可重复代码和模型方法

我们现在已经到了可以将模型/训练代码分发给他人使用的阶段。您可以用软盘分发或与他人共享代码，但这并不理想。是这样吗？也许很多年前，这是理想的做法，但现在不是了。与他人共享代码和协作的首选方式是使用源代码管理系统。Git 是最流行的源代码管理系统之一。那么，假设你已经学会了 Git，并正确地格式化了代码，编写了适当的文档，还开源了你的项目。这就够了吗？不，还不够。因为你在自己的电脑上写的代码，在别人的电脑上可能会因为各种原因而无法运行。因此，如果您在发布代码时能复制自己的电脑，而其他人在安装您的软件或运行您的代码时也能复制您的电脑，那就再好不过了。为此，如今最流行的方法是使用 Docker 容器（Docker Containers）。要使用 Docker 容器，你需要安装 Docker。

让我们用下面的命令来安装 Docker。

```
sudo apt install docker.io
sudo systemctl start docker
sudo systemctl enable docker
sudo groupadd docker
sudo usermod -aG docker $USER
```

这些命令可以在 Ubuntu 18.04 上运行。Docker 最棒的地方在于它可以安装在任何机器上：Linux、Windows、OSX。因此，如果你一直在 Docker 容器中工作，哪台机器都没关系！

Docker 容器可以被视为小型虚拟机。你可以为你的代码创建一个容器，然后每个人都可以使用和访问它。让我们看看如何创建可用于训练模型的容器。我们将使用在自然语言处理一章中训练的 BERT 模型，并尝试将训练代码容器化。

首先，你需要一个包含 python 项目需求的文件。需求包含在名为 requirements.txt 的文件中。文件名是 thestandard。该文件包含项目中使用的所有 python 库。也就是可以通过 PyPI (pip) 下载的 python 库。用于 训练 BERT 模型以检测正/负情感，我们使用了 torch、transformers、tqdm、scikit-learn、pandas 和 numpy。 让我们把它们写入 requirements.txt 中。你可以只写名称，也可以包括版本。包含版本总是最好的，这也是你应该做的。包含版本后，可以确保其他人使用的版本与你的版本相同，而不是最新版本，因为最新版本可能会更改某些内容，如果是这样的话，模型的训练方式就不会与你的相同了。

下面的代码段显示了 requirements.txt。

```
# requirements.txt
pandas==1.0.4
scikit-learn==0.22.1
torch==1.5.0
transformers==2.11.0
```

现在，我们将创建一个名为 Dockerfile 的 Docker 文件。没有扩展名。Dockerfile 有几个元素。让我们来看看。

```
# Dockerfile
# First of all, we include where we are getting the image
# from. Image can be thought of as an operating system.
# You can do "FROM ubuntu:18.04"
# this will start from a clean ubuntu 18.04 image.
# All images are downloaded from dockerhub
# Here are we grabbing image from nvidia's repo
# they created a docker image using ubuntu 18.04
# and installed cuda 10.1 and cudnn7 in it. Thus, we don't have to
# install it. Makes our life easy.
FROM nvidia/cuda:10.1-cudnn7-runtime-ubuntu18.04
# this is the same apt-get command that you are used to
# except the fact that, we have -y argument. Its because
# when we build this container, we cannot press Y when asked for
RUN apt-get update && apt-get install -y \
    git \
    curl \
    ca-certificates \
    python3 \
    python3-pip \
    sudo \
    && rm -rf /var/lib/apt/lists/*
# We add a new user called "abhishek"
# this can be anything. Anything you want it
# to be. Usually, we don't use our own name,
# you can use "user" or "ubuntu"
RUN useradd -m abhishek
# make our user own its own home directory
RUN chown -R abhishek:abhishek /home/abhishek/
# copy all files from this direrctory to a
# directory called app inside the home of abhishek
# and abhishek owns it.
COPY --chown=abhishek *.* /home/abhishek/app/
# change to user abhishek
USER abhishek
RUN mkdir /home/abhishek/data/
# Now we install all the requirements
# after moving to the app directory
# PLEASE NOTE that ubuntu 18.04 image
# has python 3.6.9 and not python 3.7.6
# you can also install conda python here and use that
# however, to simplify it, I will be using python 3.6.9
# inside the docker container!!!!
```

```
RUN cd /home/abhishek/app/ && pip3 install -r requirements.txt
# install mkl. its needed for transformers
RUN pip3 install mkl
# when we log into the docker container,
# we will go inside this directory automatically
WORKDIR /home/abhishek/app
```

创建好 Docker 文件后，我们就需要构建它。构建 Docker 容器是一个非常简单的命令。

```
docker build -f Dockerfile -t bert:train .
```

