Machine Learning with Spark

Brock Wilson
Dept. of Electrical and Computer Engineering
University of Texas at San Antonio
San Antonio, Texas, USA
brock.wilson@my.utsa.edu

Jim Foster
Dept. of Computer Science
University of Texas at San Antonio
San Antonio, Texas, USA
jim.foster@my.utsa.edu

Robert A. Klar
Dept. of Electrical and Computer Engineering
University of Texas at San Antonio
San Antonio, Texas, USA
robert.klar@my.utsa.edu

***Abstract*—Apache Spark is an open-source data processing framework that can quickly perform tasks on large datasets and distribute computing across multiple nodes. These attributes make it well suited for Machine Learning (ML) applications. It overcomes limitations of other distributed frameworks like MapReduce, for example. Spark includes the ML and MLlib libraries that provide a wide range of learning algorithms. As part of this project, we provisioned several Amazon Elastic Compute Cloud (EC2) instances and used them with Spark to investigate some ML applications. We evaluated several ML classifiers (decision tree, random forest, and multilayer perceptron neural network) using the well-known Iris dataset [1,2]. This paper summarizes our efforts.**

Keywords—Spark, Machine Learning, Cloud Computing

I. Introduction (*Heading 1*)

Apache Spark began as a project at the University of California at Berkeley (UCB) in 2009 and has been refined over the past decade and has quickly become one the most popular distributed data frameworks in the world [3]. It is used by many companies, large and small, for daily business. Spark can be deployed quickly and supports bindings for popular computing languages including Java, Scala, Python and R. It includes a rich set of libraries to support database manipulation, streaming, graph processing, and machine learning (ML).

ML with Spark offers many advantages, including distributed computing for handling large datasets, scalability, and parallel processing. Spark includes two libraries (ML and MLlib) that provide support for a wide range of algorithms, making it convenient for ML tasks such as classification, regression, and clustering. Spark's capability to do in-memory processing makes it possible to perform iterative algorithms efficiently, crucial for many machine learning tasks.

In this project, we implemented several ML classifiers (decision tree, random forest, and multilayer perceptron neural network) using the Python interface to Spark and the well-known Iris dataset [1,2]. For our experimentation, we provisioned several Amazon Elastic Compute Cloud (EC2) instances.

Since this was a group project, we shared in the responsibilities to complete the required work. Robert Klar created an account with Amazon Web Services (AWS), provisioned the EC2 instances, and installed the Spark software. Jim Foster created a classifier for the multilayer perceptron and tested it on the virtual machines (VMs). Brock Wilson created classifiers based on tree-based approaches and tested them on the VMs. Each of us contributed to the writing of this report. In the sections that follow, we elaborate on the effort and describe some of our results from testing.

II. Provisioning Cloud Resources

To make use of the AWS resources, it was necessary to sign up for an account. AWS offers tiers of service, including a "Free" tier that allows limited use of the cloud computing services that they provide. A credit card is required for registration so that any deviations from the base service can be billed.

After creating and verifying your account via email, it is possible to log in to the AWS web interface console. From the console, there is a block titled "Build a solution" with a subtopic titled "Launch a virtual machine". By clicking on this, one can navigate to the EC2 console that allows for provisioning of EC2 instances. By default, we were routed to the "U.S. East" AWS datacenter located in Ohio.

The process to provision an EC2 instance was relatively simple and straightforward. First, one must provide a name for the instance. In our case, we simply named our instances aws-group13-1, aws-group13-2, and aws-group13-3.

Next, one chooses an Application Machine Image (AMI). For the first instance, we picked an AMI for Ubuntu 22.04 LTS since it was available, and we were familiar with it. For subsequent instances, we were able to make use of an AMI that we saved off that was already configured with Hadoop and Spark.

Next, one chooses the instance type. Initially, we selected the t2.micro option, since this is the only free one - it has a single processor and 1 Gibibyte (GiB) of memory. We later found that this caused some undesirable behaviors lock up behaviors while running our Spark applications. We converted them first to t2.small and later to t2 large to avoid such issues. The t2.large instance type has 1 processor and 8 GiB of memory.

