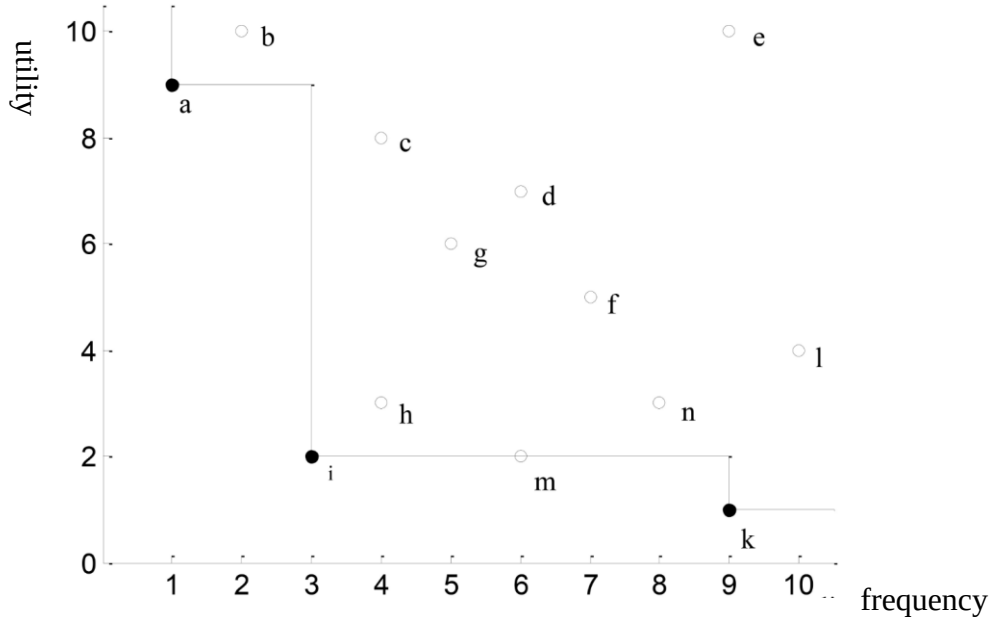
SFU-CE: Skyline Frequent-Utility Itemset Discovery Using the Cross-Entropy Method

Wei Song, Chuanlong Zheng
North China University of Technology
songwei@ncut.edu.cn

Skyline frequent-utility itemset

Skyline frequent-utility itemset(SFUI)

- Itemset mining is used to discover interesting and useful patterns
- Consider both frequency and profit of itemset
- SFUIs are itemsets that are not dominated by any other itemsets in terms of both frequency and utility



Basic Idea of SFU-CE

- The frequency of itemset X in D is defined as $f(X)$
  - $f(AB) = 2$
- The utility of itemset X in D is defined as $u(X) = \sum_{X \subseteq T_d \wedge T_d \in D} u(X, T_d)$
  - $u(A) = 6 \times 1 + 6 \times 4 = 30$
- The transaction weighted utility of itemset is defined as $TWU(X) = \sum_{X \subseteq T_d \wedge T_d \in D} TU(T_d)$
  - $TWU(AB) = 30 + 20 = 50$
- An itemset X dominates another itemset Y in D, if $f(X) \geq f(Y)$ and $u(X) > u(Y)$, or $f(X) > f(Y)$ and $u(X) \geq u(Y)$.

Example database

TID	Transactions	TU
1	(A, 1) (B, 3) (C, 1) (E,3) (G,1)	30
2	(B, 1) (C, 1) (F, 2)	10
3	(B, 2) (C, 1) (D, 1) (E, 2)	20
4	(A, 1) (B, 1) (C, 1) (D, 1) (E, 1) (F, 1) (G,1)	20
5	(D, 2) (E, 2)	10
6	(G,1)	4

