Managing Machine Learning Workflow with Amazon SageMaker

New Services, Use Cases, and Best Practices

Provectus

Introduction

Over the past several years, artificial intelligence and machine learning have gone from hype buzzwords circulating in tech media to transformative forces upending entire industries and generating enormous value for those spearheading the transformation.

However, as AI and ML become almost mainstream, the requirements for technologies and tools used for building AI solutions get more complex and nuanced. Companies want to empower their IT teams to drive AI transformations and develop specific ML systems to deliver measurable results faster and more efficiently. They naturally expect AI/ML stack to rapidly evolve to meet their needs.

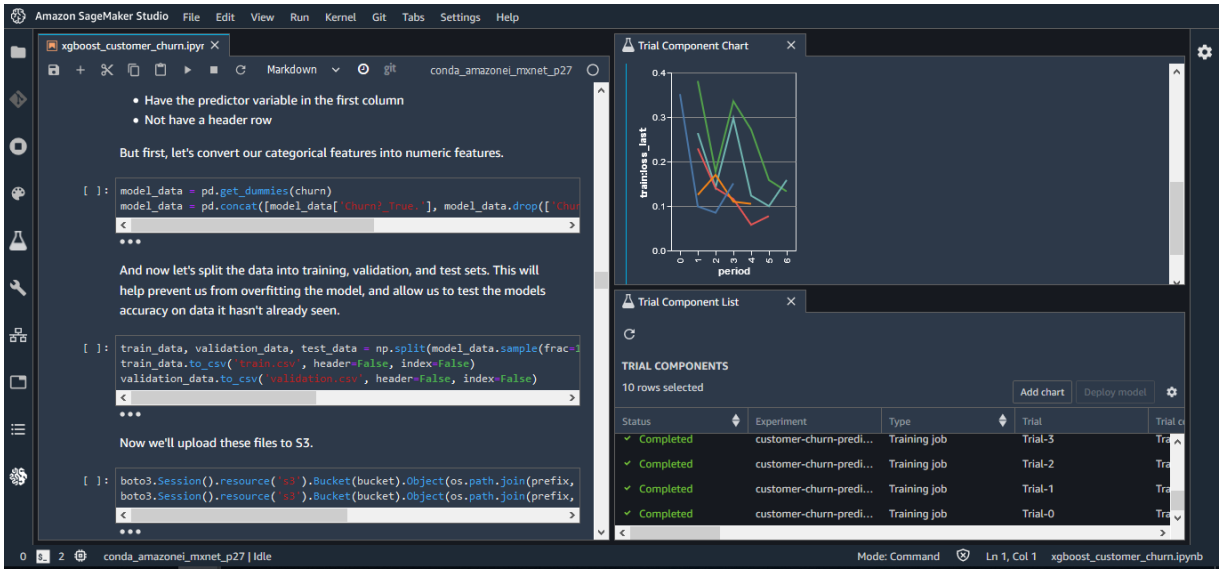
At the recent re:Invent conference, Amazon Web Services (AWS) revealed Amazon SageMaker Studio, a web-based IDE that tightly integrates all components of the ML ecosystem within a single interface, to allow for faster and more efficient code editing, debugging, tracking and tuning of training jobs.

In this whitepaper, we will explore Amazon SageMaker Studio and look into its newly integrated services, including Amazon SageMaker Experiments, Amazon SageMaker Autopilot, Amazon SageMaker Debugger, Amazon SageMaker Model Monitor, as well as Amazon SageMaker Notebooks.

Amazon SageMaker Studio

> *SageMaker Studio is JupyterLab on steroids, which helps developers write code, track experiments, visualize data, and perform debugging and monitoring all within a single, integrated visual interface.*

Amazon SageMaker Studio is a web-based integrated development environment (IDE) for machine learning, designed to build, train, debug, deploy and monitor ML models. It provides all the tools needed to efficiently take ML models from experimentation to production in a single unified visual interface.



ML model being built in SageMaker Studio

Use Cases

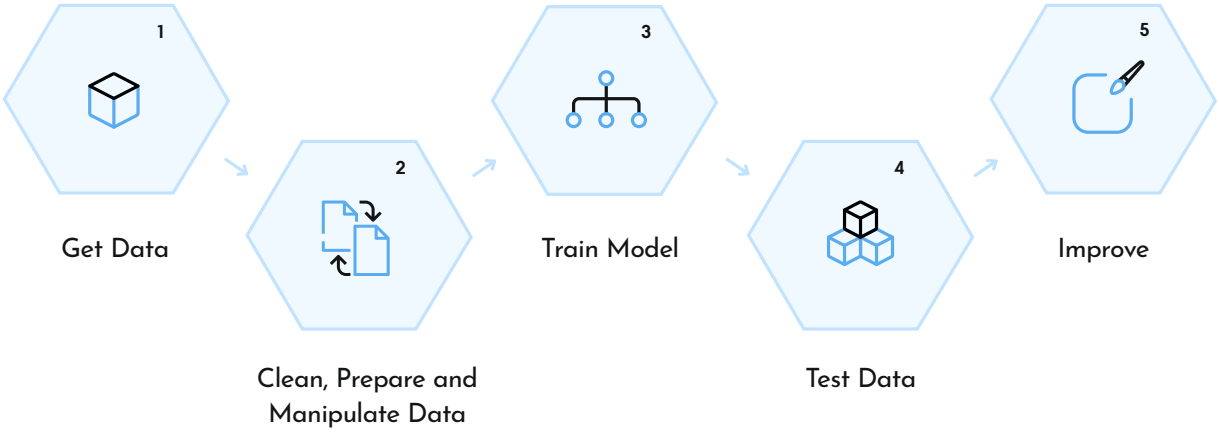
Amazon SageMaker Studio enables ML engineers to:

Write and execute code in Jupyter notebooks	Build and train machine learning models
Deploy models and monitor data drift to check models' performance	Track and debug machine learning experiments

An all-around tool, Studio enables engineers to work and navigate through the following stages of machine learning workflow:

- Data preparation
- Model development and debugging

- Model training, tuning and evaluation
- Experiment management
- Model deployment
- Model monitoring



ML Workflow – Fully accessible and manageable with SageMaker Studio

Amazon SageMaker Studio allows for deployment of ML models as SageMaker endpoints, and also for monitoring models to see how production data is different compared to those used for training.

Studio features notebook instances by default. When launching an EC2 instance, Jupyter Notebook or JupyterLab are automatically pulled and configured, based on settings, to shortcut to model experimentation. Let's look at Studio's specifics, features and services in more detail.

Onboarding & Access

Amazon SageMaker Studio allows for two modes of authentication:

- **AWS Single Sign-On (SSO)** – For those who have an AWS enterprise account and have exported client accounts
- **AWS Identity and Access Management (IAM)** – Access is similar to other AWS tools and products

Access to Amazon SageMaker Studio can be provided without exposing AWS Console to users. They sign in via a link using a login and password sent to a specified email address.

Features

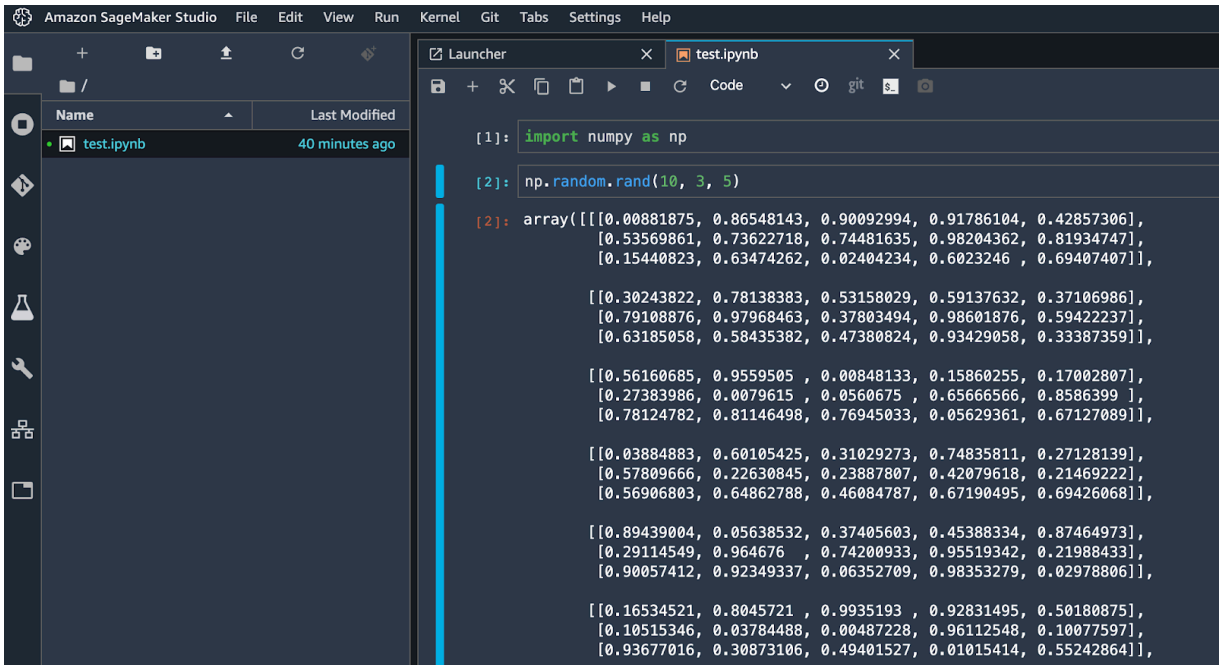
SageMaker Studio Notebooks, SageMaker Experiments, SageMaker Autopilot, SageMaker Debugger, and SageMaker Model Monitor are at the core of SageMaker Studio IDE.

AWS ML Stack

Notebooks

By default, Amazon SageMaker Studio provides ML engineers with next-generation notebooks, which are more effective and efficient than instance-based notebooks.

The new notebooks have near-instant startup times and can be easily shared among users. To share a notebook, simply take a snapshot of the notebook and share it with another user. This overrides the requirement to share the notebook itself. The snapshot can include cell outputs and GitHub details, to accelerate and simplify collaboration.

