Training External Xgboost model with BangDB CLI

Use case: Online Shoppers Purchasing Intention
Source: Kaggle
https://archive.ics.uci.edu/ml/datasets/Online+Shoppers+Purchasing+Intention+Dataset
Expected time to complete the exercise : 20 min

Introduction:

On bangdb, We are capable of even training an algorithm which is not supported by it. This document explains all the steps and requirements for training a model using a custom external algorithm. Here we will be taking an example to demonstrate the workflow for training a model for better understanding.

Users can try these use cases as they go through this guide; this will help in getting familiar with the workflow. To do this Users have to set up the BangDB server and CLI first.

Users can download BangDB from https://bangdb.com/download/ and follow the instructions on https://bangdb.com/server-install/ to install and start the server and CLI.

Please also check https://bangdb.com/developer/ for more information.

Get Started with ML through CLI:

The objective of this document is to show users how easy it is to train a model using custom external algorithms. it will help users

- To understand how to train a model using a custom external algorithm.
- To understand the training details required for training

Use-case: Online Shoppers Purchasing Intention

Problem statement:

The main objective revolved around the identification of key metrics which contribute the most towards predicting a shopper's behavior and to suggest prioritized critical recommendations and performance improvements on the same. Revenue is the attribute of interest which identifies if a purchase was made or not.

Objective:

Build a prediction model that will classify if a customer will end up not shopping

Data description:

The dataset consists of feature vectors belonging to 12,330 sessions.The dataset was formed so that each session would belong to a different user in a 1-year period to avoid any tendency to a specific campaign, special day, user profile, or period. The dataset consists of 10 numerical and 8 categorical attributes.The 'Revenue' attribute can be used as the class label.Of the 12,330 sessions in the dataset, 84.5% (10,422) were negative class samples that did not end with shopping, and the rest (1908) were positive class samples ending with shopping.

"Administrative", "Administrative Duration", "Informational", "Informational Duration", "Product Related" and "Product Related Duration" represent the number of different types of pages visited by the visitor in that session and total time spent in each of these page categories. The values of these features are derived from the URL information of the pages visited by the user and updated in real time when a user takes an action, e.g. moving from one page to another. The "Bounce Rate", "Exit Rate" and "Page Value" features represent the metrics measured by "Google Analytics" for each page in the e-commerce site. The value of "Bounce Rate" feature for a web page refers to the percentage of visitors who enter the site from that page and then leave ("bounce") without triggering any other requests to the analytics server during that session. The value of "Exit Rate" feature for a specific web page is calculated as for all pageviews to the page, the percentage that were the last in the session. The "Page Value" feature represents the average value for a web page that a user visited before completing an e-commerce transaction. The 'Special Day" feature indicates the closeness of the site visiting time to a specific special day (e.g. Mother’s Day, Valentine's Day) in which the sessions are more likely to be finalized with transactions. The value of this attribute is determined by considering the dynamics of e-commerce such as the duration between the order date and delivery date. The dataset also includes operating system, browser, region, traffic type, visitor type as returning or new visitor, a Boolean value indicating whether the date of the visit is weekend, and month of the year.

Algorithm:

This is a classification problem and we are going to solve this problem by using the Custom External algorithm “XGBoost” that is by uploading external python code.

Approach:

We can train model using algorithm which are not in built by uploading training and prediction python code it has to follow some basic protocol and here is the detail:

1. The training file should take the first attribute as the target attribute.
2. The Python file should implement the “main” function, this should be the entry point.
3. The Python file should take Model_script_path, Training_source_path arguments and it should return a string.

4. While training using external python scripts, make sure you have only one version of python3 running on the DB server. It is recommended that you have only one version of python running on the server.
5. When training using an external python script. Make sure all the libraries being used/imported in the script are installed. To check for installed libraries, you can use the help function in python to get the list of modules installed. Get into the python prompt and type the following command " **help("modules")** " this will list all the modules installed in the system.

