Productionizing a Model with Docker and SageMaker

Introduction

In this codealong, you'll see an example of the steps needed in order to productionize your own model with AWS SageMaker.

How to Use This Notebook

This notebook contains a lot of *boilerplate code* provided by Amazon which you'll need to make use of pretty much anytime you need to productionize a model. Much of this can be copied and pasted over, although it's likely that some modifications will be needed on a project-by-project basis. Going forward, use this notebook as a reference to remember the necessary steps for productionizing a model with AWS SageMaker -- you are *not* expected to remember or even understand most of the code in this notebook, as you aren't yet familiar with AWS and its quirks. That's okay -- notebooks like these are meant to help you through productionizing a model until you've done it a few times and start to understand the process and the necessary code needed to make it all work!

This notebook borrows it's general structure, as well as all boilerplate code, from this training repo provided publicly by AWS. For more examples of how to use various AWS services, check out the AWS-Samples Repository!

Overview of Process

When productionizing a machine learning model using AWS, you'll typically use the following workflow:

1. Explore and preprocess data
2. Build SageMaker container (Docker)
3. Test training and inference code on your local machine
4. Train and deploy model with SageMaker

To help simplify this process, AWS has provided some example repos containing tutorials and boilerplate code that we will make use of to help us through this process. Any code found in this notebook is primarily concerned with Step 4, training and deploying the model to SageMaker. However, the code in this notebook will only work if you have worked through the sample labs provided by AWS for steps 1 - 3, which will help you gain familiarity working with the platform. You'll find the link to these sample labs below.

Step 0: Complete Lead-Up Notebooks from AWS Training Team

IMPORTANT: Before beginning on this lab, you need to complete labs 1, 2, and 3 from this AWS Training Repo. Read them thoroughly, and complete each step for each lab. These labs handle much of the basic setup needed in order to make sure the code in this notebook will work, so ***do not skip them!***

Once you have completed those labs, come back here and move on to Step 1 below.

Step 1: Building and Registering the Container

If you are at the `container` folder within this repo, you'll see it contains some Docker images. These are docker images that you'll need to make sure you use in your own projects when productionizing a model on AWS.

Everything in the cell below is boilerplate code that uses the `container` folder in order to create and register the docker image needed on AWS.

At this point, it is best that you upload this notebook in your Jupyter's instance of SageMaker in order to run the following cells!

After you have successfully uploaded this notebook, if you are asked to choose a kernel, use the same kernel which runs the `sagemaker_keras_text_classification.ipynb` notebook.

> NOTE: If you deactivated this process (stopped your Jupyter instance, like *Step 8* below) and then came back to continue, you'll need to go back and start from Lab 2, because you need the Docker instance running in order to run the following cells.

```
%%sh

# The name of our algorithm
algorithm_name=sagemaker-keras-text-classification

cd container

chmod +x sagemaker_keras_text_classification/train
chmod +x sagemaker_keras_text_classification/serve

account=$(aws sts get-caller-identity --query Account --output text)

# Get the region defined in the current configuration (default to us-west-2 if none
region=$(aws configure get region)
region=${region:-us-west-2}

fullname="${account}.dkr.ecr.${region}.amazonaws.com/${algorithm_name}:latest"

# If the repository doesn't exist in ECR, create it.

aws ecr describe-repositories --repository-names "${algorithm_name}" > /dev/null 2>&

if [ $? -ne 0 ]
then
    aws ecr create-repository --repository-name "${algorithm_name}" > /dev/null
fi

# Get the login command from ECR and execute it directly
$(aws ecr get-login --region ${region} --no-include-email)

# Build the docker image locally with the image name and then push it to ECR
# with the full name.

# On a SageMaker Notebook Instance, the docker daemon may need to be restarted in or
# to detect your network configuration correctly.  (This is a known issue.)
if [ -d "/home/ec2-user/SageMaker" ]; then
```

```
    sudo service docker restart
fi

docker build  -t ${algorithm_name} .
docker tag ${algorithm_name} ${fullname}

docker push ${fullname}
```

Step 2: Setting Up the Environment

Once we've created the container, we'll need to set up the environment. The cell below contains more boilerplate code, which is used to handle a couple sticking points in order to set up the environment.

```
# S3 prefix
prefix = 'sagemaker-keras-text-classification'

# Define IAM role
import boto3
import re

import os
import numpy as np
import pandas as pd
from sagemaker import get_execution_role

role = get_execution_role()
```

Step 3: Creating the Session

Now that we've created the container and set up our environment, the next step is to create a SageMaker session.

```
import sagemaker as sage
from time import gmtime, strftime

sess = sage.Session()
```

Step 4: Upload the Data for Training

Steps 4 and 5 are where you'll add the code unique to your project.In this step, make sure have a folder called `'data'` that contains the data you'll be working with. The actual structure of the data is up to you, as you'll be the one consuming it to train your model in step 5.

```
WORK_DIRECTORY = 'data'

data_location = sess.upload_data(WORK_DIRECTORY, key_prefix=prefix)
```

Step 5: Fitting the Model

This is the part where you'll do the brunt of the work. You'll train your own model on the data you uploaded in the previous step. Note that in the sample code below, the first 3 lines are boilerplate code. The actual creation and training of the model happen on the last two lines of code, where `tree` is instantiated and used.

NOTE: You may have noticed that the code in the cell below uses an `Estimator` from `sage` (which is just an alias we set for `sagemaker` up above), the SageMaker library for python, rather than a model from scikit-learn. The `sagemaker` library contains a massive amount of useful models that we can use directly. Under the hood, the `sagemaker` library wraps in the same open-source frameworks such as scikit-learn, Keras, and TensorFlow that you're used to using. The code below is an example from AWS of how to use one of their `Estimator` objects for training. If you read the output of the cell when you run everything, you'll notice that much of it is warning messages or other printouts from sklearn and keras!

