EECS 4415 - Task 2

Pattern Discovery in Spark

Setup

Let's set up Spark on your Colab environment. Run the cell below!

```
!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/python3.10/dist-packages (3.5.3)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
openjdk-8-jdk-headless is already the newest version (8u422-b05-1~22.04).
0 upgraded, 0 newly installed, 0 to remove and 49 not upgraded.
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Now we authenticate a Google Drive client to download the file we will be processing in our Spark job.

Make sure to follow the interactive instructions.

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
WARNING:root:pydrive is deprecated and no longer maintained. We recommend that you migrate your projects to pydrive2, the maintained fork of pydrive
```

```
id='1dhi1F78ssqR8gE6U-AgB80ZW7V_9snX4'
downloaded = drive.CreateFile({'id': id})
downloaded.GetContentFile('products.csv')

id='1KZBNEalyMTcsRV817us6uLZgm-Mii8oU'
downloaded = drive.CreateFile({'id': id})
downloaded.GetContentFile('order_products__train.csv')
```

If you executed the cells above, you should be able to see the dataset we will need for this Colab under the "Files" tab on the left panel.

더블클릭 또는 Enter 키를 눌러 수정

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import pyspark
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark import SparkContext, SparkConf
```

Let's initialize the Spark context.

```
# create the session
conf = SparkConf().set("spark.ui.port", "4050")

# create the context
sc = pyspark.SparkContext(conf=conf)
spark = SparkSession.builder.getOrCreate()
```

Your task

If you run successfully the setup stage, you are ready to work with the **3 Million Instacart Orders** dataset. In case you want to read more about it, check the official Instacart blog post about it, a concise schema description of the dataset, and the download page.

In this Colab, we will be working with a subset training dataset (~131K orders) to perform fast and scalable Frequent Pattern Mining.

```
products = spark.read.csv('products.csv', header=True, inferSchema=True)
orders = spark.read.csv('order_products__train.csv', header=True, inferSchema=True)
```

```
products.printSchema()
```

```
root
 |-- product_id: integer (nullable = true)
 |-- product_name: string (nullable = true)
 |-- aisle_id: string (nullable = true)
 |-- department_id: string (nullable = true)
```

```
orders.printSchema()
```

```
root
 |-- order_id: integer (nullable = true)
 |-- product_id: integer (nullable = true)
 |-- add_to_cart_order: integer (nullable = true)
 |-- reordered: integer (nullable = true)
```

Use the Spark Dataframe API to join 'products' and 'orders', so that you will be able to see the product names in each transaction (and not only their ids). Then, group by the orders by 'order_id' to obtain one row per basket (i.e., set of products purchased together by one customer). Display the top 20 rows.

```
''' 2-3 lines of code expected '''
# YOUR CODE HERE
product_order = products.join(orders, products.product_id == orders.product_id).groupBy('order_id').agg(collect_set('product_name').alias('basket'))
product_order.show(20)
```

```
+--------+--------------------+
|order_id|              basket|
+--------+--------------------+
|       1|[Bag of Organic B...|
|      96|[Roasted Turkey, ...|
|     112|[Umcka Elderberry...|
|     218|[Okra, Black Plum...|
|     456|[Petite Peas, Lar...|
|     473|[Organic Whole Mi...|
|     631|[Organic Cilantro...|
|     762|[Organic Cucumber...|
|     774|[Nacho Cheese Sau...|
|     844|[Organic Red Radi...|
|     904|[Zero Calorie Col...|
|     988|[Whipped Light Cr...|
|    1032|[Organic Living B...|
|    1077|[Sparkling Water,...|
|    1119|[Shallot, Large L...|
|    1139|[Cinnamon Rolls w...|
|    1143|[Water, Natural P...|
|    1145|[Mexican Casserol...|
|    1275|[Small Hass Avoca...|
|    1280|[Vanilla Soy Milk...|
+--------+--------------------+
only showing top 20 rows
```

In this Colab we will first explore MLlib, Apache Spark's scalable machine learning library. Specifically, you should use its implementation of the FP-Growth algorithm to perform fast Frequent Pattern Mining in Spark. Use the Python example in the documentation, and train a model with `minSupport=0.01` and `minConfidence=0.5`

```
''' 3 lines of code expected '''
# YOUR CODE HERE
from pyspark.ml.fpm import FPGrowth
fpGrowth = FPGrowth(itemsCol="basket", minSupport=0.01, minConfidence=0.5)
model = fpGrowth.fit(product_order)
```

Compute and print how many frequent itemsets and association rules were generated by running FP-growth alongside visalizing top frequent itemsets and association rules. Show top 20 rows.

