LightGBM Classifier in Python

LightGBM is a fast, distributed, high performance gradient boosting framework based on decision tree algorithms, used for ranking, classification and many other machine learning tasks.

Table of Contents

1. Introduction to LightGBM

- LightGBM is a gradient boosting framework that uses tree based learning algorithms

- Faster training speed and higher efficiency.

  - Lower memory usage.
  - Better accuracy.
  - Support of parallel and GPU learning.
  - Capable of handling large-scale data.
- At present, decision tree based machine learning algorithms dominate Kaggle competitions. The winning solutions in these competitions have adopted an alogorithm called **XGBoost**.

Light GBM can handle the large size of data and takes lower memory to run.

- Another reason why Light GBM is so popular is because it focuses on accuracy of results. LGBM also supports GPU learning and thus data scientists are widely using LGBM for data science application development.

- It is not advisable to use LGBM on small datasets. Light GBM is sensitive to overfitting and can easily overfit small data.

3. XGBoost Vs LightGBM

Table of Contents

- XGBoost is a very fast and accurate ML algorithm. But now it's been challenged by LightGBM — which runs even faster with comparable model accuracy and more hyperparameters for users to tune.
- The key difference in speed is because **XGBoost split the tree nodes one level at a time** and **LightGBM does that one node at a time**.

4. LightGBM Parameters

4.1 Control Parameters

4.2 Core Parameters

4.3 Metric Parameter

4.4 IO Parameter

4.1 Control Parameters

Table of Contents

- **max_depth** : It describes the maximum depth of tree. This parameter is used to handle model overfitting. If you feel that your model is overfitted, you should to lower max_depth.
- **min_data_in_leaf** : It is the minimum number of the records a leaf may have. The default value is 20, optimum value. It is also used to deal with overfitting.
- **feature_fraction**: Used when your boosting is random forest. 0.8 feature fraction means LightGBM will select 80% of parameters randomly in each iteration for building trees.
- **bagging_fraction** : specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting.
- **early_stopping_round** : This parameter can help you speed up your analysis. Model will stop training if one metric of one validation data doesn't improve in last early_stopping_round rounds. This will reduce excessive iterations.
- **lambda** : lambda specifies regularization. Typical value ranges from 0 to 1.

- **min_gain_to_split** : This parameter will describe the minimum gain to make a split. It can used to control number of useful splits in tree.
- **max_cat_group** : When the number of category is large, finding the split point on it is easily over-fitting. So LightGBM merges them into 'max_cat_group' groups, and finds the split points on the group boundaries, default:64.

5. LightGBM implementation in Python

In [3]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


import os
for dirname, _, filenames in os.walk(r"C:\Users\Hanshu\Desktop\kaggle_dataset"):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
C:\Users\Hanshu\Desktop\kaggle_dataset\Breast_cancer_data.csv
C:\Users\Hanshu\Desktop\kaggle_dataset\lightgbm-classifier-in-python.ipynb
C:\Users\Hanshu\Desktop\kaggle_dataset\naive-bayes-classifier-in-python.ipynb
C:\Users\Hanshu\Desktop\kaggle_dataset\adult.csv\adult.csv
```

In [4]:

```
# ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

Read dataset

In [5]:

```
# Load and preview data

df = pd.read_csv(r'c:\Users\Hanshu\Desktop\kaggle_dataset\Breast_cancer_data.csv
df.head()
```

Out[5]:

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnos
0	17.99	10.38	122.80	1001.0	0.11840	
1	20.57	17.77	132.90	1326.0	0.08474	
2	19.69	21.25	130.00	1203.0	0.10960	
3	11.42	20.38	77.58	386.1	0.14250	
4	20.29	14.34	135.10	1297.0	0.10030	

View summary of dataset

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   mean_radius      569 non-null    float64
 1   mean_texture     569 non-null    float64
 2   mean_perimeter   569 non-null    float64
 3   mean_area        569 non-null    float64
 4   mean_smoothness  569 non-null    float64
 5   diagnosis        569 non-null    int64  
dtypes: float64(5), int64(1)
memory usage: 26.8 KB
```

- We can see that there are 6 columns in the dataset and there are no missing values.

Check the distribution of target variable

- target variable is `diagnosis`
- check the distribution of the target variable.

In [7]:
```
# check the distribution of the target variable

