SageMaker Processing
Lab 2. Train, Tune and Deploy XGBoost
Lab 3. Bring your own model
Lab 3a. Bring your own Script (TensorFlow)
Lab 3b. Bring your own Script (PyTorch)
Lab3c. Bring your own Container
Lab 4. Autopilot, Debugger and Model Monitor
Autopilot
Debugger
Model Monitor
Lab 5. Bias and Explainability
Bias and Explainability-Tabular Data
Lab 6. SageMaker Pipelines
Lab 7. Real Time ML inference on Streaming Data
Lab 8. Build ML Model with No Code Using Sagemaker Canvas
Lab 9. Amazon SageMaker JumpStart
Lab 10. ML Governance Tools for Amazon SageMaker
Lab 11. SageMaker Notebook Instances

Content preferences
Language
English

Model Monitor

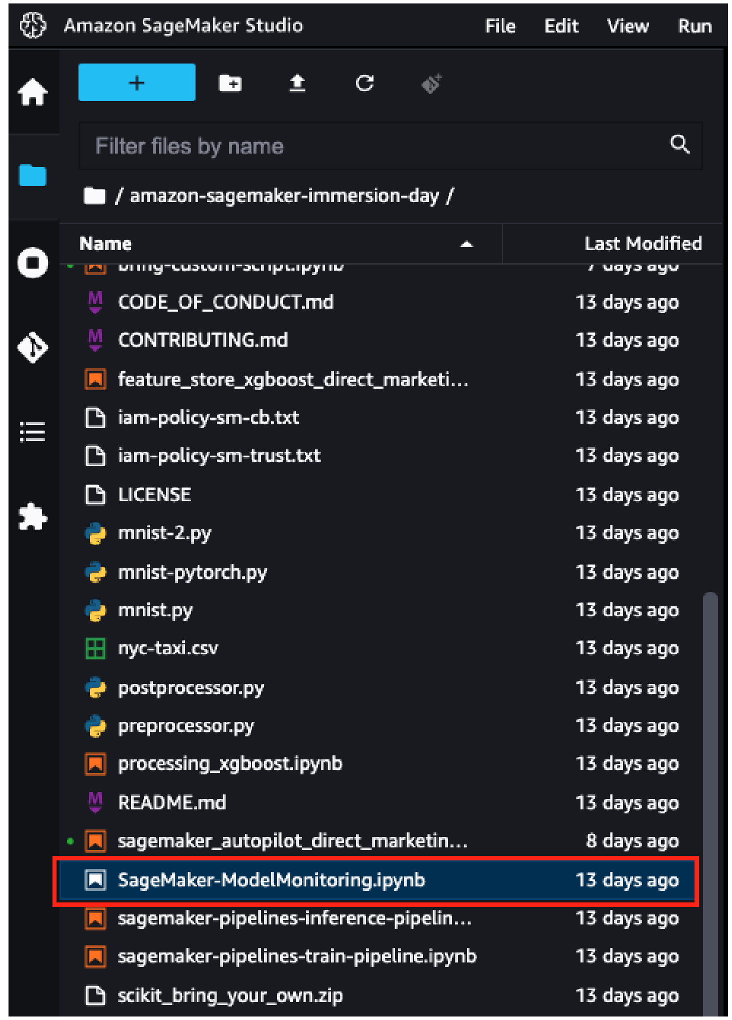
Model Monitoring

Overview

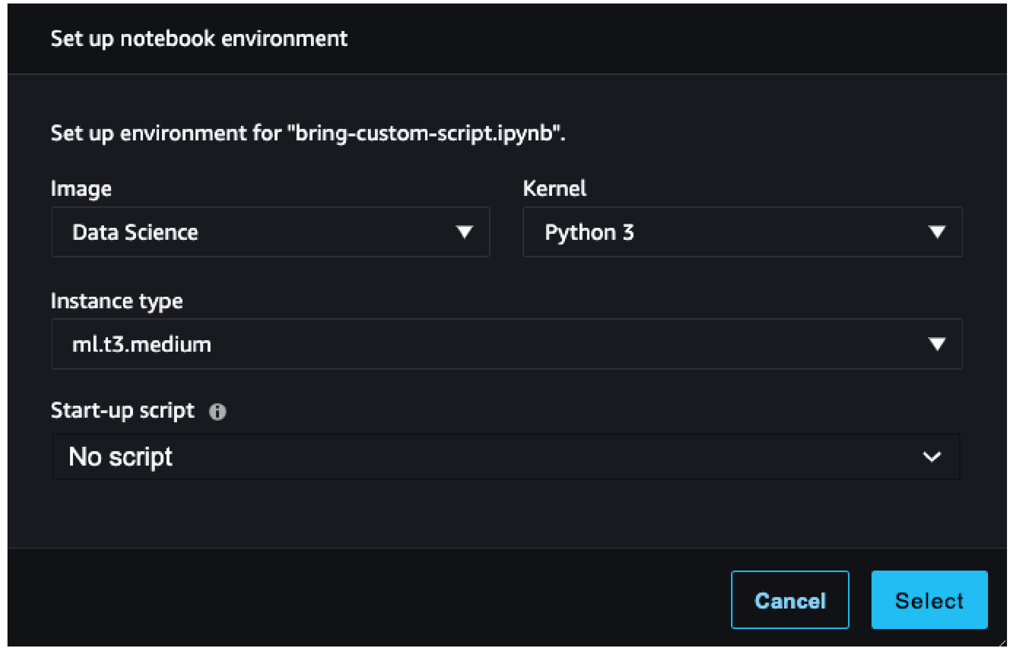
Amazon SageMaker ModelMonitor enables you to capture the input, output and metadata for invocations of the models that you deploy. It also enables you to analyze the data and monitor its quality. In this notebook, you learn how Amazon SageMaker enables these capabilities.

PART A: Capturing real-time inference data from Amazon SageMaker endpoints

1. Click on the “amazon-sagemaker-immersion-day” folder and then double click on the following file: “SageMaker-ModelMonitoring.ipynb” notebook.



2. You will be prompted to choose an image and instance type. Choose 'Data Science' image and 'ml.t3.medium' instance type and click "Select".



3. You will then have the notebook opened. You can verify the Kernel CPU and Memory states on the top right of the notebook.

Home | SageMaker-ModelMonitoring-

Code | 2 vCPU + 4 GiB | Cluster | Python 3 (Data Science) | Share

Amazon SageMaker Model Monitor

This notebook shows how to:

- Host a machine learning model in Amazon SageMaker and capture inference requests, results, and metadata
- Analyze a training dataset to generate baseline constraints
- Monitor a live endpoint or batch transforms for violations against constraints

Background

