Taysols Data Science

CHURN PREDICTION WITH BUILDIN SAGEMAKER XGBOOST

Train and Deploy build-in XGBoost with Batch Transform

Yiran Jing

June 28, 2019

Contents

1 Dataset description

The customer churn dataset includes 7021 customers (after remove duplicates) with 20 features, and target column is churn. Since the origin data is dirty with missing values and apparently wrong records, I clean the dataset before the following steps. Look **Churn example** (*Click me to open this notebook*) for details about clean data and feature engineering.

Amazon SageMaker XGBoost can train on data in either a CSV or LibSVM format. For this example, we use CSV. It should have the following:

1. Have the predictor variable in the first column (as per AWS requirements)
2. Not have a header row (as per AWS requirements)
3. No customer_id (Useless column)
4. Numerical entry only (cleaned dataset as per AWS requirements)

Data Description :We need four datasets (you can find them in S3 churn buckets):

1. train.csv (70%) (include target) (*Click me to see the train dataset*)
2. validation.csv (20%) (include target) *: validation is the test dataset despite the confusing name* (*Click me to see the validation dataset*)
3. test.csv (10%) (include target) *: is for evaluating the model once deployed* (*Click me to see the test dataset*)
4. test_data_Batch.csv (10%) (remove the target column from test.csv) *: this is test.csv with prediction col removed* (*Click me to see the batch input dataset*)

In *train.csv*, *validation.csv*, and *test.csv*, the **first column is target**, and **no header**, also removed column *customer_id*, and the dataset contain **numerical entry only**. Based on the input data format of Batch transformation, I create *test_data_Batch.csv* dataset, which contain the same predictors information with *test.csv*, but remove the target column.

The test data prediction will be in ***test_data_Batch.csv.out*** after batch transformation. It gives the probability for each observation in test dataset. (*Click me to see the test-output file once batch transformation is finished*)
Also, you can check the output file once training is finished (*Click me to see the train-output file path*)

2 Train Model

All following code screenshots are from **BuildIn XGBoost Example**(*Click me to open this notebook*).

We need two dataset *train.csv* and *validation.csv* for training model.

2.1 Set role and model container

See figure 1, the first step we need to do is get execution role and get the XGBoost container. You can paste the following code in a cell in the Jupyter notebook you created for any XGBoost modelling.

```
# import useful packages
import boto3
import sagemaker
from sagemaker.amazon.amazon_estimator import get_image_uri
from sagemaker import get_execution_role

# The get_execution_role function retrieves the IAM role you created when you created your notebook instance.
role = get_execution_role()
# get the XGBoost container
container = get_image_uri(boto3.Session().region_name, 'xgboost')
```

Figure 1: Input package, set role and model container

2.2 Read in train and validation data from S3 bucket

See figure 2, the second step is find the data path and then read in train and validation data. Note that we **must have two data files train file and validation file, and only one csv dataset in each data file**. We cannot specify the name of train or validation dataset, but we need to specify the dataset type (csv in our case). That is,

$$s3\_input\_train = bucket/prefix/train/$$

any CSV dataset with in train file will be read as train data, note that **Only one csv datafile** shall be put in train data file. Same as validation file.

```
# The S3 bucket and prefix that you want to use for training and model data.
bucket = 'taysolsdev'
prefix = 'datasets/churn'

# read in data from S3
s3_input_train =sagemaker.s3_input(s3_data='s3://{}/{}/train/'.format(bucket, prefix), content_type='csv')
s3_input_validation = sagemaker.s3_input(s3_data='s3://{}/{}/validation/'.format(bucket, prefix), content_type='csv')
```

Figure 2: Read data from S3 bucket

2.3 Specific Output path and Create estimator instance

See figure 3, firstly, we create session object, and then create the estimator instance. Note that we need to give the output file path here. You can paste the following code in a cell in the Jupyter notebook

```
#  The session object that manages interactions with Amazon SageMaker APIs and any other AWS service that the training
sess = sagemaker.Session()

# Create an instance of the sagemaker.estimator.Estimator class
# output_path – The path to the S3 bucket where Amazon SageMaker stores the training results.
# train_instance_count: generally use only a single training instance.
xgb = sagemaker.estimator.Estimator(container,
                                    role,
                                    train_instance_count=1,
                                    train_instance_type='ml.m4.xlarge',
                                    output_path='s3://{}/{}/output'.format(bucket, prefix),
                                    sagemaker_session=sess)
```

