Big Data Lab

April 16, 2018

Log in to Dumbo

Spark’s MLlib

- MLlib is Spark’s machine learning library
- Goal is to make practical machine learning scalable and easy
- Includes common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, lower-level optimization primitives and higher-level pipeline APIs

FP-Growth

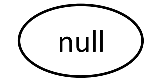
- “FP”=“frequent pattern”
- Like apriori-like algorithms, the first step of FP-growth is to calculate item frequencies and identify frequent items
- Unlike apriori-like algorithms, the second step of FP-growth uses a suffix tree (FP-tree) structure to encode transactions without generating candidate sets explicitly
- Based on the paper Han, Pei, and Yin, “Mining frequent patterns without candidate generation”, SIGMOD, 2000.

FP-Growth

- After the second step, the frequent itemsets can be extracted from the FP-tree.
- spark.mllib implements a parallel version of FP-growth called PFP, as described in Li et al., PFP: Parallel FP-growth for query recommendation
- PFP distributes the work of growing FP-trees based on the suffices of transactions, and hence more scalable than a single-machine implementation
- Spark's FP-Growth implementation takes the following parameters:
  - minSupport: the minimum support for an itemset to be identified as frequent. For example, if an item appears 3 out of 5 transactions, it has a support of 3/5=0.6.
  - numPartitions: the number of partitions used to distribute the work

Han, Pei, and Yin, “Mining frequent patterns without candidate generation”, SIGMOD, 2000

Example



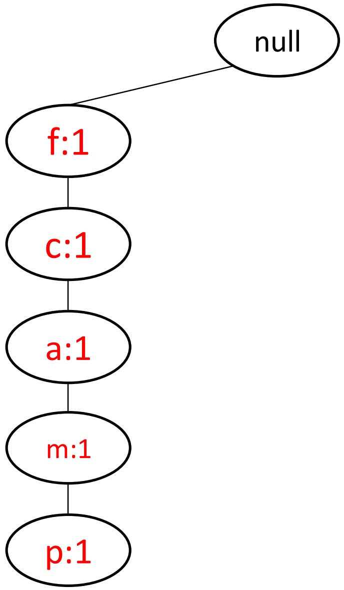
TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

- Create root of the tree, labeled “null”

Han, Pei, and Yin, “Mining frequent patterns without candidate generation”, SIGMOD, 2000

Example

TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

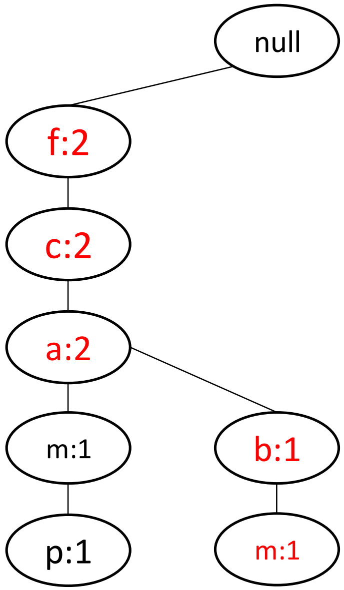


- Create root of the tree, labeled “null”
- Second scan of the DB:
- Scan of first transaction leads to first branch of the tree: <(f:1), (c:1), (a:1), (m:1), (p:1)>

Han, Pei, and Yin, “Mining frequent patterns without candidate generation”, SIGMOD, 2000

Example

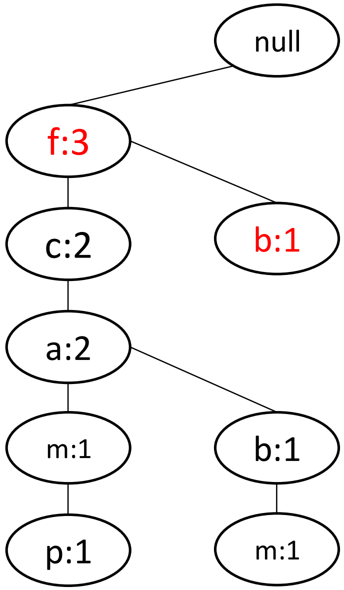
TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p



