„Mining top-K frequent itemsets from data streams“

R.C.-W. Wong A.W.-C. Fu

Gliederung

1. Einleitung
2. Chernoff-basierter Algorithmus
3. top-K lossy counting Algorithmus
4. Empirischer Vergleich
5. Mining top-K itemsets in a sliding window
6. Zusammenfassung
7. Fragen / Diskussion

Definitionen

- Itemset
  - Eine Menge bestehend aus l Items (l-itemset)
- K-th frequent itemset
  - l-itemset, das nach Sortierung der Frequenz des Auftretens auf Position k ist
- Top K-frequent itemsets
  - alle l-itemsets die mindestens am k-häufigsten aufgetreten sind

Anwendungen

- Das Wissen über frequent itemsets kann nützlich sein bei:
  - Click-Streams
  - Netzwerk-Monitoring
  - Finanzmarkt-Monitoring
  - Analyse von Bestellungen
  - und vieles mehr

Einleitung

- Das Finden von frequent itemsets ist im Bereich Data Mining gut erforscht (→ statische Datenmenge)
- Aber:
  - Data Stream ist potentiell unendlich lang
    → die Daten können nicht alle gespeichert werden
  - Die Ergebnisse werden in Real Time erwartet
    (Im Data Mining gibt es normalerweise Ergebnisse erst am Ende)
- Es werden also neue Methoden gebraucht um frequent itemsets in Data Streams zu finden

Problem A

- „Finde alle Pattern, die eine Frequenz größer als s haben"
- Problem:
  - Benutzer muss eine Grenze s setzen
    - s zu hoch: kaum frequent-pattern werden gefunden
    - s zu tief: zu viele frequent-pattern werden gefunden
  - Für jede Datenmenge muss ein neues s gefunden werden, das brauchbare Ergebnisse liefert
- Alternative:
  Benutzer gibt an wie viele Ergebnisse er haben möchte → Problem B

Problem B

- „Finde die K am häufigsten auftretenden I-Itemsets“
- In Datenströmen ist es i.a. nicht möglich die Frequenzen exakt zu bestimmen (→ Speicherplatz, Laufzeit)
- Frequenz der Itemsets muss geschätzt werden
- Es muss eine Frequenz-Schranke geschätzt werden um weniger häufige Itemsets abschneiden zu können

Gliederung

1. Einleitung
2. **Chernoff-basierter Algorithmus**
3. top-K lossy counting Algorithmus
4. Empirischer Vergleich
5. Mining top-K itemsets in a sliding window
6. Zusammenfassung
7. Resumé

Chernoff-basierter Algorithmus

- Wird gesteuert über 2 Parameter
  - Fehlerschranke (ε)
  - Zuverlässigkeit (δ)
- Grundversion geht von Unabhängigkeit der Daten aus
- Basiert auf der Chernoff-Schranke

Chernoff-Schranke (I)

- Schranke für die Wahrscheinlichkeit, dass eine Zufallsvariable um einen bestimmten Wert vom erwarteten Wert abweicht
- Voraussetzung:
  Eine Folge von Beobachtungen einzelner Bernoulli-Experimente o_i

Chernoff-Schranke (II)

beobachtete rel. Häufigkeit

Abweichungsfaktor

$$P(|\tilde{x}-x| \geq x\gamma) \leq 2\,e^{\frac{-nx\gamma^2}{2}}$$

tatsächliche Wahrscheinlichkeit

$$\epsilon := x\gamma$$

$$P(|\tilde{x}-x| \geq \epsilon) \leq \underline{2\,e^{\frac{-n\epsilon^2}{2x}}}$$

$$\delta = 2\,e^{\frac{-n\epsilon^2}{2x}}$$

$$\boxed{\epsilon = \sqrt{\frac{2\cdot x\cdot \ln(2/\delta)}{n}}}$$

Verwendung der Chernoff-Schranke

- Die Bernoulli-Experimente werden als Transaktionen interpretiert
- Das Ergebnis ist jeweils ob ein Itemset X in der Transaktion existiert
- x ist dann der erwartete Support von X
  (Support: Anteil der Transaktionen die X enthalten)
- δ ist die Wahrscheinlichkeit, dass der Support über- oder unterschätzt wird
- ε gibt die Abweichung von erwarteten und beobachteten Support an

Chernoff-basierter Algorithmus (I)