Figure 3: Create estimator instance

2.4 Set hyperparameter for GXBoost

See figure 4, before actually train model, we can set optiminal hyperparameter in this step. Note that ***silent* parameter must be integer**, cannot be none. And we **must set parameter *num_round*** in this step.
You can change **objective** parameter if use GXBoost for regression or multi-classification. Since our case is binary classification, I use *binary:logistic*, and output is probability for each observation.

```
# Set the hyperparameter values for the XGBoost training job
xgb.set_hyperparameters(max_depth=3,
                        verbosity=1,
                        random_stae=960428,
                        gamma=0,
                        subsample=1,
                        reg_lambda=1,
                        silent=0, # silent must be integer, cannot be none
                        colsample_bytree=1,
                        min_child_weight=1,
                        learning_rate = 0.02,
                        tree_method='hist',
                        n_estimators=200,
                        class_weight='balanced',
                        objective='binary:logistic',#logistic regression for binary classification, output probability
                        num_round=50 #The number of rounds for boosting (only used in the console version of XGBoost)
                        )
```

Figure 4: Set hyperparameter for GXBoost

2.5 Begin train model

See figure 5, since sagemaker uses **lazy evaluation**, we can only actually begin train model when we call *.fit()* method. You can check the S3 output dataset with in output file after training job finish. You can paste the following code in a cell in the Jupyter notebook.

```
# start model training
xgb.fit({'train': s3_input_train, 'validation': s3_input_validation}, logs=True)
```

Figure 5: Begin train model

2.6 Another way to train model: SDK format

JSON format for configuring model: The version in this section follows the SDK format that calls to the AWS API for building an ML model. Another way to do this would be to configure the model using JSON format as below.

2.6.1 User-defined Model name

In this method, since we need to configuring model, we need to know the name of the model we are running, and we can also define the model name by ourselves. You are open Amazon SamgeMaker interface to check the model name.

```
%%time
from time import gmtime, strftime

job_name = 'CHURN-xgboost-regression-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print("Training job", job_name)
bucket_path = 's3://{}/{}/output'.format(bucket, prefix)
```

Figure 6: User-defined Model name

2.6.2 configuring model

Ensure that the training and validation data folders generated above are reflected in the **InputDataConfig** parameter below. See figure 7 and figure 8:

```
create_training_params = \
{
    "AlgorithmSpecification": {
        "TrainingImage": container,
        "TrainingInputMode": "File"
    },
   "RoleArn": role,
    "OutputDataConfig": {
        "S3OutputPath": bucket_path
    },
    "ResourceConfig": {
        "InstanceCount": 1,
        "InstanceType": "ml.m4.4xlarge",
        "VolumeSizeInGB": 5
    },
    "TrainingJobName": job_name,
    "HyperParameters": {
        #"max_depth":"3",
        #"gamma":"0",
        #"min_child_weight":"1",
        #"silent":"None",
        "num_round":"50",
        "objective":"binary:logistic",
        "class_weight":"balanced",
        "n_estimators":"200",
        "learning_rate":"0.02",
        "tree_method":"hist",
        "random_state":"960428",
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 3600
    },
```

Figure 7: configuring model part 1

```
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri" : 's3://{}/{}/train/'.format(bucket, prefix),
                    "S3DataDistributionType": "FullyReplicated"
                }
            },
            "ContentType": "csv",
            "CompressionType": "None"
        },
        {
            "ChannelName": "validation",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri" : 's3://{}/{}/validation/'.format(bucket, prefix),
                    "S3DataDistributionType": "FullyReplicated"
                }
            },
            "ContentType": "csv",
            "CompressionType": "None"
        }
    ]
}
```

Figure 8: configuring model part 2

2.6.3 Begin train configuring model

```
client = boto3.client('sagemaker', region_name=region)
client.create_training_job(**create_training_params)

import time

status = client.describe_training_job(TrainingJobName=job_name)['TrainingJobStatus']
print(status)
while status !='Completed' and status!='Failed':
    time.sleep(60)
    status = client.describe_training_job(TrainingJobName=job_name)['TrainingJobStatus']
    print(status)
```

Figure 9: Begin train configuring model

3 Deploy model with Batch Transformation

We need one dataset *testdataBatch.csv* in this step.