- Create root of the tree, labeled “null”
- Second scan of the DB:
- Scan of first transaction leads to first branch of the tree: <(f:1), (c:1), (a:1), (m:1), (p:1)>
- Second transaction: since its ordered frequent item list shares a common prefix with <f, c, a> with the existing path <f, c, a, m, p>, the count of each node along the prefix is incremented by 1 and one new node (b:1) is created and linked as a child of (a:2) and another new node (m:1) is create and linked as the child of (b:1)

Example

TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

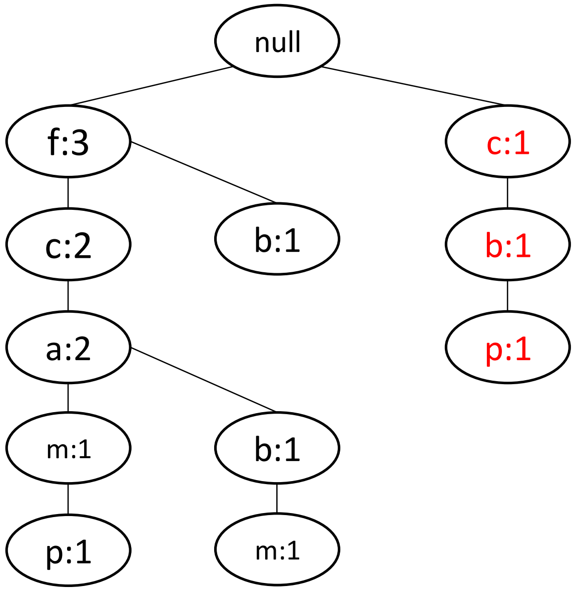


- Create root of the tree, labeled “null”
- Second scan of the DB:
- Scan of first transaction leads to first branch of the tree: <(f:1), (c:1), (a:1), (m:1), (p:1)>
- Second transaction: since its ordered frequent item list shares a common prefix with <f, c, a> with the existing path <f, c, a, m, p>, the count of each node along the prefix is incremented by 1 and one new node (b:1) is created and linked as a child of (a:2) and another new node (m:1) is create and linked as the child of (b:1)
- Third transaction: f’s count is incremented by 1, (b:1) is created as a child of (f:3)

Han, Pei, and Yin, “Mining frequent patterns without candidate generation”, SIGMOD, 2000

Example

TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

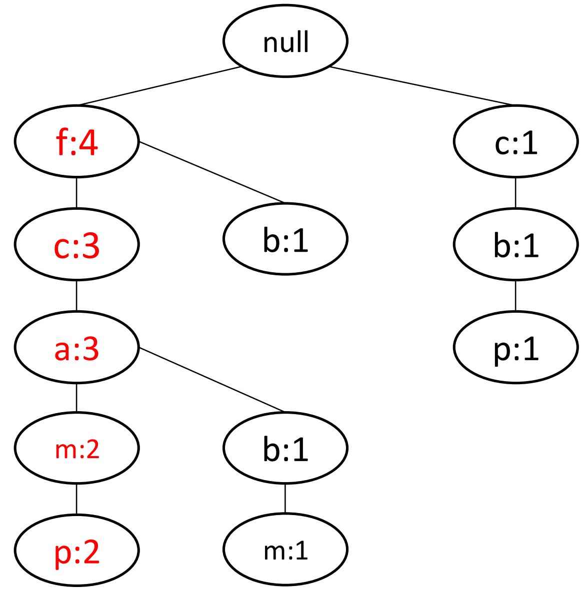


- Create root of the tree, labeled “null”
- Second scan of the DB:
- Scan of first transaction leads to first branch of the tree: <(f:1), (c:1), (a:1), (m:1), (p:1)>
- Second transaction: since its ordered frequent item list shares a common prefix with <f, c, a> with the existing path <f, c, a, m, p>, the count of each node along the prefix is incremented by 1 and one new node (b:1) is created and linked as a child of (a:2) and another new node (m:1) is create and linked as the child of (b:1)
- Third transaction: f’s count is incremented by 1, (b:1) is created as a child of (f:3)
- Fourth transaction: Since no prefix in common, create the second branch of the tree <(c:1), (b:1), (p:1)>

