

Big Data Lab

May 2, 2016

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

Example



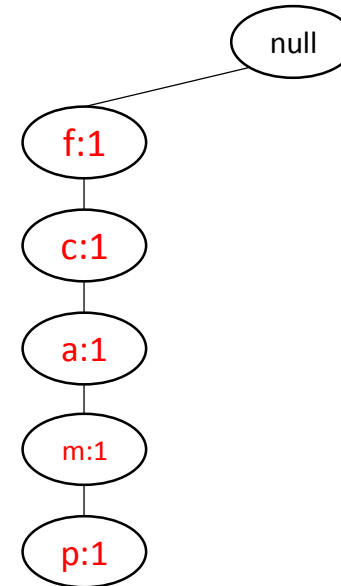
null

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”

Example

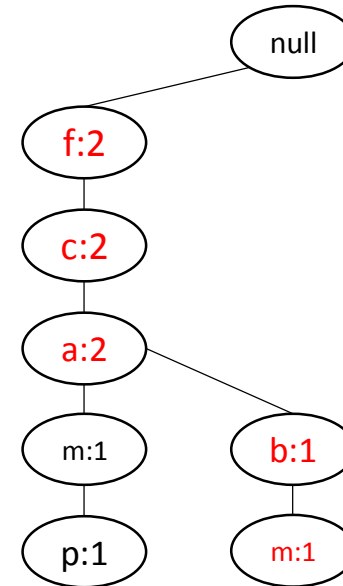
TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p



- 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)>

Example

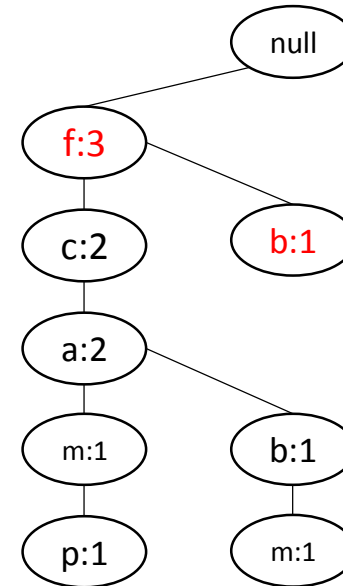
TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p



- 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

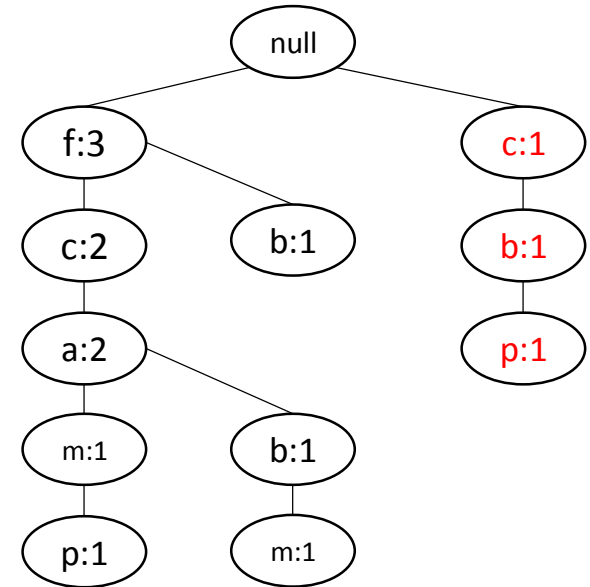
TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p



- 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)

Example

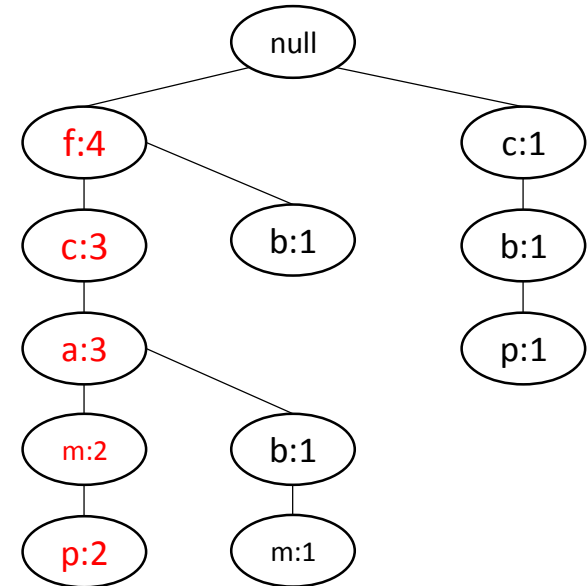
TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p



