XGBoost Training Report by SageMaker Debugger

Crea
ted on 2021-12-04 09:52:32

The SageMaker Debugger `CreateXgboostReport` built-in rule auto-generates this report. This report provides a summary of the XGBoost model training evaluation results, insights of the model performance, and interactive graphs.

Legal disclaimer: In this report, plots and and recommendations are provided for informational purposes only and are not definitive. You are responsible for making your own independent assessment of the information.] For more information, see the following documentation:

- Amazon SageMaker Developer Guide (https://docs.aws.amazon.com/sagemaker/latest/dg/train-debugger.html)

If you want to use the notebook that generated this report, you need to install the following libraries:

- SageMaker Debugger Client Library Github (https://github.com/awslabs/sagemaker-debugger)
- The Bokeh Python Visualization Tool (http://docs.bokeh.org/en/0.11.0/docs/installation.html)

In [3]:
```
# Parameters
path = "/opt/ml/processing/input/tensors"
plot_step = 995
s3_path = "s3://sagemaker-us-east-1-791618627463/dogs-cats-data/xgboost_model/
sagemaker-xgboost-2021-12-04-09-44-07-609/debug-output"
```

The following parameters are the default values auto-generated by the `CreateXgboostReport` built-in rule.

- `path (str)` - The local path where Debugger has saved output tensors in the training container.
- `plot_step (int)` - The step for which the rule has created the training report.
- `s3_path (str)` - The S3 bucket URI where Debugger has saved the output tensors.

Table of Contents

(https://bokeh.org) BokehJS 2.2.3 successfully loaded.

Distribution of True Labels of the Dataset

In the following graph, you can check the histogram of the true (target) labels of your raw data. You can see if the distribution of the target labels for prediction are skewed (in case of regression) or imbalanced (in case of classification).

- If the data for regression is skewed, you might want to reduce the skewness by taking the log or power transformation on the right-scewed or the left-scewed data respectively. Using the Python NumPy (https://numpy.org/) library, you can simply apply numpy.log (https://numpy.org/doc/stable/reference/generated/numpy.log.html?highlight=log#numpy.log) or numpy.exp (https://numpy.org/doc/stable/reference/generated/numpy.exp.html?highlight=exp#numpy.exp) to your target values.
- If the data for classification is imbalanced, you might want to improve your sample by collecting more data, resampling, or generating synthetic samples. This can also be mitigated at the step of evaluating performance metrics, such as Confusion Matrix, Precision, Recall, F-score, and Receiver Operating Characteristic curves.

Loss vs Step Graph

SageMaker Debugger automatically captures loss values of the XGBoost Learning Task Parameters (https://xgboost.readthedocs.io/en/latest/parameter.html#learning-task-parameters).

The loss curves provide the following insights.

- An **underfitting** model *(high bias)* has high training error and high validation error. Underfitting means that your model is not converging while training, resulting in a generalization failure on both training and validation data sets.
- An **overfitting** model *(high variance)* has extremely low training error but a high validation error. Overfitting happens when your model is trained too well to fit the noisy training data, resulting in negative impacts on your model performance on validation set.

Suggestions:

- If underfitting, the model is not suitable to your dataset, or the model parameters are not properly set up. You might want to consider:
  - adding more features
  - increasing the complexity of your model — increase the depth of each tree, `max_depth`
  - tuning the hyperparameters — decrease the `gamma` and `eta` parameters
  - decreasing regularization parameters — the `lambda` and `alpha` parameters
  - or even training and comparing with other algorithms.
- If overfitting, the model is too complex and trains to fit on the noisiness of the training set. You might want to consider:
  - reducing the depth of each tree, max_depth
  - increasing the min_child_weight parameter
  - tuning the hyperparameters — increase the `gamma` and `eta` parameters
  - increasing regularization parameters — the `lambda` and `alpha` parameters
  - increasing the `subsample` and `colsample_bytree` parameters
  - pruning a tree to remove the least significant feature — referring to the next sections where the feature importance scores are provided.

