**CS643 Programming Assignment #2 : Wine Quality Predictions**

**Name: Vedant Rakesh Abrol**

**CS643**

**GitHub Link:** <https://github.com/vedant-abrol/CS643-pa2>

**DockerHub Link:**

<https://hub.docker.com/r/va398/cs643-prog-assgn-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**

1. **On the AWS Management Console**, navigate to **Services → EC2 → Launch Instances**.
2. **Enter 5 for the number of instances**. Select the “Ubuntu Server 20.04 LTS…” with the AMI ID ami-04505e74c0741db8d. Select the t2.large instance type (useful for Docker purposes).
   * **Note**: The t2.large instances allow for fewer issues related to running out of memory. During testing, using t2.micro instances led to memory-related errors with the Java runtime environment.
3. **Select Create a new key pair** and name it ProgAssgn2. Click **Download key pair**.
4. Under **Network Settings → Security groups (Firewall)**, check **Allow SSH traffic from [Anywhere 0.0.0.0/0]**.
5. For **Configure storage**, increase it from 8 GiB to 16 GiB (to install and configure modules/packages on the EC2 instances). Keep all other options as default and click **Launch Instance**.
6. 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>

Perform a pull from the GitHub repository:  
 git clone https://github.com/va398/CS643-AWS-ProgAssgn-2.git

Navigate to the CS643-AWS-ProgAssgn-2 directory 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 Bash Scripts for Apache Spark Setup**

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-bin-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 node** to operate the Apache Spark Cluster:  
  
 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, or 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\_WORKER\_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 set up the Apache Spark clusters packages for both the master and the four slave nodes:  
  
 ./automate.sh

Then, run the **master.sh** script to set up the Apache Spark Clusters for the master node:  
  
 ./master.sh [for master node]

Run the worker<x>.sh scripts to set up the workers as well:  
  
 ./worker<x>.sh [for worker nodes, where <x> is 1, 2, 3, or 4]

### **Training**

Run these pip install commands before proceeding to run 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

1. 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**
2. Since the LogisticRegression Model has a higher score than the RandomForestClassifier Model, we will 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
  + At prompt ‘Do you accept the license terms? [yes|no]’, type ‘yes’ and press .
  + Press 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 set up 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>

1. 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 the model, makes predictions of wine quality for TestDataset, and finally outputs the F1 score.
2. The highest F1 score I observed (using the ValidationDataset) for the predictions model is: **0.5729445029855991**.

### **Prediction with Docker**

#### **Creating Docker Image for Prediction Application**

1. Initialize Docker on your EC2 instance using these steps.

Add the user ubuntu to the ‘docker’ group:  
  
 sudo usermod -aG docker $USER

Verify the user ubuntu has been added to the docker group:  
  
 groups

* + ubuntu adm dialout cdrom floppy sudo audio dip video plugdev netdev lxd docker

Login with the following command:  
  
 docker login

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/

Build this Docker image with:  
  
 docker build -t va398/aws-cs643-progassgn-2 .

Verify the Docker image has been created after it has been built:  
  
 docker images

Run this image build with:  
  
 docker run -v /app/:/data va398/aws-cs643-progassgn-2:latest

1. After running the predictions with Docker, here is the output result received from the code:  
    **F1 Score for our Model**: **0.562631807944308**
2. Push this image to Docker Hub.
3. It can then be seen on Docker Hub.