



AWS Machine Learning Blog

Using model attributes to track your training runs on Amazon SageMaker

by Sumit Thakur | on 02 AUG 2019 | in [Artificial Intelligence](#), [AWS Re:Invent](#), [SageMaker](#) | [Permalink](#) | [Comments](#) | [Share](#)

With a few clicks in the [Amazon SageMaker](#) console or a few one-line API calls, you can now quickly search, filter, and sort your machine learning (ML) experiments using key model attributes, such as hyperparameter values and accuracy metrics, to help you more quickly identify the best models for your use case and get to production faster. The [new Amazon SageMaker model tracking capability](#) is available through both the console and AWS SDKs in all [available](#) AWS Regions, at no additional charge.

Developing an ML model requires experimenting with different combinations of data, algorithm, and parameters—all the while evaluating the impact of small, incremental changes on performance and accuracy. This iterative fine-tuning exercise often leads to data explosion, with hundreds or sometimes thousands of experiments spread across many versions of a model.

Managing these experiments can significantly slow down the discovery of a solution. It also makes it tedious to trace back the lineage of a given model version so that the exact ingredients that went into brewing it can be identified. This adds unnecessary extra work to auditing and compliance verifications. The result is that new models don't move to production fast enough to provide better solutions to problems.

With Amazon SageMaker's new model tracking capabilities, you can now find the best models for your use case by searching on key model attributes—such as the algorithm used, hyperparameter values, and any custom tags. Using custom tags lets you find the models trained for a specific project or created by a specific data science team, helping you meaningfully categorize and catalog your work.

You can also rank and compare your model training attempts based on their performance metrics, such as training loss and validation accuracy. Do this right in the Amazon SageMaker console to more easily pick the best models for your use case. Finally, you can use the new model tracking capability to trace the lineage of a model all the way back to the dataset used in training and validating the model.

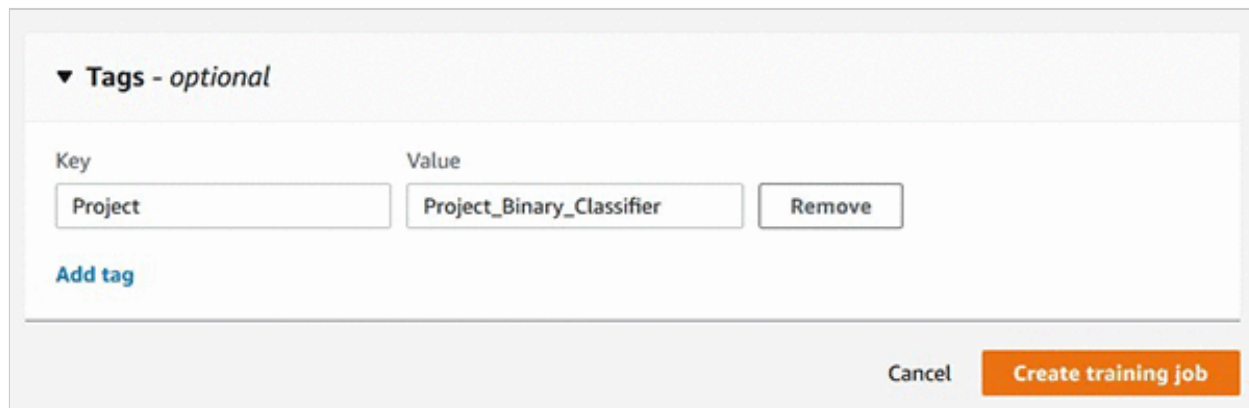
Now, I dive into the step-by-step experience of using this new capability.

Find and evaluate model training experiments

In this example, you train a simple binary classification model on the MNIST dataset using the [Amazon SageMaker Linear Learner algorithm](#). The model predicts whether a given image is of the digit 0 or otherwise. You tune the hyperparameters of the Linear Learner algorithm, such as `mini_batch_size`, while evaluating the `binary_classification_accuracy` metric that measures the accuracy of predictions made by the model. You can find the code for this example in the sample notebook in the [amazon-sagemaker-examples](#) GitHub repo.

Step 1: Set up the experiment tracking by choosing a unique label for tagging all of the model training runs

You can also add the tag using the Amazon SageMaker Python SDK API while you are creating a training job using the [Amazon SageMaker estimator](#).



Key	Value	
Project	Project_Binary_Classifier	Remove

[Add tag](#)

Cancel [Create training job](#)

You can also add the tag using the Amazon SageMaker Python SDK API while you are creating a training job using [SageMaker estimator](#).

```
linear_1 = sagemaker.estimator.Estimator(  
    linear_learner_container, role,  
    train_instance_count=1, train_instance_type = 'ml.c4.xlarge',  
    output_path=<you model output S3 path URI>,  
    tags=[{"Key": "Project", "Value": "Project_Binary_Classifier"}],  
    sagemaker_session=sess)
```