该命令根据提供的 Dockerfile 构建一个容器。Docker 容器的名称是 bert:train。输出结果如下：

```
❱ docker build -f Dockerfile -t bert:train .
Sending build context to Docker daemon  19.97kB
Step 1/7 : FROM nvidia/cuda:10.1-cudnn7-ubuntu18.04
---> 3b55548ae91f
Step 2/7 : RUN apt-get update && apt-get install -y   git  curl     ca-
certificates     python3 python3-pip     sudo     && rm -rf
/var/lib/apt/lists/*
.
.
.
.
Removing intermediate container 8f6975dd08ba
---> d1802ac9f1b4
Step 7/7 : WORKDIR /home/abhishek/app
---> Running in 257ff09502ed
Removing intermediate container 257ff09502ed
---> e5f6eb4cddd7
Successfully built e5f6eb4cddd7
Successfully tagged bert:train
```

请注意，我删除了输出中的许多行。现在，您可以使用以下命令登录容器。

```
docker run -ti bert:train /bin/bash
```

你需要记住，一旦退出 shell，你在 shell 中所做的一切都将丢失。你还可以在 Docker 容器中使用。

```
docker run -ti bert:train python3 train.py
```

输出情况：

```
Traceback (most recent call last):
File "train.py", line 2, in <module>
import config
File "/home/abhishek/app/config.py", line 28, in <module>
do_lower_case=True
File "/usr/local/lib/python3.6/dist-
packages/transformers/tokenization_utils.py", line 393, in
from_pretrained
return cls._from_pretrained(*inputs, **kwargs)
File "/usr/local/lib/python3.6/dist-
packages/transformers/tokenization_utils.py", line 496, in
_from_pretrained
list(cls.vocab_files_names.values()),
OSError: Model name '../input/bert_base_uncased/' was not found in
tokenizers model name list (bert-base-uncased, bert-large-uncased, bert-
base-cased, bert-large-cased, bert-base-multilingual-uncased, bert-base-
multilingual-cased, bert-base-chinese, bert-base-german-cased, bert-
large-uncased-whole-word-masking, bert-large-cased-whole-word-masking,
bert-large-uncased-whole-word-masking-finetuned-squad, bert-large-cased-
whole-word-masking-finetuned-squad, bert-base-cased-finetuned-mrpc, bert-
base-german-dbmdz-cased, bert-base-german-dbmdz-uncased, bert-base-
finnish-cased-v1, bert-base-finnish-uncased-v1, bert-base-dutch-cased).
We assumed '../input/bert_base_uncased/' was a path, a model identifier,
or url to a directory containing vocabulary files named ['vocab.txt'] but
couldn't find such vocabulary files at this path or url.
```

哎呀，出错了！

我为什么要把错误印在书上呢？

因为理解这个错误非常重要。这个错误说明代码无法找到目录".../input/bert_base_cased"。为什么会出现这种情况呢？我们可以在没有 Docker 的情况下进行训练，我们可以看到目录和所有文件都存在。出现这种情况是因为 Docker 就像一个虚拟机！它有自己的文件系统，本地机器上的文件不会共享给 Docker 容器。如果你想使用本地机器上的路径并对其进行修改，你需要在运行 Docker 时将其挂载到 Docker 容器上。当我们查看这个文件夹的路径时，我们知道它位于名为 input 的文件夹的上一级。让我们稍微修改一下 config.py 文件！

```
# config.py
import os
import transformers
# fetch home directory
# in our docker container, it is
# /home/abhishek
```

```
HOME_DIR = os.path.expanduser("~")
# this is the maximum number of tokens in the sentence
MAX_LEN = 512
# batch sizes is low because model is huge!
TRAIN_BATCH_SIZE = 8
VALID_BATCH_SIZE = 4
# let's train for a maximum of 10 epochs
EPOCHS = 10
# define path to BERT model files
# Now we assume that all the data is stored inside
# /home/abhishek/data
BERT_PATH = os.path.join(HOME_DIR, "data", "bert_base_uncased")
# this is where you want to save the model
MODEL_PATH = os.path.join(HOME_DIR, "data", "model.bin")
# training file
TRAINING_FILE = os.path.join(HOME_DIR, "data", "imdb.csv")
TOKENIZER =
transformers.BertTokenizer.from_pretrained(BERT_PATH,do_lower_case=True )
```

现在，代码假定所有内容都在主目录下名为 data 的文件夹中。

请注意，如果 Python 脚本有任何改动，都意味着需要重建 Docker 容器！因此，我们重建容器，然后重新运行 Docker 命令，但这次要有所改变。不过，如果我们没有英伟达™（NVIDIA®）Docker 运行时，这也是行不通的。别担心，这只是一个 Docker 容器。你只需要做一次。要安装英伟达™（NVIDIA®）Docker 运行时，可以在 Ubuntu 18.04 中运行以下命令。