Han, Pei, and Yin, “Mining frequent patterns without candidate generation”, SIGMOD, 2000

Example

TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

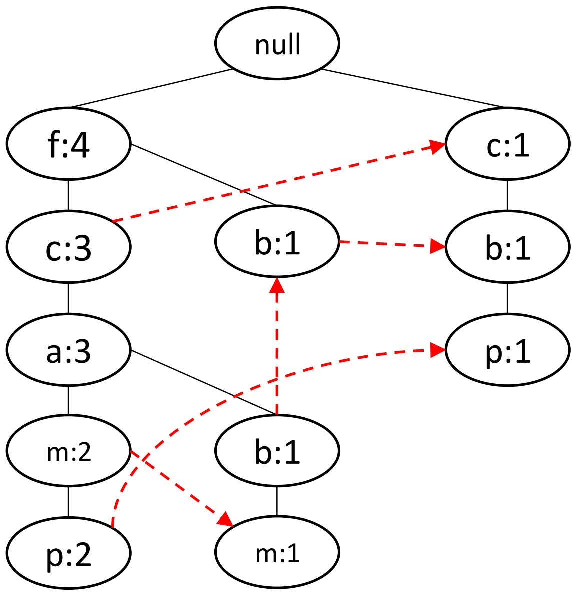


- Create root of the tree, labeled “null”
- Second scan of the DB:
- Scan of first transaction leads to first branch of the tree: <(f:1), (c:1), (a:1), (m:1), (p:1)>
- Second transaction: since its ordered frequent item list shares a common prefix with <f, c, a> with the existing path <f, c, a, m, p>, the count of each node along the prefix is incremented by 1 and one new node (b:1) is created and linked as a child of (a:2) and another new node (m:1) is create and linked as the child of (b:1)
- Third transaction: f’s count is incremented by 1, (b:1) is created as a child of (f:3)
- Fourth transaction: Since no prefix in common, create the second branch of the tree <(c:1), (b:1), (p:1)>
- Fifth transaction: Since same as the first, each count incremented by 1

Han, Pei, and Yin, “Mining frequent patterns without candidate generation”, SIGMOD, 2000

Example

TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p



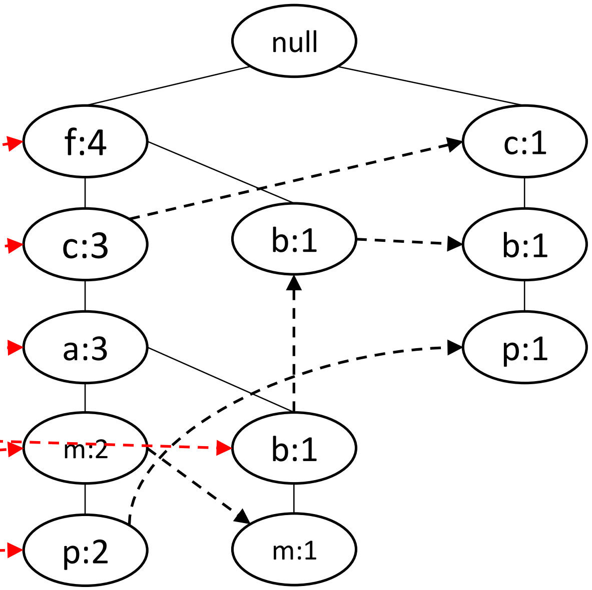
- Create root of the tree, labeled “null”
- Second scan of the DB:
- Scan of first transaction leads to first branch of the tree: <(f:1), (c:1), (a:1), (m:1), (p:1)>
- Second transaction: since its ordered frequent item list shares a common prefix with <f, c, a> with the existing path <f, c, a, m, p>, the count of each node along the prefix is incremented by 1 and one new node (b:1) is created and linked as a child of (a:2) and another new node (m:1) is create and linked as the child of (b:1)
- Third transaction: f’s count is incremented by 1, (b:1) is created as a child of (f:3)
- Fourth transaction: Since no prefix in common, create the second branch of the tree <(c:1), (b:1), (p:1)>
- Fifth transaction: Since same as the first, each count incremented by 1
- Nodes with the same item name are linked via “node-links”