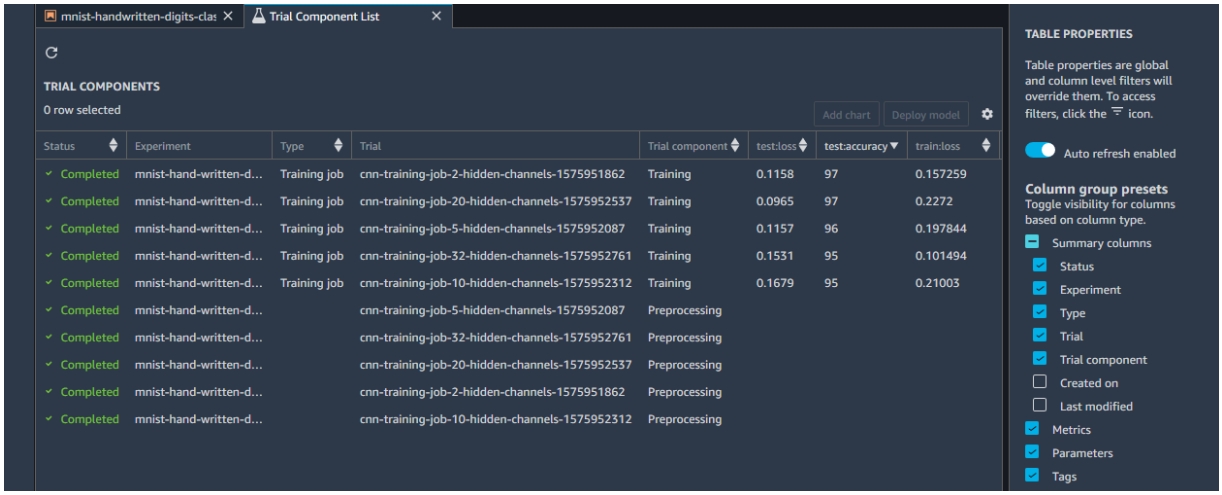


SageMaker Notebooks

Every user receives specific EFS storage that is independent of a particular instance. New notebooks can be accessed via AWS SSO.

Experiments

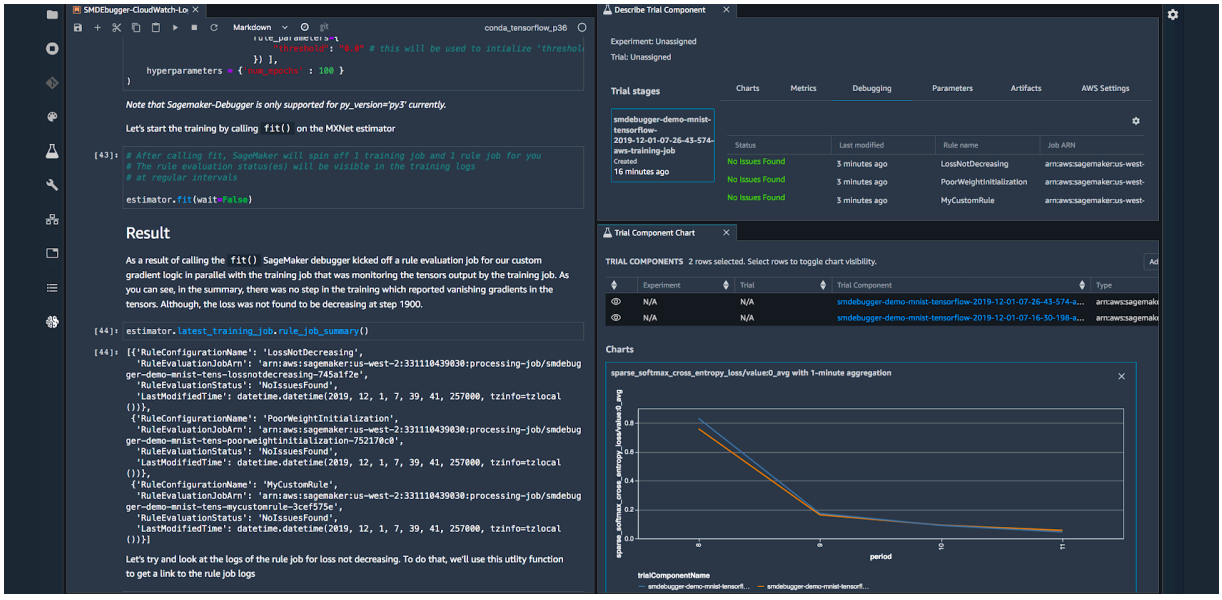
Amazon SageMaker Studio allows users to organize, track, compare and evaluate machine learning (ML) experiments and model versions. This makes it easier to deal with hundreds of thousands of jobs: keep track of metrics, group jobs by experiment, compare jobs in the same experiment or across experiments, query past jobs, and more.



SageMaker Experiments

Debugger

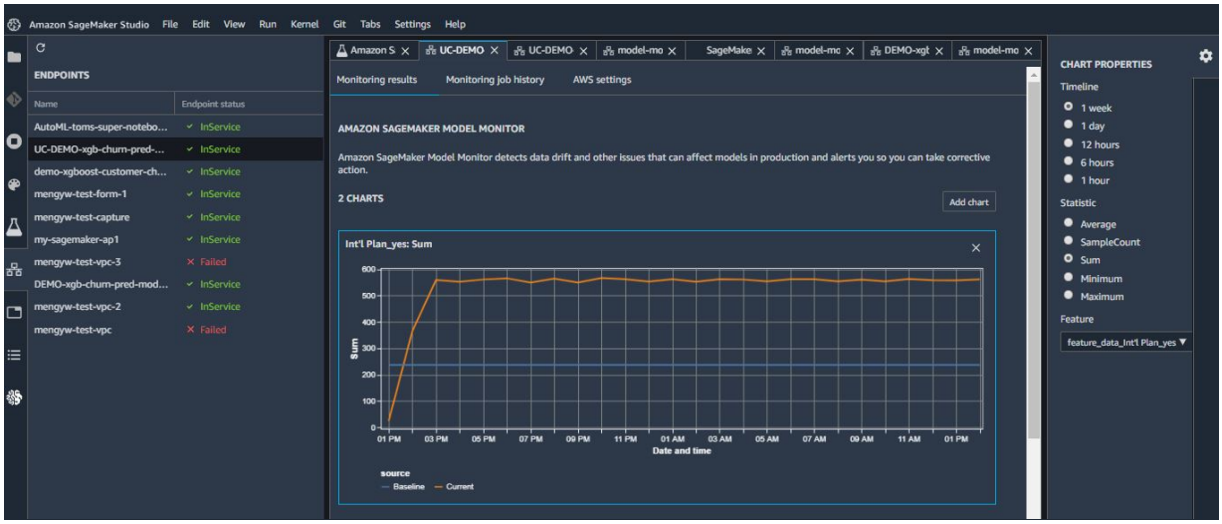
The overwhelming majority of failed training jobs is caused by inappropriate initialization of parameters, a poor combination of hyperparameters, design issues in code, and other issues. Debugger makes it possible to automatically identify (and make visible) complex issues arising in machine learning training jobs, so those issues can be proactively addressed and a robust training process maintained.



SageMaker Debugger

Model Monitor

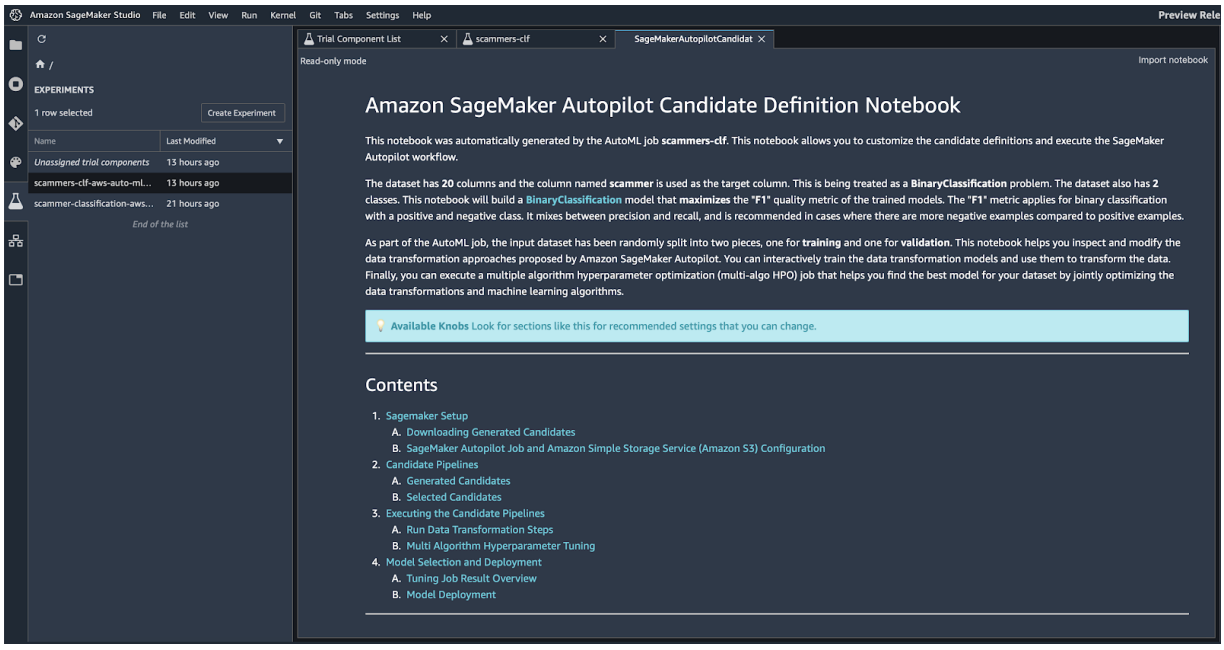
Model Monitor automatically monitors ML models in production and flags data quality issues as they appear, helping to combat ‘data drift’ and maintaining high prediction accuracy without having to ‘attach’ a bundle of data capture, statistical analysis, rule management and alert tools.



SageMaker Model Monitor

Autopilot

An autoML component of Amazon SageMaker, Autopilot automatically trains and tunes ML models for classification or regression, based on customer data. It allows engineers to maintain visibility and full control over the process.



SageMaker Autopilot

Brief Summary

ML stack is available in one single service	Shareable notebooks simplify collaboration	Based on JupyterLab, with all its benefits

Now that we have taken a broad look at Amazon SageMaker Studio's features, let's discuss them in more detail, starting with Amazon SageMaker Experiments.

Amazon SageMaker Experiments

Amazon SageMaker Experiments was announced in December 2019. According to AWS, SageMaker Experiments is the ultimate tool for engineers, allowing them to easily organize, track, and compare machine learning training jobs.

Before Amazon Sagemaker Experiments

Before going deep into SageMaker Experiments, let's review how engineers previously supplemented some of its basic features with other tools like TensorFlow.

