**Complete Guide To LightGBM Boosting Algorithm in Python**

Gradient Boosting Decision Tree (GBDT) is a popular machine learning algorithm. It has quite effective implementations such as XGBoost and many optimization techniques have been adopted from this algorithm. However, the efficiency and scalability are still unsatisfactory when the data has high dimensionality. The major reason is that the algorithm needs to scan all the data instances in each feature to estimate the information gain of all possible split points, which is very time-consuming.

To tackle this problem, the LightGBM (Gradient Boosting Machine) uses two techniques, namely Gradient-Based One-Side sampling (GOSS) and Exclusive Feature Bundling (EFB).

GOSS excludes the significant portion of data instances with small gradients and only uses the remaining data to estimate the information gain. Since the data instances with large gradients play a more important role in the computation of information gain, GOSS can obtain quite accurate information gain with a relatively much smaller dataset.

The LightGBM official document states that it grows the tree vertically while another tree-based learning algorithm grows horizontally; LightGBM grows trees leaf-wise, and it chooses max delta loss to grow. It can be best explained by the following visual.

A picture containing text, clock

Description automatically generated

The LightGBM offers many advantages;

* Faster training speed with higher accuracy,
* Lower memory usage,
* Better accuracy than any other boosting algorithm specially handles the overfitting very well when working with a small dataset,
* Compatibility with large datasets, and
* Parallel learning support.

With such features and advantages, LightGBM has become the facto algorithm in machine learning when working with tabular data for both kinds of problems, regression and classification.

**Implementing LightGBM in Python**

LightGBM can be installed using Python Package manager *pip install lightgbm*.

The dataset used here is the Titanic Passengers data.

*Importing all dependencies*

*import lightgbm as lgb*

*import pandas as pd*

*import numpy as np*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn import metrics*

*Loading the data:*

*data = pd.read\_csv('SVMtrain.csv')*

*data.head()*

Table

Description automatically generated

We have 8 columns out of which PassengerID will be dropped and Embarked will be chosen as the target variable for the classification problem.

**Loading the variables:**

*# define input and output feature*

*x = data.drop(['Embarked','PassengerId'],axis=1)*

*y = data.Embarked*

*# train test split*

*x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.33,random\_state=42)*

**Loading and fitting the model:**

The process of Initializing the model is like that of normal model initializing; however we have quite a few parameters that we can play with

While initializing the model we will define the learning rate, max\_depth and random\_state.

*model = lgb.LGBMClassifier(learning\_rate=0.09,max\_depth=-5,random\_state=42)*

*model.fit(x\_train,y\_train,eval\_set=[(x\_test,y\_test),(x\_train,y\_train)],*

*verbose=20,eval\_metric='logloss')*

In the fit method, we have passed eval\_set and eval\_metrix to evaluate our model during

training itself.

A screenshot of a computer

Description automatically generated with low confidence

**Evaluating the model:**

First we check whether this model is overfitted; if not, we will move to further evaluation.

*print('Training accuracy {:.4f}'.format(model.score(x\_train,y\_train)))*

*print('Testing accuracy {:.4f}'.format(model.score(x\_test,y\_test)))*

A screenshot of a computer

Description automatically generated with low confidence

As we can see, there is no significant difference between both the accuracy, so the model has estimated the information to be nearly good.

LightGBM comes with additional plotting functionality such as plotting the feature importance, plotting the metric evaluation, and plotting the tree. Below we will see the feature importance and metric evaluation.

*lgb.plot\_importance(model)*

A picture containing calendar

Description automatically generated

If you do not mention the eval\_set in the fit method, then you will get the error while plotting the metric evaluation.

*lgb.plot\_metric(model)*

Chart, line chart

Description automatically generated

As you can see, the validation curve tends to increase after the 100th iteration; this can be fixed by setting and tuning the hyperparameters in the model setup.

Below now lets plot the few metrics using sklearn library;

*metrics.plot\_confusion\_matrix(model,x\_test,y\_test,cmap='Blues\_r')*

Chart, treemap chart

Description automatically generated

*print(metrics.classification\_report(y\_test,model.predict(x\_test)))*

A picture containing text, receipt

Description automatically generated

As we can see from the confusion matrix and classification report, the model struggles when predicting class 1 due to the relatively few instances available for it, but if we compare this result with other ensemble algorithms, LightGBM performs best.  The same procedure is followed for the Regression Problem; we simply need to change the estimator to LGBMRegressor().