```
''' 5 lines of code in total expected but can differ based on your style. for sub-parts of the question, creating different cells of code would be recomme
# YOUR CODE HERE
df_freq = model.freqItemsets
df_freq.show(20, truncate=False)
```

```
+-----------------------------------------------+-----+
|items                                          |freq |
+-----------------------------------------------+-----+
|[Green Onions]                                 |1445 |
|[Red Raspberries]                              |1493 |
|[Organic Banana]                               |2332 |
|[Jalapeno Peppers]                             |1899 |
|[Organic Large Extra Fancy Fuji Apple]         |2891 |
|[Organic Whole String Cheese]                  |1993 |
|[Organic Peeled Whole Baby Carrots]            |2460 |
|[Limes]                                        |6033 |
|[Limes, Large Lemon]                           |1595 |
|[Limes, Banana]                                |1331 |
|[Raspberries]                                  |3279 |
|[Hass Avocado]                                 |1633 |
|[Organic Broccoli Florets]                     |1361 |
|[Uncured Genoa Salami]                         |1788 |
|[Spring Water]                                 |2225 |
|[Michigan Organic Kale]                        |2627 |
|[Yellow Onions]                                |3762 |
|[Organic Strawberries]                         |10894|
|[Organic Strawberries, Bag of Organic Bananas]|3074 |
|[Organic Strawberries, Banana]                 |2174 |
+-----------------------------------------------+-----+
only showing top 20 rows
```

```
df_assoc = model.associationRules
df_assoc.show(20)
```

```
+----------+----------+----------+----+-------+
|antecedent|consequent|confidence|lift|support|
+----------+----------+----------+----+-------+
+----------+----------+----------+----+-------+
```

Now retrain the FP-growth model changing only `minsupport=0.001`. Compute and print how many frequent itemsets and association rules were generated. Show top 20 rows for both.

```
''' 6 lines of code in total expected but can differ based on your style. for sub-parts of the question, creating different cells of code would be recomme
# YOUR CODE HERE
fpGrowth_aux = FPGrowth(itemsCol="basket", minSupport=0.001, minConfidence=0.5)
model_aux = fpGrowth_aux.fit(product_order)
df_freq = model_aux.freqItemsets
df_freq.show(20, truncate=False)
```

```
+-------------------------------------------------+----+
|items                                            |freq|
+-------------------------------------------------+----+
|[White Cheddar Popcorn]                          |370 |
|[Organic YoKids Very Berry Smoothies]            |259 |
|[Total 0% Nonfat Greek Yogurt]                   |993 |
|[Total 0% Nonfat Greek Yogurt, Organic Avocado]  |137 |
```

```
|[Total 0% Nonfat Greek Yogurt, Bag of Organic Bananas] |157 |
|[Total 0% Nonfat Greek Yogurt, Organic Baby Spinach]   |143 |
|[Total 0% Nonfat Greek Yogurt, Banana]                 |258 |
|[Organic Large Green Asparagus]                        |730 |
|[Organic Large Green Asparagus, Organic Strawberries]  |185 |
|[Organic Large Green Asparagus, Bag of Organic Bananas]|263 |
|[Organic Large Green Asparagus, Organic Baby Spinach]  |136 |
|[Organic Large Green Asparagus, Organic Hass Avocado]  |164 |
|[Organic Cream Cheese Bar]                             |365 |
|[Organic Reduced Fat Omega-3 Milk]                     |140 |
|[Florida Orange Juice With Calcium & Vitamin D]        |283 |
|[Green Onions]                                         |1445|
|[Green Onions, Limes]                                  |206 |
|[Green Onions, Organic Avocado]                        |133 |
|[Green Onions, Bunched Cilantro]                       |187 |
|[Green Onions, Large Lemon]                            |229 |
+-------------------------------------------------------+----+
only showing top 20 rows
```

