# Deploying A Pre-Trained Sklearn Model on Amazon SageMaker

## Take Your Locally Trained Models To Production

Ram Vegiraju
Published in

Towards Data Science

.
5 min read
.
Jan 7, 2022
41
3

Image from Unsplash by Mehmet Ali Peker

I've written in the past about how you can train and deploy custom Sklearn and TensorFlow models on Amazon SageMaker. For certain use cases however you may have pre-trained models that you have trained elsewhere. In this article we'll explore how to take your pre-trained model data and deploy it on SageMaker. In addition, for this example we will mostly work with the code in a local environment to make sure you don't have too much heavy-lifting in the AWS console.

**NOTE**: For those of you new to AWS, make sure you make an account at the following **link** if you want to follow along. **This article will assume a novice to intermediate level of knowledge with AWS and SageMaker.** We will also not explore model building theory, the main focus of this article will be on deployment.

## Table of Contents

1. Setup
2. Model Deployment Script
3. Model Invocation
4. Additional Resources & Conclusion

## 1. Setup

To start let us quickly locally train a Sklearn model that we can use for deployment. You can simply run the following Python script for setup.
Local model training

The main key for this script is that we are using the joblib module to save the trained model. You should see this artifact display in your local directory.

Local model artifact (Screenshot by Author)

It's necessary to use joblib to save the model as that is the format that SageMaker is expecting for Sklearn models. Along with our local model script, one script that we can provide ahead of time is our inference script. This helps SageMaker understand how your input and output

for your model serving will be configured. There are [four default functions] you can provide:

1. **model_fn**: This will deserialize and load our joblib file.
2. **input_fn**: You can pass in the format of data (json, csv, etc) that your model can expect for input.
3. **predict_fn**: Our model prediction function.
4. **output_fn**: Processes the returned value from predict_fn and the type of response your endpoint will get.

Inference Handler Functions

The next part is to create an **appropriate SageMaker IAM role** within the AWS console. Here you want to give proper permissions for SageMaker: S3 Full Access, SageMaker Full Access, and ECR Full Access. This should be your main work within the AWS console for this example. Once you've created this role we can focus on building our model deployment script.

## 2. Model Deployment Script

To get started, have the following imports for our model deployment script. Make sure this script is in the same directory as your inference handler and model data (joblib).

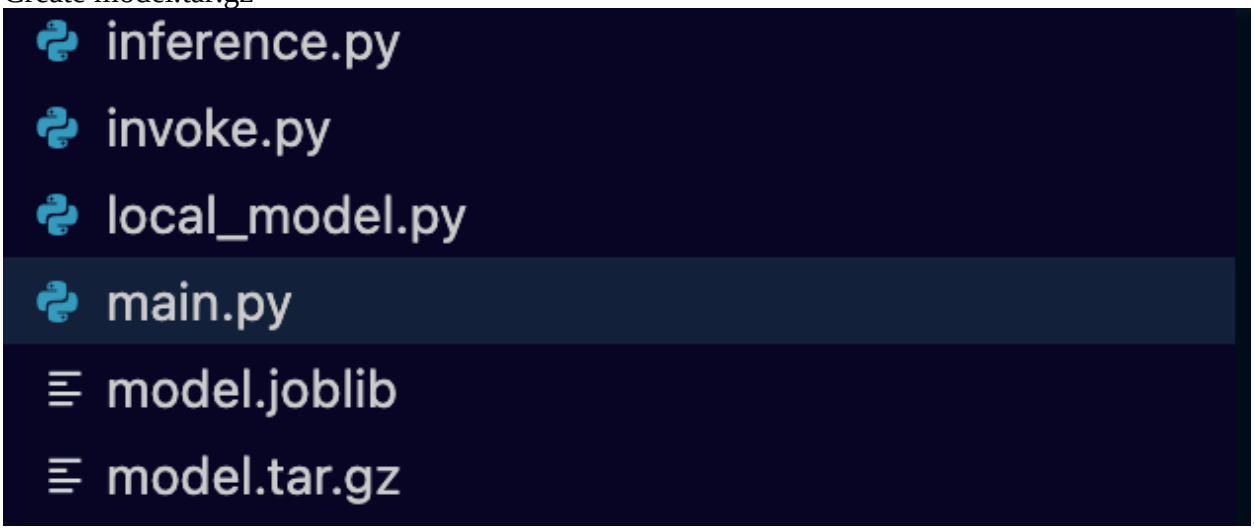
Imports

Most of our orchestration with working with AWS services is done through the SDK, in this case the Python SDK: Boto3. We work with the Boto3 client for SageMaker to orchestrate our steps for model deployment.

SageMaker setup

Here we instantiate the clients for SageMaker and the S3 client where we will store our model data for SageMaker to access. The next step is the key portion, SageMaker needs model artifacts/data in a **model.tar.gz** format. To do this we will zip the local model artifact and inference.py script into a tar file.

Create model.tar.gz




model.tar.gz locally saved (Screenshot by Author)

Next for SageMaker to understand we need to put this model artifact in a S3 location.

Upload model data to S3

Now we can focus on the three steps to build an endpoint on SageMaker.

1. Model Creation
2. Endpoint Configuration Creation
3. Endpoint Creation

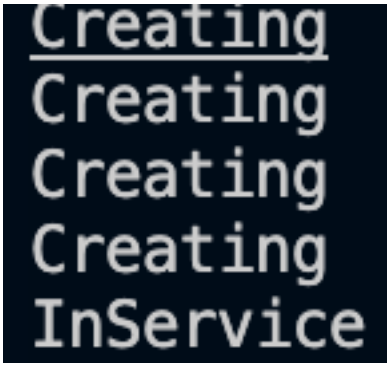
For SageMaker Model Creation we need two features: model data and our container image. In this case we can retrieve the Sklearn image for inference directly from SageMaker using the SageMaker SDK.
Retrieve Sklearn image

Now we can provide the model data and image to create our SageMaker model.
SageMaker Model Creation

Now we can take the SageMaker model and use it to create our Endpoint Configuration, here we can specify our instance type and count we want for our endpoint.
SageMaker Endpoint Configuration Creation

Now we can take this and create our endpoint which will take a few minutes to create successfully.
Create Endpoint

If you now run the script you will see an endpoint successfully created and also visible in your AWS console.



Endpoint Creation (Screenshot by Author)

## 3. Model Invocation

We can now make a separate invoke file to test our endpoint with a sample point. We don't want to add this to our main file because with each execution a new endpoint will be created. Grab your endpoint name and specify it in the following script and you will see execution.

Invoke Endpoint

Result (Screenshot by Author)

## 4. Additional Resources & Conclusion

**GitHub - RamVegiraju/Pre-Trained-Sklearn-SageMaker: Deploy a pre-trained Sklearn Model on Amazon…**

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or…

github.com

For the entire code for the example access the link above. It's very simple to deploy pre-trained models on SageMaker after understanding the format and structure of the model data. The main change that needs to occur is the image you retrieve using the SDK,

this should match the framework that you are using. In the case that the container is not supported by SageMaker check out how to Bring Your Own Container.

### Additional Resources

SageMaker Inference Examples

Multi-Model TensorFlow Endpoints with SageMaker