Amazon SageMaker provides every developer and data scientist with the ability to build, train, and deploy machine learning models quickly. Amazon SageMaker is a fully-managed service that encompasses the entire machine learning workflow. You can label and prepare your data, choose an algorithm, train a model, and then tune and optimize it for deployment. You can deploy your models to production with Amazon SageMaker to make predictions and lower costs than was previously possible.

In addition, Amazon SageMaker enables you to capture the input, output and metadata for invocations of the models that you deploy. It also enables you to analyze the data and monitor its quality. In this notebook, you learn how Amazon SageMaker enables these capabilities.

```
%%time
# cell 01

# Handful of configuration

import os
import boto3
import re
import json
from sagemaker import get_execution_role, session

region= boto3.Session().region_name

role = get_execution_role()
print("RoleArn: {}".format(role))

# You can use a different bucket, but make sure the role you chose for this notebook
# has the s3:PutObject permissions. This is the bucket into which the data is captured
bucket =  session.Session(boto3.Session()).default_bucket()
print("Demo Bucket: {}".format(bucket))
prefix = 'sagemaker/DEMO-ModelMonitor'

data_capture_prefix = '{}/datacapture'.format(prefix)
s3_capture_upload_path = 's3://{}/{}'.format(bucket, data_capture_prefix)
reports_prefix = '{}/reports'.format(prefix)
s3_report_path = 's3://{}/{}'.format(bucket,reports_prefix)
code_prefix = '{}/code'.format(prefix)
s3_code_preprocessor_uri = 's3://{}/{}/{}'.format(bucket,code_prefix, 'preprocessor.py')
s3_code_postprocessor_uri = 's3://{}/{}/{}'.format(bucket,code_prefix, 'postprocessor.py')

print("Capture path: {}".format(s3_capture_upload_path))
print("Report path: {}".format(s3_report_path))
print("Preproc Code path: {}".format(s3_code_preprocessor_uri))
print("Postproc Code path: {}".format(s3_code_postprocessor_uri))
```

```
# cell 02
# Upload some test files
boto3.Session().resource('s3').Bucket(bucket).Object("test_upload/test.txt").upload_file('test_data/upload-test-file.txt')
print("Success! You are all set to proceed.")
```