3.1 Read in test data and specify batch output path

See figure 10, firstly, we need to specify the batch input data, unlike train or validation dataset, we need to specify the name of test dataset for batch transformation. After that, we need to give the output path for batch transformation.

```
# the batch dataset used for prediction cannot have target column
batch_input = 's3://taysolsdev/datasets/churn/batch/test_data_Batch.csv' # test data used for prediction

batch_output = 's3://{}/{}/batch/batch-inference'.format(bucket, prefix) # specify the location of batch output
```

Figure 10: Read in test data for batch prediction

3.2 Run Batch Transform Job

See figure 11, to run a batch transform job, call the create_transform_job. method using the model that you trained before. *initial_instance_count* :The initial number of instances to run in the Endpoint created from this Model. You can paste the following code in a cell in the Jupyter notebook.

```
# creates a transformer object from the trained model
transformer = xgb.transformer(
                          instance_count=1,
                          instance_type='ml.m4.xlarge',
                          output_path=batch_output)

# calls that object's transform method to create a transform job
transformer.transform(data=batch_input, data_type='S3Prefix', content_type='text/csv', split_type='Line')

transformer.wait()
```

Figure 11: Run Batch Transform Job

4 Validate Model Deployed with Batch Transform

This step we need two datasets: the output prediction from batch transformation (**y_pred**) and test dataset including true target values (**y_test**).

4.1 Read in Batch output file and test data

See figure 12, to evaluate the model performance, firstly, we need to read in the batch predictions based on test dataset and true target values in test dataset. We can read in them use dataframe, note that we **must set *header=None***, otherwise we will loss the first row.
Also, the *test_data_Batch.csv.out* is different from *test_data_Batch.csv*, the first one is batch output and the second one is batch input. they are located in two different data files.

```
# test dataset with target
test_data = 's3://taysolsdev/datasets/churn/test/test.csv'
test_data = pd.read_csv(test_data, header=None, encoding = "ISO-8859-1")   # header = none

# batch output based on test data
batch_output = 's3://taysolsdev/datasets/churn/batch/batch-inference/test_data_Batch.csv.out'
batch_output = pd.read_csv(batch_output, header=None, encoding = "ISO-8859-1") # header = none
```

Figure 12: Read in Batch output file and test data

4.2 Model evaluation example

See figure 12, the *y_test* is the first column in the *test.csv*, and *y_pred* is the output dataset from batch transformation. Since we read them as dataframe, we can do any model evaluation calculation or plots using the general python code. In our binary classification example (threshold is 0.5):

```python
def get_score(y_true,y_pred):
    f1 = metrics.f1_score(y_true, y_pred)
    precision = metrics.precision_score(y_true, y_pred)
    recall = metrics.recall_score(y_true, y_pred)
    accuracy = metrics.accuracy_score(y_true, y_pred)
    tn, fp, fn, tp = metrics.confusion_matrix(y_true, y_pred).ravel()
    return precision, recall, f1, accuracy, tn, fp, fn, tp
```

```python
y_test = test_data.iloc[:, 0]
y_pred = np.round(batch_output) # threshold is 0.5


#get scores
temp_precision, temp_recall, temp_f1, temp_accuracy, tn, fp, fn, tp = get_score(y_test,y_pred)
output = [temp_precision,temp_recall,temp_f1,temp_accuracy,tp, fp, tn, fn]
output = pd.Series(output, index=['precision', 'recall', 'f1', 'accuracy', 'tp', 'fp', 'tn', 'fn'])
print(output[['accuracy', 'tp', 'fp', 'tn', 'fn']])

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
accuracy     0.798
tp         106.000
fp          41.000
tn         455.000
fn         101.000
dtype: float64
              precision    recall  f1-score   support

           0       0.82      0.92      0.87       496
           1       0.72      0.51      0.60       207

   micro avg       0.80      0.80      0.80       703
   macro avg       0.77      0.71      0.73       703
weighted avg       0.79      0.80      0.79       703
```

Figure 13: Model evaluation example

5 Reference

1. https://github.com/awslabs/amazon-sagemaker-examples/blob/master/introduction_to_applying_machine_learning/xgboost_customer_churn/xgboost_customer_churn.ipynb
2. https://docs.aws.amazon.com/batch/latest/userguide/job_states.html
3. https://sagemaker.readthedocs.io/en/stable/overview.html#sagemaker-batch-transfor
4. docs.aws.amazon.com/sagemaker/latest/dg/ex1-batch-transform.html