- 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)>

Example

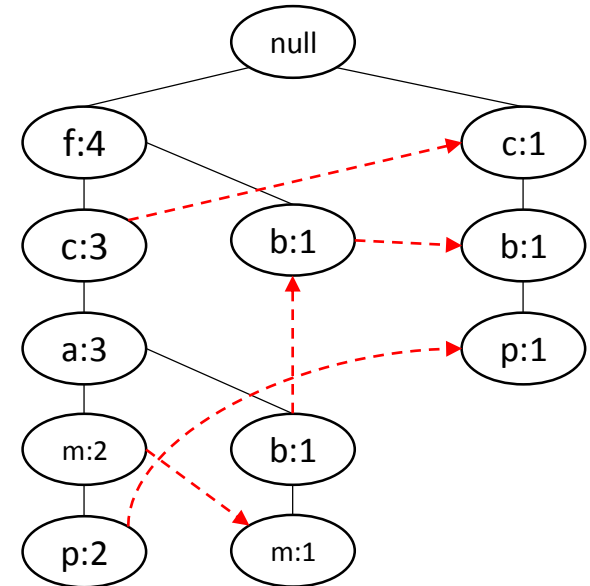
TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p



- 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

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

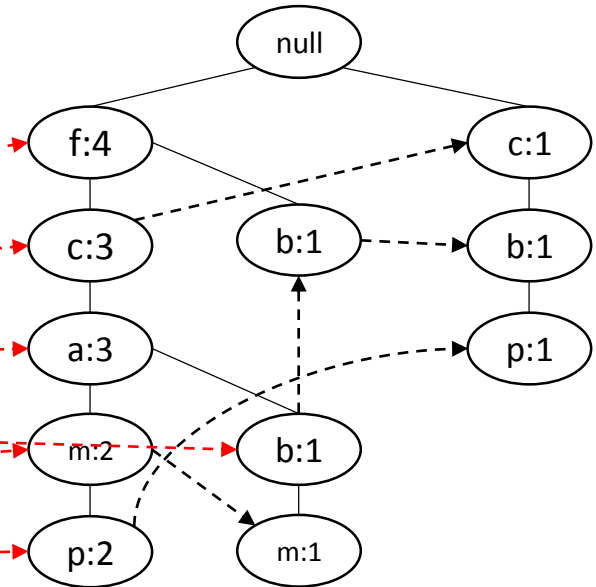


- Create root of the tree, labeled “null”
- Second scan of the DB:
- Scan of first transaction leads to first branch of the tree: $\langle (f:1), (c:1), (a:1), (m:1), (p:1) \rangle$
- Second transaction: since its ordered frequent item list shares a common prefix with $\langle f, c, a \rangle$ with the existing path $\langle f, c, a, m, p \rangle$, 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 $\langle (c:1), (b:1), (p:1) \rangle$
- Fifth transaction: Since same as the first, each count incremented by 1
- Nodes with the same item name are linked via “node-links”

Example

Header
Table

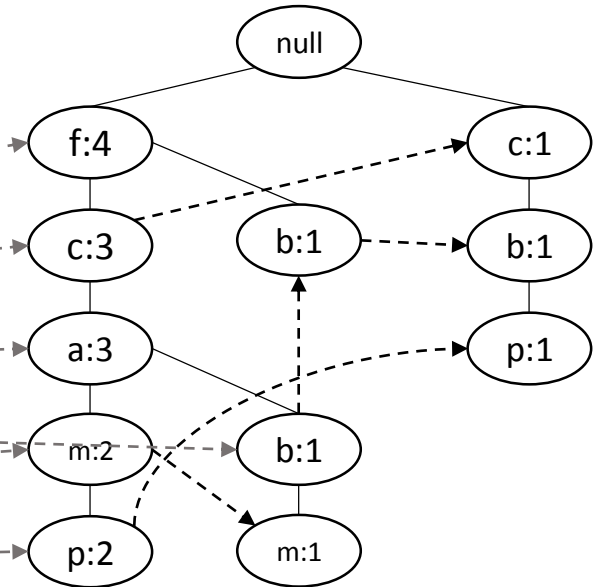
Item	Head of node-links
f	
c	
a	
b	
m	
p	