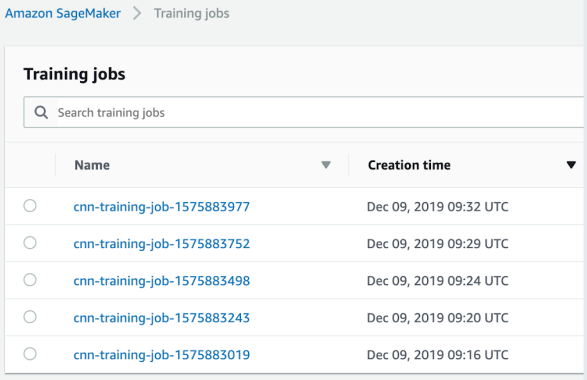
Engineers organize, track, and compare ML training jobs in TensorFlow's Tensorboard. There they can log training metrics (and other scalars), examine the execution graph, visualize hyperparameter tuning, evaluate the model with fairness indicators, and more.

Unfortunately, Tensorboard has several critical flaws:

- Represents only the training segment of the entire ML workflow
- Does not allow for tracking of parameters used to make a run
- Complex when it comes to comparing various runs
- Supports only TensorFlow and PyTorch

It also makes sense to look at Amazon SageMaker before Experiments was introduced. Back then, developers simply had a training script backed with a managing script, which would spin up the training process. After finishing a few training jobs in this fashion, it became clear that comparing training jobs would be difficult. One had to keep track of training processes, but there was no clear or convenient way to do so. By adding AWS Sagemaker Search, Amazon addressed the issue of searching for a specific training job with complex queries, but the trial comparison and experiment organization were still out of scope.

```
estimator = TensorFlow(
    role=role,
    sagemaker_session=sagemaker_session,
    train_instance_count=train_instance_count,
    train_instance_type=train_instance_type,
    hyperparameters=kwargs,
    output_path=output_path,
    model_dir=model_dir,
    entry_point="main.py",
    dependencies=["core"],
    py_version="py3",
    framework_version="1.13",
    metric_definitions=[
        {'Name': 'loss', 'Regex': r"'loss':\s([\d.]+)"},
    ],
    distributions={
        'mpi': {
            'enabled': True,
            'processes_per_host': 1,
            'custom_mpi_options': "--NCCL_DEBUG INFO",
        }
    },
    code_location=code_location,
)
```



Training jobs at Amazon SageMaker before 2019 update

After Amazon SageMaker Experiments

In their announcement at re:Invent, AWS made it clear that their primary goal is "to make it as simple as possible to create experiments, populate them with trials, and run analytics across trials and experiments."

Given that, the following features and functions were put on the table:

- Seamless integration into the existing ML workflow
- Effective tracking and management of experiments
- Decomposition of a monolith workflow into multiple steps

Core Concepts

- **Trial Component** — A single step of the machine learning workflow. For instance, data cleaning, model training, model evaluation, etc.
- **Trial** — A multi-step machine learning workflow, where each step is described by an individual trial component.
- **Experiment** — A collection of related trials. Add various trials to compare to an experiment.

Tracked Items

- Parameters
- Inputs
- Outputs
- Artifacts
- Metrics

How It Works

First define an experiment in code using Amazon SageMaker SDK. This is typically done in the management script, where all instructions for spinning up a training process are declared:

```
experiment = Experiment.create(
    experiment_name="mnist-digits-classification",
    sagemaker_boto_client=sm)
```

Once the experiment has been created, define a Tracker object, which is used to log all information related to the trial component.

```
with Tracker.create(display_name="Preprocessing", sagemaker_boto_client=sm) as tracker:
    tracker.log_input(name="mnist-dataset", media_type="s3/uri", value=inputs)
    tracker.log_parameters({
    "normalization_mean": 0.1307,
    "normalization_std": 0.3081,
})
```

Tracker automatically creates a trial component for which all information will be logged. In this example we define a “Preprocessing” trial component, where we log the input dataset and log parameters applied during dataset transformation. This trial component is not yet associated with any trials. Let’s create one.

```
cnn_trial = Trial.create(
    trial_name=trial_name,
    experiment_name=experiment.experiment_name,
    sagemaker_boto_client=sm
).add_trial_component(tracker.trial_component)
```

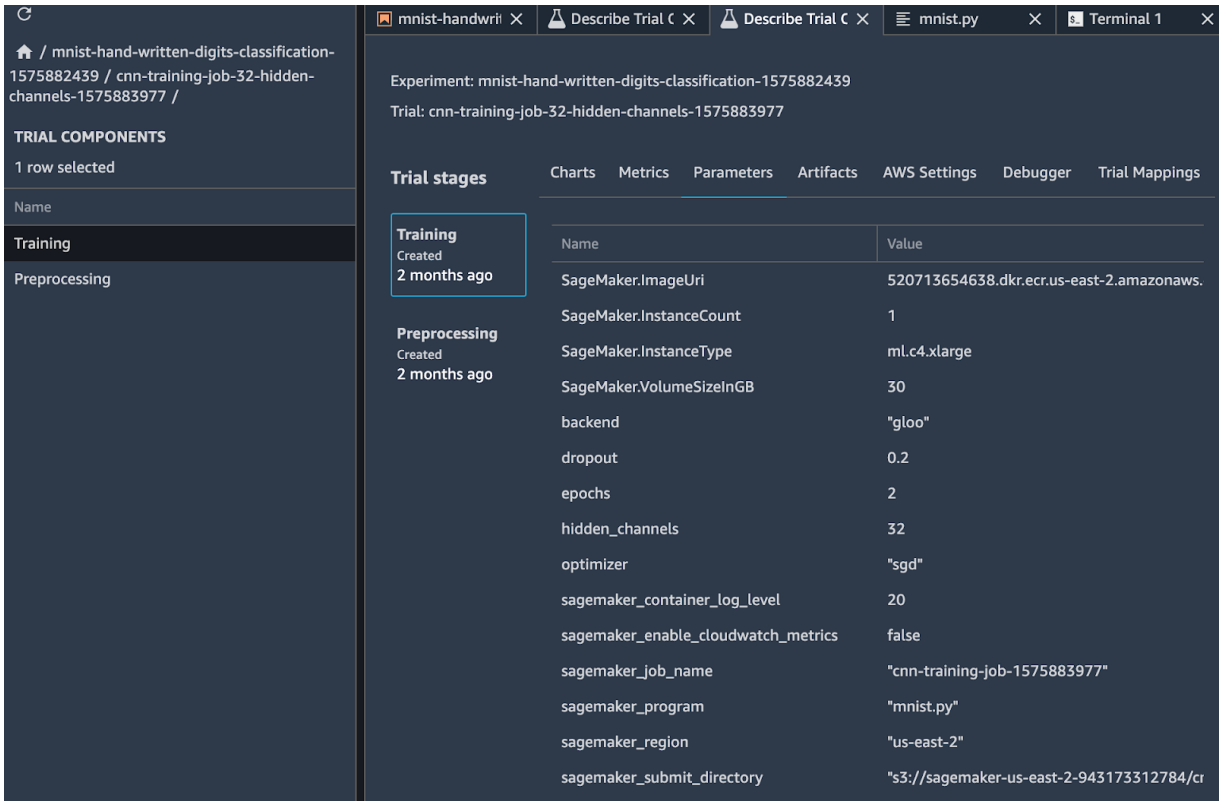
Let’s demonstrate how to use Amazon SageMaker Autopilot by launching a scammer classification experiment in Amazon SageMaker Studio.

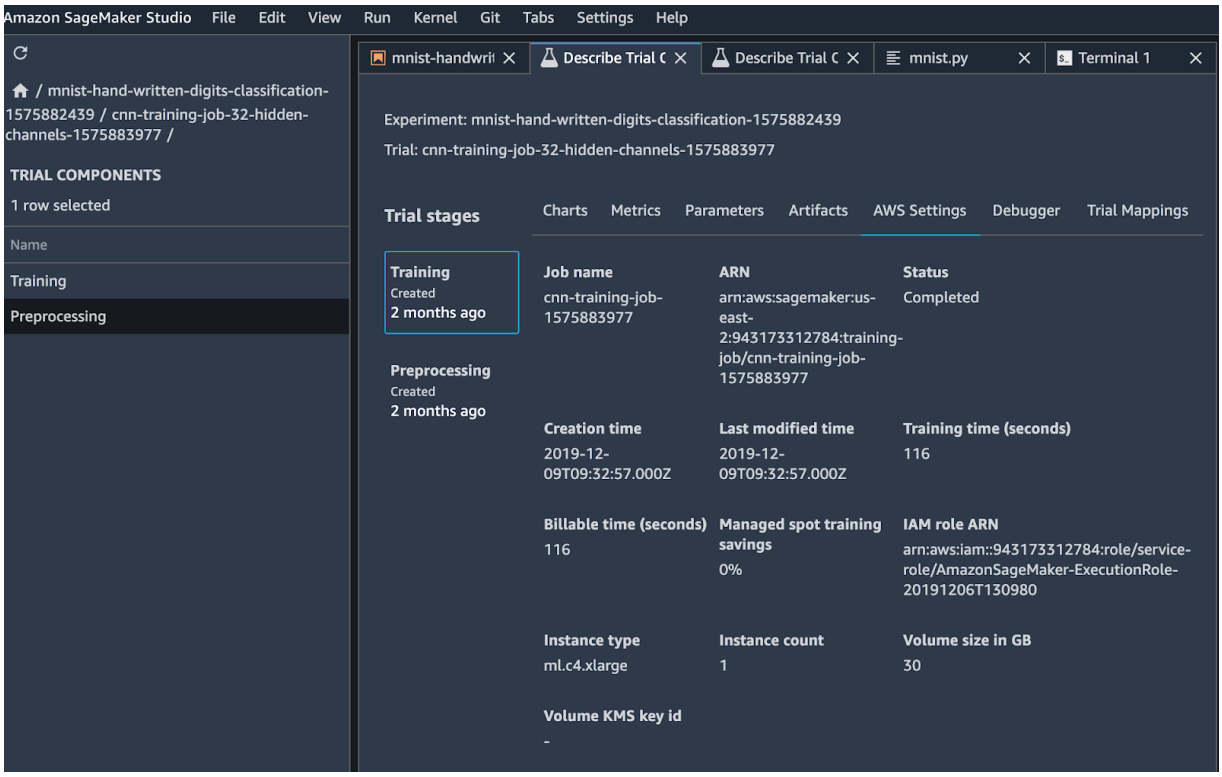
```
estimator.fit(
    inputs={'training': inputs},
    job_name=cnn_training_job_name,
    experiment_config={
        "ExperimentName": experiment.experiment_name,
        "TrialName": cnn_trial.trial_name,
        "TrialComponentDisplayName": "Training",
    }
)
```

How to Analyze Experiments

Amazon SageMaker Studio

Analyzing experiments in Amazon SageMaker Studio is easy. All experiments can be visualized in real time using predefined widgets accessible in the experiments tab. There it is possible to take a closer look at trials, trial components, metrics, parameters, etc., and also to create charts.





