Programming Assignment 5: Giraph and MLlib

1 Overview

Welcome to the Giraph and MLlib Machine Practice. This assignment builds on **Tutorial 5: Giraph** and **Tutorial 5: MLlib**. It is highly recommended that you practice these tutorials before starting the assignment.

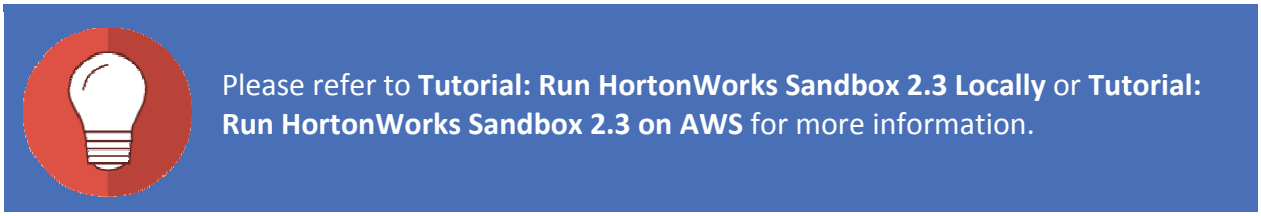
We have four goals in this assignment:

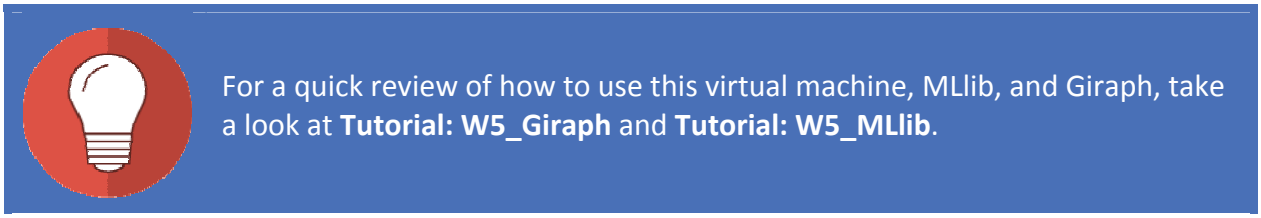
1. Find connected components of a given graph
2. Build a KMeans Clustering model for a data set
3. Find the shortest path of nodes in a given graph
4. Build a classifier model based on Random Forest for a data set

2 Requirements

All these assignments are designed to work and will be graded based on the **Hortonworks Sandbox 2.3** virtual machine. You need to have a working HortonWorks Sandbox machine either locally or on the Amazon Web Services.

Also, all assignments are designed based on **JDK 7** (included in the virtual machine).





3 Setup

Step 1: Start the virtual machine, and then connect to it through the SSH.

Step 2: After successfully logging in, you should see a prompt similar to the following:

```
[root@sandbox ~]#
```

Step 3: Download the following assignments files:

```
# git clone https://github.com/przybylek/GiraphAndMllib.git
```

(originally https://github.com/xldrx/cloudapp-mp5.git)

Step 4: Change the current folder to:

```
# cd GiraphAndMllib/
```

This folder includes some boilerplate files which you are going to fill in while completing the exercises. All the parts that need to be implemented are marked as “TODO”.

Step 6: Initialize the assignment environment using the following command:

```
# bash start.sh
```

Step 7: Finish the exercises by editing the provided template files. All you need to do is complete the parts marked with **TODO**.

- Each exercise has a Java code template. All you need to do is edit this file.
- Each exercise should be implemented in one file only. Multiple file implementation is not allowed.
- Only standard JDK 7, MLlib and Giraph libraries can be used. In other words, the code should be compiled and run on a vanilla HortonWorks Sandbox VM 2.3.0.

More information about the exercises is provided below.

Exercise A: Connected Components of Giraph

In this exercise, you are going to build a Giraph application which finds Connected Components in a graph. This first exercise is similar to **Tutorial 5: Introduction to Giraph**.