```
df_assoc = model_aux.associationRules
df_assoc.show(20, truncate=False)
```

```
+-----------------------------------------------------------------+----------------------+------------------+------------------+--------------------
|antecedent                                                       |consequent            |confidence        |lift              |support
+-----------------------------------------------------------------+----------------------+------------------+------------------+--------------------
|[Organic Whole String Cheese, Organic Hass Avocado]              |[Bag of Organic Bananas]|0.5314685314685315|4.504745125675359 |0.00115845711803306
|[Organic Broccoli, Organic Hass Avocado]                         |[Bag of Organic Bananas]|0.5048231511254019|4.278897986822536 |0.00119656426007362
|[Organic Navel Orange, Organic Raspberries]                      |[Bag of Organic Bananas]|0.5412186379928315|4.587387356098284 |0.00115083568962494
|[Organic Kiwi, Organic Hass Avocado]                             |[Bag of Organic Bananas]|0.5459770114942529|4.627719489738336 |0.00144807139754132
|[Organic D'Anjou Pears, Organic Hass Avocado]                    |[Bag of Organic Bananas]|0.5170454545454546|4.3824946411792345|0.00138709997027642
|[Organic Raspberries, Organic Hass Avocado]                      |[Bag of Organic Bananas]|0.521099116781158 |4.416853618458589 |0.00404697848470760
|[Organic Navel Orange, Organic Hass Avocado]                     |[Bag of Organic Bananas]|0.5283018867924528|4.477904539027839 |0.00149379996799000
|[Organic Raspberries, Organic Hass Avocado, Organic Strawberries]|[Bag of Organic Bananas]|0.5984251968503937|5.072272070642333 |0.00173768567704959
|[Organic Unsweetened Almond Milk, Organic Hass Avocado]          |[Bag of Organic Bananas]|0.5141065830721003|4.357584667849303 |0.00124991425893040
|[Organic Cucumber, Organic Hass Avocado, Organic Strawberries]   |[Bag of Organic Bananas]|0.546875          |4.635330870478036 |0.00106699997713571
|[Yellow Onions, Strawberries]                                    |[Banana]              |0.5357142857142857|3.753633221952670 2|0.00114321426121683
+-----------------------------------------------------------------+----------------------+------------------+------------------+--------------------
```

We ask you to inspect the resulting dataframes, and report a few results. Sort frequent items in descending order by frequency and display 50 rows.

```
# YOUR CODE HERE
df_freq.sort(desc("freq")).show(50, truncate=False)
```

```
+-----------------------------------------------+-----+
|items                                          |freq |
+-----------------------------------------------+-----+
|[Banana]                                       |18726|
|[Bag of Organic Bananas]                       |15480|
|[Organic Strawberries]                         |10894|
|[Organic Baby Spinach]                         |9784 |
|[Large Lemon]                                  |8135 |
|[Organic Avocado]                              |7409 |
|[Organic Hass Avocado]                         |7293 |
|[Strawberries]                                 |6494 |
|[Limes]                                        |6033 |
|[Organic Raspberries]                          |5546 |
|[Organic Blueberries]                          |4966 |
|[Organic Whole Milk]                           |4908 |
|[Organic Cucumber]                             |4613 |
|[Organic Zucchini]                             |4589 |
|[Organic Yellow Onion]                         |4290 |
|[Organic Garlic]                               |4158 |
|[Seedless Red Grapes]                          |4059 |
|[Asparagus]                                    |3868 |
|[Organic Grape Tomatoes]                       |3823 |
|[Organic Red Onion]                            |3818 |
|[Yellow Onions]                                |3762 |
|[Organic Baby Carrots]                         |3597 |
|[Honeycrisp Apple]                             |3551 |
|[Organic Cilantro]                             |3526 |
|[Organic Lemon]                                |3505 |
|[Sparkling Water Grapefruit]                   |3359 |
|[Raspberries]                                  |3279 |
|[Organic Fuji Apple]                           |3257 |
|[Small Hass Avocado]                           |3103 |
|[Organic Strawberries, Bag of Organic Bananas]|3074 |
```

```
|[Broccoli Crown]                                |2932 |
|[Organic Baby Arugula]                          |2923 |
|[Red Peppers]                                   |2905 |
|[Organic Large Extra Fancy Fuji Apple]          |2891 |
|[Original Hummus]                               |2858 |
|[Organic Blackberries]                          |2843 |
|[Organic Gala Apples]                           |2809 |
|[Fresh Cauliflower]                             |2651 |
|[Organic Half & Half]                           |2646 |
|[Michigan Organic Kale]                         |2627 |
|[Organic Small Bunch Celery]                    |2623 |
|[Organic Garnet Sweet Potato (Yam)]             |2568 |
|[Organic Tomato Cluster]                        |2538 |
|[Green Bell Pepper]                             |2521 |
|[Carrots]                                       |2497 |
|[Organic Peeled Whole Baby Carrots]             |2460 |
|[Half & Half]                                   |2424 |
|[Organic Hass Avocado, Bag of Organic Bananas]|2420 |
|[Cucumber Kirby]                                |2413 |
|[Organic Italian Parsley Bunch]                 |2400 |
+------------------------------------------------+-----+
only showing top 50 rows
```

Also, sort assocation rules in descending order by confidence and display 20 rows.

