UNIT–4 : Frequent Item Sets and Clustering (PYQ Solution)

AKTU · Data Analytics (BADS601) · Solutions by **Chirag Jain**, Ycotes

SECTION A Answers (≈70 Words Each)

Q1. Write the two limitations of K-Means. (Jan 2025)

1. **The number of clusters k must be fixed in advance, and the result depends on the random initial centroids, so different runs may end in different local optima.**
2. **It is sensitive to outliers and noise (the mean is pulled towards them) and works well only for spherical, similar-sized clusters; it fails for clusters of irregular shape or varying density, and cannot handle categorical data directly.**

Q2. Discuss the various categories of clustering techniques. (Jan 2025)

Clustering groups similar objects without labels. The main categories are: Partitioning (K-means, K-medoids – divide data into k clusters); Hierarchical (agglomerative and divisive, producing a dendrogram); Density-based (DBSCAN, OPTICS – dense regions form clusters); Grid-based (STING, CLIQUE – quantised grid cells); Model-based (EM, Gaussian mixtures); and Subspace/high-dimensional methods (CLIQUE, PROCLUS).

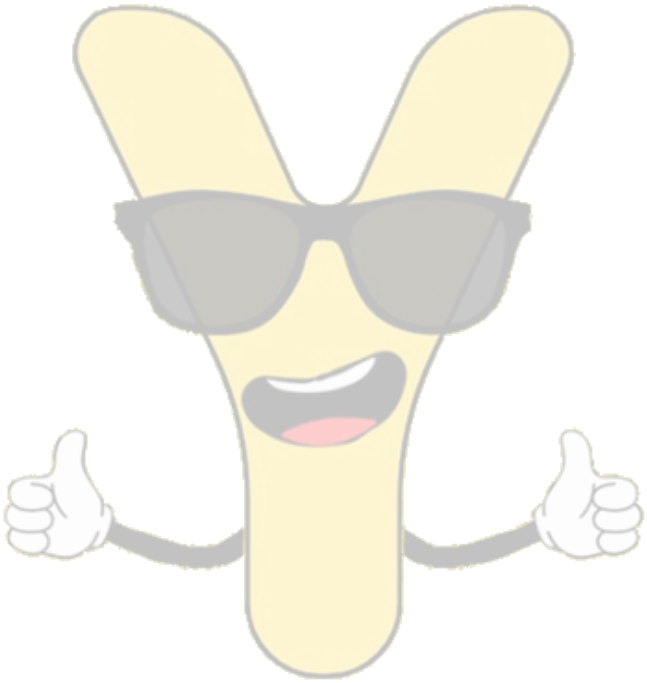
Q3. Explain the principle behind Hierarchical clustering technique. (2022, 2023)

Hierarchical clustering builds a tree of clusters (dendrogram) without fixing k in advance. In the agglomerative (bottom-up) approach every point starts as its own cluster and the two closest clusters are merged repeatedly until one cluster remains; in the divisive (top-down) approach one big cluster is split

repeatedly. Closeness is measured by single, complete or average linkage. Cutting the dendrogram at a level gives the desired clusters.

Q4. Define Lift in Association Data Mining. (2022)

Lift measures how much more often items A and B occur together than expected if they were independent: Lift(A → B) = Confidence(A → B) / Support(B) = P(A ∩ B) / (P(A) × P(B)). Lift = 1 means no relation, lift > 1 means positive association (buying A raises the chance of B), and lift < 1 means negative association. It removes the bias of confidence towards popular items.



SECTION B Answers (≈250 Words Each)

Q1. Explore the term Market Basket Analysis. Is it supervised or unsupervised? Determine how would a company use market basket analysis to improve its marketing strategies? (Jan 2025)

Market Basket Analysis (MBA) is a data-mining technique that finds **associations between items** that customers buy together in the same transaction, using **association rule mining** (Apriori, FP-growth). A rule has the form {Bread, Butter} → {Milk}, meaning customers who buy bread and butter also tend to buy milk.

Key measures

- **Support** = (transactions containing A and B) / (all transactions)
- **Confidence** = support(A ∪ B) / support(A)
- **Lift** = confidence / support(B); lift > 1 shows a real association.

Supervised or unsupervised? MBA is **unsupervised learning**. There is no target or label to predict; the algorithm just discovers frequent patterns and rules hidden in the transaction data.