In this exercise, we are going to use the "ConnectedComponentsComputation" class to label nodes. These components are exactly the same as in the tutorial.
All you need to do for this exercise is to fill in the missing parts of the code, which is exactly the same as in the tutorial. To make things easier, we have provided a boilerplate in the following file:

```
src/ ConnectedComponentsComputation.java
```

NOTE input: Data input format is similar to the format used with the Connected Components part of **Tutorial 5: Introduction to Giraph in that** each line starts with the node_id, which is then followed by its neighbor nodes, for instance:

```
0 3 4 5 6
1 2
2 1
3 0
…
```

Note output: Each line in your output will correspond to one node and the "connected component id", for example:

```
0 0
1 1
2 1
3 0
…
```

Node code: All you have to do is to complete the parts marked as "TODO".

After completing the implementation of this file, you have to build the application using the command below from the "cloudapp-mp5" directory:

```
# mvn clean package
```

Next, you have to run the application and store the output using the command below from the "cloudapp-mp5" directory:

```
# hadoop jar target/mp5-1.0-SNAPSHOT-jar-with-dependencies.jar
org.apache.giraph.GiraphRunner ConnectedComponentsComputation -vif
org.apache.giraph.io.formats.IntIntNullTextInputFormat -vip
/mp5/data/graph.data  -vof
org.apache.giraph.io.formats.IdWithValueTextOutputFormat -op
/mp5/output/part-a -w 1 -ca giraph.SplitMasterWorker=false
```

Alternatively use the following command:

```
# bash run_a.sh
```

Note: By running the above command, the output will be stored on HDFS at:

```
/mp5/output/part-a
```

For debugging purposes, please check the folder on HDFS.
Here is **a part** of a sample output of this application:

```
...
1    0
9     7
...
```

Exercise B: KMeans Clustering Model

In this exercise, you are going to build a clustering model application using MLlib, which clusters samples in a give data set. This exercise is similar to **Tutorial 5: Introduction to MLlib**.

All you need to do for this exercise is to fill in the missing parts of the code, which is exactly the same as in the tutorial. To make things easier, we have provided a boilerplate in the following file:

```
src/ KMeansMP.java
```

NOTE input: Data input format is similar to the format used in the K-means portion of **Tutorial 5: Introduction to MLlib** in that each line starts with the sample_id which is then followed its features, for instance:

```
…
"Cadillac Fleetwood",10.4,8,472,205,2.93,5.25,17.98,0,0,3,4
"Lincoln Continental",10.4,8,460,215,3,5.424,17.82,0,0,3,4
…
```

Note output: Each line in your output will correspond to one node cluster and samples which fit into that cluster, for example:

```
…
(0,["Datsun 710", "Merc 240D", "Merc 230", "Toyota Corona", "Porsche
914-2", "Lotus Europa", "Volvo 142E"])
…
```

Variable “results” contains cluster number and then iterable members:

```
JavaPairRDD<Integer, Iterable<String>> results;
```

Note code: All you have to do is to complete the parts marked as “TODO”. Please keep the following parameters as they are and only fill out the //TODO section:

```
        int k = 4;
        int iterations = 100;
        int runs = 1;
        long seed = 0;
```

These parameters correspond to the number of the cluster, the number of iterations the algorithm has to go through, and the seed point for the random number.

After completing the implementation of this file, you have to build the application using the command below from the “cloudapp-mp5” directory:

```
# mvn clean package
```

Next, you have to run the application and store the output using the command below from the "cloudapp-mp5" directory:

```
# spark-submit --class KMeansMP target/mp5-1.0-SNAPSHOT-jar-with-
dependencies.jar /mp5/data/cars.data /mp5/output/part-b/
```

Alternatively, use the following command:

```
# bash run_b.sh
```

Note: By running the above command, the output will be stored on HDFS at:

```
/mp5/output/part-b
```

For debugging purposes, please check the folder on HDFS.
Here is **a part** of a sample output of this application:

```
...
(0,["Datsun 710", "Merc 240D", "Merc 230", "Toyota Corona", "Porsche
914-2", "Lotus Europa", "Volvo 142E"])
...
```

Exercise C: Shortest Paths in Giraph

In this exercise, you are going to implement a shortest path algorithm in Giraph. To make the implementation easier, we have provided a boilerplate in the following file:

```
src/ ShortestPathsComputation.java
```

All you need to do is to make the necessary changes in the file by implementing the sections marked as “TODO”.