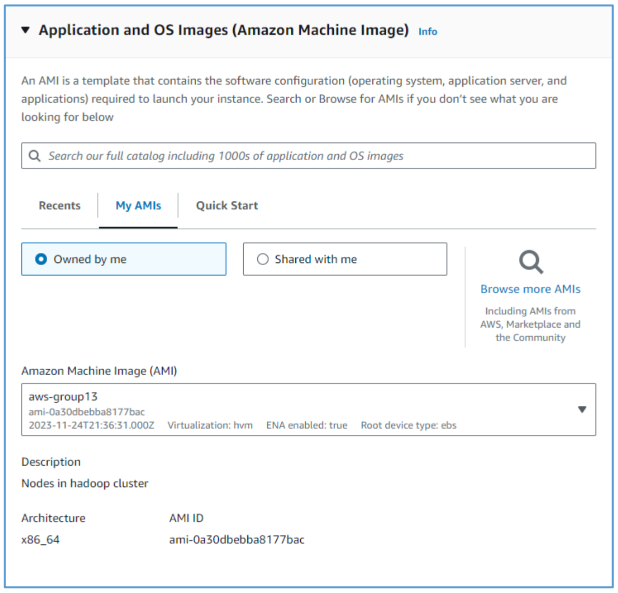
After configuring the instance type, the next step is to set up an encryption key pair that can be used to access the instances remotely via their public Internet Protocol (IP) addresses. We created one key pair that we used for all instances.

Next, one configures network settings. We chose to let AWS automatically assign public and private IP addresses. We used a single security group that allowed access to Transmission Control Protocol (TCP) ports 22 and 8080. The latter port, 8080, was used to test a simple web server application to validate connectivity.

Lastly, the storage is configured for the instance. In our case we simply allocated the default 8 GiB for each. This kept us from incurring additional charges as the "Free" tier only allows for a total for 30 GiB of general-purpose storage.

After the first instance (aws-group13-1) was created, the Hadoop 1.2.1 and Spark 1.6.1 software was installed. From the "EC2 | Instances" page on the EC2 console, an AMI image was created by selecting "Actions | Images and Templates | Create Image". This was then used later to create near duplicates for the other instances. AWS allows you to create an AMI without stopping the source instance.

Figure 1 shows the configuration block for the AMI selection with our common image selected.


▼ Application and OS Images (Amazon Machine Image) Info
Search our full catalog including 1000s of application and OS images
Recents
My AMIs
Quick Start
Owned by me
Shared with me
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Amazon Machine Image (AMI)
aws-group13
ami-0a30dbebba8177bac
2023-11-24T21:36:31.000Z Virtualization: hvm ENA enabled: true Root device type: ebs

Description
Nodes in hadoop cluster

Architecture	AMI ID
x86_64	ami-0a30dbebba8177bac

Figure 1. An example of block from the EC2 console

Once instances were created on the AWS cloud, an initial test was done using the Assignment 2 program from this class to validate the installation.

III. Creation Of Classifiers

We evaluated several ML classifiers on the Iris dataset including decision tree, random forest, and multilayer perceptron neural network.

A. Decision Tree

Decision Trees are a popular classification method because of its ease of use, high performance, and ability to capture non-linearities and feature interactions [4,5]. The method is a greedy algorithm that performs recursive binary partitioning of the feature space. Its purpose is to select the best split from a set of possible splits, to maximize the information gain at a node. As implemented, our Decision Tree model used Gini Impurity, a max depth of 5, and max bins of 32. Gini Impurity can be defined by

$$\sum_{i=1}^{C} f_1(1 - f_1) \tag{1}$$

Where f_1 is the frequency of label i at a node, and C is the number of unique labels. The maxBins parameter for this model corresponds to increasing possible splits for more "fine-grained split decisions". The maxDepth parameter for this corresponds to the maximum depth (or height) of the tree.

B. Random Forest

Random Forests should be straightforward to understand if you grasp Decision Trees because they are just an ensemble of Decision Trees. This means that they train sets of Decision Trees separately and inject randomness into training, so each Decision Tree's outcome is mildly different. Randomness in this context can be defined as subsampling original dataset to get different training set each iteration or considering different random subsets of features to split at each tree node. Random Forest then combined the predictions from each Decision Tree to reduce overall prediction variance. To classify a sample each individual Tree gives a prediction then whatever is majority vote is the overall prediction.