4. We use a pre-trained XGBoost Churn Prediction Model and Upload the pre-trained model to Amazon S3.

Upload the pre-trained model to Amazon S3

This code uploads a pre-trained XGBoost model that is ready for you to deploy. This model was trained using the XGB Churn Prediction Notebook in SageMaker. You can also use your own pre-trained model in this step. If you already have a pretrained model in Amazon S3, you can add it instead by specifying the s3_key.

```
# cell 03
model_file = open("model/xgb-churn-prediction-model.tar.gz", 'rb')
s3_key = os.path.join(prefix, 'xgb-churn-prediction-model.tar.gz')
boto3.Session().resource('s3').Bucket(bucket).Object(s3_key).upload_fileobj(model_file)
```

5. Deploy this model to Amazon SageMaker.

```
# cell 04
from time import gmtime, strftime
from sagemaker.model import Model
from sagemaker.image_uris import retrieve

model_name = "DEMO-xgb-churn-pred-model-monitor-" + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
model_url = 'https://{}.s3-{}.amazonaws.com/{}/xgb-churn-prediction-model.tar.gz'.format(bucket, region, prefix)
image_uri = retrieve(region=boto3.Session().region_name, framework='xgboost', version='0.90-2')

model = Model(image_uri=image_uri, model_data=model_url, role=role)
```

6. 'DataCaptureConfig' is required to capture model quality metrics.

```
# cell 05
from sagemaker.model_monitor import DataCaptureConfig

endpoint_name = 'DEMO-xgb-churn-pred-model-monitor-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print("EndpointName={}".format(endpoint_name))

data_capture_config = DataCaptureConfig(
                        enable_capture=True,
                        sampling_percentage=100,
                        destination_s3_uri=s3_capture_upload_path)

predictor = model.deploy(initial_instance_count=1,
                instance_type='ml.m4.xlarge',
                endpoint_name=endpoint_name,
                data_capture_config=data_capture_config)
```

7. Let's now invoke the endpoint and start capturing the model quality metrics.

```
# cell 06
from sagemaker.predictor import Predictor
import sagemaker
import time

predictor = Predictor(endpoint_name=endpoint_name, serializer=sagemaker.serializers.CSVSerializer())

# get a subset of test data for a quick test
!head -120 test_data/test-dataset-input-cols.csv > test_data/test_sample.csv
print("Sending test traffic to the endpoint {}. \nPlease wait...".format(endpoint_name))

with open('test_data/test_sample.csv', 'r') as f:
    for row in f:
        payload = row.rstrip('\n')
        response = predictor.predict(data=payload)
        time.sleep(0.5)

print("Done!")
```

8. We can view the captured data.

```
# cell 07
s3_client = boto3.Session().client('s3')
current_endpoint_capture_prefix = '{}/{}'.format(data_capture_prefix, endpoint_name)
result = s3_client.list_objects(Bucket=bucket, Prefix=current_endpoint_capture_prefix)
capture_files = [capture_file.get("Key") for capture_file in result.get('Contents')]
print("Found Capture Files:")
print("\n ".join(capture_files))
```

```
# cell 08
def get_obj_body(obj_key):
    return s3_client.get_object(Bucket=bucket, Key=obj_key).get('Body').read().decode("utf-8")

capture_file = get_obj_body(capture_files[-1])
print(capture_file[:2000])
```

9. The line contains both the input and output merged together, we can see in formatted JSON format.

```
# cell 09
import json
print(json.dumps(json.loads(capture_file.split('\n')[0]), indent=2))
```

PART B: Model Monitor - baselining and continuous monitoring

In addition to collecting the data, Amazon SageMaker provides the capability for you to monitor and evaluate the data observed by the endpoints. For this:

1. Create a baseline with which you compare the real-time traffic.

2. Once a baseline is ready, setup a schedule to continuously evaluate and compare against the baseline.

1. Constraint suggestion with baseline/training dataset

The training dataset with which you trained the model is usually a good baseline dataset. From this training dataset you can ask Amazon SageMaker to suggest a set of baseline `constraints` and generate descriptive `statistics` to explore the data.