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key
add -
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-
docker.list | sudo tee /etc/apt/sources.list.d/nvidia-docker.list
sudo apt-get update && sudo apt-get install -y nvidia-container-toolkit
sudo systemctl restart docker
```

现在，我们可以再次构建我们的容器，并开始训练过程：

```
docker run --gpus 1 -v
/home/abhishek/workspace/approaching_almost/input/:/home/abhishek/data/ -
ti bert:train python3 train.py
```

其中，-gpus 1 表示我们在 docker 容器中使用 1 个 GPU，-v 表示挂载卷。因此，我们要将本地目录 /home/abhishek/workspace/approaching_almost/input/ 挂载到 docker 容器中的 /home/abhishek/data/。这一步要花点时间，但完成后，本地文件夹中就会有 model.bin。

这样，只需做一些简单的改动，你的训练代码就已经 "dockerized "了。现在，你可以在（几乎）任何你想要的系统上使用这些代码进行训练。

下一部分是将我们训练好的模型 "提供 "给最终用户。假设您想从接收到的推文流中提取情感信息。要完成这项任务，您必须创建一个 API，用于输入句子，然后返回带有情感概率的输出。使用 Python 构建 API 的最常见方法是使用 Flask，它是一个微型网络服务框架。

```
# api.py
import config
import flask

import time
import torch
import torch.nn as nn
from flask import Flask
from flask import request
from model import BERTBaseUncased

app = Flask(__name__)

MODEL = None

DEVICE = "cuda"
def sentence_prediction(sentence):
    tokenizer = config.TOKENIZER
    max_len = config.MAX_LEN

    review = str(sentence)
    review = " ".join(review.split())

    inputs = tokenizer.encode_plus(
        review,
        None,
        add_special_tokens=True,
        max_length=max_len
    )

    ids = inputs["input_ids"]
    mask = inputs["attention_mask"]
    token_type_ids = inputs["token_type_ids"]

    padding_length = max_len - len(ids)
    ids = ids + ([0] * padding_length)
    mask = mask + ([0] * padding_length)
    token_type_ids = token_type_ids + ([0] * padding_length)
```

```
    ids = torch.tensor(ids, dtype=torch.long).unsqueeze(0)
    mask = torch.tensor(mask, dtype=torch.long).unsqueeze(0)
    token_type_ids = torch.tensor(token_type_ids,
                                  dtype=torch.long).unsqueeze(0)

    ids = ids.to(DEVICE, dtype=torch.long)
    token_type_ids = token_type_ids.to(DEVICE, dtype=torch.long)
    mask = mask.to(DEVICE, dtype=torch.long)

    outputs = MODEL(ids=ids, mask=mask, token_type_ids=token_type_ids)
    outputs = torch.sigmoid(outputs).cpu().detach().numpy()
    return outputs[0][0]

@app.route("/predict", methods=["GET"])
def predict():
    sentence = request.args.get("sentence")
    start_time = time.time()
    positive_prediction = sentence_prediction(sentence)
    negative_prediction = 1 - positive_prediction
    response = {}
    response["response"] = {
        "positive": str(positive_prediction),
        "negative": str(negative_prediction),
        "sentence": str(sentence),
        "time_taken": str(time.time() - start_time),
    }
    return flask.jsonify(response)

if __name__ == "__main__":
    MODEL = BERTBaseUncased()
    MODEL.load_state_dict(torch.load(
        config.MODEL_PATH, map_location=torch.device(DEVICE)
    ))
    MODEL.to(DEVICE)
    MODEL.eval()
    app.run(host="0.0.0.0")
```

然后运行 "python api.py "命令启动 API。API 将在端口 5000 的 localhost 上启动。cURL 请求及其响应示例如下。

```
❱ curl
$'http://192.168.86.48:5000/predict?sentence=this%20is%20the%20best%20boo
k%20ever'
{"response":{"negative":"0.0032927393913269043","positive":"0.99670726","
sentence":"this is the best book
ever","time_taken":"0.029126882553100586"}}
```

可以看到，我们得到的输入句子的正面情感概率很高。输入句子的正面情感概率很高。 您还可以访问 http://127.0.0.1:5000/predict?sentence=this%20book%20is%20too%20complicated%20for%20me。这将再次返回一个 JSON 文件。

```
{
    response: {
        negative: "0.8646619468927383",
        positive: "0.13533805",
        sentence: "this book is too complicated for me",
        time_taken: "0.03852701187133789"
    }
}
```

现在，我们创建了一个简单的应用程序接口，可以用来为少量用户提供服务。为什么是少量？因为这个 API 一次只服务一个请求。gunicorn 是 UNIX 上的 Python WSGI HTTP 服务器，让我们使用它的 CPU 来处理多个并行请求。Gunicorn 可以为 API 创建多个进程，因此我们可以同时为多个客户提供服务。您可以使用 "pip install gunicorn "安装 gunicorn。

为了将代码转换为与 gunicorn 兼容，我们需要移除 init main，并将其中的所有内容移至全局范围。此外，我们现在使用的是 CPU 而不是 GPU。修改后的代码如下。