Market basket analysis

How a company uses it to improve marketing

- **Product placement:** put frequently co-bought items (chips and cold drinks, diapers and baby wipes) close together, or place them far apart to make customers walk past other products.
- **Cross-selling and bundling:** create combo offers and "customers also bought" recommendations on e-commerce sites.
- **Promotions and pricing:** discount one item and raise the margin on its partner; avoid discounting both.
- **Catalogue and store layout design:** arrange aisles and web pages using discovered associations.
- **Targeted campaigns:** send personalised coupons for the item most likely to be bought next, based on past baskets.
- **Inventory planning:** stock complementary items together to avoid stock-outs.
- **Customer segmentation:** group customers by their typical baskets.

Example: a supermarket finds {Diaper} → {Beer} with confidence 70% and lift 1.8. It places beer near the baby section and runs a combined offer, increasing sales of both.

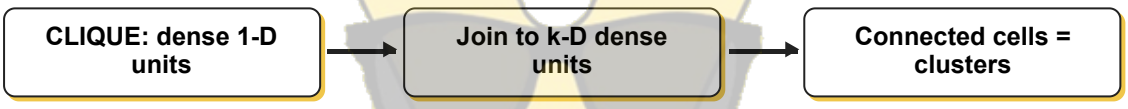
Conclusion: market basket analysis is an unsupervised association-rule technique that helps retailers understand buying behaviour and plan layout, offers and recommendations that raise sales.

Q2. Differentiate between CLIQUE and PROCLUS clustering. (Jan 2025)

Both are **subspace clustering** algorithms for high-dimensional data, where clusters exist only in some dimensions and full-space distance measures fail (curse of dimensionality).

- **CLIQUE (CLustering In QUEst):** a **grid- and density-based, bottom-up** method. Each dimension is divided into ξ equal intervals; a unit is **dense** if it contains more than τ fraction of points. Dense units are found first in 1-D, then joined Apriori-style to form dense units in 2-D, 3-D and higher subspaces (a k-D unit can be dense only if all its (k−1)-D projections are dense). Connected dense units form clusters, described by minimal DNF expressions.
- **PROCLUS (PROjected CLUStering):** a **partitioning, top-down, k-medoid-based** method. It picks k medoids, finds for each medoid the subset of dimensions in which its neighbouring points are closest (smallest average distance), assigns points using **Manhattan segmental distance** in those dimensions, and iteratively replaces bad medoids.

Feature	CLIQUE	PROCLUS
Approach	Grid + density, bottom-up	Partitioning (k-medoids), top-down
Input parameters	Grid size ξ, density threshold τ	Number of clusters k, average dimensions l
Cluster shape	Arbitrary (union of dense cells)	Roughly spherical in its subspace
Overlap	A point may belong to clusters in several subspaces	Each point in exactly one cluster (or outlier)
Outliers	Points in sparse cells	Explicit outlier set
Scalability	Linear in data size, exponential in dimensions of clusters	Faster on very high dimensions
Output	Clusters + DNF description	Partition + relevant dimensions per cluster



CLIQUE works bottom-up; PROCLUS starts from k medoids and refines

Conclusion: CLIQUE automatically finds clusters in all subspaces and handles arbitrary shapes, while PROCLUS gives a clean partition into k projected clusters with less computation.

Q3. What are the advantages of PCY algorithm over Apriori Algorithm? / Why PCY algorithm is preferred over Apriori algorithm? (2022, 2023)

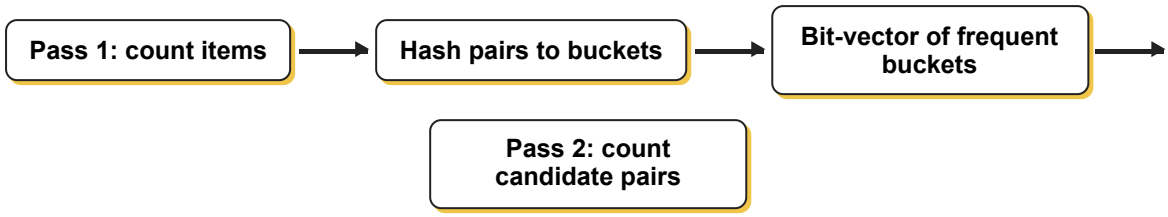
Apriori finds frequent itemsets level by level. In pass 1 it counts single items; in pass 2 it must count **every pair of frequent items**, so the candidate pairs C_2 can be huge (for m frequent items, m(m − 1)/2 pairs). In large data this exhausts main memory.

PCY (Park-Chen-Yu) improves pass 1 by using the **idle memory** to hash pairs.

