Boosting algorithm(CatBoosting algorithm)

The term CatBoost is an acronym that stands for "Category" and "Boosting."

According to the CatBoost documentation, CatBoost supports **numerical**, **categorical**, and **text features** but has a good handling technique for categorical data.

The CatBoost algorithm has quite a number of parameters to tune the features in the processing stage.

"Boosting" in CatBoost refers to the gradient boosting machine learning. Gradient boosting is a machine learning technique for regression and classification problems.

Which produces a prediction model in an ensemble of weak prediction models, typically **decision trees.**

Gradient boosting is a robust machine learning algorithm that performs well when used to provide solutions to different types of business problems such as

- **Fraud detection,**
- **Recommendation system,**
- **Forecasting.**

Again, it can return an outstanding result with relatively **fewer** data.

But other machine learning algorithms that only perform well after learning from extensive data.

Here we would look at the various features the CatBoost algorithm offers and why it stands out.

Robust

CatBoost can improve the **performance** of the model while **reducing overfitting** and the time spent on tuning.

CatBoost has several parameters to tune. Still, it reduces the need for extensive **hyper-parameter tuning** because the **default parameters** produce a great result.

Overfitting is a common problem in gradient boosting, especially when the dataset is small or noisy. CatBoost has several features that help reduce overfitting.

One of them is a novel gradient-based regularization technique called ordered boosting, which penalizes complex models that overfit the data. Another feature is the use of **per-iteration learning rate**, which allows the model to adapt to the complexity of the problem at each iteration.

Automatic Handling of Missing Values

Missing values are a common problem in real-world datasets. Traditional gradient boosting frameworks require imputing missing values before training the model. CatBoost, however, can handle **missing values automatically.**

During training, it learns the optimal direction to move along the gradient for each missing value, based on the patterns in the data.

Accuracy

The CatBoost algorithm is a high performance and **greedy novel** gradient boosting implementation.

Hence, CatBoost (when implemented well) either leads or ties in competitions with standard benchmarks.

Categorical Features Support

Usually in machine learning algorithms after preprocessing and cleaning your data, the data has to be converted into **numerical features** so that the machine can understand and make predictions.

This is same like, for any text related models we convert the text data into to numerical data it is known as word embedding techniques, or OneHotEncoder, OrdinalEncoder.

This process of encoding or conversion is time-consuming. CatBoost supports working with non-numeric factors, and this saves some time plus improves your training results.

Easy Implementation

CatBoost offers easy-to-use interfaces. The CatBoost algorithm can be used in Python with scikit-learn, R, and command-line interfaces.

Fast and scalable GPU version: the researchers and machine learning engineers designed CatBoost at Yandex to work on data sets as large as tens of thousands of objects without lagging.

Training your model on **GPU** gives a better speedup when compared to training the model on CPU.

To crown this improvement, the larger the dataset is, the more significant the speedup. CatBoost efficiently supports **multi-card configuration**. So, for large datasets, use a multi-card configuration.

Faster Training & Predictions

Before the improvement of servers, the maximum number of GPUs per server is 8 GPUs. Some data sets are more extensive than that, but CatBoost uses distributed GPUs.

This feature enables CatBoost to learn faster and make predictions 13-16 times faster than other algorithms.

Interpretability

CatBoost provides some level of interpretability. It can output feature importance scores, which can help understand which features are most relevant for the prediction.

It also supports visualization of decision trees, which can help understand the structure of the model.

Supporting Community of User

Good community support for CatBoost.

CatBoost has a growing community where the developers lookout for feedbacks and contributions.

Is tuning required in CatBoost?

The answer is **not straightforward** because of the type and features of the dataset. The **default** settings of the parameters in CatBoost would do a good job.

CatBoost produces good results without extensive **hyper-parameter tuning**. However, some important parameters can be tuned in CatBoost to get a better result.

These features are easy to tune and are well-explained in the CatBoost documentation. Here are some of the parameters that can be optimized for a better result;

- cat_ features,
- one_hot_max_size,
- learning_rate & n_estimators,
- max_depth,
- subsample,
- colsample_bylevel,
- colsample_bytree,
- colsample_bynode,
- l2_leaf_reg,
- random_strength.