```
# api.py
import config
import flask
import time
import torch
import torch.nn as nn
from flask import Flask
from flask import request
from model import BERTBaseUncased
app = Flask(__name__)
DEVICE = "cpu"
MODEL = BERTBaseUncased()
MODEL.load_state_dict(torch.load(config.MODEL_PATH,
map_location=torch.device(DEVICE)))
MODEL.to(DEVICE)
```

```
MODEL.eval()
def sentence_prediction(sentence):

    return outputs[0][0]


@app.route("/predict", methods=["GET"])
def predict():

    return flask.jsonify(response)
```

我们使用以下命令运行这个应用程序接口。

```
gunicorn api:app --bind 0.0.0.0:5000 --workers 4
```

这意味着我们在提供的 IP 地址和端口上使用 4 个 Worker 运行我们的 flask 应用程序。由于有 4 个 Worker，我们现在可以同时处理 4 个请求。请注意，现在我们的终端使用的是 CPU，因此不需要 GPU 机器，可以在任何标准服务器/虚拟机上运行。不过，我们还有一个问题：我们已经在本地机器上完成了所有工作，因此必须将其坞化。看看下面这个未注释的 Dockerfile，它可以用来部署这个应用程序接口。请注意用于培训的旧 Dockerfile 和这个 Dockerfile 之间的区别。区别不大。

```
# CPU Dockerfile
FROM ubuntu:18.04
RUN apt-get update && apt-get install -y \
git \
curl \
ca-certificates \
python3 \
python3-pip \
sudo \
&& rm -rf /var/lib/apt/lists/*
RUN useradd -m abhishek
RUN chown -R abhishek:abhishek /home/abhishek/
COPY --chown=abhishek *.* /home/abhishek/app/
USER abhishek
RUN mkdir /home/abhishek/data/
RUN cd /home/abhishek/app/ && pip3 install -r requirements.txt
RUN pip3 install mkl
WORKDIR /home/abhishek/app
```

让我们构建一个新的 Docker 容器。

```
docker build -f Dockerfile -t bert:api
```

当 Docker 容器构建完成后，我们就可以使用以下命令直接运行 API 了。

```
docker run -p 5000:5000 -v
/home/abhishek/workspace/approaching_almost/input/:/home/abhishek/data/ -
ti bert:api /home/abhishek/.local/bin/gunicorn api:app --bind
0.0.0.0:5000 --workers 4
```

请注意，我们将容器内的 5000 端口暴露给容器外的 5000 端口。如果使用 docker-compose，也可以很好地做到这一点。Dockercompose 是一个可以让你同时在不同或相同容器中运行不同服务的工具。你可以使用 "pip install docker-compose "安装 docker-compose，然后在构建容器后运行 "docker-compose up"。要使用 docker-compose，你需要一个 docker-compose.yml 文件。

```
# docker-compose.yml
# specify a version of the compose
version: '3.7'
# you can add multiple services
services:
# specify service name. we call our service: api
api:
# specify image name
image: bert:api
# the command that you would like to run inside the container
command: /home/abhishek/.local/bin/gunicorn api:app --bind
0.0.0.0:5000 --workers 4
# mount the volume
volumes:
-
/home/abhishek/workspace/approaching_almost/input/:/home/abhishek/data/
# this ensures that our ports from container will be
# exposed as it is
network_mode: host
```

现在，您只需使用上述命令即可重新运行 API，其运行方式与之前相同。恭喜你，现在，你也已经成功地将预测 API 进行了 Docker 化，可以随时随地部署了。在本章中，我们学习了 Docker、使用 flask 构建 API、使用 gunicorn 和 Docker 服务 API 以及 docker-compose。关于 docker 的知识远不止这些，但这应该是一个开始。其他内容可以在学习过程中逐渐掌握。 我们还跳过了许多工具，如 kubernetes、bean-stalk、sagemaker、heroku 和许多其他工具，这些工具如今被人们用来在生产中部署模型。"我要写什么？点击修改图 X 中的 docker 容器"？在书中描述这些是不可

行的，也是不可取的，所以我将使用不同的媒介来赞美本书的这一部分。请记住，一旦你对应用程序进行了 Docker 化，使用这些技术/平台进行部署就变得易如反掌了。请务必记住，要让你的代码和模型对他人可用，并做好文档记录，这样任何人都可以使用你开发的东西，而无需多次询问你。这不仅能节省您的时间，还能节省他人的时间。好的、开源的、可重复使用的代码在您的作品集中也非常重要。