For more information on the models and other tools included in the aws sagemaker library, check out Amazon SageMaker Python SDK Documentation!

```
account = sess.boto_session.client('sts').get_caller_identity()['Account']
region = sess.boto_session.region_name
image = '{}.dkr.ecr.{}.amazonaws.com/sagemaker-keras-text-classification'.format(acc

tree = sage.estimator.Estimator(image,
                       role, 1, 'ml.c5.2xlarge',
                       output_path="s3://{}/output".format(sess.default_bucket()),
                       sagemaker_session=sess)

tree.fit(data_location)
```

Step 6: Deploying the Model

This is where the magic happens -- we have a trained model, and now we need to actually ***deploy*** it to the AWS cloud! Notice how during this step, we include a `json_serializer` -- this is so that the model can serialize and deserialize data as needed when taking data in as input.

Running the cell below will create an endpoint for your trained model.

```
from sagemaker.predictor import json_serializer
predictor = tree.deploy(1, 'ml.t2.medium', serializer=json_serializer)
```

Step 7: Cleanup (IMPORTANT!)

As a final step for this exercise, be sure to run the following line of code to delete your endpoint! Although you are running this lab on the free tier, you don't want to leave it running, because that is how costs can accrue. Run the cell below to delete your endpoint.

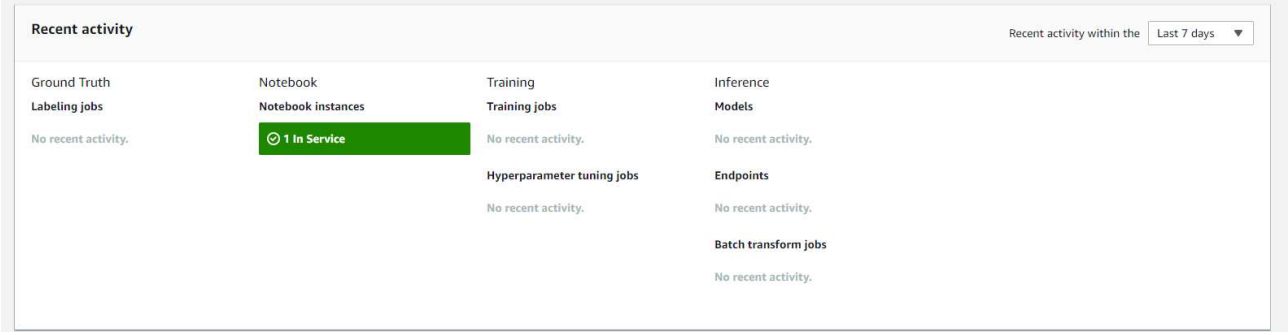
```
sess.delete_endpoint(predictor.endpoint)
```

Step 8: Deactivate Everything in AWS

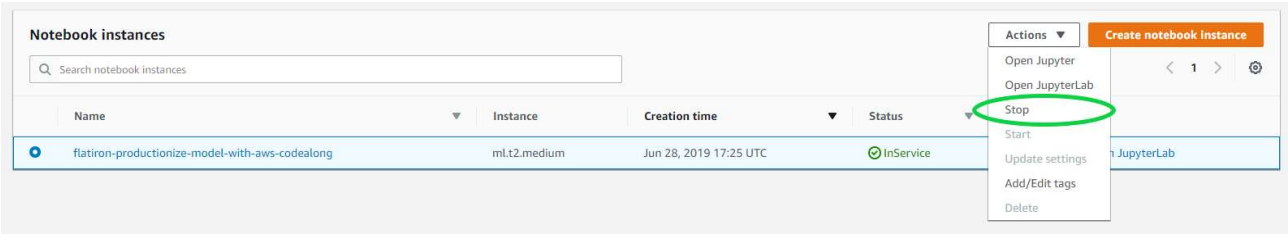
In AWS, you pay for usage. This means that anything left running is being used. While the AWS Free Tier we've signed up for allows us to do small things for free for prototyping or learning, leaving some things running may take us past the usage limits for the AWS Free Tier. In order to avoid getting charged, you'll need to do the following steps:

8.1: Deactivate the notebook in Sagemaker

First, you'll need to deactivate your notebook in SageMaker. When you enter the SageMaker platform, you'll always see the number of open notebooks you have up and running highlighted in green under the 'Recent Activity' section.



To deactivate a running notebook, select it and then go to the 'Actions' tab and select stop. Stopping the notebook instance will take a minute or two. You'll know it's done when you see the 'Status' column for the highlighted notebook change from 'InService' to 'Stopped'.



8.2: Keep an Eye on Cost Explorer

As you've seen from this lab, getting a handle on all the different services in AWS and how they interact with one another can be a bit daunting until you have some experience. It's very important that you don't leave services running when you aren't using them, because you will be charged for that. If you want to make sure that you haven't left anything running, the easiest thing to do is to check the 'Costs Explorer' page inside AWS. You can find this by searching for 'AWS Cost Explorer' in the search bar on the main page for the AWS Console. This service will show you what your usage is for everything that you can be charged for. It's quite intuitive and easy to use, and should make it easy to see if you are accruing charges because you left something running that you didn't realize. If you left something running that you aren't aware of, you'll see it here -- once you've noticed it, just navigate to the service in question and deactivate it.

Summary

In this codealong, we learned how to productionize our own model in AWS and set up an inference endpoint!

Releases

No releases published

Packages

No packages published

Contributors 3


mike-kane Mike Kane


sumedh10 Sumedh Panchadhar


alexgriff Alex Griffith

Languages

- **Python** 51.3%
- **Jupyter Notebook** 47.3%
- **Shell** 1.4%