- Kubeflow使用手册
 - o 1. 相关概念
 - 2. 使用样例
 - 2.0 准备工作
 - 2.1 使用jupyter开发模型
 - 2.2 制作Component

Kubeflow使用手册

1. 相关概念

Pipeline

在机器学习的概念中Pipeline是一套端到端的编程规范,一个Pipeline由很多个operation构成,所有的operation构成的其实是一个DAG,每一个operation的输出即为下一个operation的输入。而在Kubeflow中则将这一点发扬光大,每一个operation都是一个self-contained的docker container(当然也可以不容器化),而这些operation的原型则被称作Component,物理上也可以理解为container的image。

Component

官方定义是 self-contained set of code that performs one step in the ML workflow (pipeline)。稍微解释一下,component其实就是自定义的operation,每个operation都有自己的输入输出,能够独自完成一些工作,例如:数据预处理、数据转换、训练、预测等等。在实际使用过程中,定义一个component的工作其实就是将代码打包成docker image,有了这些image就可以去组装pipeline了。官方也提供了一些operation,但是看着全都是运行在gcp上面的,所以还是要自行打镜像。

DSL

在有了component之后如何将这些component组装成pipeline呢? Kubeflow是使用dsl实现的,其实也就是yaml文件。写这个yaml有点复杂,所以在Kubeflow pipeline SDK中包含了一个dsl-compile的工具,能够生成dsl并且打包成需要的格式。

*为了方便理解有些名词直接从TensorFlow中借鉴,如operation,并且整个套件全是谷歌出的,应该还是有意义的。

2. 使用样例

- 一个果蝇实验的demo,使用TensorFlow训练mnist。主要包含以下内容:
 - 1. 从hdfs下载训练数据
 - 2. 使用cnn训练

- 3. jupyter使用
- 4. 构建component和pipeline
- 5. 实验

2.0 准备工作

在开发过程中我们需要安装 Kubeflow Pipeline SDK,这个sdk可以帮助我们创建component和 Pipeline。

使用下面的命令安装

pip install kfp

根据需要的不同在不同的环境上安装此sdk,如果在自己电脑上的IDE中开发则需要在自己机器上安装,如果要在远程的jupyter中开发pipeline,则需要在jupyter的镜像中提前打入该包。

2.1 使用jupyter开发模型

这个其实没啥特别好说的,和普通的jupyter没啥区别。主要演示从hdfs下载训练数据,简单起见直接使用webhdfs api,实际生产使用中可以在image里面打入hdfs client。

2.2 制作Component

Kubeflow中有两种Component:

- **Reusable Component** 其实就是常用的component,在一个container中包含完整操作 code,制作完成之后可以随意分享
- **LightWeight Component** 轻量级的Component,直接将一个python的方法映射成component,在开发过程中使用比较方便

本节主要讲怎么制作Reusable component, LightWeight component比较简单直接参考官方代码样例即可。

本节要制作的component依旧是训练mnist数据,包含的功能:

- 1. 从hdfs上下载训练数据
- 2. 支持cnn或者linear classifier训练模型
- 3. 训练完的模型保存到hdfs上

Step 1. 制作self-contained镜像

这里我要创建一个名字是ml-mnist的镜像包含了上述功能,只贴一个Dockerfile的代码上来

FROM registry.cn-hangzhou.aliyuncs.com/mykf/ml-base

RUN pip install pyhdfs

```
RUN pip install tensorflow==1.7.0

ADD src/model.py /
# ADD data /tmp/data/

ENTRYPOINT ["python", "/model.py"]
```

Dockerfile中的model.py是训练的代码,和普通的TensorFlow代码没啥区别,原始代码在这里。改造后的代码增加了从hdfs下载训练数据、上传模型文件的功能,相关代码如下:

```
## 这里简单起见使用webhdfs的api, 生产使用的话还是使用native的api
def download_data_from_hdfs(client, input_data_dir):
 if not os.path.exists(input data dir):
    os.makedirs(input_data_dir)
 for data_file in client.listdir(input_data_dir):
    path = input_data_dir + '/' + data_file
    status = client.get_file_status(path)
    if status.type == 'DIRECTORY':
      download data from hdfs(client, path)
      client.copy_to_local(path, path)
 print("Download data from hdfs " + path)
def upload data to hdfs(client, output data dir):
  if not client.exists(output_data_dir):
    client.mkdirs(output data dir)
 for data_file in os.listdir(output_data_dir):
    path = output_data_dir + '/' + data_file
    if os.path.isdir(path):
      upload_data_to_hdfs(client, path)
    else:
      client.copy_from_local(path,path)
 print("Upload model to hdfs " + path)
```