How PCY works

- **Pass 1:** count each item. In addition, while reading every basket, generate all its pairs, hash each pair to a bucket, and **increment the bucket count**. After the pass, convert the bucket counts to a **bit-vector**: 1 if the bucket count ≥ support s, else 0. Frequent items are also found.

- **Pass 2:** count a pair {i, j} only if **(1) both i and j are frequent, and (2) the pair hashes to a bucket whose bit is 1**.



PCY algorithm

Advantages over Apriori

- **Fewer candidate pairs:** a bucket whose total count is below s cannot contain a frequent pair, so many pairs are eliminated before counting in pass 2.
- **Less memory in pass 2:** only pairs with both items frequent *and* a frequent bucket are counted, so the counting table is much smaller.
- **Uses spare memory:** pass 1 of Apriori leaves most of main memory empty; PCY uses it for the hash table.
- **Faster on large data** where the number of pairs is the bottleneck, and still needs only **two passes** for pairs.
- The bit-vector needs only 1 bit per bucket (versus a 4-byte count), so the memory cost is small.

Limitation: if most buckets are frequent, there is little pruning; multistage and multihash variants overcome this.

Basis	Apriori	PCY
Pass 1	Count items	Count items and hash pairs
Pass 2 candidates	All pairs of frequent items	Only pairs in frequent buckets
Memory use	Pass 1 under-used	Pass 1 memory used well
Pair counting	Larger table	Smaller table

Conclusion: PCY is preferred because it reduces the number of candidate pairs and the memory needed in the costly second pass.

SECTION C Answers (≈500 Words Each)

Q1. Cluster the following eight points into three clusters: A1(2, 10), A2(2, 5), A3(8, 4), A4(5, 8), A5(7, 5), A6(6, 4), A7(1, 2), A8(4, 9). Initial cluster centers are A1(2, 10), A4(5, 8) and A7(1, 2). The distance function is P(a, b) = |x2 − x1| + |y2 − y1|. Use K-Means Algorithm to find the three cluster centers after clustering all eight points. (Jan 2025)

K-means: (1) choose k initial centres, (2) assign each point to the nearest centre, (3) recompute each centre as the mean of its points, (4) repeat until no point changes cluster. Distance used is the **Manhattan distance**. Initial centres: C1 = (2,10), C2 = (5,8), C3 = (1,2).

Iteration 1 – distances (to C1, C2, C3)

Point	C1 (2,10)	C2 (5,8)	C3 (1,2)	Cluster
A1 (2,10)	0	5	9	C1
A2 (2,5)	5	6	4	C3
A3 (8,4)	12	7	9	C2
A4 (5,8)	5	0	10	C2
A5 (7,5)	10	5	9	C2
A6 (6,4)	10	5	7	C2
A7 (1,2)	9	10	0	C3
A8 (4,9)	3	2	10	C2

Clusters: C1 = {A1}; C2 = {A3, A4, A5, A6, A8}; C3 = {A2, A7}.
New centres: C1 = (2, 10); C2 = ((8+5+7+6+4)/5, (4+8+5+4+9)/5) = **(6, 6)**; C3 = ((2+1)/2, (5+2)/2) = **(1.5, 3.5)**.

Iteration 2 – centres (2,10), (6,6), (1.5,3.5)

Point	C1	C2	C3	Cluster
A1 (2,10)	0	8	7	C1
A2 (2,5)	5	5	2	C3
A3 (8,4)	12	4	7	C2
A4 (5,8)	5	3	8	C2
A5 (7,5)	10	2	7	C2
A6 (6,4)	10	2	5	C2
A7 (1,2)	9	9	2	C3
A8 (4,9)	3	5	8	C1

Clusters: C1 = {A1, A8}; C2 = {A3, A4, A5, A6}; C3 = {A2, A7}.
New centres: C1 = ((2+4)/2, (10+9)/2) = **(3, 9.5)**; C2 = ((8+5+7+6)/4, (4+8+5+4)/4) = **(6.5, 5.25)**; C3 = **(1.5, 3.5)**.

