CS643 Programming Assignment #2 : Wine Quality Predictions

Christian Carpena
04/28/2022
CS643-852

GitHub Link: https://github.com/cc362/CS643-AWS-ProgAssgn-2
DockerHub Link:
https://hub.docker.com/repository/docker/cc362/aws-cs643-progassgn-2

Abstract:

The objective of this final programming assignment is to create an Apache Spark MLlib application to train a machine learning model in parallel on a cluster composed of four workers and one master. This document details step-by-step procedure on how to set up the cluster, EC2 instances, and docker images. Further, the parallel training steps are specified as well as the steps to run the prediction application on both a single machine without docker and through downloading the docker image on a machine, instantiating a container, and running the container on a single machine. The code can be found on GitHub and the image can be found on Docker Hub. My implementation utilizes Apache Spark and Hadoop for this programming assignment.

Training Setup:

On the AWS Management Console, navigate to Services EC2 Launch Instances Launch Instances. Enter ***5*** for # of instances. Select the "***Ubuntu Server 20.04 LTS… ami-04505e74c0741db8d***." Select the ***t2.large*** type (useful for Docker purposes). **Note:** The t2-large instances allow for less problems relating to running out of memory. During testing, the t2-micro instances gave me a lot of errors relating to the Java runtime environment running out of memory.Select ***Create a new key pair*** and name it ***ProgAssgn2***. Hit ***Download key pair.*** Under ***Network Settings -> Security groups (Firewall),*** check ***Allow SSH traffic from [Anywhere 0.0.0.0/0]***. For ***Configure storage***, configure it from ***8*** GiB to ***16*** GiB (to install and configure modules/packages on the EC2 instances). Keep all the rest of the default options and click ***Launch instance.*** Then ***View all instances***. You will see a **Pending** status for the ***Instance State*** of the EC2 Instances.

While waiting for this to switch to **Running**, open a terminal and move the .pem file you downloaded to your home directory. Run the following command to set the correct permissions for the .pem file:

```
$ chmod 400 ProgAssgn2.pem
```

To connect to your EC2 instance (after it has started running), run the following command in your terminal (replacing <YOUR_INSTANCE_PUBLIC_DNS> with the "Public IPv4 DNS" attribute of the EC2 instance):

```
$ ssh -i ~/ProgAssgn2.pem ubuntu@<YOUR_INSTANCE_PUBLIC_DNS>
```

Now, perform a pull down from the GitHub repository:

```
$ git clone https://github.com/cc362/CS643-AWS-ProgAssgn-2.git
```

Navigate to CS632-AWS-ProgAssgn-2 that was just pulled from GitHub. Copy the two files in that directory to the /app directory (this will be used for Predictions with Docker):

```
$ cd CS643-AWS-ProgAssgn-2
$ sudo cp TrainingDataset.csv ValidationDataset.csv /app/
```

Create a bash script on both the master and the four slave nodes to run commands to install, extract and remove the apache-spark files.

```
$ vi automate.sh
```

Insert the following lines into the bash script:

```
#/bin/bash

sudo apt-get update
sudo apt-get install -y curl vim wget software-properties-common
ssh net-tools ca-certificates
sudo apt install -y default-jre
sudo wget --no-verbose -O apache-spark.tgz
"https://archive.apache.org/dist/spark/spark-3.2.0/spark-3.2.0-b
in-hadoop2.7.tgz"
sudo mkdir -p /opt/spark
sudo tar -xf apache-spark.tgz -C /opt/spark --strip-components=1
sudo rm apache-spark.tgz
```

Change permissions to execute the script:

```
$chmod 755 automate.sh
```

Now, create a script for the master to operate the Apache Spark Cluster. This will only be applicable and for the sole purpose be for the master node:

```
$vi master.sh
```

Insert the following lines into the bash script:

#/bin/bash

```
export SPARK_MASTER_PORT=7077
export SPARK_MASTER_WEBUI_PORT=8090
export SPARK_LOG_DIR=/opt/spark/logs
export SPARK_MASTER_LOG=/opt/spark/logs/spark-master.out
export JAVA_HOME=/usr/bin/java

sudo mkdir -p $SPARK_LOG_DIR
sudo touch $SPARK_MASTER_LOG
sudo ln -sf /dev/stdout $SPARK_MASTER_LOG

export SPARK_MASTER_HOST=`hostname`

cd /opt/spark/bin &&
sudo ./spark-class org.apache.spark.deploy.master.Master --ip
$SPARK_MASTER_HOST --port $SPARK_MASTER_PORT --webui-port
$SPARK_MASTER_WEBUI_PORT >> $SPARK_MASTER_LOG
```

Change permissions to execute the script:

```
$chmod 755 master.sh
```

Create a bash script for the slave nodes to run commands to install, extract and remove the apache-spark files.

```
$ vi worker<x>.sh [where <x> is 1,2,3,4]
```

