NS_COS_598_F21_HW3

December 1, 2021

1 Data Science HW3

Notebook by: Nicholas C Soucy
For: COS 598 Data Science HW3

1.1 Setting Up

This cell contains the setting up of pyspark for google colab, all imports, and creation of the spark context.

[1]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# install java
!apt-get install openjdk-8-jdk-headless -qq > /dev/null

# install spark (change the version number if needed)
!wget -q https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.
 ↪0-bin-hadoop3.2.tgz

# unzip the spark file to the current folder
!tar xf spark-3.0.0-bin-hadoop3.2.tgz

# set your spark folder to your system path environment.
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop3.2"

# install findspark using pip
!pip install -q findspark
```

```
Mounted at /content/drive
```

[2]:
```
import findspark
findspark.init()
from pyspark.sql import SparkSession

from pyspark import SparkConf, SparkContext
```

```
from pyspark.sql import SQLContext
from pyspark.sql.types import IntegerType
import pyspark.sql.functions as f
import re
from operator import add
from pyspark.sql.functions import when

spark = SparkSession.builder.appName("Google_Page_Rank").getOrCreate()
sc=spark.sparkContext
sqlContext = SQLContext(spark)
```

1.2 Task 1: Page Rank

[3]:
```
#Read data from text file into list
data = []
file_open = open('drive/MyDrive/Data Science Applications/Homework/HW3/data.
 ↪txt', "r", encoding="latin-1")
for line in file_open:
    items = line.split(" ")
    data.append([items[0].strip(), items[1].strip()])

file_open.close()

print(data)
```

```
[['A', 'B'], ['B', 'D'], ['C', 'D'], ['D', 'A'], ['A', 'C'], ['B', 'C']]
```

[4]:
```
#Parse the data into a list where each element is a vertex and all its edges␣
 ↪are in a list with it

def parseNeighbors(urls):
    """Parses a urls pair string into urls pair."""
    parts = re.split(r'\s+', urls)
    return parts[0], parts[1]

lines = spark.read.text('drive/MyDrive/Data Science Applications/Homework/HW3/
 ↪data.txt').rdd.map(lambda r: r[0])

# Loads all vertices from input file and initialize their neighbors.
edges = lines.map(lambda v: parseNeighbors(v)).distinct().groupByKey().
 ↪map(lambda x : (x[0], list(x[1])))
print(edges.collect())
```

```
[('A', ['B', 'C']), ('B', ['D', 'C']), ('C', ['D']), ('D', ['A'])]
```

[5]:
```
n = edges.count()
# initialize ranks to one
ranks = edges.map(lambda v: (v[0], 1.0))
print(ranks.collect())
```

```
[('A', 1.0), ('B', 1.0), ('C', 1.0), ('D', 1.0)]
```

[6]:
```
#Page rank algorithm
k = 10
for i in range(k):
  ranks = edges.join(ranks).flatMap(lambda x : [(i, float(x[1][1])/
 ↪len(x[1][0])) for i in x[1][0]]).reduceByKey(add)
  print(ranks.sortByKey().collect())

#divide by the number of vertices at end
ranks = ranks.map(lambda v: (v[0], v[1]/n))
print(ranks.sortByKey().collect())
```

```
[('A', 1.0), ('B', 0.5), ('C', 1.0), ('D', 1.5)]
[('A', 1.5), ('B', 0.5), ('C', 0.75), ('D', 1.25)]
[('A', 1.25), ('B', 0.75), ('C', 1.0), ('D', 1.0)]
[('A', 1.0), ('B', 0.625), ('C', 1.0), ('D', 1.375)]
[('A', 1.375), ('B', 0.5), ('C', 0.8125), ('D', 1.3125)]
[('A', 1.3125), ('B', 0.6875), ('C', 0.9375), ('D', 1.0625)]
[('A', 1.0625), ('B', 0.65625), ('C', 1.0), ('D', 1.28125)]
[('A', 1.28125), ('B', 0.53125), ('C', 0.859375), ('D', 1.328125)]
[('A', 1.328125), ('B', 0.640625), ('C', 0.90625), ('D', 1.125)]
[('A', 1.125), ('B', 0.6640625), ('C', 0.984375), ('D', 1.2265625)]
[('A', 0.28125), ('B', 0.166015625), ('C', 0.24609375), ('D', 0.306640625)]
```

[7]:
```
#Print results and save to text file

print_list = []
for (v, rank) in ranks.sortBy(lambda x: x[0]).collect():
    print("%s has rank: %s." % (v, rank))
    print_list.append(str(v) + " has rank: " +str(rank) + ".")

textfile = open("task_1.txt", "w")
for element in print_list:
  textfile.write(element + "\n")
textfile.close()
```

```
A has rank: 0.28125.
B has rank: 0.166015625.
C has rank: 0.24609375.
D has rank: 0.306640625.
```

1.3 Task 2: Spark SQL

1.3.1 a)

Create a Spark data frame from the RDD containing the page ranks of the vertices 0, 1, 2, and 3 computed in Task 1.

[8]:
```
#Parallelize and convert list to dataframe
db = sqlContext.createDataFrame(ranks.sortByKey(),['Index', 'Rank'])
#changing letters to numbers for later task
db = db.replace('A', str(0))
db = db.replace('B', str(1))
db = db.replace('C', str(2))
db = db.replace('D', str(3))
db.show()

db.createOrReplaceTempView('db')
```

```
+-----+-----------+
|Index|       Rank|
+-----+-----------+
|    0|    0.28125|
|    1|0.166015625|
|    2| 0.24609375|
|    3|0.306640625|
+-----+-----------+
```

1.3.2 b)

Write a Spark SQL query to find the page rank of vertex 2. Print it to the screen.

[9]:
```
query = sqlContext.sql("SELECT * FROM db WHERE db.Index = 2")
print(query.collect())
```