We are going to test this application on a graph data set that is stored in the file “graph.txt”.

NOTE input: The data input format is similar to the format used with the Connected Components part of **Tutorial 5: Introduction to Giraph** in that each line starts with the node_id which is then followed by its neighbor nodes, for instance:

```
0 3 4 5 6
1 2
2 1
3 0
…
```

Note output: Each line in your output will correspond to one node and its shortest path length, for example:

```
...
0     2
8     2147483647
...
```

Note that if there is no path between two nodes you should return a big int, in JAVA:

```
Integer.MAX_VALUE
```

After completing the implementation of this file, you have to build the application using the command below from the “cloudapp-mp5” directory:

```
# mvn clean package
```

Next, you have to run the application and store the output using the command below from the "cloudapp-mp5" directory:

```
# hadoop jar target/mp5-1.0-SNAPSHOT-jar-with-dependencies.jar
org.apache.giraph.GiraphRunner ShortestPathsComputation -vif
org.apache.giraph.io.formats.IntIntNullTextInputFormat -vip
/mp5/data/graph.data  -vof
org.apache.giraph.io.formats.IdWithValueTextOutputFormat -op
/mp5/output/part-c -w 1 -ca giraph.SplitMasterWorker=false -ca
SimpleShortestPathsVertex.sourceId=3
```

Alternatively, use the following command:

```
# bash run_c.sh
```

Note that this command assumes you are giving the input file name as an input argument. If needed, you can change the command accordingly.

Note: By running above command, the output will be stored on HDFS at:

```
/mp5/output/part-c
```

For debugging purposes, please check the folder on HDFS.
Here is **a part** of a sample output of this application:

```
...
0     2
8     2147483647
...
```

Exercise D: Random Forest Classifier

In this exercise, we are going to use Random Forest models for classifying the dataset and finding the accuracy of the dataset.

In this exercise, we are going to use the "RandomForestMP" class to classify samples. These components are exactly the same as in the tutorial.
All you need to do for this exercise is to fill in the missing parts of the code, which is exactly the same as in the tutorial. To make things easier, we have provided a boilerplate in the following file:

```
src/RandomForest.java
```

All you have to do is complete the parts marked as "TODO".

NOTE input: The data input format is similar to the format used with the Connected Components part of **Tutorial 5: Introduction to MLlib** in that each line has a feature set with a last number corresponding to label (0 or 1), for instance:

```
…
1,97,66,15,140,23.2,0.487,22,0
13,145,82,19,110,22.2,0.245,57,1
…
```

Note output: The output will be the result of testing the trained model on a testing dataset. Results for each sample in the testing data set will include the predicted label and the actual label along with the feature set, for example:

```
...
(1.0,[1.0,122.0,84.0,47.0,240.0,45.8,0.551,31.0])
(0.0,[5.0,110.0,80.0,35.0,3.0,29.0,0.263,29.0])
...
```

Note: You only need to fill out the //TODO part and build the model; testing is done in the grader.

After completing the implementation of this file, you have to build the application using the command below from the "cloudapp-mp5" directory:

```
# mvn clean package
```

Next, you have to run the application and store the output using the command below from the "cloudapp-mp5" directory:

```
spark-submit --class RandomForestMP target/mp5-1.0-SNAPSHOT-jar-with-
dependencies.jar  /mp5/data/training.data /mp5/data/test.data
/mp5/output/part-d/
```

Alternatively, use the following command:

```
# bash run_d.sh
```

Note: By running above command, the output will be stored on HDFS at:

```
/mp5/output/part-d
```

For debugging purposes, please check the folder on HDFS.
Here is **a part** of a sample output of this application:

```
...
(1.0,[1.0,122.0,84.0,47.0,240.0,45.8,0.551,31.0])
(0.0,[5.0,110.0,80.0,35.0,3.0,29.0,0.263,29.0])
...
```