C. Multilayer Perceptron Neural Network

A Multilayer Perceptron (MLP) Classifier is a feedforward neural network classification model [6]. The MLP model consists of layers of nodes fully connected to the next layer in the network. An MLP model begins with an input layer. This layer must have the same number of nodes as input features. Then the model will contain at least one hidden layer. These hidden layers are made up of nodes that map their inputs to their outputs through a linear combination of the inputs, node weights, and node biases, and then applying a sigmoid activation function defined by

$$f(z_i) = \frac{1}{1+e^{-z_i}} \tag{2}$$

Finally, there is an output layer. This layer must have the same number of nodes as sample labels. Similar to the hidden layers, the output layer performs a linear combination, but applies a softmax activation function instead of sigmoid function. The softmax activation function is defined by

$$f(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{N} e^{z_k}} \tag{3}$$

Where N is the number of nodes in the output layer. As implemented, our MLP Classifier model has a 4-node input layer, two hidden layers consisting of 5 and 4 nodes respectively, and a 3-node output layer. The model is trained over 100 iterations, with a block size of 128. The blockSize parameter for this model corresponds to the size of the partitions in which input data as matrices are stacked.

IV. Testing Of Classifiers

We ran several trials of each classifier. Each model was used to classify the Iris dataset [1] in a random data split where 60% (90) of the samples were used for training, and 40% (60) of the samples were used for testing. The dataset's four features are all continuous, and there are three different classes. The training time for our models was recorded for each run, and the metrics used for evaluation were Precision, Recall, and F1.

A. *Decision Tree*

The results of our Decision Tree model's runs can be seen in Table I. Each run of the Decision Tree model will generate a Decision Tree Diagram, like the one shown in Figure 2. This Diagram is what the model generated during training, and later used in testing to predict the classes of the test samples.

TABLE I. DECISION TREE RESULTS

Run #	Model Evaluation Metrics			
	Precision	***Recall***	***F1***	***Time (s)***
1	94.23	94.23	94.22	2.92
2	98.25	98.25	98.24	2.99
3	86.89	86.89	87.01	3.09
4	96.92	96.92	96.92	3.03
5	93.62	93.62	93.62	3.00

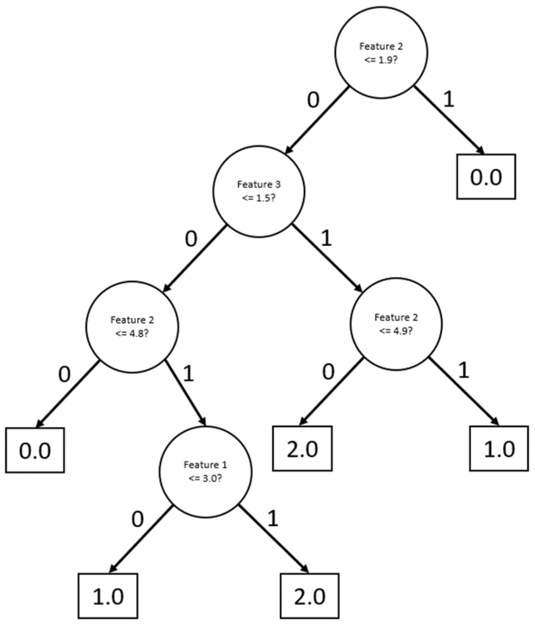


Figure 2. Decision Tree Resultant Diagram

B. *Random Forest*

The results of our Random Forest model's runs can be seen in Table II.

TABLE II. RANDOM FOREST RESULTS

Run #	Model Evaluation Metrics			
	Precision	***Recall***	***F1***	***Time (s)***
1	96.67	96.67	96.67	3.42
2	96.49	96.49	96.47	3.54
3	96.23	96.23	96.23	3.43
4	93.85	93.85	93.91	3.27
5	100.00	100.00	100.00	3.28

In most of the Random Forest model's runs, 20 trees (similar to that shown in Figure 2) were generated for each run. Despite this, the training time between the Decision Tree model and Random Forest model is at most half a second longer.

Another interesting element to note in Table II is the scores of the 5th run. In this run, the Random Forest model was able to classify all 60 test samples correctly. The size of the dataset is partly responsible for this perfect run. The Iris dataset is limited in sample size, consisting of only 150 total samples. A much larger dataset would likely cause this model to produce multiple false classifications based on a greater chance that the dataset contains a greater number of samples with varied features.

C. *Multilayer Perceptron Neural Network*

The results of our MLP model's runs can be seen in Table III.