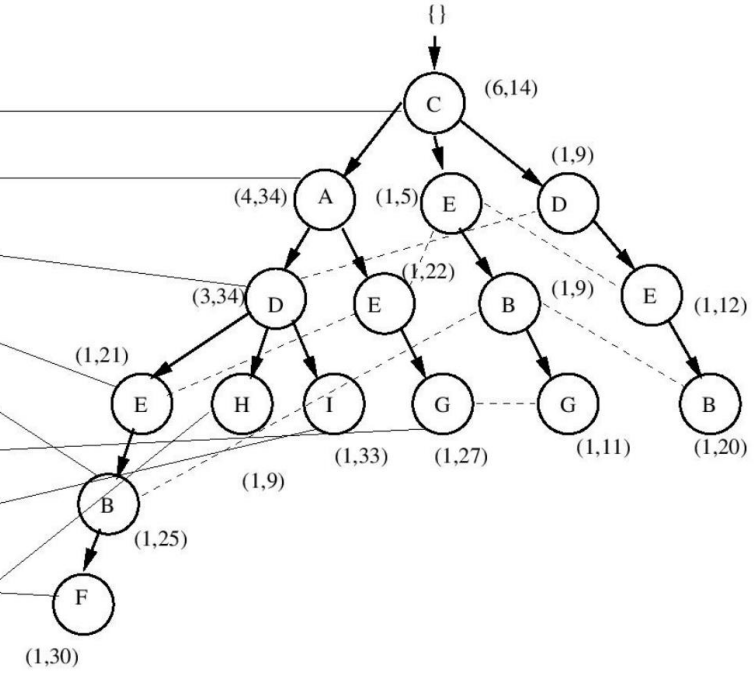
Profit table

Item	A	B	C	D	E	F	G
Profit	2	5	3	3	2	1	4

Related work

- Goyal et al. introduced SFUIs and proposed the SKYMINE algorithm using utility pattern-tree (UP-Tree) for mining SFUIs
- Using the utility list structure (UL), Pan et al. proposed the SFU-Miner algorithm to discover SFUIs

Item	TWU	Link
C	130	
A	99	
D	92	
E	88	
B	61	
G	38	
I	33	
F	30	
H	9	



E		
tid	*iutil*	*rutil*
2	4	14
4	4	2
5	8	14

C		
tid	*iutil*	*rutil*
1	2	7
2	3	11
3	2	9
4	2	0
6	1	11

B		
tid	*iutil*	*rutil*
1	2	5
2	2	9
5	4	10
6	8	3

A		
tid	*iutil*	*rutil*
2	4	5
3	4	5
5	5	5
6	3	0

D		
Tid	*lutil*	*Rutil*
1	*5*	*0*
2	*5*	*0*
3	*5*	*0*
5	*5*	*0*
7	*5*	*0*

Related work

- Compared with the UP-Tree, the UL does not generate a large number of candidates, so the efficiency is improved
- Lin et al. proposed SKYFUP-D and SKYFUP-B for mining SFUIs by depth-first and breadth-first traversing the search space, respectively
- We modeled the SFUI mining problem using cross entropy, proposed critical utility pruning for unpromising itemsets and used random mutation to increase the sample diversity. Finally, the overall performance is improved

Cross-Entropy Method

- Let $Y = (y_1, y_2, ..., y_n)$ be an n-dimensional binary vector, the goal of the CE method is to reconstruct the unknown vector Y by maximizing the function $S(X)$ using a random search algorithm

$$S(X) = n - \sum_{j=1}^{n} |x_j - y_j|$$

- In CE method, we use a probability vector(PV) to represent the probability that each bit in vector Y is the SFUI

$$P_{t,j} = \sum_{i=1}^{N} I_{\{s(X_i) \geq \gamma_t\}} \times I_{X_{ij}=1} \Big/ I_{\{s(X_i) \geq \gamma_t\}}$$

- When the probability vector becomes a binary vector or reaches the maximum number of iterations, the iteration ends

Critical Utility Pruning

- **Definition 1.** The critical utility of SFUIs (*CUS*) in transaction database *D* is the maximal utility of single items that have the highest frequency, and is defined as

$$CUS = \max\{u(i)|f(i) = f_{max}\}$$

- **Corollary 1.** Let *X* be an itemset. If *TWU*(*X*) < *CUS*, *X* and all itemsets containing *X* are not SFUIs

- Using Corollary 1, once a 1-itemset is found to have *TWU* lower than the *CUS*, this itemset and all its supersets can be pruned safely. In the running example, because *TWU*(*F*) = 30 < *CUS* = 35, *F* will be deleted from the database

Random Mutation

- To increase the sample diversity, the random mutation is used
- It should be noted that only the bits corresponding to probabilities strictly higher than 0.5 perform *RM*

PV: <0.3, 0.8, 0.7, 0.2, 0.6, 0.1>

↓ Random bits: 1,4

IV: <010010>

↓ ABCDEF Represent

Itemset: BE

SFU-CE algorithm

Algorithm 1 describes the proposed SFU-CE algorithm.

Algorithm 1	**SFU-CE**
Input	Transaction database ***D***, sample numbers N, quantile parameter ρ, maximum number of iterations *max_iter*, mutation factor α
Output	SFUIs
1	Initialization( );
2	**while** $t \leq max\_iter$ **and** P_t is not a binary vector **do**
3	Calculate P_t using Eq. 3;
4	Perform RM;
5	Generate the remaining IVs according to P_t;
6	Sort the N itemsets in utility-descending order and denote them by $X_1, X_2, \ldots, X_N$;
7	**for** $i = 1$ to N **do**
8	$CSFUI$ = SFUI-Filter(X_i, $CSFUI$);
9	**end for**
10	Calculate γ_t using Eq. 2;
11	t ++;
12	**end while**
13	Output all SFUIs in *CSFUI*.

SFU-CE algorithm

Algorithm 2 is initialization algorithm.