另外看一下参数接收的地方,所有这里定义的参数都可以通过container的启动命令传入,也和普通的python代码没有区别

```
def parse_arguments():
 parser = argparse.ArgumentParser()
 parser.add argument('--webhdfs-hosts',
                      type=str,
                      help='hdfs host:port')
 parser.add_argument('--tf-data-dir',
                      type=str,
                      default='/tmp/data/mnist',
                      help='hdfs path')
 parser.add_argument('--tf-model-dir',
                      type=str,
                      help='hdfs path or local directory.')
 parser.add_argument('--tf-export-dir',
                      type=str,
                      default='/tmp/model/mnist',
                      help='hdfs path to export model')
```

```
parser.add_argument('--tf-model-type',
                    type=str,
                    default='CNN',
                    help='Tensorflow model type for training.')
parser.add_argument('--tf-train-steps',
                    type=int,
                    default=200,
                    help='The number of training steps to perform.')
parser.add_argument('--tf-batch-size',
                    type=int,
                    default=100,
                    help='The number of batch size during training')
parser.add_argument('--tf-learning-rate',
                    type=float,
                    default=0.01,
                    help='Learning rate for training.')
args = parser.parse_args()
return args
```

我是用阿里云的容器镜像服务构建的镜像,提交代码后自动触发构建。公司内部的话棱镜应该也是可以做到这一点的,作为算法开发的人员来说也不用关心具体怎么构建镜像,定义好代码结构规范、数据目录规范、模型目录规范这些后,只管写代码模型代码就行了。

为了说明和验证这个镜像是self-contaned的,可以直接运行这个镜像

```
docker run --add-host spark-docker:10.28.47.211 registry.cn-
hangzhou.aliyuncs.com/mykf/ml-mnist && \
    --webhdfs=10.28.47.211:9870 && \
    --tf-data-dir=/tmp/data/mnist && \
    --tf-export-dir=/tmp/model/mnist
```

提前将训练数据放到hdfs的/tmp/data/mnist目录,执行结束后会在hdfs的/tmp/model/mnist目录中生成模型文件。

Step 2. 创建Component

有了镜像之后,就可以使用制作好的镜像创建component,有两种方式

- 1. 在定义pipeline的时候,使用kfp定义component
- 2. 提前通过component.yml描述文件定义

第一种方式的component并不能重用,使用代码如下:

```
import kfp.dsl as dsl
...

train = dsl.ContainerOp(
   name='train',
   image='registry.cn-hangzhou.aliyuncs.com/mykf/ml-mnist', ## 这里就是使用Step 1制作的镜像
   arguments=[
```

```
"--tf-export-dir", model_export_dir,
    "--tf-train-steps", train_steps,
    "--tf-batch-size", batch_size,
    "--tf-learning-rate", learning_rate
]
)
```

第二种方式和第一种没有啥本质区别,就是一个描述文件告诉kubeflow怎么运行这个component, 使用哪个镜像,有哪些入参和出参,这样就可以分享给别人使用,具体的规范参考这里。

component.yml

```
name: ml-mnist
description: ml-mnist
inputs:
- {name: Webhdfs hosts, type: String, description: 'Webhdfs hosts'}
- {name: Training data input hdfs dir, type: String, description: 'hdfs input data dir'}
- {name: TF model dir, type: String, description: 'tf model dir'}
- {name: Model data output hdfs dir, type: String, description: 'hdfs output data dir'}
- {name: TF model type, type: String, description: 'CNN or LINEAR'}
- {name: Train steps, type: Integer, description: 'The number of training steps to
perform'}
- {name: Train batch size, type: Integer, description: 'The number of batch size during
training'}
- {name: Train learning rate, type: Float, description: 'Learning rate for training'}
- {name: Model data output hdfs dir, type: String, description: 'hdfs output data dir'}
implementation:
  container:
    image: registry.cn-hangzhou.aliyuncs.com/mykf/ml-mnist
    command: [python, /model.py]
   args: [
     --webhdfs-hosts,
                         {inputValue: Webhdfs hosts},
                       {inputValue: Training data input hdfs dir},
      --tf-data-dir,
     --tf-model-dir, {inputValue: TF model dir},
     --tf-export-dir, {inputValue: Model data output hdfs dir},
      --tf-model-type,
                         {inputValue: TF model type},
      --tf-train-steps, {inputValue: Train steps},
     --tf-batch-size,
                         {inputValue: Train batch size},
     --tf-learning-rate, {inputValue: Train learning rate}
    fileOutputs:
     Model data output hdfs dir: /tmp/model/mnist
```

有了这个component.yml之后,在定义pipeline的时候可以通过如下代码加载component operation:

```
## 从本地文件加载operation
dummy_op = kfp.components.load_component_from_file(os.path.join(component_root,
'component.yaml'))
## 从远程地址加载
confusion_matrix_op =
```

components.load_component_from_url('https://raw.githubusercontent.com/kubeflow/pipelines/
1f65a564d4d44fa5a0dc6c59929ca2211ebb3d1c/components/local/confusion_matrix/component.yaml
')

在实际使用过程中,component描述文件随镜像代码一起发布,在镜像build完成并发布到内网的 registry之后,就可以在任意代码中加载这个 component.yaml 文件,也就达到了重用的目的。