- Alle Itemsets werden in 2 Gruppen aufgeteilt:
  - potential K-frequent itemsets
  - unpromising itemsets
- Bedingung für potential K-frequent itemsets:
  - s_k = beobachtete Support des k-th frequent itemsets
  - dann gilt: $\epsilon_{s_k} = \sqrt{\frac{2 s_k \ln(2/\delta)}{n}}$
  - $\tilde{s}(X)$ = beobachte Support eines Itemsets X
  - $\tilde{s}(X) \geq s_k - 2 \cdot \epsilon_{s_k}$ $\rightarrow$ X ist potential K-frequent

Chernoff-basierter Algorithmus (II)

$n = 0$, $P_l = \emptyset$, $F_l = \emptyset$

for every Batch of R transactions **do**

  $n = n + R$

  for all l such that $1 <= l <= L$ **do**

    find potential K-frequent itemsets in the current batch and store them in P_l

    $F_l = P_l \cup F_l$ and update the support of each entry in F_l

    prune unpromising itemsets from F_l if $|Fl| > n_{0,l}$

  endfor

endfor

Mathematische Analyse (I)

- Aufgrund der Verwendung der Chernoff-Schranke können Garantien für die Güte des Alorithmus bewiesen werden
- Hier aber nur die Ergebnisse ohne Beweis:
  - Die Wahrscheinlichkeit, dass der beobachtete Support um mehr als ε vom tatsächlichen abweicht ist höchstens δ
  - Wenn X zu den tatsächlichen top-K frequent itemsets gehört, dann wird X vom Algorithmus mit der Wahrscheinlichkeit von mind. 1-δ gefunden

Mathematische Analyse (II)

- Speicherbedarf (für itemset der Größe l)
  - $$O\left(\frac{2\left[C_l^{S_T}+4\ln\left(2/\delta\right)\right]}{s_k}\right)$$
  - sK ist der Support des K-th frequent itemsets
  - C_l^{sT} ist die Anzahl der Kombinationen von l Objekten aus einer Menge von sT Objekten
- Der Speicher ist also unabhängig von der Länge des Data Streams

Gliederung

1. Einleitung
2. Chernoff-basierter Algorithmus
3. **top-K lossy counting Algorithmus**
4. Empirischer Vergleich
5. Mining top-K itemsets in a sliding window
6. Zusammenfassung
7. Fragen / Diskussion

top-K lossy counting Algorithmus (I)

- basiert auf dem „Lossy counting Algorithm“
- Der Data-Stream wird in Batches bestehend aus R Transaktionen eingeteilt
- Jeder Batch wird in Buckets der Größe ω eingeteilt (mit $\omega = [1/\varepsilon]$)
- Itemsets werden in der Form (set, f, Δ)
- unpromising entry, wenn: $f + \Delta \leq [\frac{n}{\omega}]$

top-K lossy counting Algorithmus (II)

$n = 0$, $P_l = \emptyset$, $F_l = \emptyset$

for every Batch of R transactions **do**

  $n = n + R$

  for all l such that $1 <= l <= L$ **do**

    find all itemsets of size l with frequency count greater than or equal to β and store them in P_l

    $F_l = P_l \cup F_l$

    update the support of each entry stored in F_l

    remove all unpromising entries just updated in F_l

    $P_l = \emptyset$

  endfor

endfor

Gliederung

1. Einleitung
2. Chernoff-basierter Algorithmus
3. top-K lossy counting Algorithmus
4. **Empirischer Vergleich**
5. Mining top-K itemsets in a sliding window
6. Zusammenfassung
7. Fragen / Diskussion

Empirischer Vergleich

- Verwendetes System:
  Pentium IV 2,2 GHz, 1 GB RAM
- Vergleich der beiden vorgestellten Algorithmen mit:
  - BOMO algorithm (Alle Daten des Streams werden als ein Batch angesehen)
  - Zipf-Verteilung
  - Space-Saving Algorithm
- Tests wurden mit 2 synthetischen und einigen echten Datensets durchgeführt

Empirischer Vergleich (II)

- Verwendete Qualitätsmaße:
  - Recall ($\frac{|T \cap O|}{|T|}$)
    - Anteil der korrekt gefundenen frequent itemsets an allen wahren

