GraphFrame_example

August 22, 2019

[33]:
```
from functools import reduce
from graphframes import *
from pyspark import *
from pyspark.sql import *
from pyspark.sql.functions import col, lit, when
```

1 Creating GraphFrames

[34]:
```
vertices = sqlContext.createDataFrame([
  ("a", "Alice", 34),
  ("b", "Bob", 36),
  ("c", "Charlie", 30),
  ("d", "David", 29),
  ("e", "Esther", 32),
  ("f", "Fanny", 36),
  ("g", "Gabby", 60)], ["id", "name", "age"])
```

[35]:
```
edges = sqlContext.createDataFrame([
  ("a", "b", "friend"),
  ("b", "c", "follow"),
  ("c", "b", "follow"),
  ("f", "c", "follow"),
  ("e", "f", "follow"),
  ("e", "d", "friend"),
  ("d", "a", "friend"),
  ("a", "e", "friend")
], ["src", "dst", "relationship"])
```

[36]:
```
g = GraphFrame(vertices, edges)
print(g)
```

```
GraphFrame(v:[id: string, name: string ... 1 more field], e:[src: string, dst:
string ... 1 more field])
```

[37]:
```
from graphframes.examples import Graphs
same_g = Graphs(sqlContext).friends()
```

```
print(same_g)
```

```
GraphFrame(v:[id: string, name: string ... 1 more field], e:[src: string, dst:
string ... 1 more field])
```

2 Querying graph and DataFrame

[38]:
```
vertices.show()
```

```
+---+-------+---+
| id|   name|age|
+---+-------+---+
|  a|  Alice| 34|
|  b|    Bob| 36|
|  c|Charlie| 30|
|  d|  David| 29|
|  e| Esther| 32|
|  f|  Fanny| 36|
|  g|  Gabby| 60|
+---+-------+---+
```

[39]:
```
edges.show()
```

```
+---+---+------------+
|src|dst|relationship|
+---+---+------------+
|  a|  b|      friend|
|  b|  c|      follow|
|  c|  b|      follow|
|  f|  c|      follow|
|  e|  f|      follow|
|  e|  d|      friend|
|  d|  a|      friend|
|  a|  e|      friend|
+---+---+------------+
```

[40]:
```
# incoming degree of the vertices
g.inDegrees.show()

# outgoing degree of the vertices
g.outDegrees.show()

g.degrees.show()
```

```
+---+--------+
| id|inDegree|
+---+--------+
|  f|       1|
|  e|       1|
|  d|       1|
|  c|       2|
|  b|       2|
|  a|       1|
+---+--------+

+---+---------+
| id|outDegree|
+---+---------+
|  f|        1|
|  e|        2|
|  d|        1|
|  c|        1|
|  b|        1|
|  a|        2|
+---+---------+

+---+------+
| id|degree|
+---+------+
|  f|     2|
|  e|     3|
|  d|     2|
|  c|     3|
|  b|     3|
|  a|     3|
+---+------+
```

[41]:
```
#find the age of the youngest person in the graph
youngest = g.vertices.groupBy().min("age")
youngest.show()
```

```
+--------+
|min(age)|
+--------+
|      29|
+--------+
```

[42]:
```
# count the number of 'follow' relationships in the graph
numFollows = g.edges.filter("relationship = 'follow'").count()
```

```
print("The number of follow edges is", numFollows)
```

```
The number of follow edges is 4
```

3 Motif finding

[43]:
```
# Search for pairs of vertices with edges in both directions between them.
motifs = g.find("(a)-[e]->(b); (b)-[e2]->(a)")
motifs.show()
```

```
+----------------+--------------+----------------+--------------+
|               a|             e|               b|            e2|
+----------------+--------------+----------------+--------------+
|[c, Charlie, 30]|[c, b, follow]|    [b, Bob, 36]|[b, c, follow]|
|    [b, Bob, 36]|[b, c, follow]|[c, Charlie, 30]|[c, b, follow]|
+----------------+--------------+----------------+--------------+
```

[44]:
```
#  find all the reciprocal relationships in which one person is older than 30
filtered = motifs.filter("b.age > 30 or a.age > 30")
filtered.show()
```

```
+----------------+--------------+----------------+--------------+
|               a|             e|               b|            e2|
+----------------+--------------+----------------+--------------+
|[c, Charlie, 30]|[c, b, follow]|    [b, Bob, 36]|[b, c, follow]|
|    [b, Bob, 36]|[b, c, follow]|[c, Charlie, 30]|[c, b, follow]|
+----------------+--------------+----------------+--------------+
```