1. Upload the training data to s3

```
# cell 11
training_data_file = open("test_data/training-dataset-with-header.csv", 'rb')
s3_key = os.path.join(baseline_prefix, 'data', 'training-dataset-with-header.csv')
boto3.Session().resource('s3').Bucket(bucket).Object(s3_key).upload_fileobj(training_data_file)
```

2. Suggest_baseline() function generates the baseline constraints.

```
# cell 12
from sagemaker.model_monitor import DefaultModelMonitor
from sagemaker.model_monitor.dataset_format import DatasetFormat

my_default_monitor = DefaultModelMonitor(
    role=role,
    instance_count=1,
    instance_type='ml.m5.xlarge',
    volume_size_in_gb=20,
    max_runtime_in_seconds=3600,
)

my_default_monitor_baseline = my_default_monitor.suggest_baseline(
    baseline_dataset=baseline_data_uri+'/training-dataset-with-header.csv',
    dataset_format=DatasetFormat.csv(header=True),
    output_s3_uri=baseline_results_uri,
    wait=True
)
```

3. We can explore this generated baseline

```
# cell 14
import pandas as pd

baseline_job = my_default_monitor.latest_baselining_job
schema_df = pd.json_normalize(baseline_job.baseline_statistics().body_dict["features"])
schema_df.head(10)
```

```
# cell 15
constraints_df = pd.json_normalize(baseline_job.suggested_constraints().body_dict["features"])
constraints_df.head(10)
```

2. Analyzing collected data for data quality issues

When you have collected the data above, analyze and monitor the data with hourly Monitoring Schedules

1. CronExpressionGenerator class will create an hourly schedule

```
# cell 17
from sagemaker.model_monitor import CronExpressionGenerator
from time import gmtime, strftime

mon_schedule_name = 'DEMO-xgb-churn-pred-model-monitor-schedule-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
my_default_monitor.create_monitoring_schedule(
    monitor_schedule_name=mon_schedule_name,
    endpoint_input=predictor.endpoint_name,
    #record_preprocessor_script=pre_processor_script,
    post_analytics_processor_script=s3_code_postprocessor_uri,
    output_s3_uri=s3_report_path,
    statistics=my_default_monitor.baseline_statistics(),
    constraints=my_default_monitor.suggested_constraints(),
    schedule_cron_expression=CronExpressionGenerator.hourly(),
    enable_cloudwatch_metrics=True,

)
```

2. Below code starts a thread to send some traffic to the endpoint. Thus we can use the baseline resources (constraints and statistics) to compare against this real-time traffic.

```
# cell 18
from threading import Thread
from time import sleep
import time

endpoint_name=predictor.endpoint_name
runtime_client = boto3.client('runtime.sagemaker')

# (just repeating code from above for convenience/ able to run this section independently)
def invoke_endpoint(ep_name, file_name, runtime_client):
    with open(file_name, 'r') as f:
        for row in f:
            payload = row.rstrip('\n')
            response = runtime_client.invoke_endpoint(EndpointName=ep_name,
                                                      ContentType='text/csv',
                                                      Body=payload)
            response['Body'].read()
            time.sleep(1)

def invoke_endpoint_forever():
    while True:
        invoke_endpoint(endpoint_name, 'test_data/test-dataset-input-cols.csv', runtime_client)

thread = Thread(target = invoke_endpoint_forever)
thread.start()

# Note that you need to stop the kernel to stop the invocations
```

3. Describe and inspect monitoring schedule

```
# cell 19
desc_schedule_result = my_default_monitor.describe_schedule()
print('Schedule status: {}'.format(desc_schedule_result['MonitoringScheduleStatus']))
```

4. List the executions

```
# cell 20
mon_executions = my_default_monitor.list_executions()
print("We created a hourly schedule above and it will kick off executions ON the hour (plus 0 - 20 min buffer.\nWe will have to wait till we hit the hour...")

while len(mon_executions) == 0:
    print("Waiting for the 1st execution to happen...")
    time.sleep(60)
    mon_executions = my_default_monitor.list_executions()
```

⚠ This step can take more than one hour to complete.