Amazon SageMaker SDK

Alternatively, all information about the experiment can be exported to a Pandas DataFrame and create graphs and charts there, to compare different experiments and trials.

```
>>> from sagemaker.analytics import ExperimentAnalytics
>>> trial_component_analytics = ExperimentAnalytics(
...     sagemaker_session=sess,
...     experiment_name=some_experiment.experiment_name
... )
>>> analytic_table = trial_component_analytics.dataframe()
```

```
>>> for col in analytic_table.columns:
...     print(col)

TrialComponentName
DisplayName
SourceArn
dropout
epochs
hidden_channels
optimizer
test:accuracy - Min
test:accuracy - Max
test:accuracy - Avg
test:accuracy - StdDev
test:accuracy - Last
test:accuracy - Count
```

Amazon SageMaker Experiments: Downsides & Weaknesses

- **Does not allow building of complex DAGs** — It supports only sequential execution. Several stages cannot be run in parallel; different execution paths cannot be declared, which is a critical limitation.
- **Lack of instruments for configuring robust pipelines** — Currently, Experiments does not allow to define logic for timeouts and retries.
- **Out of the box, UI is available only in Amazon SageMaker Studio** — The UI has numerous bugs, and it needs to be enhanced from a UX perspective.

Open source alternatives: Metaflow, Kubeflow, MLFlow.

AWS SageMaker Debugger

Amazon SageMaker Debugger was announced in December 2019 as a new capability of Amazon SageMaker. It automatically identifies complex issues developing in machine learning training jobs, and comes in a single package with Notebooks, Experiments, Autopilot and Model Monitor.

Because the training process can be corrupted by a variety of obscure issues, from inappropriate initialization or a poor combination of parameters to a design issue in code, data scientists need an automated tool to reduce time wasted fixing errors, **thus running experiments more effectively and cost-efficiently.**

How to Debug?

Code

- Unit tests
- Logging
- Peer review

Experiments

- Unit tests
- Logging
- Peer review

- Asserts for model parameters
- Tracking loss curves / metrics during training
- Checking model outputs

Amazon SageMaker Debugger allows users to monitor, track, and log specific training metrics and loss curves, and to check model outputs. In comparison with Jupyter Notebooks, where one needs to log all metrics, loss curves, and other details on their own in a separate tab, AWS offers a much more efficient and scalable solution.

```
# Accuracy report
print('Accuracy:', sess.run(accuracy, feed_dict={X: mnist.test.images, Y: mnist.test.labels}))
```

```
cost : 5.745171: 100%|██████████| 550/550 [00:01<00:00, 319.00it/s]
cost : 1.780057: 100%|██████████| 550/550 [00:01<00:00, 309.36it/s]
cost : 1.122779: 100%|██████████| 550/550 [00:01<00:00, 317.76it/s]
cost : 0.872012: 100%|██████████| 550/550 [00:01<00:00, 334.38it/s]
cost : 0.738203: 100%|██████████| 550/550 [00:01<00:00, 303.13it/s]
cost : 0.654729: 100%|██████████| 550/550 [00:01<00:00, 303.78it/s]
cost : 0.596024: 100%|██████████| 550/550 [00:01<00:00, 310.52it/s]
cost : 0.552217: 100%|██████████| 550/550 [00:01<00:00, 308.68it/s]
cost : 0.518255: 100%|██████████| 550/550 [00:01<00:00, 307.12it/s]
cost : 0.491113: 100%|██████████| 550/550 [00:01<00:00, 306.57it/s]
cost : 0.084112:  16%|▌         | 87/550 [00:00<00:02, 226.18it/s]
```

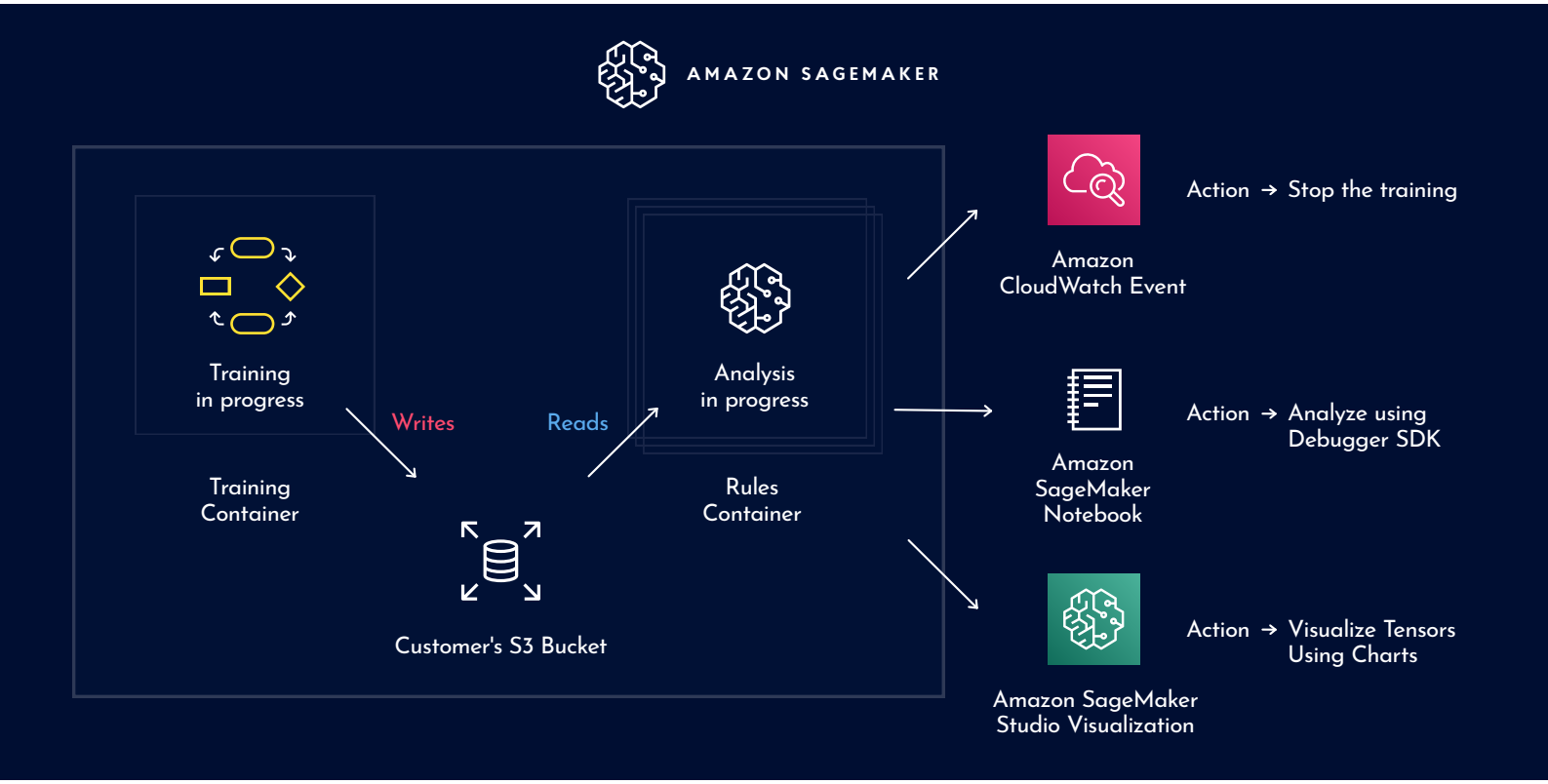
```
index = []
ori = []
pred = []

labels = sess.run(tf.argmax(mnist.test.labels, 1))
predictions = sess.run(tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images})

for i in range(0,mnist.test.num_examples):
    if labels[i] != predictions[i]:
        index.append(i)
        ori.append(labels[i])
        pred.append(predictions[i])
```

Logging in Jupyter Notebooks – All details will be lost when launching the experiment anew

Amazon SageMaker Debugger combines the best of the two worlds – Logging and Statistics – to enable data scientist and ML engineers to:

- Track various model parameters and metrics during training
- Upload parameters to Amazon S3 for storage and further use
- Confirm that values satisfy predetermined rules (e.g. vanishing gradient, class imbalance)
- Explore and visualize tracked values in Amazon SageMaker Studio

How Amazon SageMaker Debugger Works



Amazon SageMaker Debugger – How it works

In Amazon SageMaker, a machine learning model is trained in a separate training container. Debugger embeds into the training process to write logs to Amazon S3. In the meantime, other containers are used to monitor data that is generated and stored in Amazon S3. If any training rules are violated, alerts are pushed to Amazon CloudWatch and the training is terminated. The logic behind this dictates that the experiment should stop immediately when any one of the rules has been violated, since it no longer makes sense to continue the training. Here is where we can process and analyze alerts to identify specific activities leading to errors in training using Amazon SageMaker Notebook. Data stored in Amazon S3 can be visualized using Amazon SageMaker Studio Visualization.

Debugger API

- **Trial** – An aggregation of metrics for one training job, either in Amazon S3 or locally, that is different from Experiment trials.
- **Tensor** – A type of a saved parameter that is being tracked in the training process (e.g. loss)
- **Hook** – A callback for saving parameters. It gets embedded into the training process to push them to Amazon S3.
- **Rule** – A predicate that verifies if a constraint is satisfied or not.