Iteration 3 – centres (3,9.5), (6.5,5.25), (1.5,3.5)

Point	C1	C2	C3	Cluster
A1 (2,10)	1.5	9.25	7	C1
A2 (2,5)	5.5	4.75	2	C3
A3 (8,4)	10.5	2.75	7	C2
A4 (5,8)	3.5	4.25	8	C1
A5 (7,5)	8.5	0.75	7	C2
A6 (6,4)	8.5	1.75	5	C2
A7 (1,2)	9.5	9.75	2	C3
A8 (4,9)	1.5	6.75	8	C1

A4 moves to C1. Clusters: C1 = {A1, A4, A8}; C2 = {A3, A5, A6}; C3 = {A2, A7}.
New centres: C1 = ((2+5+4)/3, (10+8+9)/3) = **(3.67, 9)**; C2 = ((8+7+6)/3, (4+5+4)/3) = **(7, 4.33)**; C3 = **(1.5, 3.5)**.

Iteration 4 – centres (3.67,9), (7,4.33), (1.5,3.5)

- A1: 2.67, 10.67, 7 → C1; A2: 5.67, 5.67, 2 → C3; A3: 9.33, 1.33, 7 → C2; A4: 2.33, 5.67, 8 → C1; A5: 7.33, 0.67, 7 → C2; A6: 7.33, 1.33, 5 → C2; A7: 9.67,

8.33, 2 → C3; A8: 0.33, 8.33, 8 → C1.

- No point changes its cluster, so the algorithm **stops**.



K-means steps

Final result

Cluster	Points	Centre
1	A1, A4, A8	(3.67, 9)
2	A3, A5, A6	(7, 4.33)
3	A2, A7	(1.5, 3.5)

Conclusion: after convergence the three cluster centres are (3.67, 9), (7, 4.33) and (1.5, 3.5).

Q2. The database has 6 transactions. Assume Support threshold = 50%, Confidence = 60%. (TID 10: Beer, Nuts, Diaper; 20: Beer, Coffee, Diaper; 30: Beer, Diaper, Eggs; 40: Nuts, Eggs, Milk; 50: Nuts, Coffee, Diaper, Eggs, Milk; 60: Beer, Nuts, Diaper). i) Use Apriori algorithm to find all frequent itemsets. ii) Show all the strong association rules (with support and confidence). (Jan 2025)

Apriori principle: every subset of a frequent itemset must be frequent. Minimum support = 50% of 6 = **3 transactions**; minimum confidence = 60%.

Step 1 – C1 and L1 (count of single items)

Item	Count	Frequent?
Beer	4 (TID 10, 20, 30, 60)	Yes
Nuts	4 (10, 40, 50, 60)	Yes
Diaper	5 (10, 20, 30, 50, 60)	Yes
Coffee	2	No
Eggs	3 (30, 40, 50)	Yes
Milk	2	No

L1 = {Beer, Nuts, Diaper, Eggs}.

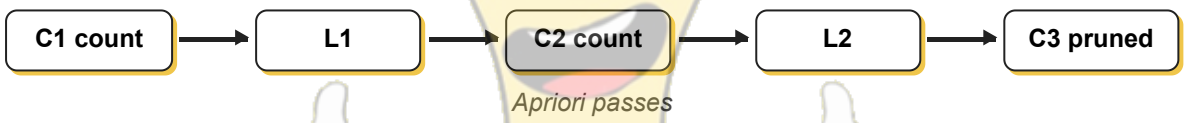
Step 2 – C2 and L2 (pairs from L1)

Pair	Count	Frequent?
Beer, Nuts	2 (10, 60)	No
Beer, Diaper	4 (10, 20, 30, 60)	Yes
Beer, Eggs	1 (30)	No
Nuts, Diaper	3 (10, 50, 60)	Yes
Nuts, Eggs	2 (40, 50)	No
Diaper, Eggs	2 (30, 50)	No

L2 = {Beer, Diaper} and {Nuts, Diaper}.

Step 3 – C3 and L3. Joining L2 gives {Beer, Nuts, Diaper}, but its subset {Beer, Nuts} is infrequent, so it is pruned (actual count is 2 < 3). L3 = empty; the algorithm stops.

(i) All frequent itemsets: {Beer}, {Nuts}, {Diaper}, {Eggs}, {Beer, Diaper}, {Nuts, Diaper}.



Apriori passes

(ii) Association rules from the frequent 2-itemsets
Support of a rule = support of the itemset; Confidence(X → Y) = count(X ∪ Y) / count(X).

Rule	Support	Confidence	Strong (≥ 60%)?
Beer → Diaper	4/6 = 66.7%	4/4 = 100%	Yes
Diaper → Beer	4/6 = 66.7%	4/5 = 80%	Yes
Nuts → Diaper	3/6 = 50%	3/4 = 75%	Yes
Diaper → Nuts	3/6 = 50%	3/5 = 60%	Yes

All four rules satisfy both thresholds, so all are **strong association rules**.