5. Inspect latest executions

```
# cell 21
latest_execution = mon_executions[-1] # latest execution's index is -1, second to last is -2 and so on..
time.sleep(60)
latest_execution.wait(logs=False)

print("Latest execution status: {}".format(latest_execution.describe()['ProcessingJobStatus']))
print("Latest execution result: {}".format(latest_execution.describe()['ExitMessage']))

latest_job = latest_execution.describe()
if (latest_job['ProcessingJobStatus'] != 'Completed'):
        print("====STOP==== \n No completed executions to inspect further. Please wait till an execution completes or investigate previously reported failures.")
```

6. List the generated reports

```
# cell 23
from urllib.parse import urlparse
s3uri = urlparse(report_uri)
report_bucket = s3uri.netloc
report_key = s3uri.path.lstrip('/')
print('Report bucket: {}'.format(report_bucket))
print('Report key: {}'.format(report_key))

s3_client = boto3.Session().client('s3')
result = s3_client.list_objects(Bucket=report_bucket, Prefix=report_key)
report_files = [report_file.get("Key") for report_file in result.get('Contents')]
print("Found Report Files:")
print("\n ".join(report_files))
```

7. See violations compared to baseline

```
# cell 24
violations = my_default_monitor.latest_monitoring_constraint_violations()
pd.set_option('display.max_colwidth', None)
constraints_df = pd.json_normalize(violations.body_dict["violations"])
constraints_df.head(10)
```

Cleanup

You can keep your endpoint running to continue capturing data. If you do not plan to collect more data or use this endpoint further, you should delete the endpoint to avoid incurring additional charges. Note that deleting your endpoint does not delete the data that was captured during the model invocations. That data persists in Amazon S3 until you delete it yourself.But before that, you need to delete the schedule first.

```
# cell 26
my_default_monitor.delete_monitoring_schedule()
time.sleep(60) # actually wait for the deletion
```

```
# cell 27
predictor.delete_endpoint()
```

```
# cell 28
predictor.delete_model()
```

Model monitoring with Batch transform (Optional)

This is an optional section that walks through setup of model monitoring for batch use case.

PART A: Capturing data from Amazon SageMaker endpoints

1. We use a pre-trained XGBoost Churn Prediction Model and Upload the pre-trained model to Amazon S3.

```
model_file = open("model/xgb-churn-prediction-model.tar.gz", "rb")
s3_key = os.path.join(prefix, "xgb-churn-prediction-model.tar.gz")
boto3.Session().resource("s3").Bucket(bucket).Object(s3_key).upload_fileobj(model_file)
```

```
from time import gmtime, strftime
from sagemaker.model import Model
from sagemaker.image_uris import retrieve

model_name = "DEMO-xgb-churn-pred-model-monitor-" + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
model_url = "https://{}.s3-{}.amazonaws.com/{}/xgb-churn-prediction-model.tar.gz".format(
    bucket, region, prefix
)

image_uri = retrieve("xgboost", boto3.Session().region_name, "0.90-1")

model = Model(image_uri=image_uri, model_data=model_url, role=role)
```

2. We upload the batch inference data to S3

```
!aws s3 cp test_data/test-dataset-input-cols.csv s3://{bucket}/transform-input/test-dataset-input-cols.csv
```

3. We create a Batch Transform job

```
from sagemaker.inputs import BatchDataCaptureConfig
```

```
transfomer = model.transformer(
    instance_count=1,
    instance_type="ml.m4.xlarge",
    accept="text/csv",
    assemble_with="Line",
)

transfomer.transform(
    "s3://{}/transform-input".format(bucket),
    content_type="text/csv",
    split_type="Line",
    # configure the data capturing
    batch_data_capture_config=BatchDataCaptureConfig(
        destination_s3_uri=s3_capture_upload_path,
    ),
    wait=True,
)
```

!