Debugger for SageMaker SDK Model

```
import sagemaker as sm
from sagemaker.debugger import rule_configs, Rule, CollectionConfig

# Choose a built-in rule to monitor your training job
rule = Rule.sagemaker(
    rule_configs.exploding_tensor(),
    # configure your rule if applicable
    rule_parameters={"tensor_regex": ".*"},
    # specify collections to save for processing your rule
    collections_to_save=[
        CollectionConfig(name="weights"),
        CollectionConfig(name="losses"),
    ],
)

# Pass the rule to the estimator
sagemaker_simple_estimator = sm.tensorflow.TensorFlow(
    entry_point="script.py",
    role=sm.get_execution_role(),
    framework_version="1.15",
    py_version="py3",
    # argument for smdebug below
    rules=[rule],
)

sagemaker_simple_estimator.fit()
tensors_path = sagemaker_simple_estimator.latest_job_debugger_artifacts_path()

import smdebug.trials as smd
trial = smd.create_trial(out_dir=tensors_path)
```

```
estimator = Estimator(
    ...
    rules = Rules.custom(
        name='VGRule',
        image_uri='864354269164.dkr.ecr.us-east-1.amazonaws.com/s
        instance_type='ml.t3.medium', # instance type to run the
        source='rules/vanishing_gradient_rule.py', # path to the
        rule_to_invoke='VanishingGradientRule', # name of the cla
        volume_size_in_gb=30, # EBS volume size required to be at
        collections_to_save=[CollectionConfig("gradients")], # co
        rule_parameters={
                "threshold": "20.0" # this will be used to initia
        }
)
```

How to embed Debugger into a SageMaker model

Sagemaker Debugger supports the ingestion of rules and hooks to the training jobs running on AWS Sagemaker with only a few adjustments to the training code.

Debugger for Local Models

```
import smdebug.tensorflow as smd
hook = smd.KerasHook(out_dir='~/smd_outputs/')

model = tf.keras.models.Sequential([ ... ])
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
)

# Add the hook as a callback
model.fit(x_train, y_train, epochs=2, callbacks=[hook])
model.evaluate(x_test, y_test, callbacks=[hook])

# Create a trial to inspect the saved tensors
trial = smd.create_trial(out_dir='~/smd_outputs/')
```

```
from smdebug.rules import invoke_rule
from smdebug.trials import create_trial

trial = create_trial('s3://smdebug-dev-test/mnist-job/')
rule_obj = VanishingGradientRule(trial, threshold=0.0001)
invoke_rule(rule_obj, start_step=0, end_step=None)
```

How to use Debugger locally, on Keras

Sagemaker Debugger also supports ingestions of hooks in local mode. This allows to develop and train models locally while still leveraging all the power Debugger provides with Amazon Sagemaker Studio.

Rules

- **Vanishing gradients** – As more layers with specific activation functions are added to neural networks, gradients of the loss function approach zero. As it gets harder to train the network,

Debugger starts sending alerts.

- **Overfitting** – The machine learning model learns the noise in the training data to the point where it affects the performance of the model on new data.
- **Poor weight initialization** – As layer activation outputs explode or entirely vanish in the network, loss gradients become either too large or too small to flow backwards beneficially, thus preventing the network from converging.
- **Saturated activations** – Neurons output values close to the asymptotic ends of the bounded activation function, thus damaging both the information capacity and the learning ability of the network.
- **Overpruned trees** – Disproportionately large sections of the decision tree get removed, limiting the capacity to accurately classify instances, underfitting, and inaccurate performance.

How to Use Amazon SageMaker Debugger

1. **Save Tensors** – Add hooks as parameters to SageMaker API Estimator, or add them to the code of the training pipeline
2. **Apply Rules** – Add rules as parameters to invoke on AWS, or manually invoke them locally
3. **Handle CloudWatch Alerts** – Keep track of alerts and process all notifications, training stops and other alerts.

Integrations: Keras, TensorFlow, PyTorch, MXNet.

Amazon SageMaker Autopilot

AWS envisions Amazon SageMaker Autopilot as an autoML tool for creating machine learning models with full visibility and control. Using a single API call, or with a few clicks, engineers can automatically train and tune machine learning models for classification and regression.

According to AWS, Amazon SageMaker Autopilot should be able to:

- Accurately inspect the dataset
- Identify the optimal combination of data preprocessing steps, machine learning algorithms, and hyperparameters
- Train an inference pipeline that can be deployed either on a real-time endpoint or for batch processing

- Generate Python code to demonstrate how data has been preprocessed, for better explainability and further reuse, or for manual tuning

With Autopilot, data scientists and ML engineers receive a fully-managed infrastructure to create and deploy ML models automatically. This includes auto data cleaning and preprocessing, auto algorithm selection, auto hyperparameter optimization, auto instance and cluster size selection, on top of virtually zero use of code.

Optimized time and resources spent on building ML models	Minimal knowledge of the specifics of machine learning

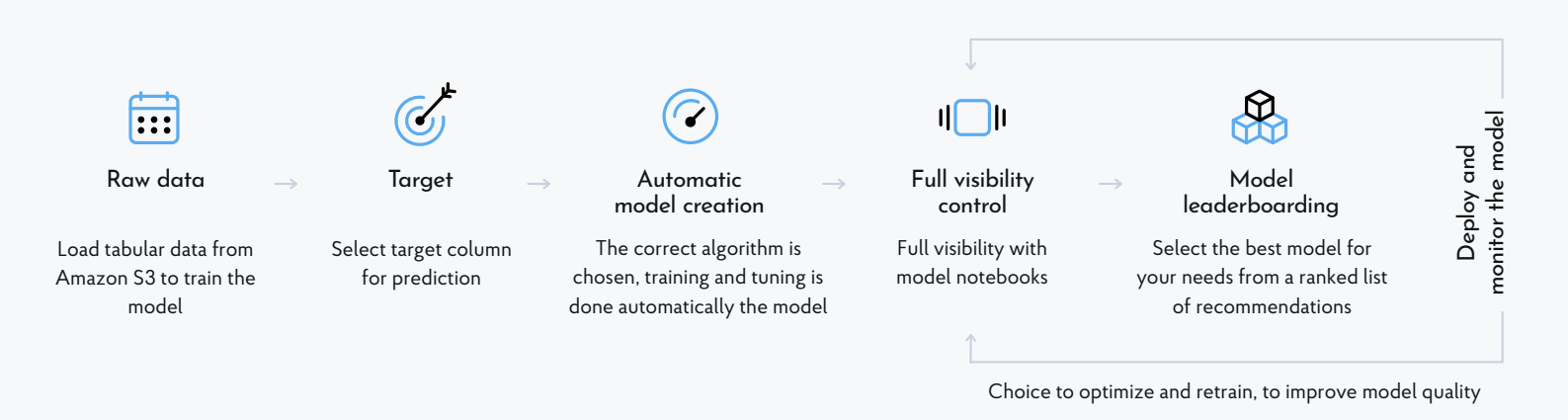
However, Amazon SageMaker Autopilot also has a few major restrictions:

- Input data must only be in tabular format (non-structured data is not supported)
- Only three types of models are supported (linear regression, binary classification, multi-class classification)
- Limited set of combinations of data preprocessors and algorithms (custom combinations are not allowed)
- Limited set of metrics (additional metrics cannot be added)

Given all that, Amazon SageMaker Autopilot is hardly the most flexible tool in Amazon SageMaker Studio, but it is well-suited for standardized ML tasks and activities.

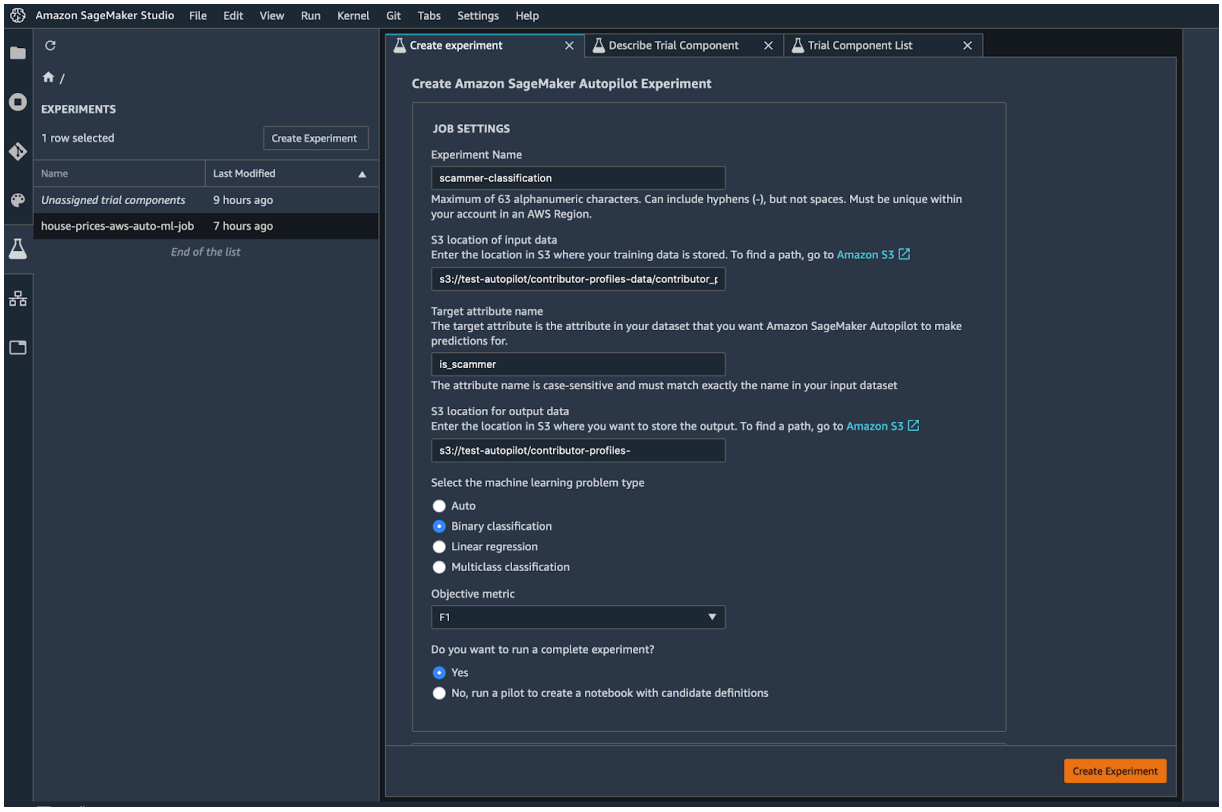
How to Use Amazon SageMaker Autopilot

To launch an experiment using Amazon SageMaker Autopilot, access Amazon SageMaker Studio, define the problem to solve, upload tabular data in Amazon S3 bucket, define target variable(s) to predict, define the metric(s) to evaluate the model's quality, and launch the experiment.