Insert the following lines into the bash script:

```
#/bin/bash

export SPARK_MASTER_PORT=7077
export SPARK_MASTER_WEBUI_PORT=8080
export SPARK_LOG_DIR=/opt/spark/logs
export SPARK_MASTER_LOG=/opt/spark/logs/spark-master.out
export JAVA_HOME=/usr/bin/java

export SPARK_MASTER_PORT=7077 \
export SPARK_MASTER_WEBUI_PORT=8080 \
export SPARK_LOG_DIR=/opt/spark/logs \
export SPARK_WORKER_LOG=/opt/spark/logs/spark-worker.out \
export SPARK_WORKER_WEBUI_PORT=8080 \
export SPARK_WORKER_PORT=7000 \
export SPARK_MASTER="spark://ip-172-31-25-126:7077" \
export SPARK_LOCAL_IP
```

```
. "/opt/spark/bin/load-spark-env.sh"

sudo mkdir -p $SPARK_LOG_DIR
sudo touch $SPARK_WORDER_LOG
sudo ln -sf /dev/stdout $SPARK_WORKER_LOG

cd /opt/spark/bin
sudo ./spark-class org.apache.spark.deploy.worker.Worker
--webui-port $SPARK_WORKER_WEBUI_PORT $SPARK_MASTER >>
$SPARK_WORKER_LOG
```

Run the automate.sh script to setup the apache-spark clusters packages for both the master and four slave nodes.

```
$./automate.sh
```

Then, run the master.sh script to setup the Apache Spark Clusters, which will be used to open up on the browser. Run the worker<x>.sh scripts to set up the workers as well :

```
$./master.sh [for master node]
$./worker<x>.sh [for worker nodes, where <x> is 1,2,3,4]
```

Training:

Run these pip install commands before proceeding to run through the Training.py script:

```
$sudo apt install python3-pip
$pip install numpy
$pip install pandas
$pip install quinn
$pip install pyspark
$pip install findspark
```

Run the python script 'Training.py' to obtain the output results of both the LogisticRegressionModel and the RandomForestClassifier Model after they are trained on all four nodes.

```
$python3 /home/ubuntu/CS643-AWS-ProgAssgn-2/Training.py
```

After running the Training.py script, here is the output result received from the code:

```
F1 Score for LogisticRegression Model: 0.5729445029855991
F1 Score for RandomForestClassifier Model: 0.5035506965944272
Since the LogisticRegression Model has a higher score than the
RandomForestClassifier Model, we use this one in our prediction
application.
```

Prediction without Docker:

While SSH'ed into your EC2 created in the previous step, run the following command to install and configure Java:

```
$ export JAVA_HOME=/usr/bin/java
```

Run the following commands to install Anaconda:

```
$ cd /tmp
$ curl -O
https://repo.anaconda.com/archive/Anaconda3-2020.11-Linux-x86_64
.sh
$ bash Anaconda3-2020.11-Linux-x86_64.sh
```

At prompt 'Please, press ENTER to continue', press <Enter>

At prompt 'Do you accept the license terms? [yes|no]', type 'yes' and press <Enter>,

Press <Enter> to confirm the location and begin anaconda installation.

At prompt, 'Do you wish the installer to initialize Anaconda3 by running conda init? [yes|no]', type 'yes'

Now, add the following lines of code to the end of your ~/.bashrc file without modifying anything else in the file:

```
function snotebook ()
{
SPARK_PATH=~/opt/spark/spark-3.2.0-bin-hadoop2.7

export PYSPARK_DRIVER_PYTHON="jupyter"
export PYSPARK_DRIVER_PYTHON_OPTS="notebook"

export PYSPARK_PYTHON=python3

$SPARK_PATH/bin/pyspark --master local[2]
}
```

Run the following line of code to load the changes to your .bashrc file:

```
$ source ~/.bashrc
```

Close out of your SSH instance and SSH again into your EC2. Run the following code to configure Jupyter with a password (of your choice when prompted) and start up the Jupyter Notebook (without browser) on the EC2 instance.

```
$ jupyter notebook password
$ jupyter notebook --no-browser
```

Open up a new terminal tab on your local machine and run the following command to setup an SSH tunnel so that you can open the Jupyter Notebook on the EC2:

```
$ ssh -i "ProgAssgn2.pem" -N -f -L localhost:8888:localhost:8888 ubuntu@<YOUR_INSTANCE_PUBLIC_DNS>
```

Now navigate to localhost:8888 in your browser and enter the Jupyter password that you set for your EC2 in the previous step. Once you're in, create a new Python 3 notebook (New -> Python3). In this notebook, paste the code from the prediction.py file (located here) into the first notebook cell. You will see several things printed as the code reads the data, formats it, trains our model, makes the predictions of wine quality for TestDataset, and finally outputs the F1 score.

The highest F1 score I observed (using the ValidationDataset) for the predictions model is: 0.5729445029855991.

Prediction with Docker:

A. Creating Docker Image for Prediction Application

1. Initialize docker on your ec2-instance using these steps.

2. Add the user ubuntu to the 'docker' group

```
$ sudo usermod -aG docker $USER
```

3. Verify user ubuntu has been added to the docker group

```
$ groups
ubuntu adm dialout cdrom floppy sudo audio dip video plugdev netdev lxd docker
```

4. Login with the follow command:

```
$ docker login
```

Proceed to enter your credentials

5. Navigate to the folder where the Dockerfile is saved. This folder should also have the model, the Prediction.py application, and the dataset to utilize for prediction.

```
$ cd /home/ubuntu/CS643-AWS-ProgAssgn-2/
```

6. Build this docker image with:

```
$ docker build -t cc362/aws-cs643-progassgn-2 .
```

7. Verify the docker image has been created after it has been built

```
$ docker images
```

8. Run this image build with:

```
$ docker run -v /app/:/data
cc362/aws-cs643-progassgn-2:latest
```

9. After running the Predictions with Docker, here is the output result received from the code:
```
F1 Score for our Model: 0.562631807944308
```

10. Push this image to Docker Hub with:

```
$ docker push cc362/aws-cs643-progassgn-2:latest
```

11. It can then be seen on Docker Hub through here.