```
# YOUR CODE HERE
df_assoc.sort(desc("confidence")).show(20, truncate=False)
```

```
+------------------------------------------------------------------+-----------------------+------------------+------------------+--------------------
|antecedent                                                        |consequent             |confidence        |lift              |support
+------------------------------------------------------------------+-----------------------+------------------+------------------+--------------------
|[Organic Raspberries, Organic Hass Avocado, Organic Strawberries]|[Bag of Organic Bananas]|0.5984251968503937|5.072272070642333 |0.00173768567704959
|[Organic Cucumber, Organic Hass Avocado, Organic Strawberries]   |[Bag of Organic Bananas]|0.546875          |4.635330870478036 |0.00106699997713571
|[Organic Kiwi, Organic Hass Avocado]                             |[Bag of Organic Bananas]|0.5459770114942529|4.627719489738336 |0.00144807139754132
|[Organic Navel Orange, Organic Raspberries]                      |[Bag of Organic Bananas]|0.5412186379928315|4.587387356098284 |0.00115083568962494
|[Yellow Onions, Strawberries]                                    |[Banana]               |0.5357142857142857|3.7536332219526702|0.00114321426121683
|[Organic Whole String Cheese, Organic Hass Avocado]              |[Bag of Organic Bananas]|0.5314685314685315|4.504745125675359 |0.00115845711803306
|[Organic Navel Orange, Organic Hass Avocado]                     |[Bag of Organic Bananas]|0.5283018867924528|4.477904539027839 |0.00149379996799000
|[Organic Raspberries, Organic Hass Avocado]                      |[Bag of Organic Bananas]|0.521099116781158 |4.416853618458589 |0.00404697848470760
|[Organic D'Anjou Pears, Organic Hass Avocado]                    |[Bag of Organic Bananas]|0.5170454545454546|4.3824946411792345|0.00138709997027642
|[Organic Unsweetened Almond Milk, Organic Hass Avocado]          |[Bag of Organic Bananas]|0.5141065830721003|4.357584667849303 |0.00124991425893040
|[Organic Broccoli, Organic Hass Avocado]                         |[Bag of Organic Bananas]|0.5048231511254019|4.278897986822536 |0.00119656426007362
+------------------------------------------------------------------+-----------------------+------------------+------------------+--------------------
```

What are your observations by varying support parameters? Answer in 2 sentences.

As minSupport decreases, the number of association rules found tends to increases. minSupport is a threshold for association rules to identified as "frequent enough".

Your next task is to develop the **SON Map Reduce algorithm** to compute frequent itemsets by dividing the data into 10 chunks. Assume the *global* support to be: `minsupport=0.001`.

Hint:

1. To simulate the SON Map Reduce algorithm for frequent itemset mining in Apache Spark, even if the input file is located on a single node, you can divide the file into smaller chunks (partitions) and process each partition as though it were distributed across multiple nodes. Spark will treat these partitions as separate subsets of the data, effectively mimicking how the SON algorithm would work in a true distributed environment.
2. The repartition function redistributes the data into the desired number of partitions (use 10 in this case). Example:

   ```
   data_partitioned = data.repartition(10)
   ```

The .repartition(10) method repartitions the DataFrame into 10 partitions, but it does not change the underlying structure of data. It remains a Spark DataFrame, just with a different partitioning scheme.

3. Apply FP-Growth to each partition without converting to RDD to generate candidates. Get the DataFrame for each partition by using e.g., the filter function(). If you want to apply FP-Growth to each partition while keeping everything within the DataFrame API, you must avoid using RDD-based operations like .rdd.mapPartitions().

Perform map and reduce operations using Spark transformations and actions.

In Phase 1, find candidate itemsets.