df['diagnosis'].value_counts()
```

Out[7]:
```
diagnosis
1    357
0    212
Name: count, dtype: int64
```

- The target variable is `diagnosis`. It contains 2 values - 0 and 1.
- `0` is for **Negative prediction** and `1` for **Positive prediction**.
- We can see that the problem is binary classification task.

Declare feature vector and target variable

In [9]:
```
x = df[['mean_radius', 'mean_texture', 'mean_perimeter', 'mean_area', 'mean_smoc
y = df['diagnosis']
```

Split dataset into training and test set

In [10]:
```
# split the dataset into the training set and test set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.3, random
```

LightGBM Model Development and Training

- We need to convert our training data into LightGBM dataset format(this is mandatory for LightGBM training).
- After creating the necessary dataset, we created a python dictionary with parameters and their values.
- Accuracy of the model depends on the values we provide to the parameters.
- In the end block of code, we simply trained model with 100 iterations.

In [11]:
```
# build the lightgbm model
import lightgbm as lgb
clf = lgb.LGBMClassifier()
clf.fit(x_train, y_train)
```

```
[LightGBM] [Info] Number of positive: 249, number of negative: 149
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.000481 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 665
[LightGBM] [Info] Number of data points in the train set: 398, number of used fea
tures: 5
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.625628 -> initscore=0.513507
[LightGBM] [Info] Start training from score 0.513507
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

Out[11]:

```
▾ LGBMClassifier
LGBMClassifier()
```

In [20]:

```python
# the above oupput have some noisy so i can do

from lightgbm import LGBMClassifier
clf = LGBMClassifier(n_estimator=100, verbose=-1)
```

```
clf.fit(x_train, y_train)

#clf.get_params()
```

Out[20]:

```
LGBMClassifier

LGBMClassifier(n_estimator=100, verbose=-1)
```

In [ ]:

```
from lightgbm import LGBMClassifier

#create a classifier
#clf = LGBMClassifier(n_estimator=100, boosting_type='gbdt', learning_rate=0.1,

#fit the model
clf.fit(x_train, y_train)

# print the model then it show parameters
#print(clf)
clf.get_params()
```

Out[ ]:

```
{'boosting_type': 'gbdt',
 'class_weight': None,
 'colsample_bytree': 1.0,
 'importance_type': 'split',
 'learning_rate': 0.1,
 'max_depth': -1,
 'min_child_samples': 20,
 'min_child_weight': 0.001,
 'min_split_gain': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'num_leaves': 31,
 'objective': None,
 'random_state': 42,
 'reg_alpha': 0.0,
 'reg_lambda': 0.0,
 'subsample': 1.0,
 'subsample_for_bin': 200000,
 'subsample_freq': 0,
 'n_estimator': 100}
```

Model Prediction

In [19]:

```
# predict the results
y_pred = clf.predict(x_test)
```

View Accuracy

In [21]:

```
# view accuracy

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_pred, y_test)
print('LightGBM Model accuracy score: {0:0.4f}'.format(accuracy_score(y_test, y_
```

```
LightGBM Model accuracy score: 0.9298
```

Here, `y_test` are the true class labels and `y_pred` are the predicted class labels in the test-set.

Compare train and test set accuracy

- Now, I will compare the train-set and test-set accuracy to check for overfitting.

In [ ]:
```
# print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train,
```

In [26]:
```
from sklearn.metrics import accuracy_score

# predictions
y_train_pred = clf.predict(x_train)
y_test_pred = clf.predict(x_test)

#accuracy
train_acc = accuracy_score(y_train, y_train_pred)
test_acc = accuracy_score(y_test, y_test_pred)

print("Training accuracy:", train_acc)
print('Testing accuracy:', test_acc)
```

```
Training accuracy: 1.0
Testing accuracy: 0.9298245614035088
```

Check for Overfitting

In [28]:
```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(clf.score(x_train, y_train)))

print('Test set score: {:.4f}'.format(clf.score(x_test, y_test)))
```

```
Training set score: 1.0000
Test set score: 0.9298
```

- The training and test set accuracy are quite comparable. So, we cannot say there is overfitting.