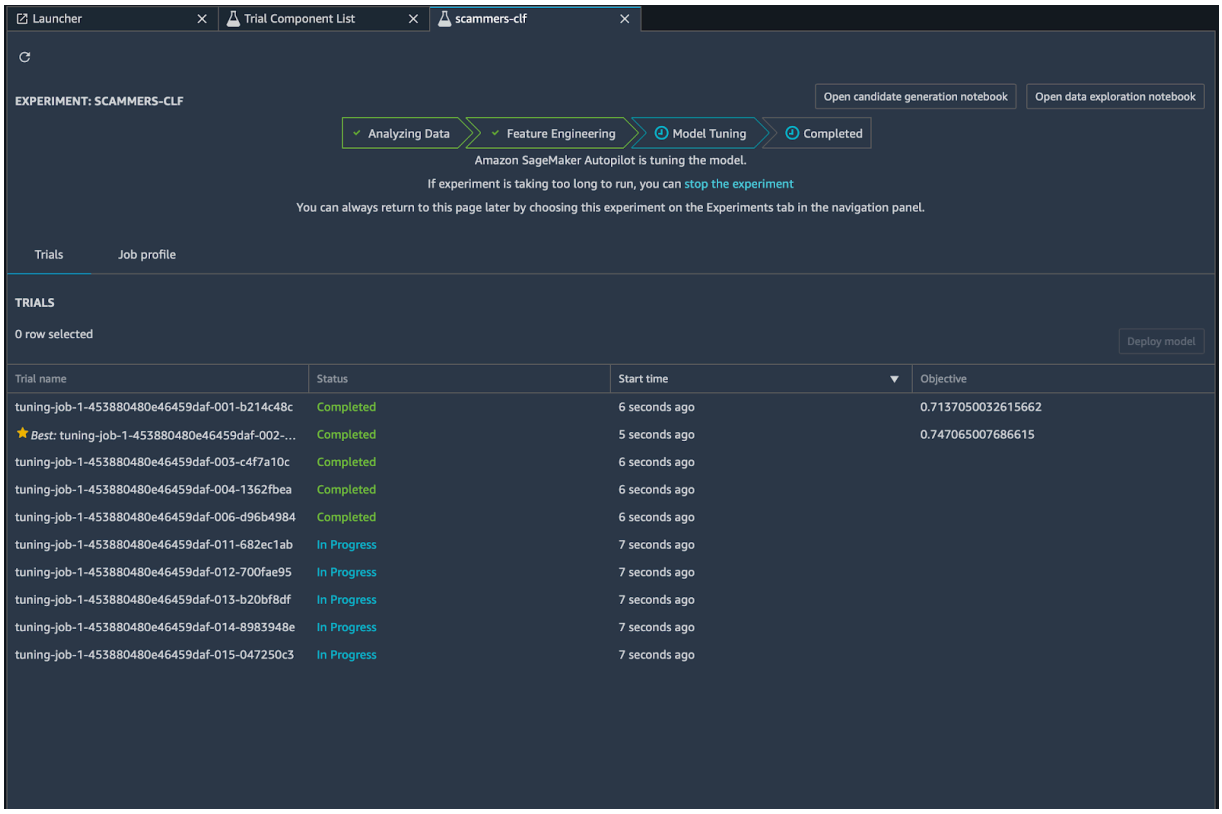


Amazon SageMaker Autopilot – How it works

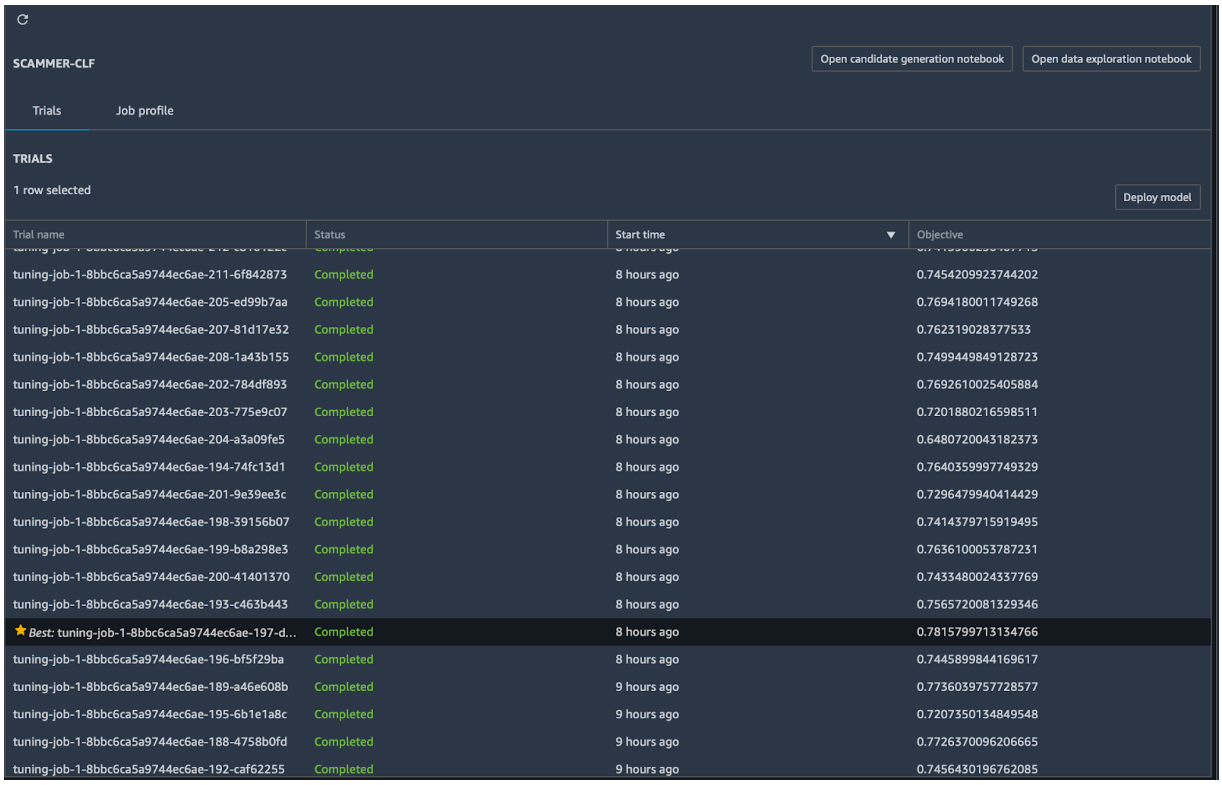
Let's demonstrate how to use Amazon SageMaker Autopilot by launching a scammer classification experiment in Amazon SageMaker Studio.



Creating an experiment: S3 location (input & output), Target Attribute, Problem type, Metric



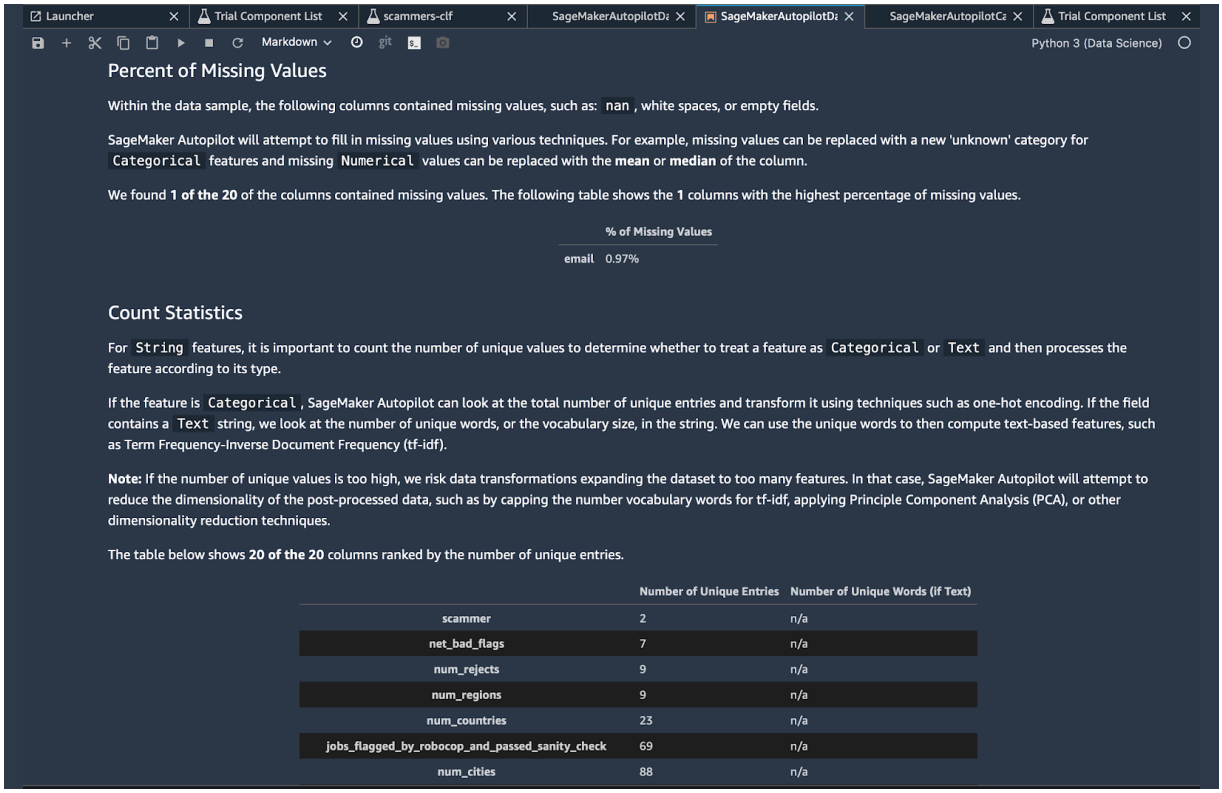
Monitoring the process: Data analysis, Feature Engineering, Model Tuning