TABLE III. MLP RESULTS

Run #	Model Evaluation Metrics			
	Precision	***Recall***	***F1***	***Time (s)***
1	63.0769	63.0769	51.3287	7.328815
2	59.7015	59.7015	46.806	7.131229
3	62.9032	62.9032	51.1265	7.56553
4	67.2414	67.2414	56.5204	7.58692
5	94.9153	94.9153	94.7818	11.17737
6	98.1481	98.1481	98.1326	10.97613
7	96.875	96.875	96.8561	10.35164
8	94.3396	94.3396	94.3754	11.83687
9	92.5373	92.5373	92.2428	11.36455

Figure 3 and Figure 4 provide alternative visualizations of the data presented in Table III. From these figures we can determine a few noteworthy characteristics of the data.

First, a correlation between training time and metric scores is clearly shown. The first four runs of the MLP model produced Precision and Recall scores of around 0.6, and F1 scores of around 0.5. These same runs only trained for under 8 seconds. Meanwhile, the subsequent five runs of the MLP model produced Precision, Recall, and F1 scores all over 0.9, and took around 11 seconds to train.

Second, the MLP model performed best on its 7th run. It should first be noted that the metric scores for the 6th run are the highest, at just above 98. Meanwhile, the metric scores for the 7th run are at just under 97. However, the 6th run took half a second longer than the 7th run. Thus, the 7th has the optimal training time-to-correctness ratio.

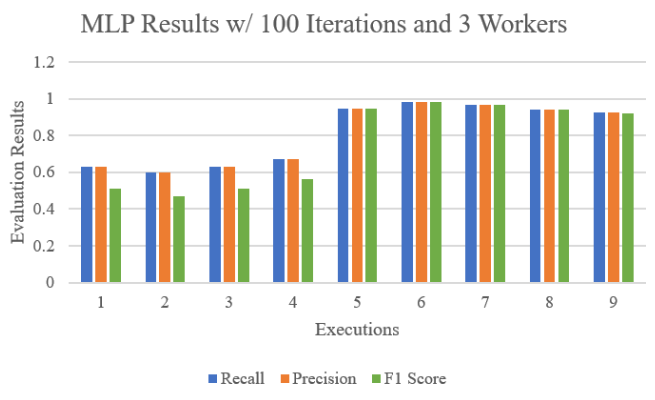


Figure 3. MLP Classifier Evaluation Results

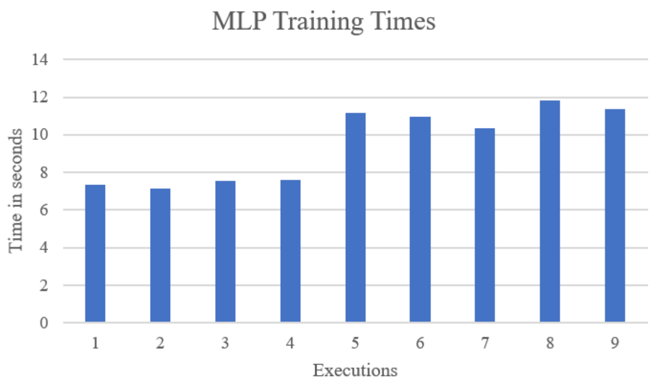


Figure 4. MLP Classifier Training Times

For the Iris dataset, MLP completed quickly. For 100 iterations, we achieved good ML performance with only a few seconds of compute time.

V. CONCLUSION

Spark and its supporting ML libraries make it possible to perform iterative tasks such as model training efficiently. This makes it very powerful when combined with rapidly scalable cloud computing resources.

REFERENCES

[1] Iris Dataset. University of California, Irvine. Donated June 1988. https://archive.ics.uci.edu/dataset/53/iris

[2] A. Unwin and K. Kleinman, 'The iris data set: In search of the source of virginica', Significance, vol. 18, 2021.

[3] I. Pointer, "What is Apache Spark? The big data platform that crushed Hadoop." InfoWorld. March 2023. https://www.infoworld.com/article/3236869/what-is-apache-spark-the-big-data-platform-that-crushed-hadoop.html

[4] Decision Trees – spark.mllib (apache.org) https://spark.apache.org/docs/1.6.1/mllib-decision-tree.html

[5] Ensembles – spark.mllib https://spark.apache.org/docs/1.6.1/mllib-ensembles.html

[6] Classification and regression - spark.ml (apache.org) https://spark.apache.org/docs/1.6.1/ml-classification-regression.html#multilayer-perceptron-classifier