Feature Importance

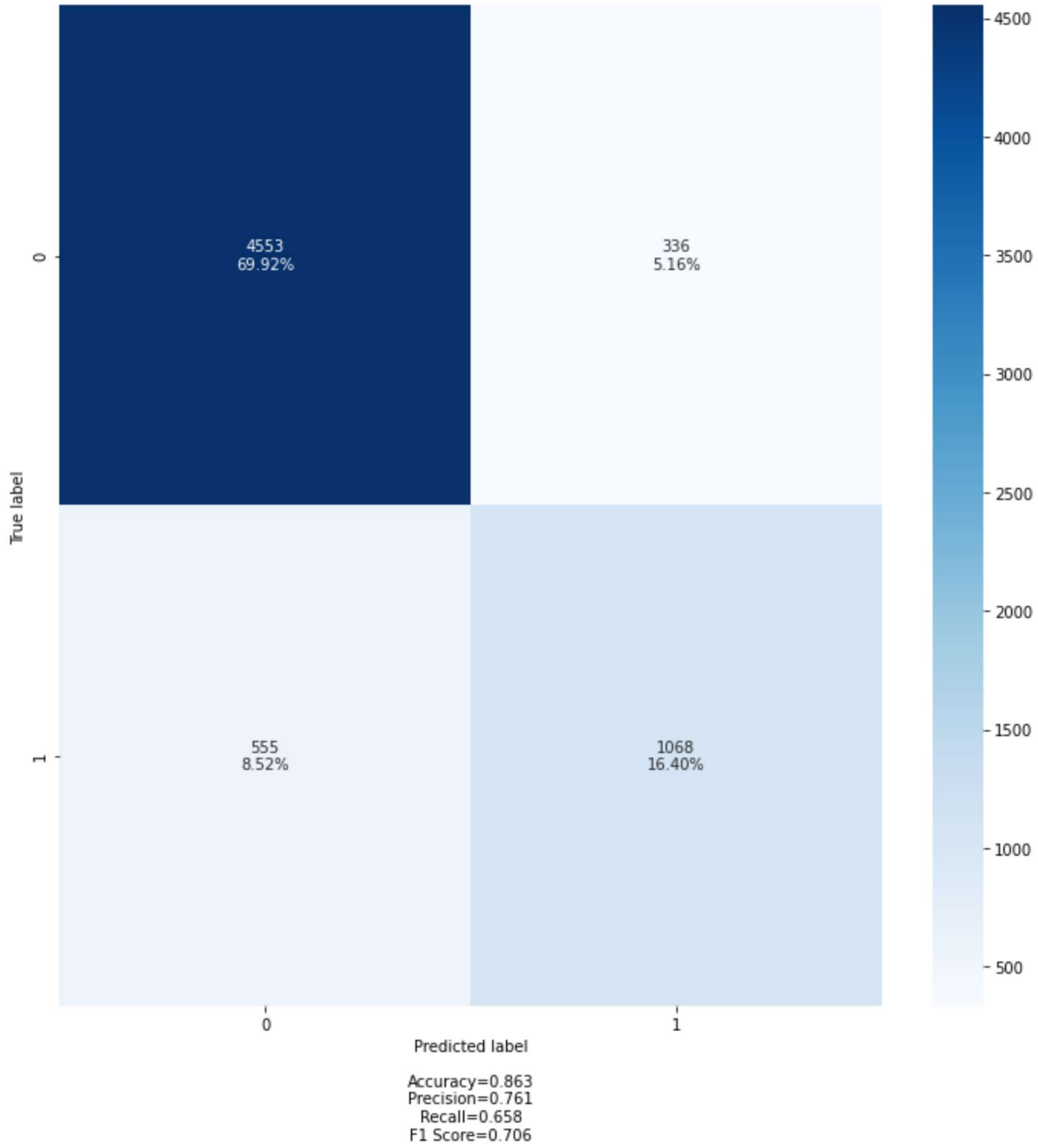
Feature importance refers to techniques that assign scores to input features based on how useful they are at predicting a target variable. The supported importance types for XGBoost are:

- **Weight** (or **Frequency**) is the percentage representing the relative number of times a particular feature occurs in the trees of the model.
- **Gain** is the improvement in accuracy brought by a feature to the branches it is on. *The Gain is the most relevant attribute to interpret the relative importance of each feature.*
- **Coverage** measures the relative quantity of observations concerned by a feature.

Confusion Matrix

The following Confusion Matrix shows a breakdown of the classification results.

- The cells on the principal diagonal shows the True Positive counts.
- The off-diagonal cells count the number of misclassified predictions.



Evaluation of the Confusion Matrix

The following statistics summary of the confusion matrix is provided using the Scikit-learn Metrics and Scoring APIs (https://scikit-learn.org/stable/modules/model_evaluation.html). You can use the following score metrics to evaluate the performance of your model: accuracy, precision, recall, and F1-score.

For more information, see the following Scikit-learn documentation:

- Accuracy Score (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html)
- Precision Score (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html#sklearn.metrics.pre
- Recall Score (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html#sklearn.metrics.recall_s
- F1-Score (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score)

Overall Accuracy

Overall Accuracy: 0.863

Micro Performance Metrics

Performance metrics calculated globally by counting the total true positives, false negatives, and false positives.

Micro Precision: 0.863
Micro Recall: 0.863
Micro F1-score: 0.863

Macro Performance Metrics

Performance metrics calculated for each label, and find their unweighted mean.
This does not take the class imbalance problem into account.

Macro Precision: 0.826
Macro Recall: 0.795
Macro F1-score: 0.808

Weighted Performance Metrics

Performance metrics calculated for each label and their average weighted by support
(the number of true instances for each label).
This extends the macro option to take the class imbalance into account.
It might result in an F-score that is not between precision and recall.

Weighted Precision: 0.859
Weighted Recall: 0.863
Weighted F1-score: 0.86

Classification Report

The summary of the precision, recall, and F1-score for each class.

	precision	recall	f1-score	support
0.0	0.89	0.93	0.91	4889
1.0	0.76	0.66	0.71	1623
accuracy			0.86	6512
macro avg	0.83	0.79	0.81	6512
weighted avg	0.86	0.86	0.86	6512

Accuracy Rate of Each Diagonal Element over Iteration

The following graph shows the progression in accuracy rate of each class over iterations.

- Each line is calculated by dividing the count of each diagonal element over the total population at validation steps captured by Debugger.
- This plot provides visibility into the progression in the class-level accuracy, and you can evaluate which class is not well classified. If a certain class is under-performing, this might be, for example, due to the imbalanced class problem. It is recommended to see the first section, The Distribution of True Labels of the Dataset, for more information about your data and suggested actions.

Receiver Operating Characteristic Curve

The *Receiver Operating Characteristic* curve shows the performance of binary classification.

- It measures the ratio of TPR over FPR, and the **Area Under Curve** (AUC) approaches to 1 as the model performance improves. The lowest AUC value is 0.5 (the area under the line of no-discrimination).
- If the AUC value is less than 0.6, it means that your model significantly under-performs. Consider improving your data, tuning the model parameters, pruning the trees, or trying other classifiers.

Distribution of Residuals at the Last Saved Step

The following histogram shows the distribution of residuals captured at the last step by Debugger.

The residual is calculated as **predicted minus true** in this visualization.

- For **regression**, the distribution should show a normal distribution centered at zero if the model is well trained.
- For **classification**, the distribution shows how the number of correct prediction counts (at the zero residual bin) dominates over the number of wrong prediction counts.