Han, Pei, and Yin, “Mining frequent patterns without candidate generation”, SIGMOD, 2000

Example

Header Table

Item	Head of node-links
f	
c	
a	
b	
m	
p	

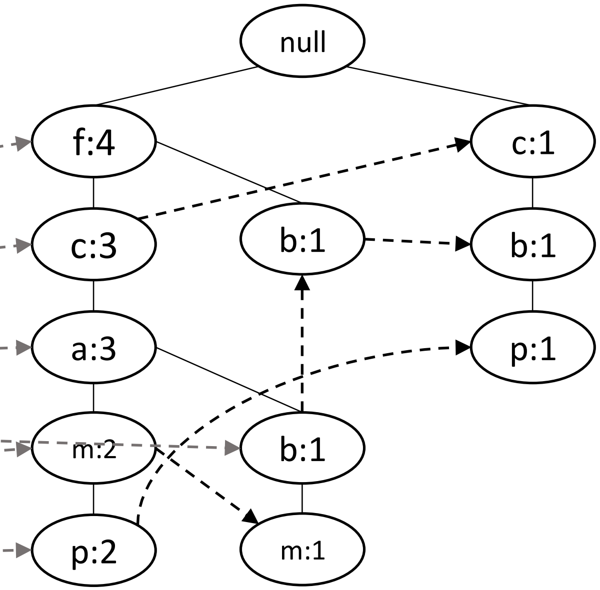


Han, Pei, and Yin, “Mining frequent patterns without candidate generation”, SIGMOD, 2000

Example

Header Table

Item	Head of node-links
f	
c	
a	
b	
m	
p	

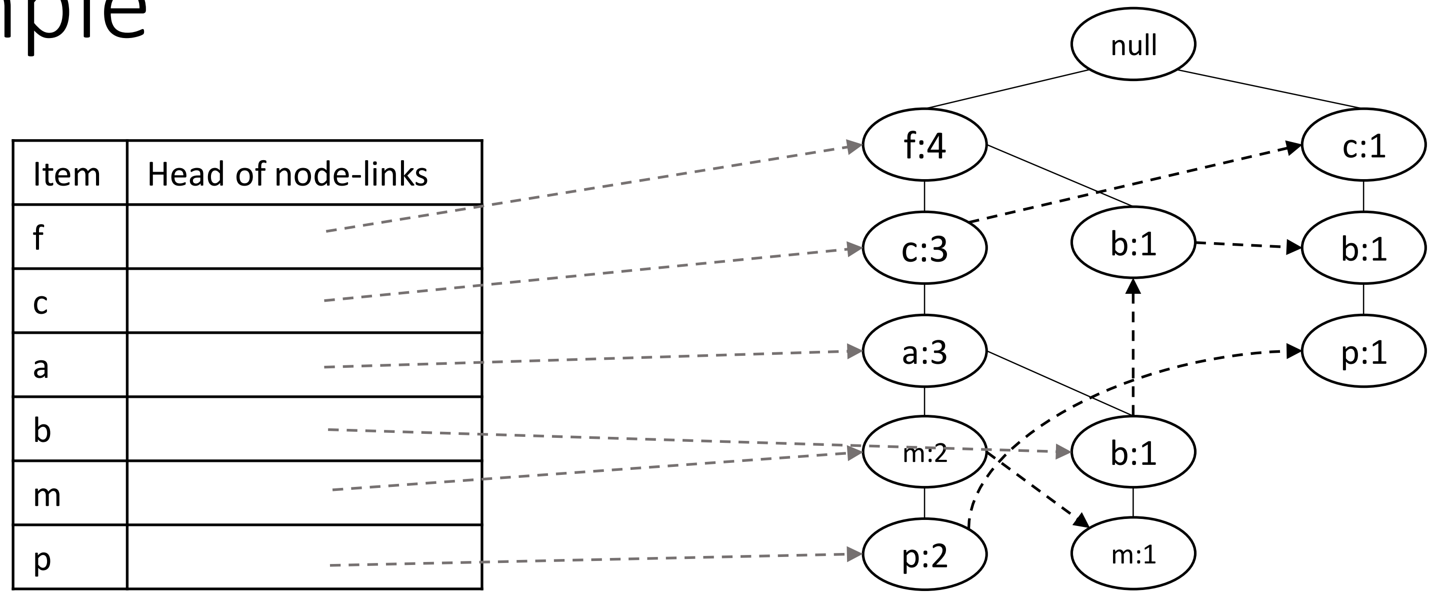


- Mining frequent patterns
- Collect all patterns that a node x participates in by starting from x’s head (in the header table) and following x’s node-links