If the libraries are not installed, you have to install it first. Example: Let's install Xgboost library, type " **pip3 install xgboost** " on the command line and if the pre-requirements are satisfied the library will be installed. [It is recommended to visit official website of the libraries and read the instruction and requirements before installing a library.]

Run the python script from the command line to check for errors in code. If you are able to execute the script with no errors on the command line then you can train using the same scripts.

Creating external python script:

Write two python scripts, one for training and other for prediction.Users can copy the script given below.

Training python Script

```
import pandas as pd
import sys
import xgboost as xgb
import joblib

def main(argv):
        narg = len(argv)
        if narg<3:
        print("argument list is improper")
        exit(1)
        model_name = argv[1]
        train_data = argv[2]
        try:
                dataset = pd.read_csv(train_data)
                x = dataset.iloc[:,1:17]
                y = dataset.iloc[:,0]
                model = xgb.XGBClassifier(max_depth=3,n_estimators = 300,
                        learning_rate=0.05)
                model.fit(x, y)
                print("model trained")
                # save the model to disk
                joblib.dump(model, model_name)
                print("model [ ", model_name, " ] saved to disk")
```

```
        return "SUCCESS"
        except Exception as e:
        print("error in execution", e)
return "ERROR"
if __name__ == "__main__":
try:
        if len(sys.argv) < 3:
        print ("Usage: Wrong input )
        exit (1)
        main(sys.argv[0:])
        except KeyboardInterrupt:
        print("Interrupt Received.. Shutting Down..")
exit(1)
```

As we can see in the example python script, all required libraries are imported. In the main function, the argument contains [“training_script_name”, “model_name”, “training_data_file_name”].

As per requirement of the user, the part written in red colour in the script can be changed.

Prediction python script:

```
import pandas as pd
import numpy as np
import sys
import xgboost as xgb
import joblib

def main(argv):
    narg = len(argv)
    if narg<4:
            print("num of arg is not proper, exiting")
            exit(1)
    model_name = argv[1]
    input_op = argv[2]
    if input_op == "1" and narg < 5:
            print("for input as file, we need output file name as well, less number of arg supplied")
            return "ERROR_ARGS"

    loaded_model = joblib.load(model_name)
    try:
            if input_op == "2":
                    input_list = [float(i) for i in argv[3].split(',')]
                    input_list = np.array(input_list).reshape(1, -1)
                    result = loaded_model.predict(input_list)
                    print(result.item(0))
                    return '%.2f' % result.item(0)
            elif input_op == "1":
                    fname = argv[3]
```

```
                    out_fname = argv[4]
                    of = open(out_fname, "w")
                    lines = [line.rstrip('\n') for line in open(fname)]
                    for line in lines:
                            line = [float(i) for i in line.split(',')]
                            line = line[:17]
                            line = np.array(line).reshape(1, -1)
                            print('line:', line)
                            result = loaded_model.predict(line)
                            print('result:', result)
                            np.savetxt(of, result, fmt="%2.3f")
                            of.close()
                    return "SUCCESS"
    except Exception as e:
            print("error in execution", e)
    return "ERROR"


if __name__ == "__main__":
    try:
            if len(sys.argv) < 4:
                    print("Usage: Input")
                    exit(1)
            main(sys.argv[0:])
    except KeyboardInterrupt:
            print("Interrupt Received.. Shutting Down..")
            exit(1)
```

Once the scripts are ready we have to upload them. Here also the procedure is the same: three ways, one is to register the mega_data directly or by following the workflow or training using api. For this use-case we will be following the second method.

Now we can train a model.

There are three ways to train a model on bangdb.

1. One is to directly register the meta_data for training (we call it json request which contains all the details about the model),

2. The second is to create mage_data for training by following the workflow on CLI and the last one is to train using api's.

Here we will be training the model using CLI.