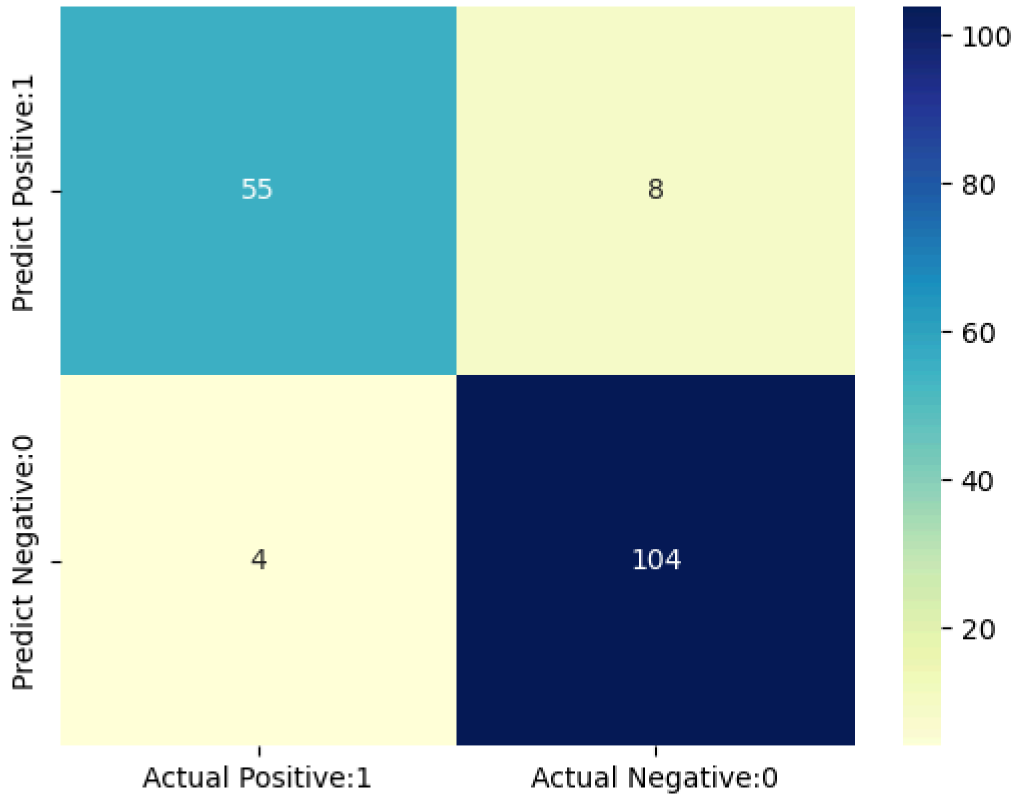
Confusion-matrix

In [29]:
```
# view confusion-matrix
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion matrix\n\n', cm)
print('\nTrue Positives(TP) = ', cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])
print('\nFalse Positives(FP) = ', cm[0,1])
print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[ 55   8]
 [  4 104]]

True Positives(TP) =  55

True Negatives(TN) =  104

False Positives(FP) =  8

False Negatives(FN) =  4
```

In [30]:
```python
# visualize confusion matrix with seaborn heatmap

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative
                                 index=['Predict Positive:1', 'Predict Negative:

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Out[30]: <Axes: >



In [31]:
```python
# **Classification Metrices**
```

In [32]:
```python
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.87      0.90        63
           1       0.93      0.96      0.95       108

    accuracy                           0.93       171
   macro avg       0.93      0.92      0.92       171
weighted avg       0.93      0.93      0.93       171
```

6. LightGBM Parameter Tuning

Table of Contents

- In this section, I will discuss some tips to improve LightGBM model efficiency.
- Following set of practices can be used to improve your model efficiency.
  - 1 **num_leaves** : This is the main parameter to control the complexity of the tree model. Ideally, the value of num_leaves should be less than or equal to 2^(max_depth). Value more than this will result in overfitting.
  - 2 **min_data_in_leaf** : Setting it to a large value can avoid growing too deep a tree, but may cause under-fitting. In practice, setting it to hundreds or thousands is enough for a large dataset.
  - 3 **max_depth** : We also can use max_depth to limit the tree depth explicitly.

For Faster Speed

- Use bagging by setting `bagging_fraction` and `bagging_freq` .
- Use feature sub-sampling by setting `feature_fraction` .
- Use small `max_bin` .
- Use `save_binary` to speed up data loading in future learning.

For better accuracy

- Use large `max_bin` (may be slower).
- Use small `learning_rate` with `large num_iterations`
- Use large `num_leaves` (may cause over-fitting)
- Use bigger training data
- Try `dart`
- Try to use categorical feature directly.

To deal with over-fitting

- Use small `max_bin`

- Use small `num_leaves`
- Use `min_data_in_leaf` and `min_sum_hessian_in_leaf`
- Use bagging by set `bagging_fraction` and `bagging_freq`
- Use feature sub-sampling by set `feature_fraction`
- Use bigger training data
- Try `lambda_l1` , `lambda_l2` and `min_gain_to_split` to regularization
- Try `max_depth` to avoid growing deep tree