**Frequent Pattern Mining - spark.mllib**

Mining frequent items, itemsets, subsequences, or other substructures is usually among the first steps to analyze a large-scale dataset, which has been an active research topic in data mining for years. We are using FP-Growth algorithm for pattern mining.

**FP-growth**

The FP-growth algorithm is described in the paper [Han et al., Mining frequent patterns without candidate generation](http://dx.doi.org/10.1145/335191.335372), where “FP” stands for frequent pattern. Given a dataset of transactions, the first step of FP-growth is to calculate item frequencies and identify frequent items. Different from [Apriori-like](http://en.wikipedia.org/wiki/Apriori_algorithm) algorithms designed for the same purpose, the second step of FP-growth uses a suffix tree (FP-tree) structure to encode transactions without generating candidate sets explicitly, which are usually expensive to generate. After the second step, the frequent itemsets can be extracted from the FP-tree. Inspark.mllib, its been implemented a parallel version of FP-growth called PFP, as described in [Li et al., PFP: Parallel FP-growth for query recommendation](http://dx.doi.org/10.1145/1454008.1454027). PFP distributes the work of growing FP-trees based on the suffices of transactions, and hence more scalable than a single-machine implementation. We refer users to the papers for more details.

spark.mllib’s FP-growth implementation takes the following (hyper-)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.

Below is the implementation of algorithm for our heart bit data to find out the pattern on particular day to check working of heart bit, whether its behaving normal or not. The output for our sample file is included below it.