CatBoost vs. XGBoost Comparison

These three popular machine learning algorithms are based on gradient boosting techniques. Hence, a greedy and very powerful.

Several Kagglers have won a Kaggle competition using one of these accuracy-based algorithms.

Before we dive into the several differences that these algorithms possess, it should be noted that the CatBoost algorithm does not require the conversion of the data set to any specific format. Precisely numerical format, unlike XGBoost

The oldest of these three algorithms is the XGBoost algorithm. It was introduced sometime in **March 2014** by Tianqi Chen, and the model became famous in **2016**.

Microsoft introduced **lightGBM** in January 2017. Then Yandex open sources the CatBoost algorithm later in April 2017.

The algorithms differ from one another in implementing the boosted trees algorithm and their technical compatibilities and limitations.

XGBoost was the first to improve GBM's training time. Followed by LightGBM and CatBoost, each with its techniques mostly related to the splitting mechanism.

Split

The split function is a useful technique, and there are different ways of splitting features for these three machine learning algorithms.

One right way of splitting features during the processing phase is to inspect the characteristics of the column.

The CatBoost algorithm introduced a unique system called Minimal Variance Sampling (MVS), which is a weighted sampling version of the widely used approach to regularization of boosting models, Stochastic Gradient Boosting.

Also, Minimal Variance Sampling (MVS) is the new default option for subsampling in CatBoost.

With this technique, the number of examples needed for each iteration of boosting decreases, and the quality of the model improves significantly compared to the other gradient boosting models.

The features for each boosting tree are sampled in a way that maximizes the accuracy of split scoring.

In contrast to the two algorithms discussed above, XGBoost **does not** utilize any **weighted sampling** techniques.

This is the reason why the splitting process is slower compared to the GOSS of LightGBM and MVS of CatBoost.

Leaf Growth

A significant change in the implementation of the gradient boosting algorithms such as XGBoost, LightGBM, CatBoost, is the method of tree construction, also called **leaf growth**.

The CatBoost algorithm grows a balanced tree. In the tree structure, the feature-split pair is performed to choose a leaf.

The split with the smallest penalty is selected for all the level's nodes according to the penalty function. This method is repeated level by level until the leaves match the **depth of the tree**.

Missing Values Handling

CatBoost supports three modes for processing

1. missing values,
2. "Forbidden,”
3. "Min," and "Max.”

For "Forbidden,” CatBoost treats missing values as not supported.

The presence of the missing values is interpreted as errors. For "Min,” missing values are processed as the minimum value for a feature.

With this method, the split that separates missing values from all other values is considered when selecting splits.

"Max" works just the same as "Min,” but the difference is the change from minimum to maximum values.

The method of handling missing values for LightGBM and XGBoost is similar. The missing values will be allocated to the side that reduces the loss in each split.

Categorical Features Handling

CatBoost uses one-hot encoding for handling categorical features. By default, CatBoost uses one-hot encoding for categorical features with a small number of different values in most modes.

The number of categories for one-hot encoding can be controlled by the **one_hot_max_size** parameter in Python and R.

On the other hand, the CatBoost algorithm categorical encoding is known to make the model slower.

However, the engineers at Yandex have in the documentation stated that one-hot encoding should not be used during pre-processing because it affects the model’s speed.

LightGBM uses **integer-encoding** for handling the categorical features. This method has been found to perform better than one-hot encoding.

The categorical features must be encoded to non-negative integers (an integer that is either positive or zero).

The parameter that refers to handling categorical features in LightGBM is categorical_features.

XGBoost was not engineered to handle categorical features. The algorithm supports only numerical features.

This, in turn, means that the encoding process would be done manually by the user.

Some manual methods of encoding include label encoding, mean encoding, and one-hot.

When and When Not to Use CatBoost

We have discussed all of the goods of the CatBoost algorithm without addressing the procedure for using it to achieve a better result.

When To Use CatBoost

Short training time on a robust data

CatBoost performs well with a small data set.

However, it is advisable to be mindful of **overfitting**. A little tweak to the parameters might be needed here.

Working on a small data set

This is one of the significant strengths of the CatBoost algorithm. Suppose your data set has categorical features, and converting it to numerical format seems to be quite a lot of work.