Step 2: Perform multiple model training runs with new hyperparameter settings

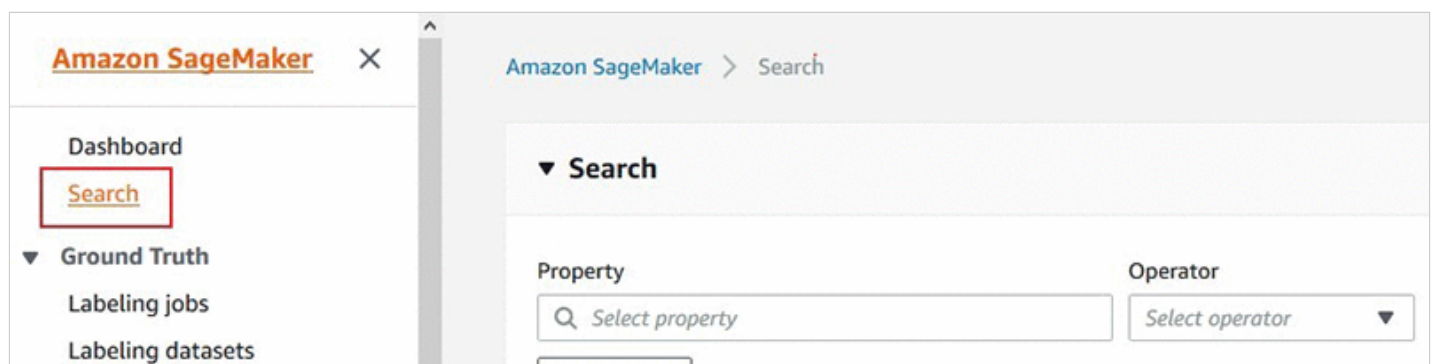
For demonstration purposes, try three different `batch_sizes` values of 100, 200, and 300. Here is some example code:

```
linear_1.set_hyperparameters(feature_dim=784,predictor_type='binary_classifier')
linear_1.fit({'train': <your training dataset S3 URI>})
```

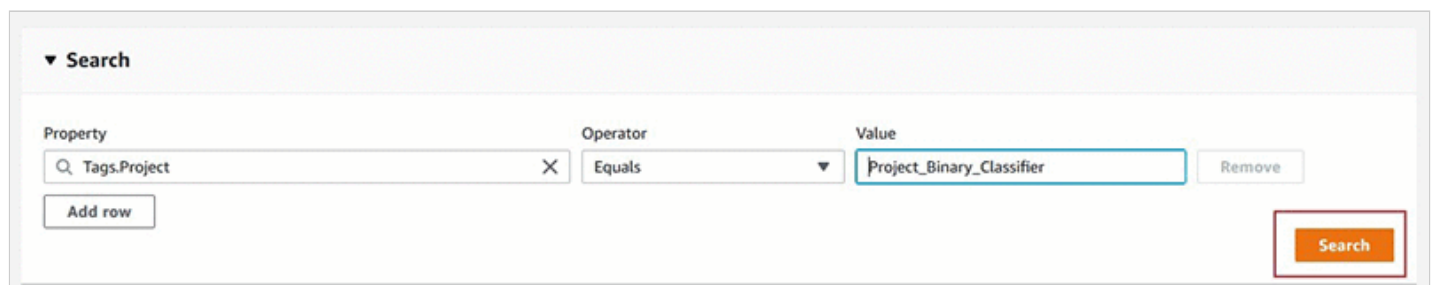
You are consistently tagging all three model training runs with the same unique label so you can track them under the same project. In the next step, I show how you can find and group all model training runs labeled with the “Project” tag.

Step 3: Find the relevant training runs for further evaluation

You can find the training runs on the Amazon SageMaker console.



You can search for the tag used in Steps 1 and 2.



This lists all labeled training runs in a table.

Results: Training jobs

Name	Algorithm	Data sources	Model artifact location	Hyperparameters
linear-learner- [redacted]	[redacted].dkr.ecr.us-west-2.amazonaws.com/linear-learner:1	train : s3://[redacted] test/sagemaker/DEMO-linear-mnist/train/recordio-pb-data	s3://[redacted]/sagemaker/DEMO-linear-mnist/output3/linear-learner-[redacted]/output/model.tar.gz	78
linear-learner- [redacted]	[redacted].dkr.ecr.us-west-2.amazonaws.com/linear-learner:1	train : s3://[redacted] test/sagemaker/DEMO-linear-mnist/train/recordio-pb-data	s3://[redacted]/sagemaker/DEMO-linear-mnist/output2/linear-learner-[redacted]/output/model.tar.gz	78
linear-learner- [redacted]	[redacted].dkr.ecr.us-west-2.amazonaws.com/linear-learner:1	train : s3://[redacted] test/sagemaker/DEMO-linear-mnist/train/recordio-pb-data	s3://[redacted]/sagemaker/DEMO-linear-mnist/output/linear-learner-[redacted]/output/model.tar.gz	78

You can also use the [AWS SDK API for Amazon SageMaker](#).

```
.....
search_params={
    "MaxResults": 10,
    "Resource": "TrainingJob",
    "SearchExpression": {
        "Filters": [{
            "Name": "Tags.Project",
            "Operator": "Equals",
            "Value": "Project_Binary_Classifier"
        }],
        "SortBy": "Metrics.train:binary_classification_accuracy",
        "SortOrder": "Descending"
    }
}
smclient = boto3.client(service_name='sagemaker')
results = smclient.search(**search_params)
```

While I have demonstrated searching by tags, you can search using any metadata for model training runs. This includes the learning algorithm used, training dataset URIs, and ranges of numerical values for hyperparameters and model training metrics.

