Programming Assignment: Machine Problem 5: Spark - GraphFrames and MLLib

✔ Passed · 4/4 points

Deadline The assignment was due on May 12, 09:59 PM PDT
You can still pass this assignment before the course ends.

Instructions

My submission

Discussions

Machine Problem 5: GraphFrames and MLLib

1 Overview

Welcome to the Graphframes and MLLib Machine Problem

We have four goals in this assignment -

- Find the connected components of a given graph
- Build a K-means clustering model for a data set
- Find the shortest path between nodes in a given graph
- Build a random forest classifier model for a data set

2 Requirements

This assignment will be graded based on **Python 3.7.**

All these assignments are designed to work on the **Docker image** that we provide. We ***highly recommend*** that you use this docker image and the given instructions to run the docker container. This will reduce the amount of time you put in setting up the environment for the MP substantially, and also ensure that you do not face any difficulty due to differences in your local environment and the autograder environment.

3 Set up the environment

Step 1: Start the "default" Docker machine that you created when following the "Tutorial: Docker installation" in week 4, run:

```
docker-machine start default
docker-machine env
# follow the instruction to configure your shell: eval $(...)
```

Step 2: Download the Dockerfile and related files for this MP, change the current folder, build, and run the docker image. You can do this by running the following commands -

```
git clone https://github.com/UIUC-CS498-Cloud/mp5-starter-pack.git
cd mp5-starter-pack
docker build -t docker_mp5 .
docker run -v $(pwd):/mp5 -it docker_mp5 /bin/bash
# Use the following instead for Windows
# docker run -it -v //c/path-to-mp5-starter-pack:/mp5 docker_mp5
```

Note that modifying the files locally (on the host) will modify the files in the docker container when you run it. If you don't want this to happen, you can remove the **-v $(pwd):./mp5** arguments passed while calling docker run. If **$(pwd)** does not work for you, you can simply replace it by the mp5-starter-pack directory.

When you run your container, the templates and datasets will be available in the directory **/mp5**

4 Procedures

Step 4: The starter pack includes templates for all parts of this MP. Here is the relevant command to get the repository if you did not clone the repository previously here.

```
git clone https://github.com/UIUC-CS498-Cloud/mp5-starter-pack.git
cd mp5-starter-pack
```

Step 4: Finish the exercises by editing the provided template files. All you need to do is complete the parts marked with **TODO**. Additional details on each exercise, and the input and output formats are given below.

Step 5: After you are done with the assignment, if you want to check if your results are correct, please put all your python files (part_a.py, part_b.py, part_c.py, part_d.py) into a zip file "**solution.zip**", and submit the solution.zip file to the autograder.

You can also create the zip file by running the following command from the **/mp5** directory in the docker container. Make sure that no other files are included in your submission.

```
zip solution.zip part_a.py part_b.py part_c.py part_d.py
```

Exercise A: Connected Components

In this exercise, you are going to build an application using GraphFrames which finds Connected Components in a given graph.

Input Format:

The input for the exercise is given in **dataset/graph.data.** Each line in the input starts with a vertex id which is followed by the ids of the vertices which are connected to it. For instance,

```
0 1 2 # There are edges from vertex 0 to vertices 1 and 2
1 0 2 # There are edges from vertex 1 to vertices 0 and 2
```

You can assume that all edges are bidirectional i.e. if there is an edge from vertex a to vertex b, then there will be an edge from vertex b to vertex a.

Output Format:

Print all the vertex ids of a component in a single line. For example, if nodes {1, 2, 3} form a component, and nodes {4, 5, 6} form a component. Then the output would be

```
1 2 3
4 5 6
```

The order of the components and the order of the vertices in each component do not matter.

Running Exercise A:

In order you complete this exercise, you need to complete the parts marked as "TODO" in the given template.

After completing this exercise, you can run it using the command

```
spark-submit --packages graphframes:graphframes:0.7.0-spark2.4-s_2.11 part_a.py 2> /dev/null
```

Sample Output:

Running this application using the given data and the command given above should have the following output

```
4 5
0 1 2 3
```

Note that the order of vertex ids in each line, and the order of different components (order of the lines) can be different.

Exercise B: K-Means Clustering

In this exercise, you are going to build a clustering model application using MLlib, which clusters samples in a given data set.

All you need to do for this exercise is to fill in the missing parts of the code. To make things easier, we have provided a boilerplate in the following file: **part_b.py**

Input Format:

The input for this exercise is given in **dataset/cars.data.** Each line starts with the sample_id, which is then followed by its features. For instance:

```
"Mazda RX4",21,6,160,110,3.9,2.62,16.46,0,1,4,4
"Mazda RX4 Wag",21,6,160,110,3.9,2.875,17.02,0,1,4,4
```

"Mazda RX4" and "Mazda RX4 Wag" are the sample ids. The numbers following this in each line are the corresponding features.