Example

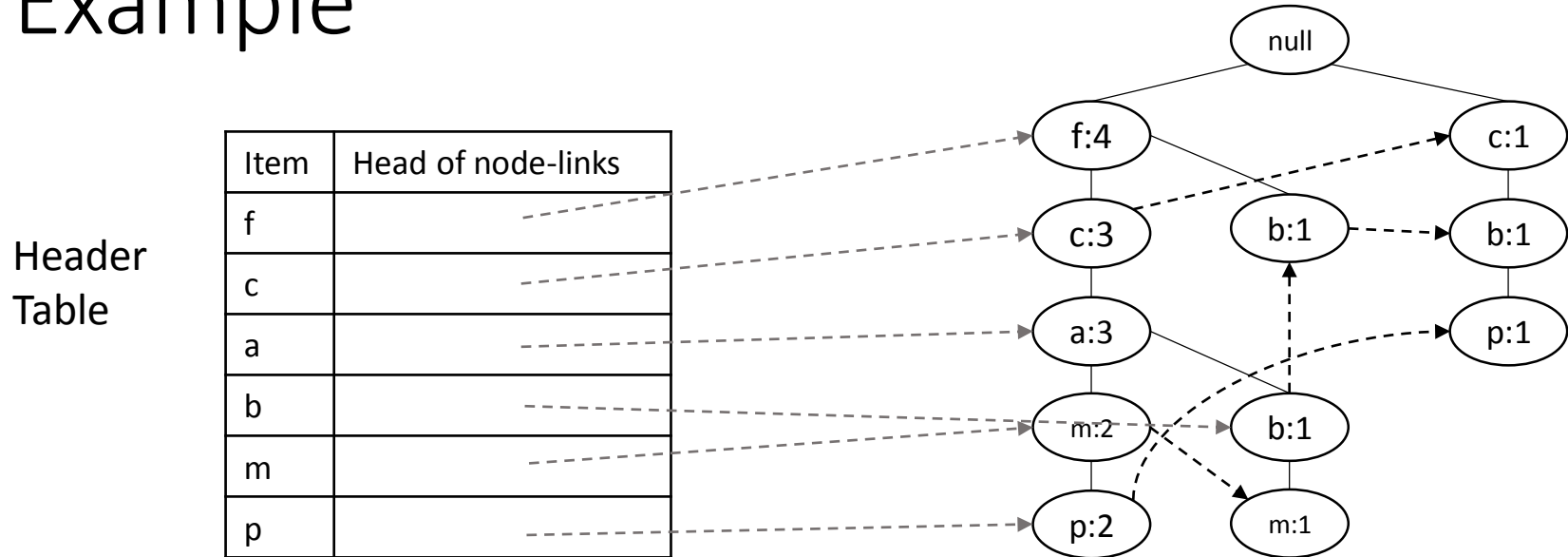
Header
Table

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

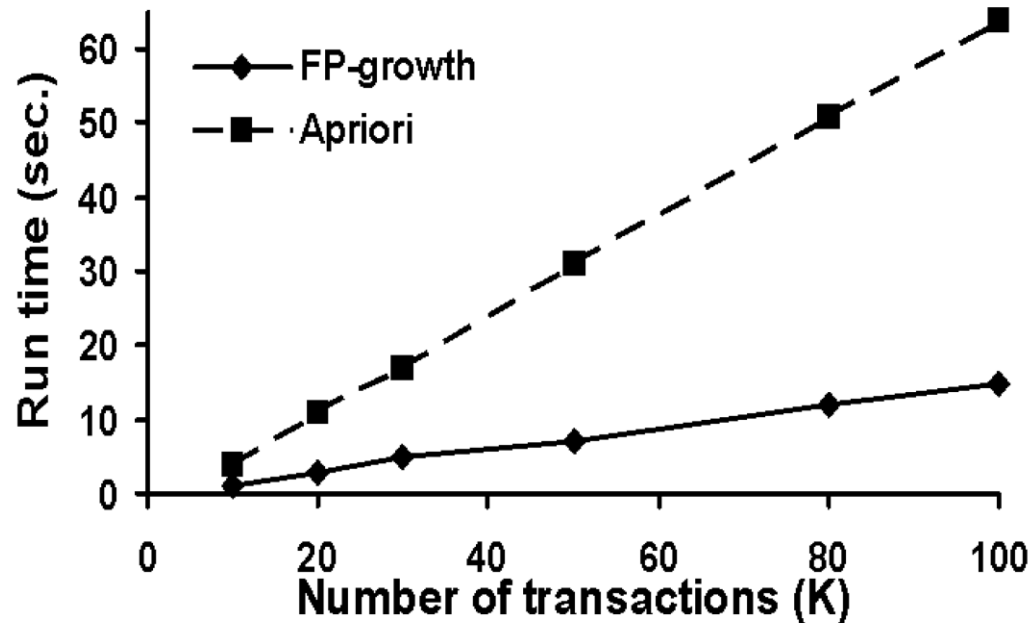
Example



- 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: $\langle f:4, c:3, a:3, m:2, p:2 \rangle$ and $\langle c:1, b:1, p:1 \rangle$
 - 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



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(' '))
```

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

```
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(' '))
```

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

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

Returns an FPGrowthModel object

```
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(' '))
```

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

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

Returns an FPGrowthModel object

```
result = model.freqItemsets().collect()
```

Returns (items, freq) tuples

```
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(' '))
```

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

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

Returns an FPGrowthModel object

```
result = model.freqItemsets().collect()
```

Returns (items, freq) tuples

```
for fi in result:
```

```
    print(fi)
```

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>