[45]:
```
# Find chains of 4 vertices.
chain4 = g.find("(a)-[ab]->(b); (b)-[bc]->(c); (c)-[cd]->(d)")

# Query on sequence, with state (cnt)
#  (a) Define method for updating state given the next element of the motif.
def cumFriends(cnt, edge):
  relationship = col(edge)["relationship"]
  return when(relationship == "friend", cnt + 1).otherwise(cnt)

#  (b) Use sequence operation to apply method to sequence of elements in motif.
#   In this case, the elements are the 3 edges.
edges = ["ab", "bc", "cd"]
numFriends = reduce(cumFriends, edges, lit(0))
```

```
chainWith2Friends2 = chain4.withColumn("num_friends", numFriends).
 ↪where(numFriends >= 2)
chainWith2Friends2.show()
```

```
+--------------+-------------+--------------+-------------+--------------+-
------------+---------------+-----------+
|             a|           ab|             b|           bc|             c|
cd|              d|num_friends|
+--------------+-------------+--------------+-------------+--------------+-
------------+---------------+-----------+
| [d, David, 29]|[d, a, friend]| [a, Alice, 34]|[a, e, friend]|[e, Esther,
32]|[e, f, follow]|  [f, Fanny, 36]|          2|
|[e, Esther, 32]|[e, d, friend]| [d, David, 29]|[d, a, friend]| [a, Alice,
34]|[a, e, friend]| [e, Esther, 32]|          3|
| [d, David, 29]|[d, a, friend]| [a, Alice, 34]|[a, e, friend]|[e, Esther,
32]|[e, d, friend]|  [d, David, 29]|          3|
| [d, David, 29]|[d, a, friend]| [a, Alice, 34]|[a, b, friend]|   [b, Bob,
36]|[b, c, follow]|[c, Charlie, 30]|          2|
|[e, Esther, 32]|[e, d, friend]| [d, David, 29]|[d, a, friend]| [a, Alice,
34]|[a, b, friend]|    [b, Bob, 36]|          3|
| [a, Alice, 34]|[a, e, friend]|[e, Esther, 32]|[e, d, friend]| [d, David,
29]|[d, a, friend]|  [a, Alice, 34]|          3|
+--------------+-------------+--------------+-------------+--------------+-
------------+---------------+-----------+
```

4 Subgraphs

[46]:
```
g2 = g.filterEdges("relationship = 'friend'").filterVertices("age > 30").
 ↪dropIsolatedVertices()
g2.vertices.show()
g2.edges.show()
```

```
+---+------+---+
| id|  name|age|
+---+------+---+
|  e|Esther| 32|
|  b|   Bob| 36|
|  a| Alice| 34|
+---+------+---+

+---+---+------------+
|src|dst|relationship|
+---+---+------------+
|  a|  e|      friend|
|  a|  b|      friend|
```

```
+---+---+------------+
```

5 Standard graph algorithms

5.1 Breadth-first search (BFS)

[47]:
```
# Search from "Esther" for users of age < 32
paths = g.bfs("name = 'Esther'", "age < 32")
paths.show()
```

```
+--------------+-------------+--------------+
|          from|           e0|            to|
+--------------+-------------+--------------+
|[e, Esther, 32]|[e, d, friend]|[d, David, 29]|
+--------------+-------------+--------------+
```

[48]:
```
# limited the search by edge filters and maximum path lengths
filteredPaths = g.bfs(
  fromExpr = "name = 'Esther'",
  toExpr = "age < 32",
  edgeFilter = "relationship != 'friend'",
  maxPathLength = 3)
filteredPaths.show()
```

```
+--------------+-------------+--------------+-------------+----------------+
|          from|           e0|            v1|           e1|              to|
+--------------+-------------+--------------+-------------+----------------+
|[e, Esther, 32]|[e, f, follow]|[f, Fanny, 36]|[f, c, follow]|[c, Charlie, 30]|
+--------------+-------------+--------------+-------------+----------------+
```

5.2 Connected components

[49]:
```
sc.setCheckpointDir("/tmp/graphframes-example-connected-components")
result = g.connectedComponents()
result.show()
```

```
+---+-------+---+------------+
| id|   name|age|   component|
+---+-------+---+------------+
|  a|  Alice| 34|412316860416|
|  b|    Bob| 36|412316860416|
|  c|Charlie| 30|412316860416|
|  d|  David| 29|412316860416|
```

```
|  e| Esther| 32|412316860416|
|  f|  Fanny| 36|412316860416|
|  g|  Gabby| 60|146028888064|
+---+-------+---+------------+
```

5.3 Strongly connected components

[50]:
```
result = g.stronglyConnectedComponents(maxIter=10)
result.select("id", "component").show()
```

```
+---+-------------+
| id|    component|
+---+-------------+
|  g| 146028888064|
|  b|1047972020224|
|  e| 670014898176|
|  a| 670014898176|
|  f| 412316860416|
|  d| 670014898176|
|  c|1047972020224|
+---+-------------+
```