Output Format:

Each line in your output will correspond to the sample_ids in one cluster. For example, if sample_ids “A” and “B” form a cluster, and sample_ids “C” and “D” form another cluster, then the output would be

```
1  “A”,“B”
2  “C”,“D"
```

All you have to do is to complete the parts marked as “TODO”. Please use the following parameters with "**random**" initialisation in your model. These values are also included in the part_b.py template.

```
1  NUM_CLUSTERS = 4
2  SEED = 0
3  MAX_ITERATIONS = 100
4  INITIALIZATION_MODE = "random"
```

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

Running Exercise B:

After completing the implementation of this file, you have to run the application using the command below from the “/mp5” directory in the docker container

```
1  spark-submit part_b.py 2> /dev/null
```

Here is the sample output of this application:

```
1  "Mazda RX4","Mazda RX4 Wag","Hornet 4 Drive","Valiant","Merc
   280","Ferrari Dino"
2  "Datsun 710","Merc 240D","Merc 230","Fiat 128","Toyota
   Corolla","Toyota Corona","Fiat X1-9","Porsche 914-2","Lotus
   Europa","Volvo 142E"
3  "Hornet Sportabout","Duster 360","Lincoln Continental","Chrysler
   Imperial"
4  "Merc 450SE","Merc 450SL","Merc 450SLC","Dodge Challenger","AMC
   Javelin","Camaro Z28","Ford Pantera L","Maserati Bora"
```

The order of the clusters and the order of the entries in each cluster does not matter.

Note: Make sure you do not have a **collect()** in your code until the very end when you actually aggregate your results.

Exercise C: Shortest Paths in a Graph

In this exercise, you are going to find the shortest path in a graph using GraphFrames

All you need to do for this exercise is to fill in the missing parts of the code. To make things easier, we have provided a boilerplate in the following file: **part_c.py**

Input Format:

Please refer to the input format for Part A. The input format for the exercise is the same as that for exercise A.

Output Format:

Each line in your output will correspond to a given vertex id and its distance to the destination vertex. The destination vertex is fixed to the vertex with id "1".

If no path exists for a given vertex to the destination vertex, print “-1” instead (without the quotation marks). A sample output is given further below.

Running Exercise C:

After filling in the missing parts in the code, you can run the application using the following command after navigating to the "/mp5" directory in the docker container.

```
spark-submit --packages graphframes:graphframes:0.7.0-spark2.4-s_2
    .11 part_c.py 2> /dev/null
```

Running this command using the given data should output the following

```
0 1
1 0
2 1
3 2
4 -1
5 -1
```

The order of the vertices may be different.

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.

All you need to do for this exercise is fill in the missing parts of the code. To make things easier, we have provided a boilerplate in the following file: **part_d.py**

Input Format:

Sample training data is available in **dataset/training.data**. Each line in your training data contains k+1 numbers. The first k numbers correspond to the features of a sample. The last number corresponds to the target label of the sample point. For instance,

```
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
...
```

There are two files corresponding to the test data – **dataset/test-features.data** and **dataset/test-labels.data**. Each line in *test-features.data* contains k numbers corresponding to the features of a sample. For instance,

```
1,122,84,47,240,45.8,0.551,31
2,110,92,18,10,22.7,0.235,48
...
```

Each line in *test-labels.data* contains the label corresponding to each test sample. For instance,

```
1
0
1
...
```

Output Format:

Each line in your output will correspond to the predicted label of each test case. For instance, for the given data, your ideal output (assuming 100% accuracy) will be the same as *test-labels.data* –

```
1
0
1
...
```

Note that it is quite likely that your model will not have 100% accuracy however, so a few entries will be different.

Running Exercise D

After completing the implementation of this file, you have to run the application and store the output using the command below from the "/mp5" directory:

```
spark-submit part_d.py 2> /dev/null
```

Sample Output:

Running this application using the command given above and the given data should output the following

```
1
0
1
0
1
0
1
0
```

Your results might be slightly different depending on the parameters you use to train your model. Try to tune your parameters to maximize the accuracy of the model on the test dataset you have. Some parameters you can vary for random forests are **numTrees** and **maxDepth**. You can fix the **seed** parameter to ensure you get the same output every time you train the random forest.

If the autograder gives an "Incorrect result" output for your submission to this exercise, try changing your model parameters and submitting again.

5 Additional Resources

You can play around with MLLib and GraphFrames in the PySpark shell by running the following command in the docker container

```
pyspark --packages graphframes:graphframes:0.7.0-spark2.4-s_2.11
```

The following documents are relevant for this MP for MLLib

- Classification: https://spark.apache.org/docs/latest/mllib-classification-regression.html
- Clustering: https://spark.apache.org/docs/latest/mllib-clustering.html

Relevant resources for GraphFrames

- GraphFrames Python User Guide - https://docs.databricks.com/spark/latest/graph-analysis/graphframes/user-guide-python.html

How to submit

When you're ready to submit, you can upload files for each part of the assignment on the "My submission" tab.