Algorithm 2	**Initialization( )**
1	Scan ***D*** once to delete items with TWU values lower than the CUS;
2	Represent the database using a bitmap;
3	$P_0 = (1/2, 1/2, \ldots, 1/2)$;
4	Generate all itemsets of the first iteration with P_0;
5	Sort the N itemsets in utility-descending order and denote them by $X_1, X_2, \ldots, X_N$;
6	$CSFUI = \varnothing$;
7	**for** $i = 1$ to N **do**
8	$CSFUI$ = SFUI-Filter(X_i, $CSFUI$);
9	**end for**
10	Calculate γ_t using Eq. 2;
11	$t = 1$;

SFU-CE algorithm

Algorithm 3 is the utility filter algorithm for updating the *CSFUI* list.

Algorithm 3	**SFUI-Filter(*X*, CSFUI)**
Input	Itemset *X*, set of candidate SFUIs *CSFUI*
Output	*CSFUI*
1	**if** $u(X) \geq$ CUS **then**
2	Remove all itemsets dominated by *X* in *CSFUI*;
3	**if** *X* is dominated by any itemset in *CSFUI* **then**
4	**return** *CSFUI*;
5	**end if**
6	$X \rightarrow$ *CSFUI*;
7	**end if**
8	**return** *CSFUI*;

Experimental environment

- **4-Core 3.40 GHz CPU**
- **8 GB memory**
- **64-bit Microsoft Windows 10**
- **Java programming language**
- **Algorithms for comparison**
  - SKYMINE, SFU-Miner , SKYFUP-D, SKYFUP-B
  - Codes downloaded from SPMF an open-source data mining library http://www.philippe-fournier-viger.com/spmf/

Datasets & parameters

Dataset details

- We use three parameters to describe four different datasets, these datasets can also be downloaded or generated from SPMF

Datasets	Avg. trans. length	No. of items	No. of trans.
T25I50D10k	25	50	10,000
T35I10050k	35	100	50,000
Chess	37	75	3,156
Connect	43	129	67,557

Runtime

- We compare the efficiency of the five algorithms in four different datasets. From the results, we can find that SFU-CE is the most efficient algorithm in all datasets

Unit (Sec)	SKYMINE	SFU-Miner	SKYFUP-D	SKYFUP-B	SFU-CE
T25I50D10k	20.03	1915.78	77.75	-	7.82
T35I10050k	163.67	-	1006.42	-	69.28
Chess	-	167.53	37.92	36.22	16.20
Connect	-	-	6523.72	-	673.28

Accuracy

- We compare the percentage of SFUIs discovered by SFU-CE to the actual SFUIs , the results shows that SFU-CE discovered SFUIs with 100% accuracy, except on the Connect dataset

Datasets	*Num_CE*	*Num*	*Acc* (%)
T25I50D10k	6	6	100
T35I10050k	4	4	100
Chess	35	35	100
Connect	42	46	91.30

Diversity

- To verify the effect of *RM* proposed, we evaluated the degree of diversity of the mining results using the bit edit distance (*BED*). The results demonstrated that the proposed *RM* strategy improved the diversity of the samples in most datasets

(a) BED for the T25I50D10k dataset

Percentage of total number of iterations (%)	SFU-Base		SFU-CE	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*
25	4.58	18.0	4.89	18.0
50	2.46	13.0	2.73	15.0
75	1.56	9.0	1.64	9.0
100	0.0	0.0	0.27	1.0

(b) BED for the T35I10050k dataset

Percentage of total number of iterations (%)	SFU-Base		SFU-CE	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*
25	11.82	23.0	10.85	24.0
50	1.97	9.0	2.92	13.0
75	1.06	4.0	1.55	6.0
100	0.0	0.0	0.0	0.0

(c) BED for the Chess dataset

Percentage of total number of iterations (%)	SFU-Base		SFU-CE	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*
25	13.40	28.0	11.46	28.0
50	5.83	16.0	6.36	16.0
75	1.97	7.0	3.95	13.0
100	0.0	0.0	2.91	12.0

(d) BED for the Connect dataset

Percentage of total number of iterations (%)	SFU-Base		SFU-CE	
	Ave_BED	*Max_BED*	*Ave_BED*	*Max_BED*
25	12.73	26.0	8.22	24.0
50	7.62	19.0	7.39	19.0
75	2.43	9.0	5.10	18.0
100	0.0	0.0	3.92	16.0

Conclusions

- **In this paper, we studied the SFUIM problem from the perspective of cross entropy and proposed an SFUIM algorithm called SFU-CE :**
  - we used utility as the optimization object of CE and proposed critical utility to filter intermediate results
  - To improve the diversity of each sample, we designed *RM* to generate some new itemsets
  - Experiments on publicly available datasets demonstrated that the SFU-CE algorithm was efficient, accurate, and produced diverse samples

Thank You!