Han, Pei, and Yin, “Mining frequent patterns without candidate generation”, SIGMOD, 2000

Example

Header Table

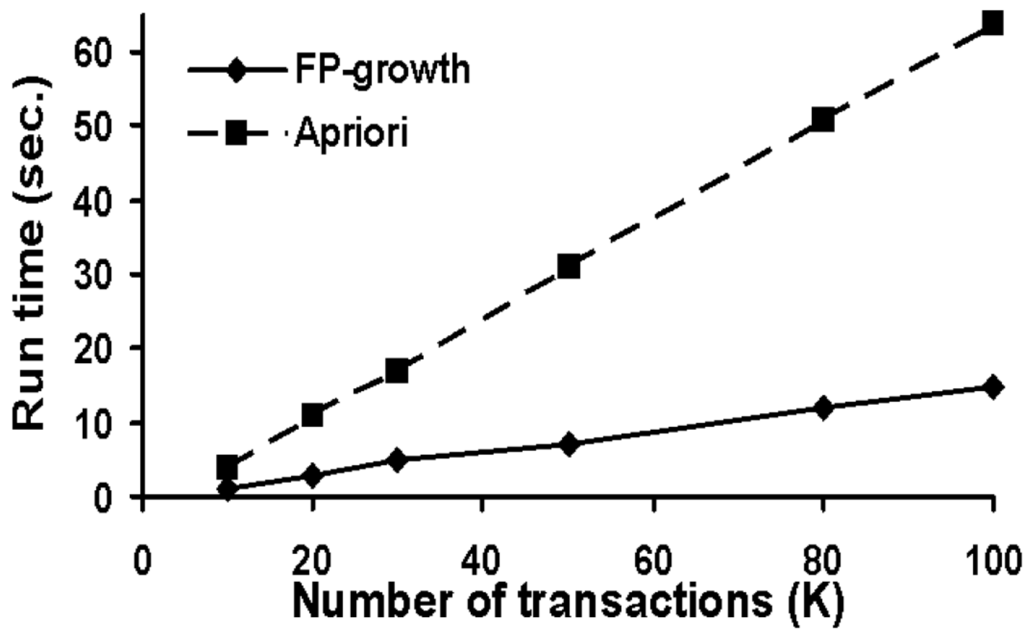


- Mining frequent patterns
- Collect all patterns that a node x participates in by starting from x’s head (in the header table) and following x’s node-links

- Example: item p
- Node p derives a frequent pattern (p:3) and two paths in the FP tree: <f:4, c:3, a:3, m:2, p:2> and <c:1, b:1, p:1>
  - The first path indicates that the string (f,c,a,m,p) appears twice in the DB
  - Second path indicates that (c,b,p) appears once in the DB
- Since both paths contain (c,p), this is a frequent pattern, (cp:3)

Benefits

- Apriori-like algorithms can generate an exponential number of candidates in the worst case, but size of an FP-tree is bounded by the size of its database
- Can lead to faster runtime



Han, Pei, and Yin, "Mining frequent patterns without candidate generation", SIGMOD, 2000

In PySpark

```
from pyspark.mllib.fpm import FPGrowth

data = sc.textFile("data/mllib/sample_fpgrowth.txt")

transactions = data.map(lambda line: line.strip().split(' '))

model = FPGrowth.train(transactions, minSupport=0.2, numPartitions=10)



result = model.freqItemsets().collect()

for fi in result:
        print(fi)
```

In PySpark

```
from pyspark.mllib.fpm import FPGrowth

data = sc.textFile("data/mllib/sample_fpgrowth.txt")

transactions = data.map(lambda line: line.strip().split(' '))

model = FPGrowth.train(transactions, minSupport=0.2, numPartitions=10)



result = model.freqItemsets().collect()

for fi in result:
        print(fi)
```