5.4 Label Propagation

[51]:
```
result = g.labelPropagation(maxIter=5)
result.show()
```

```
+---+-------+---+-------------+
| id|   name|age|        label|
+---+-------+---+-------------+
|  g|  Gabby| 60| 146028888064|
|  b|    Bob| 36|1047972020224|
|  e| Esther| 32| 412316860416|
|  a|  Alice| 34| 670014898176|
|  f|  Fanny| 36| 670014898176|
|  d|  David| 29| 670014898176|
|  c|Charlie| 30|1382979469312|
+---+-------+---+-------------+
```

5.5 PageRank

[52]:
```
# Identify important vertices in a graph based on connections
results = g.pageRank(resetProbability=0.15, tol=0.01)

results.vertices.show()
results.edges.show()
```

```
+---+-------+---+------------------+
| id|   name|age|          pagerank|
+---+-------+---+------------------+
|  g|  Gabby| 60| 0.1799821386239711|
|  b|    Bob| 36|  2.655507832863289|
|  e| Esther| 32|0.37085233187676075|
|  a|  Alice| 34|0.44910633706538744|
|  f|  Fanny| 36| 0.3283606792049851|
|  d|  David| 29| 0.3283606792049851|
|  c|Charlie| 30| 2.6878300011606218|
+---+-------+---+------------------+

+---+---+------------+------+
|src|dst|relationship|weight|
+---+---+------------+------+
|  a|  b|      friend|   0.5|
|  b|  c|      follow|   1.0|
|  e|  f|      follow|   0.5|
|  e|  d|      friend|   0.5|
|  c|  b|      follow|   1.0|
|  a|  e|      friend|   0.5|
|  f|  c|      follow|   1.0|
|  d|  a|      friend|   1.0|
+---+---+------------+------+
```

[53]:
```
# Run PageRank for a fixed number of iterations.
iter_10 = g.pageRank(resetProbability=0.15, maxIter=10)

iter_10.vertices.show()
```

```
+---+-------+---+------------------+
| id|   name|age|          pagerank|
+---+-------+---+------------------+
|  g|  Gabby| 60|0.17073170731707318|
|  b|    Bob| 36| 2.7025217677349773|
|  e| Esther| 32| 0.3613490987992571|
|  a|  Alice| 34| 0.4485115093698443|
|  f|  Fanny| 36|0.32504910549694244|
```

```
|  d|  David| 29|0.32504910549694244|
|  c|Charlie| 30| 2.6667877057849627|
+---+-------+---+-------------------+
```

[54]:
```
# Run PageRank personalized for vertex "a"
vertex_a = g.pageRank(resetProbability=0.15, maxIter=10, sourceId="a")

vertex_a.vertices.show()
```

```
+---+-------+---+-------------------+
| id|   name|age|           pagerank|
+---+-------+---+-------------------+
|  g|  Gabby| 60|                0.0|
|  b|    Bob| 36| 0.3366143039702568|
|  e| Esther| 32|0.07657840357273027|
|  a|  Alice| 34|0.17710831642683564|
|  f|  Fanny| 36|0.03189213697274781|
|  d|  David| 29|0.03189213697274781|
|  c|Charlie| 30| 0.3459147020846817|
+---+-------+---+-------------------+
```

5.6 Shortest paths

[55]:
```
#Computes shortest paths to the given set of landmark vertices(specified by␣
 ↪vertex ID)
results = g.shortestPaths(landmarks=["a", "d"])
results.show()
```

```
+---+-------+---+----------------+
| id|   name|age|       distances|
+---+-------+---+----------------+
|  g|  Gabby| 60|              []|
|  b|    Bob| 36|              []|
|  e| Esther| 32|[d -> 1, a -> 2]|
|  a|  Alice| 34|[a -> 0, d -> 2]|
|  f|  Fanny| 36|              []|
|  d|  David| 29|[d -> 0, a -> 1]|
|  c|Charlie| 30|              []|
+---+-------+---+----------------+
```

5.7 Triangle Count

[56]:
```
# Computes the number of triangles passing through each vertex
results = g.triangleCount()
results.show()
```

```
+-----+---+-------+---+
|count| id|   name|age|
+-----+---+-------+---+
|    0|  g|  Gabby| 60|
|    0|  f|  Fanny| 36|
|    1|  e| Esther| 32|
|    1|  d|  David| 29|
|    0|  c|Charlie| 30|
|    0|  b|    Bob| 36|
|    1|  a|  Alice| 34|
+-----+---+-------+---+
```