Conclusion: the frequent itemsets are the four single items plus {Beer, Diaper} and {Nuts, Diaper}, and Beer ⇄ Diaper and Nuts ⇄ Diaper are the strong rules.

Q3. Differentiate between CLIQUE and ProCLUS clustering. (2022)

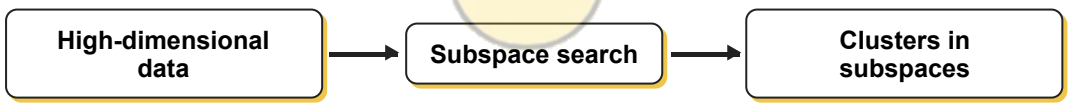
Both CLIQUE and PROCLUS are **subspace clustering** algorithms for high-dimensional data, where clusters exist only in some of the dimensions and distance in the full space loses meaning (the "curse of dimensionality").

CLIQUE (Clustering In QUEst) – a **grid-based, density-based** and **bottom-up** method.

1. Divide each dimension into ξ equal intervals, forming rectangular grid units.
2. A unit is **dense** if it contains more than τ (threshold) points.
3. Find dense units in 1-D, then, using an Apriori-like pruning ("if a unit is dense in k dimensions, its projections are dense in k − 1 dimensions"), find dense units in higher-dimensional subspaces.
4. Connected dense units in a subspace form a cluster, described by a minimal set of rules.

PROCLUS (PROjected CLUStering) – a **partition-based (k-medoid), top-down** method.

1. Choose a set of well-separated medoid candidates by greedy sampling.
2. Pick k medoids and for each find the **dimensions** in which its neighbouring points are closest (smallest spread).
3. Assign every point to the nearest medoid using distance in that medoid's own subspace (Manhattan segmental distance).
4. Replace bad medoids iteratively and refine; points far from all medoids are marked as **outliers**.



Subspace clustering

Basis	CLIQUE	PROCLUS
Type	Grid and density based	Partitioning (k-medoid) based
Strategy	Bottom-up, from 1-D upward	Top-down, from full space to projected subspaces
Input parameters	Grid size ξ and density threshold τ	Number of clusters k and average dimensionality l
Number of clusters	Found automatically	Must be given (k)
Cluster shape	Arbitrary, unions of grid cells	Roughly spherical around medoids
Subspaces	Finds clusters in all subspaces; may report overlapping clusters	Each cluster has one subspace; clusters do not overlap
Outliers	Points in non-dense cells	Explicit outlier handling
Output	Clusters with DNF (rule) descriptions	Medoids, point lists and dimension sets
Complexity	Linear in records, exponential in dimension of subspace	Linear in data size; efficient for large data
Weakness	Many overlapping results; grid sensitivity	Need to know k and l; poor for non-spherical clusters

Conclusion: CLIQUE discovers dense regions in every subspace without knowing k, whereas PROCLUS partitions the data into k disjoint projected clusters; the choice depends on whether overlapping, grid-based results or a fixed number of non-overlapping clusters is required.

Q4. Tid | Items bought: 10 Beer, Nuts, Diaper; 20 Beer, Coffee, Diaper; 30 Beer, Diaper, Eggs; 40 Nuts, Eggs, Milk; 50 Nuts, Coffee, Diaper, Eggs, Milk. Find all the Association rules from the above given transactions with given minsup = 50%, minconf = 50%. (2022)

There are **5 transactions**. Minimum support = 50% × 5 = 2.5, so an itemset must appear in **at least 3 transactions**. Minimum confidence = 50%.

Step 1 – frequent 1-itemsets (L1)

Item	Count	Frequent?
Beer	3 (10, 20, 30)	Yes
Nuts	3 (10, 40, 50)	Yes
Diaper	4 (10, 20, 30, 50)	Yes
Coffee	2	No
Eggs	3 (30, 40, 50)	Yes
Milk	2	No

Step 2 – candidate 2-itemsets (from Beer, Nuts, Diaper, Eggs)

Pair	Count	Frequent?
Beer, Nuts	1 (10)	No
Beer, Diaper	3 (10, 20, 30)	Yes
Beer, Eggs	1 (30)	No
Nuts, Diaper	2 (10, 50)	No
Nuts, Eggs	2 (40, 50)	No
Diaper, Eggs	2 (30, 50)	No

L2 = {Beer, Diaper}.

