Homework 4

by Diego Freire

The homework section is located at the end of the notebook

Week 11

Mining Association Rules

In this project, we will analyze a data set about online browsing behavior, and identify significant association rules among the items. Each line represents a browsing session of a customer, and each item is represented by a string of 8 characters.

We will implement the A Priori algorithm with PySpark. The goal is to find significant association rules with $s \geq 100$ and high confidence scores.

```
# Install Spark
# https://github.com/twistedmove/CS246/blob/master/hw1/hw1.pdf
# https://github.com/wrwwctb/Stanford-CS246-2018-2019-winter/blob/master/completed/1_2_de.py
# https://github.com/twistedmove/CS246/blob/master/hw1/hw1q2/h1q2.py
!pip install pyspark
# !pip install -U -q PyDrive
# !apt install openjdk-8-jdk-headless -qq
# import os
# os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
```

```
Requirement already satisfied: pyspark in /usr/local/lib/python3.7/dist-packages (3.2.1)
Requirement already satisfied: py4j==0.10.9.3 in /usr/local/lib/python3.7/dist-packages
```

```
# Download the browsing data
!wget http://snap.stanford.edu/class/cs246-data/browsing.txt
```

```
--2022-04-30 12:49:48--  http://snap.stanford.edu/class/cs246-data/browsing.txt
Resolving snap.stanford.edu (snap.stanford.edu)... 171.64.75.80
Connecting to snap.stanford.edu (snap.stanford.edu)|171.64.75.80|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3458517 (3.3M) [text/plain]
Saving to: ‘browsing.txt’

browsing.txt        100%[===================>]   3.30M  1.47MB/s    in 2.2s

2022-04-30 12:49:50 (1.47 MB/s) - ‘browsing.txt’ saved [3458517/3458517]
```

```
# Create a Spark context
import re, sys, operator
import pyspark
```

```
# Load the data file as an RDD object
sc = pyspark.SparkContext()
lines = sc.textFile('browsing.txt')
baskets = lines.map(lambda l: l.split())
N = baskets.count()
print("N:", N)
```

```
N: 31101
```

Resilient Distributed Datasets

Resilient Distributed Datasets (RDDs) are the basic building block of Spark for distributed computation. Note that RDD objects are immutable and are computed lazily.

RDD I/O

- `textFile()` : load data from a text file.
- `take(n)` : return the first n elements as an array
- `collect()` : return all elements. **Use with caution**.
- `count()` : return the number of elements.
- `saveAsTextFile()`

RDD Operations

Every RDD operation returns a new RDD.

- `map()` : apply an operation to every element of an RDD
- `flatMap()` : apply an operation to every element, and return a list of all results combined.
- `mapValues()` : apply an operation to the value of every tuple in a paired RDD.
- `flatMapValues()`
- `filter()` : apply an operation and filter the elements.
- `groupByKey()`
- `sortBy()`
- `sortByKey()`
- `join()`
- `reduce()` : combine the elements using an associative function.
- `reduceByKey()`

References:

- RDD Programming Guide
- A Video Introduction to RDD

```
baskets = baskets.map(lambda b: sorted(set(b)))
baskets.take(5)
```

```
[['ELE17451', 'ELE89019', 'FRO11987', 'GRO99222', 'SNA90258'],
 ['ELE17451',
  'ELE26917',
  'ELE52966',
  'ELE91550',
  'FRO12685',
  'FRO84225',
  'FRO90334',
  'GRO12298',
  'GRO99222',
  'SNA11465',
  'SNA30755',
  'SNA80192'],
 ['DAI22896', 'ELE17451', 'FRO86643', 'GRO73461', 'SNA99873'],
 ['ELE17451', 'ELE23393', 'ELE37798', 'FRO86643', 'GRO56989', 'SNA11465'],
 ['DAI54444',
  'ELE11375',
  'ELE17451',
  'ELE28573',
  'FRO78087',
  'FRO86643',
  'GRO39357',
  'SNA11465',
  'SNA69641']]
```

```
def singles_helper(basket):
    ret = []
    for item in basket:
        ret.append((item, 1))
    return ret


singles_support = baskets.flatMap(singles_helper)
# singles_support = baskets.map(singles_helper)
singles_support.take(20)
```

```
[('ELE17451', 1),
 ('ELE89019', 1),
 ('FRO11987', 1),
 ('GRO99222', 1),
 ('SNA90258', 1),
 ('ELE17451', 1),
 ('ELE26917', 1),
 ('ELE52966', 1),
 ('ELE91550', 1),
```

```
        ('FRO12685', 1),
        ('FRO84225', 1),
        ('FRO90334', 1),
        ('GRO12298', 1),
        ('GRO99222', 1),
        ('SNA11465', 1),
        ('SNA30755', 1),
        ('SNA80192', 1),
        ('DAI22896', 1),
        ('ELE17451', 1),
        ('FRO86643', 1)]
```

```
singles_support = singles_support.reduceByKey(lambda x, y: x + y)
singles_support.take(5)
```

```
     [('FRO11987', 104),
      ('SNA90258', 550),
      ('ELE52966', 380),
      ('ELE91550', 23),
      ('FRO84225', 74)]
```

```
print(singles_support.count())
singles_support = singles_support.filter(lambda x: x[1] >= 100)
print(singles_support.count())
```

```
     12592
     647
```

```
singles = dict(singles_support.collect())
singles
```

```
       'SNA62046': 115,
       'SNA62128': 1023,
       'SNA62579': 106,
       'SNA63157': 139,
       'SNA64534': 351,
       'SNA64706': 105,
       'SNA66583': 261,
       'SNA66588': 109,
       'SNA66979': 703,
       'SNA67745': 151,
       'SNA68538': 344,
       'SNA69641': 599,
       'SNA71244': 195,
       'SNA71332': 541,
       'SNA72163': 1090,
       'SNA74022': 286,
       'SNA74222': 158,
       'SNA77101': 102,
       'SNA77634': 131,
       'SNA79852': 108,
       'SNA80192': 258,
       'SNA80230': 201,
       'SNA80324': 3044,
```

```
 'SNA81024': 222,
 'SNA81556': 110,
 'SNA82308': 136,
 'SNA82528': 594,
 'SNA83597': 130,
 'SNA83730': 379,
 'SNA84284': 128,
 'SNA85241': 335,
 'SNA85662': 349,
 'SNA87482': 103,
 'SNA87537': 167,
 'SNA88091': 168,
 'SNA88283': 865,
 'SNA88543': 147,
 'SNA89368': 231,
 'SNA89670': 169,
 'SNA90094': 1390,
 'SNA90161': 101,
 'SNA90258': 550,
 'SNA90993': 100,
 'SNA91554': 208,
 'SNA92435': 331,
 'SNA92493': 153,
 'SNA93177': 132,
 'SNA93641': 240,
 'SNA93730': 320,
 'SNA93860': 1407,
 'SNA94104': 107,
 'SNA95666': 804,
 'SNA96271': 1295,
 'SNA96466': 506,
 'SNA97370': 119,
 'SNA97586': 178,
 'SNA99654': 234,
 'SNA99873': 2083}
```

```
def doubles_helper(basket):
    ret = []
    for i in range(len(basket)):
        if basket[i] in singles:
            for j in range(i):
                if basket[j] in singles:
                    ret.append(((basket[j], basket[i]), 1)) # basket is sorted
    return ret


doubles_support = baskets.flatMap(doubles_helper)
doubles_support.take(15)
```

```
 [(('ELE17451', 'FRO11987'), 1),
  (('ELE17451', 'GRO99222'), 1),
  (('FRO11987', 'GRO99222'), 1),
  (('ELE17451', 'SNA90258'), 1),
  (('FRO11987', 'SNA90258'), 1),
```

```
        (('GRO99222', 'SNA90258'), 1),
        (('ELE17451', 'ELE26917'), 1),
        (('ELE17451', 'ELE52966'), 1),
        (('ELE26917', 'ELE52966'), 1),
        (('ELE17451', 'GRO12298'), 1),
        (('ELE26917', 'GRO12298'), 1),
        (('ELE52966', 'GRO12298'), 1),
        (('ELE17451', 'GRO99222'), 1),
        (('ELE26917', 'GRO99222'), 1),
        (('ELE52966', 'GRO99222'), 1)]
```

```
doubles_support = doubles_support.reduceByKey(lambda x, y: x + y)
doubles_support.take(5)
```

```
     [(('ELE17451', 'GRO99222'), 148),
      (('FRO11987', 'SNA90258'), 2),
      (('ELE17451', 'ELE26917'), 314),
      (('ELE17451', 'GRO12298'), 36),
      (('ELE26917', 'GRO12298'), 17)]
```

```
print(doubles_support.count())
doubles_support = doubles_support.filter(lambda x: x[1] >= 100)
print(doubles_support.count())
```

```
     149097
     1334
```

```
def confidence_doubles_helper(double_support):
    double, support = double_support
    support = float(support)
    u, v = double
    uv_conf = support / singles[u]
    vu_conf = support / singles[v]
    return (('%s -> %s' % (u, v), uv_conf),
            ('%s -> %s' % (v, u), vu_conf))
```

```
doubles_conf = doubles_support.flatMap(confidence_doubles_helper)
doubles_conf.take(5)
```

```
     [('ELE17451 -> GRO99222', 0.03819354838709677),
      ('GRO99222 -> ELE17451', 0.16335540838852097),
      ('ELE17451 -> ELE26917', 0.08103225806451612),
      ('ELE26917 -> ELE17451', 0.13699825479930192),
      ('ELE26917 -> GRO99222', 0.08376963350785341)]
```

```
doubles_conf = doubles_conf.sortBy(lambda x: (-x[1], x[0]))
doubles_conf.take(5)
```

```
     [('DAI93865 -> FRO40251', 1.0),
      ('GRO85051 -> FRO40251', 0.999176276771005),
```

```
('GRO38636 -> FRO40251', 0.9906542056074766),
('ELE12951 -> FRO40251', 0.9905660377358491),
('DAI88079 -> FRO40251', 0.9867256637168141)]
```