4. We examine the results

```
!aws s3 ls {s3_capture_upload_path}/input/ --recursive
```

```
2023-03-15 12:40:29         99 sagemaker/DEMO-ModelMonitor/datacapture/input/2023/03/15/12/d2b7486f-1693-4b31-b891-205605cef428.json
2023-03-24 11:45:23         99 sagemaker/DEMO-ModelMonitor/datacapture/input/2023/03/24/11/1da419ce-024e-4f9e-8fc5-b25eb1640777.json
2023-03-24 12:45:27         99 sagemaker/DEMO-ModelMonitor/datacapture/input/2023/03/24/12/8834abd7-4889-4eaa-ab18-85d0af218a46.json
2023-03-24 12:46:05         99 sagemaker/DEMO-ModelMonitor/datacapture/input/2023/03/24/12/97a9ca4e-c887-4b5a-ad74-7154e654a4c7.json
2023-03-24 12:46:51         99 sagemaker/DEMO-ModelMonitor/datacapture/input/2023/03/24/12/d4990cfa-cbf4-4f08-961b-a33a5107d548.json
2023-03-24 13:46:22         99 sagemaker/DEMO-ModelMonitor/datacapture/input/2023/03/24/13/caca8d16-5b96-4ac7-9614-7e4b567f57f0.json
```

```
!aws s3 ls {s3_capture_upload_path}/output/ --recursive
```

```
2023-03-15 12:40:29        129 sagemaker/DEMO-ModelMonitor/datacapture/output/2023/03/15/12/689dcd0e-b680-4027-b029-a67323bb95ac.json
2023-03-24 11:45:23        129 sagemaker/DEMO-ModelMonitor/datacapture/output/2023/03/24/11/b10c356a-5cfa-4465-8943-cc75a301d93e.json
2023-03-24 12:46:51        129 sagemaker/DEMO-ModelMonitor/datacapture/output/2023/03/24/12/ad75cf2c-38fe-4f9f-9677-02670165a223.json
2023-03-24 12:46:05        129 sagemaker/DEMO-ModelMonitor/datacapture/output/2023/03/24/12/df05677b-2692-4805-8689-02ad744af0c6.json
2023-03-24 12:45:27        129 sagemaker/DEMO-ModelMonitor/datacapture/output/2023/03/24/12/eaa0a70a-54de-47d6-a14a-7227771b941c.json
2023-03-24 13:46:22        129 sagemaker/DEMO-ModelMonitor/datacapture/output/2023/03/24/13/a8129c9d-deba-4bc1-9fa8-e9f3dcf9611f.json
```

PART B: Model Monitor - baselining and continuous monitoring

1. In addition to collecting the data, Amazon SageMaker provides the capability for you to monitor and evaluate the data observed by Batch transform. For this:

2. Create a baseline with which you compare the realtime traffic. Once a baseline is ready, setup a schedule to continously evaluate and compare against the baseline.

1. Constraint suggestion with baseline/training dataset

The training dataset with which you trained the model is usually a good baseline dataset. From this training dataset you can ask Amazon SageMaker to suggest a set of baseline `constraints` and generate descriptive `statistics` to explore the data.

1. Upload the training data to s3

```
# copy over the training dataset to Amazon S3 (if you already have it in Amazon S3, you could reuse it)
baseline_prefix = prefix + "/baselining"
baseline_data_prefix = baseline_prefix + "/data"
baseline_results_prefix = baseline_prefix + "/results"

baseline_data_uri = "s3://{}/{}".format(bucket, baseline_data_prefix)
baseline_results_uri = "s3://{}/{}".format(bucket, baseline_results_prefix)
print("Baseline data uri: {}".format(baseline_data_uri))
print("Baseline results uri: {}".format(baseline_results_uri))
```

```
training_data_file = open("test_data/training-dataset-with-header.csv", "rb")
s3_key = os.path.join(baseline_prefix, "data", "training-dataset-with-header.csv")
boto3.Session().resource("s3").Bucket(bucket).Object(s3_key).upload_fileobj(training_data_file)
```

2. Suggest_baseline() function generates the baseline constraints.

```
from sagemaker.model_monitor import DefaultModelMonitor
from sagemaker.model_monitor.dataset_format import DatasetFormat

my_default_monitor = DefaultModelMonitor(
    role=role,
    instance_count=1,
    instance_type="ml.m5.xlarge",
    volume_size_in_gb=20,
    max_runtime_in_seconds=3600,
)

my_default_monitor.suggest_baseline(
    baseline_dataset=baseline_data_uri + "/training-dataset-with-header.csv",
    dataset_format=DatasetFormat.csv(header=True),
    output_s3_uri=baseline_results_uri,
    wait=True,
)
```