Assuming that CLI and Server are running, let's go and train the model

Train the model

Method One : Use the CLI train model workflow

1. **Enter command "train model model_name"** :- train model cxt
   Here the workflow starts. User just have to enter the training details:

STEPS AND PARAMETER EXPLANATIONS FOR THE CLI WORKFLOW FOR TRAINING

2. **What's the name of the schema for which you wish to train the model?: sml**
   [Enter the schema, its user define -name of the schema where user want to apply model]

3. **Do you wish to read the earlier saved ml schema for editing/adding? [yes | no]: no**

Here we will get a list of all the algorithms supported. As we have to perform Classification, we will be selecting classification option for next command

4. **What's the algo would you like to use (or Enter for default (1)): 5**
   [Training from algorithm which are not in built we have to select option 5 (Custom External)]

5. **What's the input (training data) source? [local file (1) | file on BRS (2) | stream (3)]: 1**
   [Here, users have to specify the source of data whether it's a file store in the local system or in BRS or its streaming data. For this use-case, the training file is store in local system **]**

6. **Enter the file name for upload (along with full path): <enter path of file "shoppers_train.csv">**

7. **Enter the external train file (python) name for upload: <enter path of file "shopper_train.py">**
   [Location of training python script on local system]

8. **Enter the external pred file (python) name for upload: <enter path of file "shopper_pred.py">**
   [Location of prediction python script on local system]

9. **What is the input data format [LIBSVM (0) | CSV (1) | JSON (3)]: 1**
   [Here, users have to define the format of the training data file]

10. **Enter the argument list (name of columns in comma separated manner ex; "name, education, address..."): revenue,admin,admin_duration,infor,infor_duration,product, product_duration,bounce_rate,page_values,special_day,month,system,browser,region,traffic_type,visitor_type,weekday**
    [Providing column names]

11. **What's the training speed you wish to select [Very fast (1) | fast (2) | medium (3) | slow (4) | very slow (5)] (or Enter for default (1)):**
    [Here, this parameter doesn't play an important role. User can select any values]

12. **What is the target index (to select the target attribute/val): 0**
    [As we are training from csv file where we use position of attribute to identify them, here the attribute at position 0 is the target attribute this is also one of the requirements for training a custom external model]

BangDB deals with categorical data on its own by converting categorical to numerical, users just have to select the proper attribute type in the option below.

13. **What's the attribute type [NUM (1) | STRING (2) | HYBRID (3)] : 1**
    [Users have to specify the nature of attributes present in the training file. If all attributes are numerical then select option 1, if all are string select 2 and for both categorical and numerical select 3.]

14. **Do you wish to scale the data? [yes | no]: yes**

15. **Do you wish to tune the params? [yes | no]: yes**

Next, we need to do the mapping. This means we need to provide the attribute name and its position in the training file. We need to add mapping for [17] attributes as we have so many dimensions

16. **Enable attr name: revenue**
    enable attr position: 0
    do you wish to add more attributes? [yes | no]: **yes**

    ……
    ……
17. **Enable attr name:** weekday
    enable attr position: **16**
    do you wish to add more attributes? [yes | no]: **yes**

18. **Do you wish to add more attributes? [yes | no]: no**

Here, we can view the meta_data which we created for training.

updated schema :

```
{
  "algo_type" : "PY",
   "pred_file" : "shopper_pred.py",
   "training_details" : {
            "training_source_type" : 1,
            "expected_format" : "CSV",
            "file_size_mb" : 1,
            "train_speed" : 5,
            "target_idx" : 0,
            "training_source" : "shoppers_train.csv",
            "train_file" : "shopper_train.py",
            "input_format" : "CSV"
   },
   "attr_list" : [
      {
         "name" : "revenue",
         "position" : 0
      },
     .............
      {
         "name" : "weekday",
         "position" : 16
      }
   ],
  "attr_type" : 1,
   "model_name" : "cxt",
   "schema-name" : "sml",
   "arg_list" : [
            "revenue,admin,admin_duration,infor,infor_duration,product, product_duration,
bounce_rate,page_values,special_day,month,system,browser,region,traffic_type,visitor_type,wee
kday"
   ]
}
```


19. **Do you wish to start training now? [yes | no]: yes**

    schema [sml] registered successfully for training
    you may check the train status by using 'show train status' command ------training started

To check training status enter
show status where schema = “sml” and model = “model_name"
Training status 25 represents that the training is completed.