Step 4: Sort on the objective performance metric of your choice to get the best model

The model training runs identified in Step 3 are presented to you in a table, with all of the hyperparameters and training metrics presented in sortable columns. Choose the column header to rank the training runs for

the objective performance metric of your choice, in this case, `binary_classification_accuracy`.

Results: Training jobs

HyperParameter mini_batch_size	HyperParameter predictor_type	Metric train:binary_f_beta	Metric train:progress	Metric train:objective_loss	Metric train:binary_classification_accuracy
300	binary_classifier	0.966639518737793	100	0.023814236745238304	0.9934399724006653
100	binary_classifier	0.9652714133262634	100	0.023504912853240967	0.993179976940155
200	binary_classifier	0.9647442698478699	100	0.023259807378053665	0.9930800199508667

You can also print the table inline in your Amazon SageMaker Jupyter notebooks. Here is some example code:

```
import pandas
headers=["Training Job Name", "Training Job Status", "Batch Size", "Binary Classification Accuracy"]
rows=[]
for result in results['Results']:
    trainingJob = result['TrainingJob']
    metrics = trainingJob['FinalMetricDataList']
    rows.append([trainingJob['TrainingJobName'],
                 trainingJob['TrainingJobStatus'],
                 trainingJob['HyperParameters']['mini_batch_size'],
                 metrics[[x['MetricName'] for x in metrics].index('train:binary_classification_accuracy')]['Value']
                ])
df = pandas.DataFrame(data=rows,columns=headers)
from IPython.display import display, HTML
display(HTML(df.to_html()))
```

As you can see in Step 3, you had already given the sort criteria in the `search()` API call for returning the results sorted on the metric of interest as follows:

```
"SortBy": "Metrics.train:binary_classification_accuracy"
"SortOrder": "Descending"
```

The previous example code parses the JSON response and presents the results in a leaderboard format, which looks like the following:

	Training Job Name	Training Job Status	Batch Size	Binary Classification Accuracy
0	linear-learner-2018-11-14-00-44-13-576	Completed	300	0.99344
1	linear-learner-2018-11-14-00-24-13-681	Completed	100	0.99318
2	linear-learner-2018-11-13-21-51-48-935	Completed	200	0.99308

Now that you have identified the best model—with `batch_size = 300`, and classification accuracy of 0.99344—you can now deploy this model to a live endpoint. The sample notebook has step-by-step instructions for deploying an Amazon SageMaker endpoint.

Tracing a model's lineage

Now I show an example of picking a prediction endpoint and quickly tracing back to the model training run used in creating the model in the first place.

Using single-click on the Amazon SageMaker console

In the left navigation pane of the Amazon SageMaker console, choose **Endpoints**, and select the relevant endpoint from the list of all your deployed endpoints. Scroll to **Endpoint Configuration Settings**, which lists all the model versions deployed at the endpoint. You see an additional hyperlink to the model training job that created that model in the first place.

Production variants			
Model name	Training job	Variant name	Instance type
linear-learner-2018-██████████	linear-learner-2018-██████████	AllTraffic	ml.m4.xlarge

Using the AWS SDK for Amazon SageMaker

You can also use few simple one-line API calls to quickly trace the lineage of a model.

```
#first get the endpoint config for the relevant endpoint
endpoint_config = smclient.describe_endpoint_config(EndpointConfigName=endpointI
```

```
#now get the model name for the model deployed at the endpoint.  
model_name = endpoint_config['ProductionVariants'][0]['ModelName']  
  
#now look up the S3 URI of the model artifacts  
model = smclient.describe_model(ModelName=model_name)  
modelURI = model['PrimaryContainer']['ModelDataUrl']  
  
#search for the training job that created the model artifacts at above S3 URI to  
search_params={  
    "MaxResults": 1,  
    "Resource": "TrainingJob",  
    "SearchExpression": {  
        "Filters": [  
            {  
                "Name": "ModelArtifacts.S3ModelArtifacts",  
                "Operator": "Equals"
```

Get started with more examples and developer support

Now that you have seen examples of how to efficiently manage the ML experimentation process and trace a model's lineage, you can try out a sample notebook in the [amazon-sagemaker-examples](#) GitHub repo. For more examples, see [our developer guide](#), or post your questions on the [Amazon SageMaker forum](#). Happy experimenting!

About the Author



Sumit Thakur is a Senior Product Manager for AWS Machine Learning Platforms where he loves working on products that make it easy for customers to get started with machine learning on cloud. In his spare time, he likes connecting with nature and watching sci-fi TV series.

This post was originally published November 28, 2018. Last updated August 2, 2019.