Selecting and deploying the best model from the list of completed trials

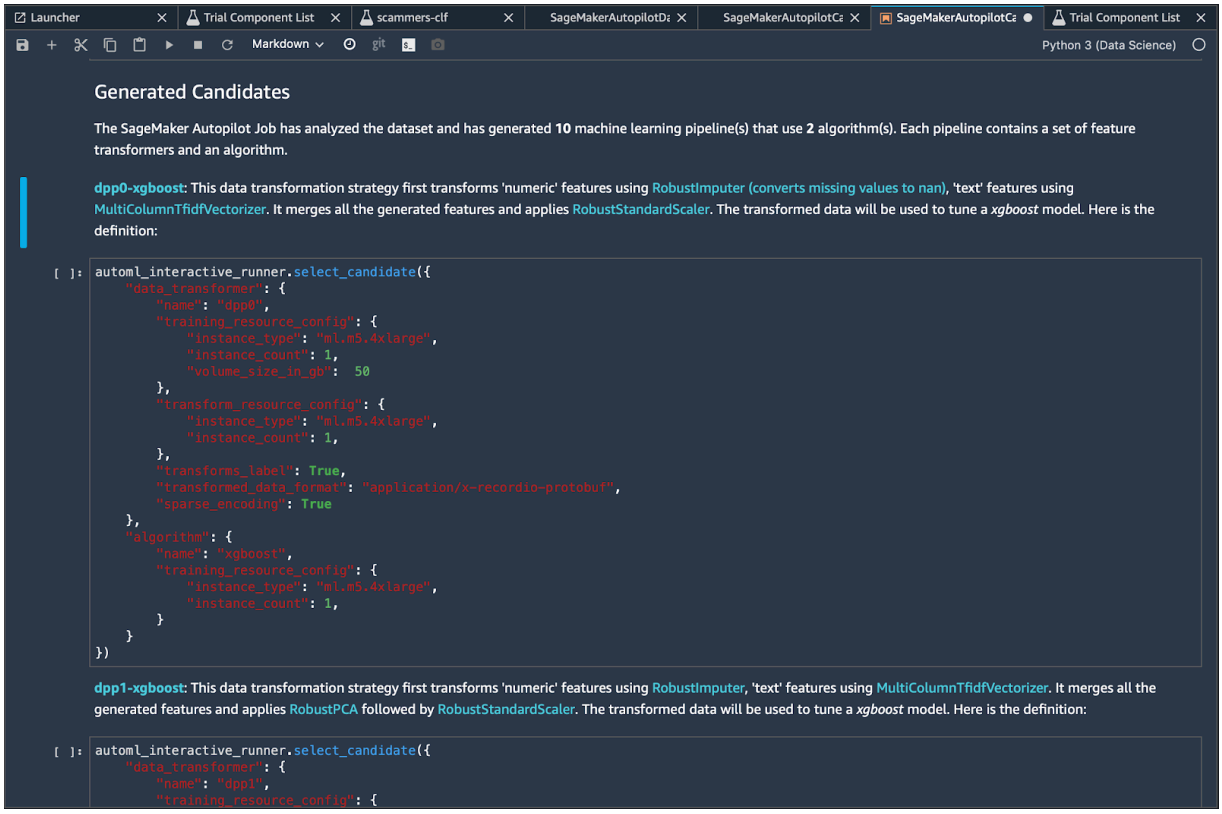
Once the experiment is completed, Amazon SageMaker automatically generates a few output items:

- **Data Exploration Notebook** featuring data preprocessing steps, to help identify issues in the dataset



Example of a data exploration notebook

- **Candidate Generation Notebook** featuring suggested preprocessing method, suggested algorithm, and suggested hyperparameter ranges



Example of a candidate generation notebook

- Model artifacts, processed data, and parameters for each candidate, all of which are uploaded to Amazon S3 bucket

Based on the examples above, it is evident that Amazon SageMaker Autopilot provides deep insight into how a specific machine learning model is built, thus simplifying ML model explainability and interpretability.

Alternatives: Cloud AutoML by Google, AutoKeras, H2O AutoML, firefly.ai, Auto-WEKA

Amazon SageMaker Model Monitor

Amazon SageMaker Model Monitor is a new capability of Amazon SageMaker that automatically monitors machine learning models and notifies developers if any issues arise.

The tool addresses the problem of data drift in non-experimental data sets; it compares training data with production data to capture "changing" attributes that shape assumptions and decisions made during the training of a model, to maintain the accuracy of predictions.

Though data scientists and ML engineers know how to handle data drift, building tools to capture data, compare it to the training set(s), define rules, detect drift, send alerts, etc. takes time and

resources. To minimize this heavy lifting, AWS came up with Amazon SageMaker Model Monitor.

Automatically monitors quality of the production data	Pushes alerts when issues in production data are detected

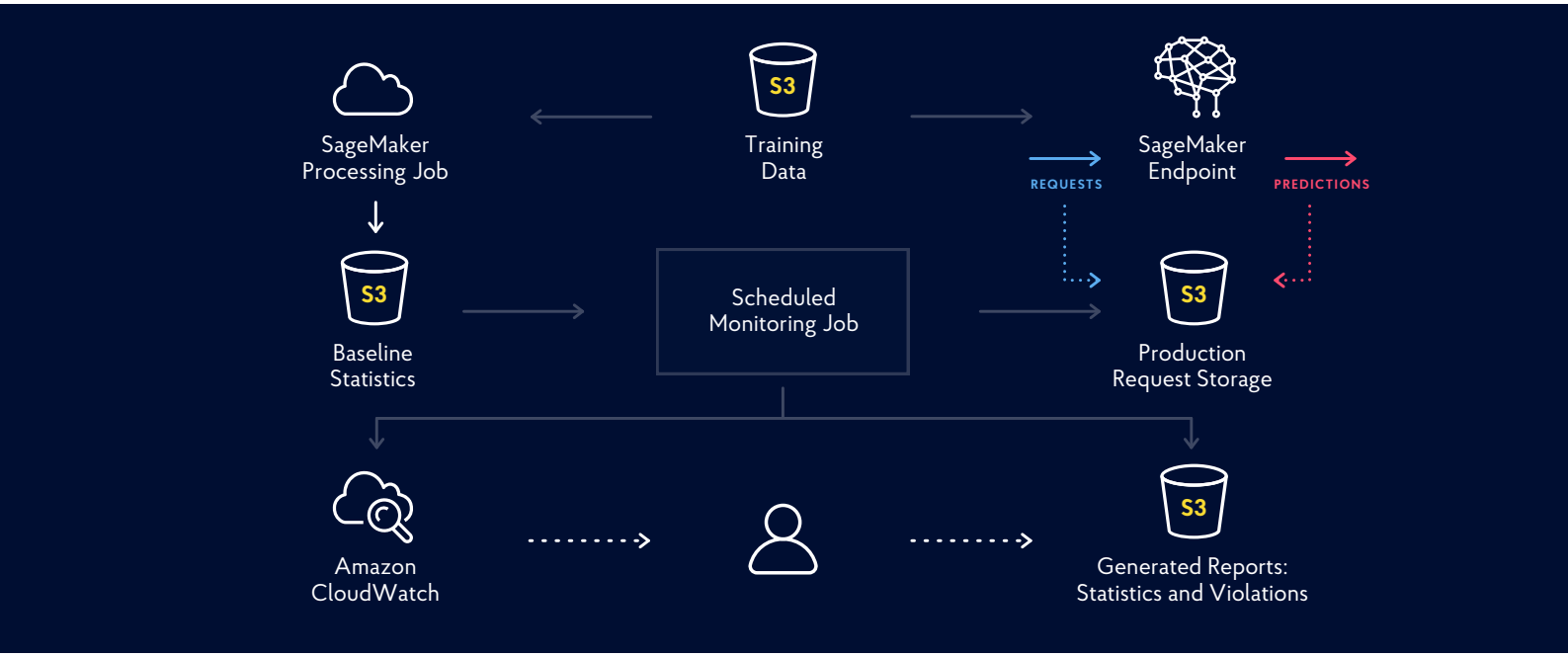
How Amazon SageMaker Model Monitor Works

Let's assume that the machine learning model, which is wrapped as a SageMaker endpoint, receives requests and generates predictions based on the training data stored in Amazon S3.

Once Amazon SageMaker Model Monitor is activated, it creates a production request storage that enables shadowing of data streams, requests and predictions, and stores them in a separate S3 bucket.

Amazon SageMaker Model Monitor launches a SageMaker processing job that retrieves schema from the training data (attributes like minimum and maximum values, mean, average, variance, and more), to generate a JSON report featuring baseline statistics.

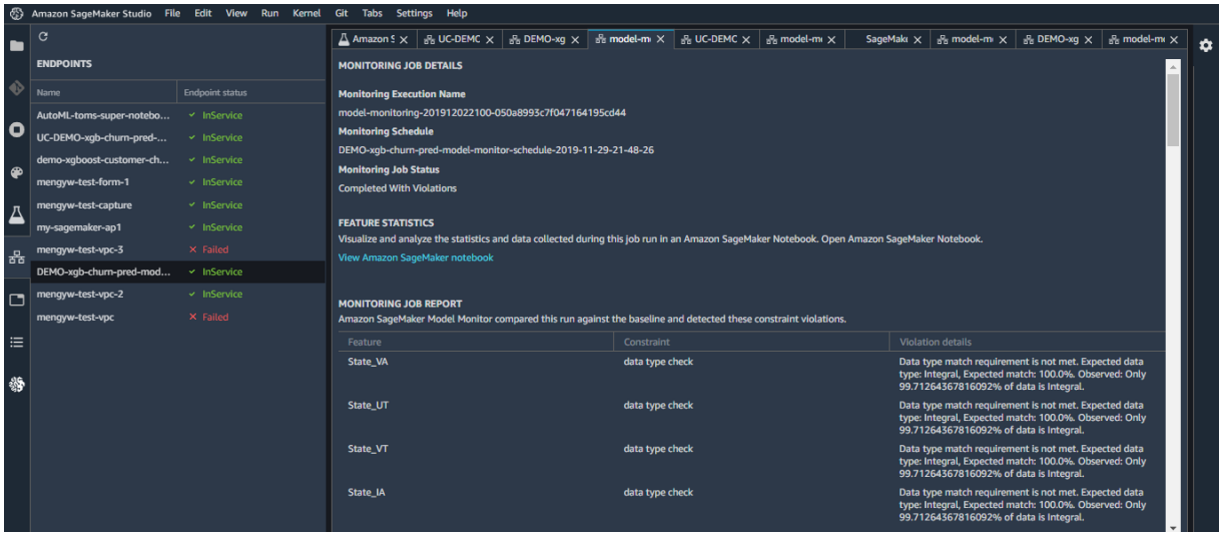
A scheduled monitoring job is created that compares requests stored in the production request storage and baseline statistics, to generate a report on statistics and violations. Some metrics are also pushed to Amazon CloudWatch, to notify users of detected errors.