3. We can explore this generated baseline

```
import pandas as pd

baseline_job = my_default_monitor.latest_baselining_job
schema_df = pd.json_normalize(baseline_job.baseline_statistics().body_dict["features"])
schema_df.head(10)
```

```
constraints_df = pd.json_normalize(
    baseline_job.suggested_constraints().body_dict["features"]
)
constraints_df.head(10)
```

When you have collected the data above, analyze and monitor the data with hourly Monitoring Schedules

1. CronExpressionGenerator class will create an hourly schedule

Content preferences
Language

```
from sagemaker.model_monitor import CronExpressionGenerator
from sagemaker.model_monitor import BatchTransformInput
from sagemaker.model_monitor import MonitoringDatasetFormat
from time import gmtime, strftime

statistics_path = "{}/statistics.json".format(baseline_results_uri)
constraints_path = "{}/constraints.json".format(baseline_results_uri)

mon_schedule_name = "DEMO-xgb-churn-pred-model-monitor-schedule-" + strftime(
    "%Y-%m-%d-%H-%M-%S", gmtime()
)
my_default_monitor.create_monitoring_schedule(
    monitor_schedule_name=mon_schedule_name,
    batch_transform_input=BatchTransformInput(
        data_captured_destination_s3_uri=s3_capture_upload_path,
        destination="/opt/ml/processing/input",
        dataset_format=MonitoringDatasetFormat.csv(header=False),
    ),
    output_s3_uri=s3_report_path,
    statistics=statistics_path,
    constraints=constraints_path,
    schedule_cron_expression=CronExpressionGenerator.hourly(),
    enable_cloudwatch_metrics=True,
)
```

2. List the executions

```
import time

mon_executions = my_default_monitor.list_executions()
print(
    "We created a hourly schedule above and it will kick off executions ON the hour (plus 0 - 20 min buffer.\nWe will have to wait till we hit the hour..."
)

while len(mon_executions) == 0:
    print("Waiting for the 1st execution to happen...")
    time.sleep(60)
    mon_executions = my_default_monitor.list_executions()
```

⚠ This step can take more than one hour to complete.

3. Inspect latest executions

```
latest_execution = mon_executions[
    -1
]  # latest execution's index is -1, second to last is -2 and so on..
# time.sleep(60)
latest_execution.wait(logs=False)

print("Latest execution status: {}".format(latest_execution.describe()["ProcessingJobStatus"]))
print("Latest execution result: {}".format(latest_execution.describe()["ExitMessage"]))

latest_job = latest_execution.describe()
if latest_job["ProcessingJobStatus"] != "Completed":
    print(
        "====STOP==== \n No completed executions to inspect further. Please wait till an execution completes or investigate previously reported failures."
    )
```

```
..........................................................!Latest execution status: Completed
Latest execution result: CompletedWithViolations: Job completed successfully with 1 violations.
```

Content preferences

Language

4. List the generated reports

```
from urllib.parse import urlparse

s3uri = urlparse(report_uri)
report_bucket = s3uri.netloc
report_key = s3uri.path.lstrip("/")
print("Report bucket: {}".format(report_bucket))
print("Report key: {}".format(report_key))

s3_client = boto3.Session().client("s3")
result = s3_client.list_objects(Bucket=report_bucket, Prefix=report_key)
report_files = [report_file.get("Key") for report_file in result.get("Contents")]
print("Found Report Files:")
print("\n ".join(report_files))
```

5. See violations compared to baseline

```
violations = my_default_monitor.latest_monitoring_constraint_violations()
pd.set_option("display.max_colwidth", -1)
constraints_df = pd.io.json.json_normalize(violations.body_dict["violations"])
constraints_df.head(10)
```

Conclusion

In this lab you have walked through the process of SageMaker model monitoring to identify model drift. You captured real-time inference data from Amazon SageMaker endpoints and use model monitor for baselining and continuous monitoring. You also looked at how to setup model monitoring with Batch Transform for batch inference use cases

Beyond the lab

- Amazon Sagemaker Model Monitor, product page.
- Detect machine learning (ML) model drift in production, video (29:50) from re:Invent 2020.

Previous Next