# A/B Testing Deployment with Amazon SageMaker

This document guides you step-by-step on implementing an A/B testing deployment strategy for machine learning models using Amazon SageMaker, including clear explanations of each step.

## Prerequisites

Ensure you have: - An AWS account with SageMaker access - Basic knowledge of Python and machine learning

## Lab Overview

In this lab, you’ll: 1. Set up the SageMaker environment. 2. Prepare and upload a small dataset. 3. Train two distinct models. 4. Deploy the models with A/B testing. 5. Evaluate performance. 6. Clean up AWS resources.

### Step 1: Environment Setup

# Importing necessary libraries for AWS SageMaker interaction and data handling  
import boto3  
import sagemaker  
import pandas as pd  
import numpy as np  
from sagemaker import get\_execution\_role  
  
# Setup SageMaker session and execution role  
session = sagemaker.Session()  
role = get\_execution\_role()

### Step 2: Data Preparation and Upload

Using the Iris dataset for simplicity and quick training.

# Loading and splitting the Iris dataset (only first 100 samples for simplicity)  
from sklearn.datasets import load\_iris  
from sklearn.model\_selection import train\_test\_split  
  
iris = load\_iris()  
X, y = iris.data[:100], iris.target[:100]  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Saving the dataset locally  
train\_data = pd.DataFrame(X\_train, columns=iris.feature\_names)  
train\_data['target'] = y\_train  
train\_data.to\_csv('train.csv', index=False, header=False)  
  
# Uploading training data to AWS S3 for SageMaker  
s3\_input\_train = session.upload\_data(path='train.csv', key\_prefix='sagemaker/iris/train')

### Step 3: Model Training

Training two different models: Random Forest and SVM.

**Random Forest Model:**

%%writefile train\_model\_a.py  
# Script for training a Random Forest model  
from sklearn.ensemble import RandomForestClassifier  
import argparse, joblib, os  
import numpy as np  
  
# Parsing script arguments  
parser = argparse.ArgumentParser()  
parser.add\_argument('--n-estimators', type=int, default=10)  
parser.add\_argument('--model-dir', type=str, default=os.environ['SM\_MODEL\_DIR'])  
parser.add\_argument('--train', type=str, default=os.environ['SM\_CHANNEL\_TRAIN'])  
args, \_ = parser.parse\_known\_args()  
  
# Loading training data  
train\_data = np.loadtxt(os.path.join(args.train, 'train.csv'), delimiter=',')  
X, y = train\_data[:, :-1], train\_data[:, -1]  
  
# Training model and saving it  
model = RandomForestClassifier(n\_estimators=args.n\_estimators)  
model.fit(X, y)  
joblib.dump(model, os.path.join(args.model\_dir, 'model.joblib'))

**SVM Model:**

%%writefile train\_model\_b.py  
# Script for training an SVM model  
from sklearn.svm import SVC  
import argparse, joblib, os  
import numpy as np  
  
# Parsing script arguments  
parser = argparse.ArgumentParser()  
parser.add\_argument('--C', type=float, default=1.0)  
parser.add\_argument('--model-dir', type=str, default=os.environ['SM\_MODEL\_DIR'])  
parser.add\_argument('--train', type=str, default=os.environ['SM\_CHANNEL\_TRAIN'])  
args, \_ = parser.parse\_known\_args()  
  
# Loading training data  
train\_data = np.loadtxt(os.path.join(args.train, 'train.csv'), delimiter=',')  
X, y = train\_data[:, :-1], train\_data[:, -1]  
  
# Training model and saving it  
model = SVC(C=args.C)  
model.fit(X, y)  
joblib.dump(model, os.path.join(args.model\_dir, 'model.joblib'))

### Step 4: Deploy Models with A/B Testing

Deploying both models simultaneously for A/B testing on SageMaker.

from sagemaker.session import production\_variant  
from sagemaker.serializers import CSVSerializer  
from sagemaker.deserializers import JSONDeserializer  
from datetime import datetime  
  
# Creating model objects from trained estimators  
model\_a = sklearn\_estimator\_a.create\_model()  
model\_b = sklearn\_estimator\_b.create\_model()  
  
# Explicitly registering models in SageMaker  
timestamp = datetime.now().strftime("%Y-%m-%d-%H-%M-%S")  
model\_a.name = f"model-a-{timestamp}"  
model\_b.name = f"model-b-{timestamp}"  
  
model\_a.\_create\_sagemaker\_model(instance\_type='ml.c4.2xlarge', accelerator\_type=None)  
model\_b.\_create\_sagemaker\_model(instance\_type='ml.c4.2xlarge', accelerator\_type=None)  
  
# Defining variants for A/B testing deployment  
variant1 = production\_variant(model\_name=model\_a.name, instance\_type="ml.c4.2xlarge",  
 initial\_instance\_count=1, variant\_name='ModelA', initial\_weight=50)  
variant2 = production\_variant(model\_name=model\_b.name, instance\_type="ml.c4.2xlarge",  
 initial\_instance\_count=1, variant\_name='ModelB', initial\_weight=50)  
  
endpoint\_name = 'iris-ab-test-endpoint'  
  
# Deploying endpoint with two production variants (A/B testing)  
session.endpoint\_from\_production\_variants(name=endpoint\_name, production\_variants=[variant1, variant2])  
  
# Predictor setup to test deployed endpoint  
predictor = sagemaker.Predictor(endpoint\_name=endpoint\_name, sagemaker\_session=session,  
 serializer=CSVSerializer(), deserializer=JSONDeserializer())

### Step 5: Evaluate Performance

You should include a script here to send data to your endpoint and measure the performance and variant distribution.

### Step 6: Clean Up AWS Resources

Always delete resources to avoid additional AWS charges.

# Delete endpoint and resources after testing  
session.delete\_endpoint(endpoint\_name='iris-ab-test-endpoint')  
  
# Delete endpoint configuration  
sm\_client = boto3.client('sagemaker')  
sm\_client.delete\_endpoint\_config(EndpointConfigName='iris-ab-test-endpoint')  
  
# Delete models explicitly  
sm\_client.delete\_model(ModelName=model\_a.name)  
sm\_client.delete\_model(ModelName=model\_b.name)

## Conclusion

You’ve successfully implemented an A/B testing deployment with Amazon SageMaker, including resource cleanup to prevent unnecessary charges.

**Best Practices:** - Always monitor AWS resource usage. - Keep models and endpoint names unique to avoid conflicts. - Clean up resources immediately after use to minimize costs.