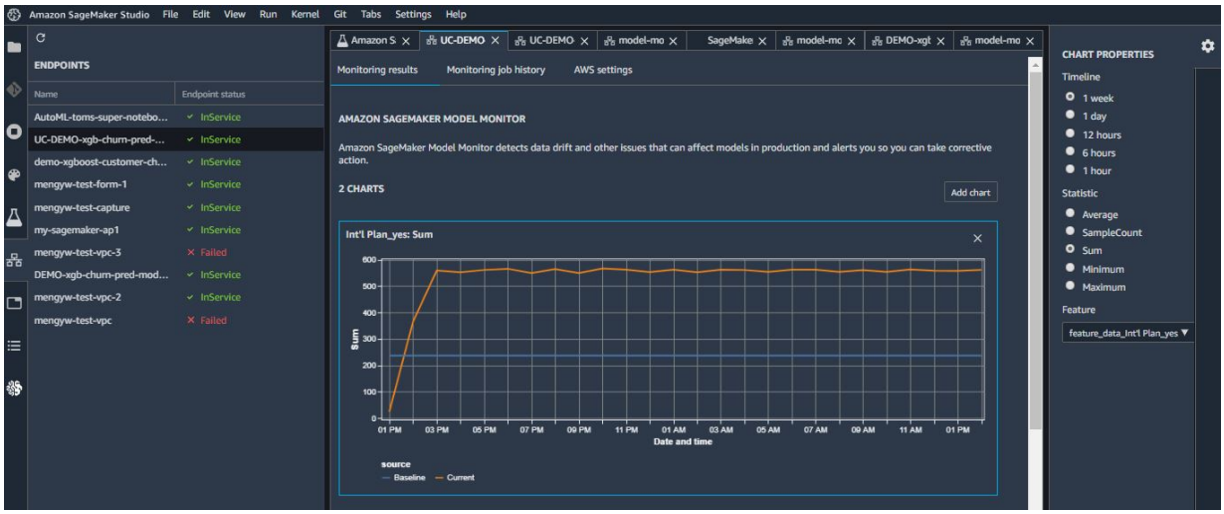
Amazon SageMaker Model Monitor Architecture

Note: Though most metrics are pushed to Amazon CloudWatch, some of them are accessible only in the generated reports bucket in Amazon S3. To access those, unparse a JSON report and search for violations manually.

Amazon SageMaker Model Monitor is part of Amazon SageMaker Studio, so one can easily check running jobs, reports, constraints, etc. All statistics, monitoring results and data collected can be viewed and further analyzed in a notebook.



Example of Dashboards and Alerts in Amazon SageMaker Model Monitor



Example of real-time model monitoring in Amazon SageMaker Model Monitor

Hidden Value of Amazon SageMaker Model Monitor

Given all that, we can assume that Scheduled Monitoring Job is a key value driver for data scientists in Amazon SageMaker Model Monitor. The baseline element of the tool, however, is its ability to handle data wrangling between its artifacts and services.

Amazon SageMaker Model Monitor is an SDK that allows to easily move and shadow data to S3 buckets while enabling fine-tuning and fixing of the Processing Job and Monitoring Job as Deequ-based containers.

In other words, SageMaker Model Monitor's hidden value lies in its ability to combine specific

artifacts and services in a fully-managed, automatic environment, to help data scientists build architectures like:

- Built-in container with schema extractor from training data
- Built-in container with Min/Max/Mean and KS test
- SDK which hides data wrangling, shadowing, job scheduling, pushing metrics to CloudWatch and retrieval of the latest job results

Amazon SageMaker Elastic Inference

Amazon SageMaker Elastic Inference (EI) is a new capability of Amazon SageMaker that allows to cost-efficiently accelerate the throughput and reduce the latency of real-time inferences from models that are deployed as Amazon SageMaker hosted models.

Amazon SageMaker Elastic Inference allows data scientists and ML engineers to:

- Add inference acceleration to a hosted endpoint at a fraction of the cost of a full GPU instance
- Add an EI accelerator in one of the available sizes to a deployable model in addition to a CPU instance type
- Add the model as a production variant to an endpoint configuration, which is used to deploy a hosted endpoint
- Add an EI accelerator to an Amazon SageMaker notebook instance, to test and evaluate inference performance when building the models

Amazon SageMaker EI: Benefits & Limitations

EI (CPU + EI Accelerator) allows to **quickly, effectively and cost-efficiently deploy machine learning models** as compared to deploys on EC2 instances or deploys as AWS Sagemaker Endpoints. It is well-suited for projects where one needs to utilize fast accelerators, support streaming pipeline and maintain cost-efficiency.

Amazon Elastic Inference surpasses the limitations of other deployment options:

- Scales from 1GB to 8GB VRAM
- Offers high TFLOPS in $/h value
- Provides autoscaling

However, EI has certain limitations. For example, it is fully available only in six regions. It supports

ONNX only through MXNET runtime. It supports only older versions of MXNET and TensorFlow versions (up to 1.14). It does not support Cuda. Eia2.xlarge is twice as slow as V100, but it costs only $0.340 per hour.

How to Use EI in ONNX Model

To start using Elastic Inference, save the PyTorch model in a standardized format. Then, since only two types of runtimes are supported, export the model to MXNET, and add a piece of code on the right, as shown below:

Deploy

```
mxnet_model = MXNetModel(model_data=model_data,
                         entry_point='resnet152.py',
                         role=role,
                         py_version='py3',
                         framework_version='1.4.1')

predictor = mxnet_model.deploy(
    initial_instance_count=1,
    instance_type='ml.m5.xlarge',
    accelerator_type='ml.eia1.medium'
)
```

```
scores = predictor.predict(input_image.asnumpy())
```

```
predictor.delete_endpoint()
```

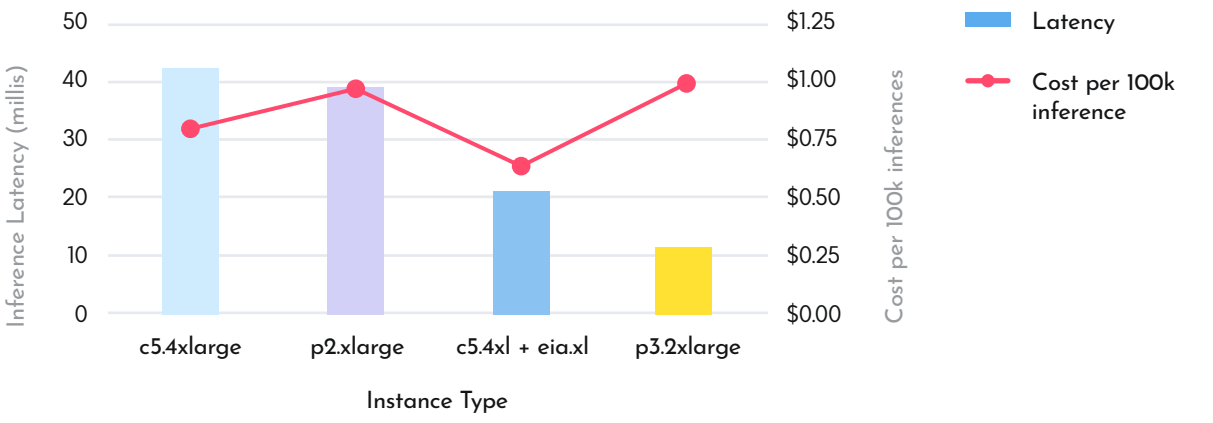
Put it in your model code:

```
ctx = mx.eia()
```

Launching EI for ONNX models

When compared in terms of inference latency and cost, EI performs faster and more cost-efficiently than c5.4xlarge and p2.xlarge. It is not as fast as p3.2xlarge, but is still more favorable cost-wise.

Initial Results



Comparison of different ML model deployment instances

EI offers the lowest price on the market while performing better than P2. It has a few limitations in frameworks and their versions, but they are compensated by such features as auto scaling and rapid inference acceleration.

Conclusion and Further Steps

As more companies are looking to implement AI and ML in one way or another, both tools and technology must evolve to enable data scientists and ML engineers to explore data and build ML models more quickly, effectively and cost-efficiently.

With the release of Amazon SageMaker Studio, AWS gives engineers the ability to launch JupyterLab-based notebooks in seconds, access notebooks with SSO, manage multiple related training jobs, create new experiments and visualize results, automatically generate and run experiments, detect and troubleshoot training problems, monitor data drift, and visualize data metrics and rules violations.

Though Amazon SageMaker Studio has some limitations, it promises to become the ultimate platform for writing code, tracking experiments, visualizing data, and performing debugging and monitoring. AWS continues to enhance Studio, so we can expect to see more and better features, and new service integrations in the near future.

References

1. aws.amazon.com/sagemaker
2. docs.aws.amazon.com/sagemaker/latest/dg/gs-studio.html
3. docs.aws.amazon.com/sagemaker/latest/dg/experiments.html
4. docs.aws.amazon.com/sagemaker/latest/dg/train-debugger.html
5. aws.amazon.com/sagemaker/autopilot
6. aws.amazon.com/blogs/aws/amazon-sagemaker-autopilot-fully-managed-automatic-machine-learning
7. docs.aws.amazon.com/sagemaker/latest/dg/autopilot-automate-model-development.html
8. docs.aws.amazon.com/sagemaker/latest/dg/model-monitor.html
9. docs.aws.amazon.com/sagemaker/latest/dg/ei.html
10. aws.amazon.com/blogs/big-data/test-data-quality-at-scale-with-deequ
11. aws.amazon.com/blogs/aws/amazon-sagemaker-studio-the-first-fully-integrated-development-environment-for-machine-learning
12. mckinsey.com/featured-insights/artificial-intelligence/notes-from-the-ai-frontier-applications-and-value-of-deep-learning
13. infoq.com/news/2019/12/aws-sagemaker-studio-ide

Appendix B

Authors and Contributors

1. **Stepan Pushkarev**
   Chief Technology Officer
   spushkarev@provectus.com

2. **Iskandar Sitdikov**
   ML Engineer
   isitdikov@provectus.com

3. **Lenar Gabdrakhmanov**
   ML Engineer
   lgabdrakhmanov@provectus.com

4. **Anton Kiselev**
   ML Engineer
   akiselev@provectus.com

5. **Marat Adayev**
   ML Engineer
   madayev@provectus.com

6. **Yuriy Gavrilin**
   ML Engineer
   ygavrilin@provectus.com

7. **Rinat Akhmetov**
   ML Engineer
   rakhmetov@provectus.com

8. **Ilnur Garifullin**
   Demo Engineer
   igarifullin@provectus.com

9. **Andrii Khakhariev**
   Copywriter
   akhakhariev@provectus.com

10. **Vlad Usatenko**
    Designer
    vusatenko@provectus.com

For more information reach us at
hello@provectus.com | provectus.com