For more info on ML, please visite https://bangdb.com

- For more commands enter “help ml”

Method two : training model by uploading training request (training meta-data)

Step 1. Prepare a file containing training meta-data
Here, we have create a json file name cxt.json and we have created the training request:
Training request: (user can copy and paste the below training request)

{"schema-name":"sml","model_name":"cxt","algo_type":"PY","pred_file":"shopper_pred.py","arg_list":["revenue,admin,admin_duration,infor,infor_duration,product, product_duration, bounce_rate,page_values,special_day,month,system,browser,region,traffic_type,visitor_type,weekday"],"training_details":{"training_source":"shoppers_train.csv","file_size_mb":1,"training_source_type":1,"train_file":"shopper_train.py","expected_format":"CSV","input_format":"CSV","train_speed":5,"target_idx":0},"attr_type":1,"attr_list":[{"name":"revenue","position":0},{"name":"admin,","position":1},{"name":"admin_duration","position":2},{"name":"infor","position":3},{"name":"infor_duration","position":4},{"name":"product","position":5},{"name":"product_duration","position":6},{"name":"bounce_rate","position":7},{"name":"page_values","position":8},{"name":"special_day","position":9},{"name":"month","position":10},{"name":"system","position":11},{"name":"browser","position":12},{"name":"region","position":13},{"name":"traffic_type","position":14},{"name":"visitor_type","position":15},{"name":"weekday","position":16}]}

Step 2. Enter command **train model from model-meta-data**
[Model-meta-data = is the location of the file containing the meta data for training with its file name.
From here the cli workflow will start]

After entering the above command the training request in the file will be displayed on the screen
Step 3. Cli will ask the path for training file on the system
upload file : shoppers_train.csv
enter the path for upload (full path of the file):
[provide the training file path with file name]

Step 4. upload file : shopper_train.py
enter the path for upload (full path of the file):

Step 5: file upload successful
upload file : shopper_pred.py
enter the path for upload (full path of the file):

Step 4. Do you wish to start training now? [yes | no]: yes
[Enter yes to start training]

The User has to understand that the training time taken depends on a lot of factors (like the parameter selected, size of data, bangdb setting etc..)

Once the training is started user can check the train status by using 'show train status' command

Step 5. show status where schema = "sml" and model = "criditRisk"

For more info on ML, please visite https://bangdb.com

Predict using the model

For prediction, user have to enter command “pred model model_name”, then we just have to enter details:-

STEPS FOR THE CLI WORKFLOW FOR PREDICTION

1. **Enter the command “pred model cxt”**
2. **What's the name of the schema for which model was trained?: sml**
3. **Do you wish to see the train request? [yes | no]: no**

model algo type is [PY] it needs [NUM] data type with [CSV] input data format

[Above comment will be displayed on the CLI, giving info about the model, its attribute type and format required by the model for prediction.]

4. **What is the input data format for the given pred file [LIBSVM (0) | CSV (1) | JSON (3)] (press Enter for default 1): 1**
   [As our test file is in CSV, we will select 1]
5. **What is the separator (SEP) for the csv file?:**
   [we have to mention the SEP for CSV test file]
6. **Do you wish to provide an attribute list? [yes | no]: no**
   [in cases where arrangement of attributes are different while training and prediction]
7. **Do you wish to consider the target (are you also supplying target value?) [yes | no]: no**
   [we select this when have target within test data]
8. **Do you wish to pred the file? Or a single event? [yes (file) | no (single event)]: yes**
9. **Do you wish to upload the file? [yes | no]: yes**
10. **Enter the test file name for upload:**<enter path of file **shoppers_test.csv>**
11. **Once it's done, we can download the test file**

Result:

- The time taken for training was approx 2 min.
- The accuracy is 89%.
- The User has to understand that the training time taken depends on a lot of factors (like the parameter selected, size of data, bangdb setting etc..)