```
[Row(Index='2', Rank=0.24609375)]
```

1.3.3 c)

Write a Spark SQL query to find the vertex with the largest page rank. Print both the vertex ID and its page rank.

[10]:
```
query2 = sqlContext.sql("SELECT Index, Rank from db ORDER BY Rank DESC LIMIT 1")
print(query2.collect())
```

```
[Row(Index='3', Rank=0.306640625)]
```

1.3.4 d)

Suppose that there is another input text file, where each line contains the meaning of a vertex. Create a Spark data frame from the input text file shown.

[11]:
```
#Read data from text file into list
name_data = []
file_open = open('drive/MyDrive/Data Science Applications/Homework/HW3/data2.
 ↪txt', "r", encoding="latin-1")
for line in file_open:
    items = line.split(" ")
    name_data.append([items[0].strip(), str(items[1].strip())])

file_open.close()

#Create edge rdd from data
name_data = sc.parallelize(name_data)

names = sqlContext.createDataFrame(name_data,['Index', 'Name'])
names.show()
names.createOrReplaceTempView('names')
```

```
+-----+-----+
|Index| Name|
+-----+-----+
|    0| Adam|
|    1| Lisa|
|    2| Bert|
|    3|Ralph|
+-----+-----+
```

1.3.5 e)

Write a Spark SQL query to join the data frame containing the page rank information and the data frame containing the meaning of each vertex (in this case, the name of the person each vertex corresponds to). Save the query result in CSV format.

[12]:
```
query3 = sqlContext.sql(
    """Select names.Index, Name, Rank FROM db, names WHERE db.Index = names.
 ↪index ORDER BY db.Index""" )
print(query3.collect())
query3.write.option("header", True).csv("results_2e")
```

```
[Row(Index='0', Name='Adam', Rank=0.28125), Row(Index='1', Name='Lisa',
Rank=0.166015625), Row(Index='2', Name='Bert', Rank=0.24609375), Row(Index='3',
Name='Ralph', Rank=0.306640625)]
```

1.4 Task 3 BONUS: Wikipedia Dataset

NOTE: Functionally I am confident this would work very well, however, my machine nor goolge colab as enough RAM to simply read the initial text file into a list. Therefore, feel free to run it on a machine that has more RAM as I had no luck. I hope I still get bonus points for my code though!

[ ]:
```
#Read data from text file into list
data = []
file_open = open('drive/MyDrive/Data Science Applications/Homework/HW3/
 ↪enwiki-2013.txt', "r", encoding="latin-1")
for line in file_open:
    items = line.split(" ")
    data.append([items[0].strip(), items[1].strip()])

file_open.close()
```

[ ]:
```
def parseNeighbors(urls):
    """Parses a urls pair string into urls pair."""
    parts = re.split(r'\s+', urls)
    return parts[0], parts[1]

lines = spark.read.text('drive/MyDrive/Data Science Applications/Homework/HW3/
 ↪data.txt').rdd.map(lambda r: r[0])

# Loads all vertices from input file and initialize their neighbors.
edges = lines.map(lambda v: parseNeighbors(v)).distinct().groupByKey().
 ↪map(lambda x : (x[0], list(x[1])))
```

[ ]:
```
n = edges.count()
# initialize ranks of them to one
ranks = edges.map(lambda v: (v[0], 1.0))
print(ranks.collect())
```

[ ]:
```
#Page rank algorithm
k = 10
for i in range(k):
  ranks = edges.join(ranks).flatMap(lambda x : [(i, float(x[1][1])/
 ↪len(x[1][0])) for i in x[1][0]]).reduceByKey(add)
  print(ranks.sortByKey().collect())

#divide by the number of vertices at end
ranks = ranks.map(lambda v: (v[0], v[1]/n))
print(ranks.sortByKey().collect())
```

[ ]:
```
#save results to text file

print_list = []
for (v, rank) in ranks.sortBy(lambda x: x[0]).collect():
    print_list.append(str(v) + " has rank: " +str(rank) + ".")

textfile = open("task_2.txt", "w")
for element in print_list:
  textfile.write(element + "\n")
textfile.close()
```

[ ]:
```
#Parallelize and convert list to dataframe
db1 = sqlContext.createDataFrame(ranks.sortByKey(),['Index', 'Rank'])

db1.createOrReplaceTempView('db1')
```

[ ]:
```
#Read data from text file into list
name_data = []
file_open = open('drive/MyDrive/Data Science Applications/Homework/HW3/data2.
 ↪txt', "r", encoding="latin-1")
for line in file_open:
    items = line.split(" ")
    name_data.append([items[0].strip(), str(items[1].strip())])

file_open.close()

#Create edge rdd from data
name_data = sc.parallelize(name_data)

names = sqlContext.createDataFrame(name_data,['Index', 'Name'])
names.show()
names.createOrReplaceTempView('names')
```

[ ]:
```
#add ID --> Names data table for joining
db2 = spark.read.option("header",True).csv('drive/MyDrive/Data Science␣
 ↪Applications/Homework/HW3/enwiki-2013-names.csv')

db2 = db2.withColumnRenamed("node_id","Index")

db2.createOrReplaceTempView('db2')
```

[ ]:
```
#query to print/save to csv the top 10 wiki pages
query3 = sqlContext.sql(
    "Select db2.node_id, name, Rank FROM db1, db2 WHERE db1.node_id = db2.
 ↪node_id ORDER BY Rank LIMIT 10")
print(query3.collect())
query3.write.option("header", True).csv("results_3")
```