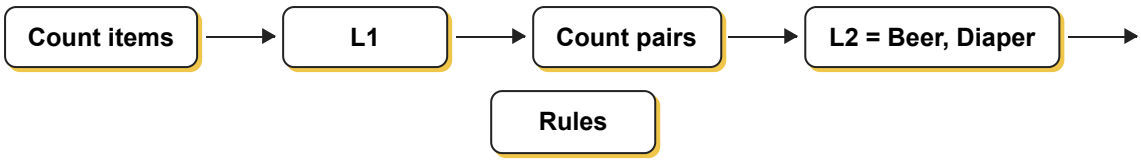
Step 3 – 3-itemsets: none can be formed from a single frequent pair, so the algorithm stops.

Frequent itemsets: {Beer}, {Nuts}, {Diaper}, {Eggs}, {Beer, Diaper}.

Step 4 – generate rules from {Beer, Diaper}

Rule	Support	Confidence = count(Beer, Diaper)/count(antecedent)	Accepted (≥ 50%)?
Beer → Diaper	3/5 = 60%	3/3 = 100%	Yes
Diaper → Beer	3/5 = 60%	3/4 = 75%	Yes

Both rules satisfy minsup 50% and minconf 50%.



Rule generation

Answer: the association rules are **Beer → Diaper (support 60%, confidence 100%)** and **Diaper → Beer (support 60%, confidence 75%)**. Lift of both = 0.6 / (0.6 × 0.8) = 1.25 (> 1), showing a positive association.

Q5. Explain SON algorithm to find all or most frequent item sets using at most two passes. (2023)

The **SON algorithm** (Savasere, Omiecinski and Navathe) finds frequent itemsets in a **large data set in two passes**, and is well suited to **parallel / MapReduce** computing. It is based on the observation that **an itemset that is frequent in the whole data must be frequent in at least one chunk (partition)** of the data.

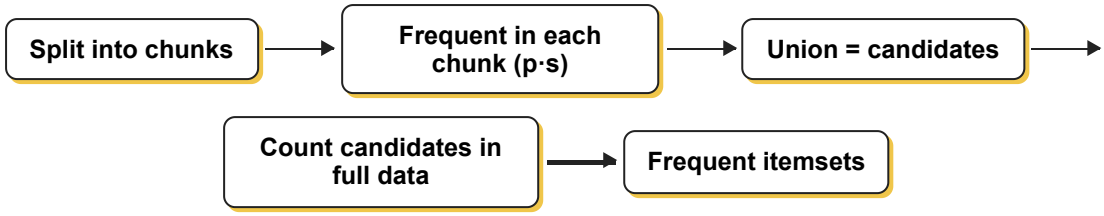
Idea: split the baskets into chunks that each fit in memory. If the chunk is a fraction p of the whole file and the global support threshold is s, use the **lower threshold p·s** in each chunk. An itemset frequent overall cannot be below p·s in every chunk.

Pass 1 – find candidates

- Read the file chunk by chunk.
- In each chunk run an in-memory algorithm (Apriori, PCY, Simple/Randomized) with threshold p·s to find the frequent itemsets of that chunk.
- Take the **union of all itemsets found frequent in any chunk** – these are the **candidate itemsets**.
- Because no truly frequent itemset can be missed (no false negatives), the candidate set contains every globally frequent itemset.

Pass 2 – count candidates

- Read the whole file again and **count the exact support of every candidate** itemset.
- Keep those whose total count ≥ s; they are the **frequent itemsets**.
- This removes the false positives, so the result is exact.



SON algorithm

MapReduce implementation

- **First Map:** each mapper gets a chunk, finds its frequent itemsets with threshold p·s and emits (F, 1). **First Reduce:** removes duplicates and outputs the candidate itemsets.
- **Second Map:** each mapper receives all candidates and one chunk, counts how many times each candidate appears and emits (C, count). **Second Reduce:** adds the counts for each C and outputs those with total ≥ s.

Example: s = 100 in 10,000 baskets with 10 chunks of 1,000 (p = 0.1) → local threshold 10. An itemset appearing 100 times overall must reach at least 10 in some chunk, so it becomes a candidate and is confirmed in pass 2.

Advantages: only two passes over the data, no false negatives, exact result, easily parallelised, and does not need main memory for the whole data set.

Limitation: many false-positive candidates may appear when the local threshold is low, making pass 2 costly. The *Toivonen* algorithm gives results with one pass but may need a repeat.

Conclusion: SON finds all frequent itemsets exactly with two passes: candidates from chunks in pass 1, true counts in pass 2.