Homework

List the top 5 association rules with highest confidence scores for itemsets of size 3. The current program should be extended to do the following:

1. Create a list of candidate 3-item sets by merging two frequent item pairs. Two item pairs can generate a 3-item set if they have one element in common.
2. Read the data again so that the frequency of those candidate sets can be counted. This step should be done using the MapReduce model.
3. Remove those candidates who don't reach the support threshold $s = 100$.
4. Compute the confidence value for the remaining sets, and output the top 5 itemsets.

Once the work is done, please convert the notebook to a PDF file via the "Print" option, and submit to Blackboard before **Wednesday, May 4th at 11:59 PM**.

1. Create a list of candidate 3-item sets by merging two frequent item pairs. Two item pairs can generate a 3-item set if they have one element in common.

```
doubles = dict(doubles_support.collect())


def triples_helper(basket):
    ret = []
    for i in range(len(basket)):
      if basket[i] in singles:
        for j in range(i):
          if basket[j] in singles:
            if (basket[j], basket[i]) in doubles:
              for k in range(j):
                  if basket[k] in singles:
                    if (basket[k], basket[j]) in doubles:
                      if (basket[k], basket[i]) in doubles:
                        ret.append(((basket[k], basket[j], basket[i]), 1)) # basket is sorted
    return ret


triples_support = baskets.flatMap(triples_helper)
```

```
triples_support.take(5)
```

```
[(('ELE17451', 'GRO99222', 'SNA90258'), 1),
 (('ELE17451', 'ELE26917', 'GRO99222'), 1),
 (('DAI22896', 'ELE17451', 'GRO73461'), 1),
 (('ELE17451', 'GRO73461', 'SNA99873'), 1),
 (('DAI22896', 'ELE17451', 'GRO73461'), 1)]
```

2. Read the data again so that the frequency of those candidate sets can be counted. This step should be done using the MapReduce model.

```
# this can be done using lamba like in the example above for doubles count or using operator.

#triples_support = triples_support.reduceByKey(operator.add)

triples_support = triples_support.reduceByKey(lambda x, y: x + y)
```

```
triples_support.take(5)
```

```
[(('ELE17451', 'ELE26917', 'GRO99222'), 32),
 (('DAI22896', 'ELE17451', 'GRO73461'), 48),
 (('DAI22177', 'ELE17451', 'ELE66810'), 9),
 (('DAI22177', 'ELE17451', 'GRO73461'), 35),
 (('ELE17451', 'ELE66810', 'GRO73461'), 29)]
```

3. Remove those candidates who don't reach the support threshold s=100 .

```
triples_support = triples_support.filter(lambda x: x[1] >= 100)
```

```
triples_support.take(5)
```

```
[(('ELE17451', 'SNA59903', 'SNA72163'), 127),
 (('DAI62779', 'ELE17451', 'FRO78087'), 121),
 (('DAI62779', 'ELE17451', 'ELE26917'), 160),
 (('DAI62779', 'ELE17451', 'SNA55762'), 157),
 (('DAI62779', 'ELE17451', 'SNA99873'), 126)]
```

4. Compute the confidence value for the remaining sets, and output the top 5 itemsets.

```
singles = dict(singles_support.collect())
doubles = dict(doubles_support.collect())
```

```
def confidence_triples_helper(triples_support):
    triples, support = triples_support
    support = float(support)
    u, v, w = triples
    uv_w = support / doubles[u, v]
    uw_v = support / doubles[u, w]
    vw_u = support / doubles[v, w]
    return ((('( %s, %s ) -> %s' % (u, v, w), uv_w),
            ('( %s, %s ) -> %s' % (u, w, v), uw_v),
            ('( %s, %s ) -> %s' % (v, w, u), vw_u))
```

```
triples_conf = triples_support.flatMap(confidence_triples_helper)

triples_conf = triples_conf.sortBy(lambda x: (-x[1], x[0]))
```

```
triples_conf.take(5)
```

```
[('( DAI23334, ELE92920 ) -> DAI62779', 1.0),
 ('( DAI31081, GRO85051 ) -> FRO40251', 1.0),
 ('( DAI55911, GRO85051 ) -> FRO40251', 1.0),
 ('( DAI62779, DAI88079 ) -> FRO40251', 1.0),
 ('( DAI75645, GRO85051 ) -> FRO40251', 1.0)]
```

✓ 3s completed at 10:35 AM