Advantages:

Faster training speed and higher efficiency
Lower memory usage
Better accuracy
Support of parallel and GPU learning
Capable of handling large-scale data

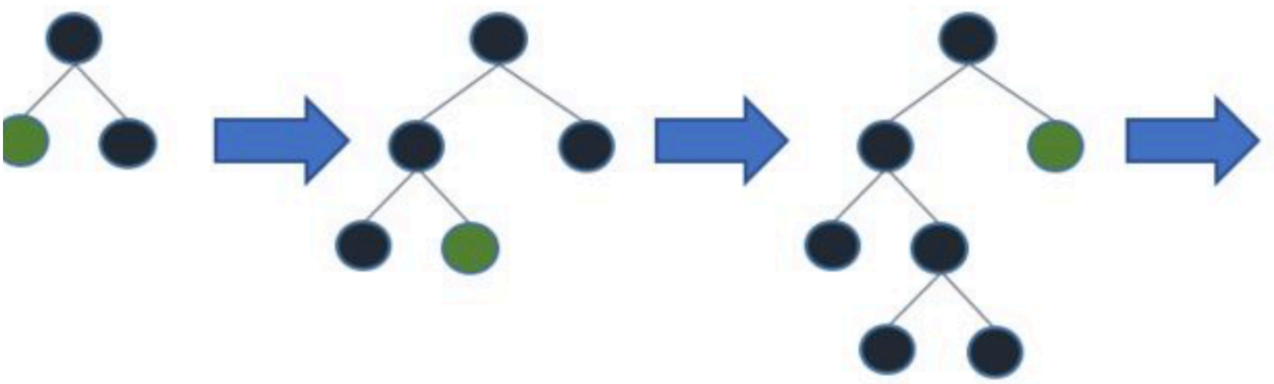
4)
Using Random Forest generates many trees, each with leaves of equal weight within the mo
Boosting introduces leaf weighting to penalize those that do not improve the model predicta
boosting also improves the bias.

ise tree growth algorithm. Leaf-wise tree growth algorithms are known to
tting.

which leads to faster training and more efficient memory usage.

odel, in order to obtain higher accuracy. On the other hand, Gradient Descent
ability. Both decision tree algorithms generally decrease the variance, while

5)LIGHTGBM:
Params:
`max_depth` the maximum depth of each tree;
`objective` which defaults to regression;
`learning_rate` the boosting learning rate;
`n_estimators` the number of decision trees to fit;
`device_type` whether you're working on a CPU or GPU.
boosting_type ==> by default **gbdt** and It can be changed to `**dart**` — Dropouts meet Sampling.



Leaf-wise tree growth

6)Random Forest and GBDT ::

Random Forest and GBDT are both based on Decision Trees. But they are using Decision Tre

Random Forest is a group / a class / a bunch of trees, which will later be averaged or n FOREST. This method is also called bagging.

Gradient Boosting Decision Tree is a sequence of trees, where each tree is built based method is called boosting.

Can you be more specific about the difference regarding the Decision Trees used in the mod

In Random Forest, each tree is DEEP. Each tree is trained on a sufficient amount of dat features, and the tree is developed as a big tree. Often times, a single tree in Random robust performance.

In GBDT, each tree is SMALL. We call it weak learners. It is weak in a way that the tree only pick a small amount of signal, and every time we are making a small progress. By

Why they are both robust if their approaches are so different?

When we are talking about machine learning models, there is one core thing that we a the balance of bias and variance.

Random Forest and GBDT are reducing the error and leveraging bias and variance trac

Random Forest: each DT has low bias and thus high variance. By averaging them toget

Multiple Additive Regression Trees, or `**goss**` — Gradient-based One-Side

e in a different way:
majority voted on. Each Decision Tree is independent. Thus the name Random

on the results of previous trees. So trees are not independent. And this

els?
ta (from a random subset of the whole training set) using random subset of
Forest is good already, and when we average all of them, we get a really

is very shallow, like 4 layers or even less. Each weak learner is supposed to
having many steps of learning, we can also get a robust performance.

always care about: the bias-variance tradeoff. Building a model is about finding

deoff in the opposite way:
ther, we can achieve a much lower variance by compromising a tiny bit of bias.

GBDT: each DT has high bias and low variance. So by combining them together sequen

What do you choose if you have a large dataset? (Usually, they are asking for parallel compu

In Random Forest, each tree can be built in parallel. So it is very fast. If I have a huge a
However, there are many great implementations of GBDT out there, like XGBoost, Ligh
to reduce the computational load. They can be equally fast and powerful.
So really, these two methods are both really good.

7)Missing vlaue handling in Trees:

Note: A very important point regarding how RF and GBM methods are handling *missing*
was proposed by its authors). CART trees are also used in Random Forests. CART handles mi
average/mode, either by an averaging/mode based on proximities. However, one can build a
CART is C4.5 proposed by Quinlan. In C4.5 the missing values are not replaced on data set. In
by penalizing the impurity score with the ratio of missing values

ntially, we can keep the low variance but also get a low bias.

uting)

mount of data I would go for Random Forest first.

ntGBM, etc. They all offer great performance by doing many engineering tricks

g data. Gradient Boosting Trees uses CART trees (in a standard setup, as it
ssing values either by imputation with average, either by rough
a GBM or RF with other types of decision trees. The usual replacement for
nstead, the impurity function computed takes into account the missing values