```
# YOUR CODE HERE (map and reduce operations)
from pyspark.ml.fpm import FPGrowth
NUM_PARTITIONS = 10
GLOBAL_MIN_SUPPORT = 0.001
LOCAL_MIN_SUPPORT = GLOBAL_MIN_SUPPORT/NUM_PARTITIONS

data_partitioned = product_order.repartition(NUM_PARTITIONS, "order_id")
data_partitioned = data_partitioned.withColumn("pid", spark_partition_id())
partitions = []
candidates = []
for pid in range(0,data_partitioned.rdd.getNumPartitions() - 1):
  df = data_partitioned.filter(data_partitioned.pid == pid)
  df = df.drop("pid")
  partitions.append(df)

fpGrowth_son = FPGrowth(itemsCol="basket", minSupport=LOCAL_MIN_SUPPORT, minConfidence=0.5)

for partition in partitions:
  model = fpGrowth_son.fit(partition)
  candidates.append(model.freqItemsets)
```

Can Phase 1 cause false negatives? Provide an answer including justification in one sentence.

An itemset that belongs to none of the candidates must have frequency less than GLOBAL_MIN_SUPPORT, which means it cannot be false negative at any occasion.

In Phase 2, find true frequent itemsets.

```
# YOUR CODE HERE (map and reduce operations)
result = candidates[0]
for i in range(1, len(candidates)):
  result = result.union(candidates[i])

result = result.groupBy("items").agg(sum("freq").alias("freq"))
result = result.filter(result.freq >= GLOBAL_MIN_SUPPORT*product_order.count())
```

What is the benefit of Phase 2? Provide an answer in one sentence.

It removes false positive cases.

:Compute and print how many frequent itemsets were generated.

```
# YOUR CODE HERE
result.count()
```

```
3772
```

Sort frequent items in descending order by frequency and display 50 rows.

```
# YOUR CODE HERE
result.sort(desc("freq")).show(50, truncate=False)
```

```
+----------------------------------------+-----+
|items                                   |freq |
+----------------------------------------+-----+
|[Banana]                                |16874|
|[Bag of Organic Bananas]                |13923|
|[Organic Strawberries]                  |9796 |
|[Organic Baby Spinach]                  |8819 |
|[Large Lemon]                           |7352 |
|[Organic Avocado]                       |6675 |
|[Organic Hass Avocado]                  |6549 |
```

```
|[Strawberries]                                |5869 |
|[Limes]                                       |5410 |
|[Organic Raspberries]                         |5038 |
|[Organic Blueberries]                         |4469 |
|[Organic Whole Milk]                          |4433 |
|[Organic Cucumber]                            |4142 |
|[Organic Zucchini]                            |4141 |
|[Organic Yellow Onion]                        |3873 |
|[Organic Garlic]                              |3772 |
|[Seedless Red Grapes]                         |3638 |
|[Asparagus]                                   |3496 |
|[Organic Red Onion]                           |3460 |
|[Organic Grape Tomatoes]                      |3424 |
|[Yellow Onions]                               |3407 |
|[Organic Baby Carrots]                        |3223 |
|[Honeycrisp Apple]                            |3189 |
|[Organic Cilantro]                            |3185 |
|[Organic Lemon]                               |3155 |
|[Sparkling Water Grapefruit]                  |3022 |
|[Raspberries]                                 |2973 |
|[Organic Fuji Apple]                          |2963 |
|[Small Hass Avocado]                          |2798 |
|[Organic Strawberries, Bag of Organic Bananas]|2750 |
|[Organic Baby Arugula]                        |2634 |
|[Red Peppers]                                 |2618 |
|[Broccoli Crown]                              |2613 |
|[Original Hummus]                             |2579 |
|[Organic Large Extra Fancy Fuji Apple]        |2576 |
|[Organic Blackberries]                        |2546 |
|[Organic Gala Apples]                         |2509 |
|[Fresh Cauliflower]                           |2393 |
|[Organic Half & Half]                         |2376 |
|[Organic Small Bunch Celery]                  |2361 |
|[Michigan Organic Kale]                       |2358 |
|[Organic Garnet Sweet Potato (Yam)]           |2319 |
|[Organic Tomato Cluster]                      |2301 |
|[Green Bell Pepper]                           |2272 |
|[Carrots]                                     |2257 |
|[Organic Peeled Whole Baby Carrots]           |2245 |
|[Organic Hass Avocado, Bag of Organic Bananas]|2197 |
|[Half & Half]                                 |2183 |
|[Cucumber Kirby]                              |2168 |
|[Organic Italian Parsley Bunch]               |2153 |
+----------------------------------------------+-----+
only showing top 50 rows
```

Write a paragraph of conclusions below summarizing your insights.

It seems that the SON mapReduce yields the same result as the regular frequent pattern mining. However, due to the nature of SON mapReduce stating that each partition can be distributed to multiple nodes to be processed, the efficiency of SON mapReduce increases if the size of dataset gets bigger.

Once you obtained the desired results, **head over to eClass and submit your solution for this Colab**!