In that case, you can capitalize on the strength of CatBoost to make the process of building your model easy.

When you are working on a categorical dataset

CatBoost is incredibly faster than many other machine learning algorithms. The splitting, tree structure, and training process are **optimized** to be faster on GPU and CPU.

Training on GPU is 40 times faster than on CPU, two times faster than LightGBM, and 20 times faster than XGBoost.

When To Not Use CatBoost

There are not many disadvantages of using CatBoost for whatever data set.

So far, the hassle why many do not consider using CatBoost is because of the slight difficulty in tuning the parameters to optimize the model for categorical features.

CatBoost Algorithm Overview in Python 3.x

Pipeline:

1. Import the libraries/modules needed
2. Import data
3. Data cleaning and preprocessing
4. Train-test split
5. CatBoost training and prediction
6. Model Evaluation

Before we build the cat boost model, Let's have

Feature	Description
PassengerId	Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
survival	Survival (0 = No; 1 = Yes)
name	name of the passenger
sex	sex of passenger
age	age of the passenger
sibsp	Number of Siblings/Spouses Aboard
parch	Number of Parents/Children Aboard
ticket	Ticket Number
fare	Passenger Fare (British pound)
cabin	Passenger Cabin
embarked	Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southam

Before we implement the CatBoost, we need to **install** the the catboost library.

- Command: **pip install catboost**

```python
## import the libraries needed
import pandas as pd
import numpy as np

# Here we import our dataset from the CatBoost dataset library
from catboost.datasets import titanic

## The titanic dataset is made up of the train and test set, so we have to separate the data
titanic_train, titanic_test = titanic()

## Here we create a list to sort the columns so that the "Survived" column comes last
## This is because "Survived" is the target
column_sort = [ 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket',
'Fare', 'Cabin', 'Embarked','Survived']

## Now we apply the sorted columns to the train data
train = titanic_train[column_sort]
train.set_index('Pclass') ## Not necessary just to get of the default index

test = titanic_test
train.head()

## Remember the target column - "Survived" - we identified in the cell above,
## it is missing in the test set
## To solve this problem, we would create a column and fill it with dummy values,
## let's say '2' so it is not dormant and we can merge the DataFrame later
test['Survived'] = 2  ## The numpy background of pandas allows this to work
test.sample(5) ## shows five random rows in the dataset

## We would combine the train and test set into one DataFrame,
## so we do not have to repeat the same process for the two sets
df = pd.concat([train,test],ignore_index = False)

## Some features (such as Name, and Age) are irrelevant so we delete them
df = df.drop(['Name', 'Age'], axis=1)

## The data is not clean so we check all the columns for missing values
df.isnull().sum(axis=0)



## "Fare", "Cabin", "Embarked", and "PassengerId" have missing values, we have to fix this
```

```python
df['Embarked'] = df['Embarked'].fillna('S') ## The missing values in Embarked is filled with "S" (for
df['Cabin'] = df['Cabin'].fillna('Undefined')
df.fillna(-999, inplace=True)

## Now that the data looks good, we have to separate the train from the test set
train = df[df.Survived != 2]

test = df[df.Survived == 2]
test = test.drop(['Survived'], axis=1) ## drop the placeholder we created earlier in the test set

## Pop out the training features from the target variable
target = train.pop('Survived')
target.head()

## Let's ensure the model is trained and fit well
cat_features_index = np.where(train.dtypes != float)[0]

## Split the data into a train and test set
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(train, target,
train_size=0.85, random_state=1234)

## Import the CatBoostClassifier to fit the model and run a prediction
from catboost import CatBoostClassifier
model = CatBoostClassifier(
    custom_loss=['Accuracy'],
    random_seed=42)

## Set the metric for evaluation
model = CatBoostClassifier(eval_metric='Accuracy',
use_best_model=True,  random_seed=42)

model.fit(X_train, y_train, cat_features=cat_features_index,
eval_set=(X_test, y_test))


from catboost import cv
from sklearn.metrics import accuracy_score

print('the test accuracy is :{:.6f}'.format(accuracy_score(
y_test, model.predict(X_test))))
```