In PySpark

```
from pyspark.mllib.fpm import FPGrowth

data = sc.textFile("data/mllib/sample_fpgrowth.txt")

transactions = data.map(lambda line: line.strip().split(' '))

model = FPGrowth.train(transactions, minSupport=0.2, numPartitions=10)



result = model.freqItemsets().collect()

for fi in result:
        print(fi)
```

takes an RDD of transactions, where each transaction is an List of items of a generic type

In PySpark

```
from pyspark.mllib.fpm import FPGrowth

data = sc.textFile("data/mllib/sample_fpgrowth.txt")

transactions = data.map(lambda line: line.strip().split(' '))

model = FPGrowth.train(transactions, minSupport=0.2, numPartitions=10)

result = model.freqItemsets().collect()

for fi in result:
        print(fi)
```

takes an RDD of transactions, where each transaction is an List of items of a generic type

Returns an FPGrowthModel object

In PySpark

```
from pyspark.mllib.fpm import FPGrowth

data = sc.textFile("data/mllib/sample_fpgrowth.txt")

transactions = data.map(lambda line: line.strip().split(' '))

model = FPGrowth.train(transactions, minSupport=0.2, numPartitions=10)

result = model.freqItemsets().collect()

for fi in result:
        print(fi)
```

takes an RDD of transactions, where each transaction is an List of items of a generic type

Returns an FPGrowthModel object

Returns (items, freq) tuples

In PySpark

```python
from pyspark.mllib.fpm import FPGrowth

data = sc.textFile("data/mllib/sample_fpgrowth.txt")

transactions = data.map(lambda line: line.strip().split(' '))

model = FPGrowth.train(transactions, minSupport=0.2, numPartitions=10)

result = model.freqItemsets().collect()

for fi in result:
        print(fi)
```

takes an RDD of transactions, where each transaction is an List of items of a generic type

Returns an FPGrowthModel object

Returns (items, freq) tuples

Could access item list by fi.items, frequency by fi.freq

Resources

- https://spark.apache.org/docs/latest/mllib-frequent-pattern-mining.html

- https://spark.apache.org/docs/latest/api/python/pyspark.mllib.html#pyspark.mllib.fpm.FPGrowth

Get the files

Once SSHed into DUMBO, get the file for the lab by typing:

hfs -get /user/ecc290/lab8/freqitems.py

You do not need to get the input file - it can be accessed from HDFS on dumbo. But if you would like to get it to view the input, you can type:

hfs -get /user/ecc290/lab8/groceries.csv

Run the Sample Program

1. Type

```
cat freqitems.py
```

to view the program

2. To run the job, use the command

```
spark-submit freqitems.py /user/ecc290/lab8/groceries.csv > freqitemsoutput.txt
```

3. Type

```
cat freqitemsoutput.txt
```

to view the output file

```
bash-4.1$ cat freqitemsoutput.txt
FreqItemset(items=['whole milk'], freq=2513)
FreqItemset(items=['other vegetables'], freq=1903)
FreqItemset(items=['rolls/buns'], freq=1809)
FreqItemset(items=['soda'], freq=1715)
FreqItemset(items=['yogurt'], freq=1372)
FreqItemset(items=['bottled water'], freq=1087)
FreqItemset(items=['root vegetables'], freq=1072)
FreqItemset(items=['tropical fruit'], freq=1032)
```

Deliverable

Due Monday, April 23, 2018, 6pm

Suppose you are deciding which items to place next to each other at the grocery store, so you only care about frequent itemsets of size 2 or greater which appear in at least 2% of the transactions.

1. Modify the freqitems.py file to meet these constraints. Your code should print itemsets of size 2 or larger that appear in at least 2% of transactions, **sorted by decreasing frequency**. (This involves both setting an appropriate minSupport and modifying the code to prune itemsets of size 1 and sort).
2. Run the job with your modified freqitems.py file using spark-submit, saving the output to the file modifiedoutput.txt
3. Submit the modifiedoutput.txt file to NYU Classes.