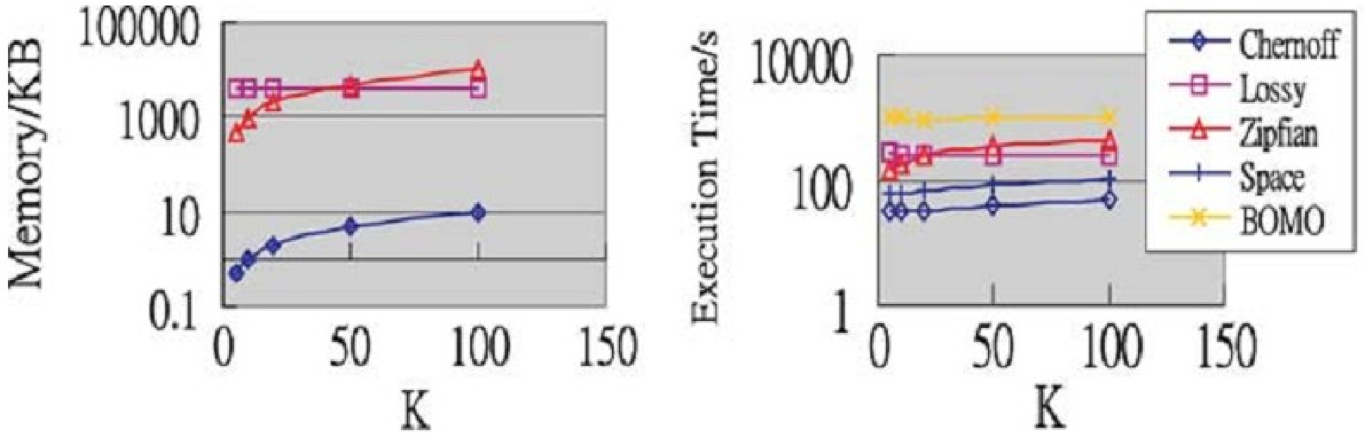
T = wahren frequent itemsets

O = vom Algorithmus bestimmten frequent itemsets

  - Precision ($\frac{|T \cap O|}{|O|}$)
    - Anteil der korrekten gefunden frequent itemsets an allen vom Algorithmus gefundenen

Empirischer Vergleich (III)

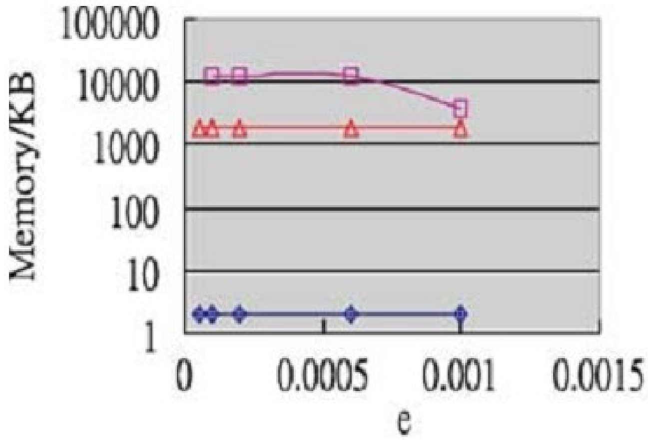
- Unterschiedliche K (Anzahl der ges. itemsets)

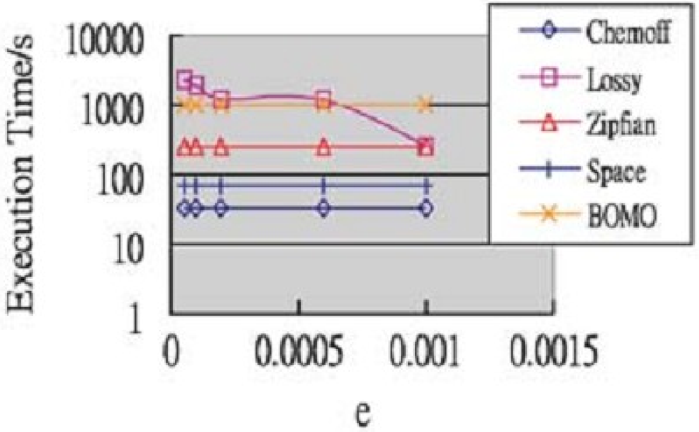


	Chernoff		Lossy		Zipfian		Space		BOMO	
K	R	P	R	P	R	P	R	P	R	P
5	0.97	0.97	1.00	1.00	1.00	1.00	0.60	1.00	1.00	1.00
10	0.95	0.95	1.00	1.00	1.00	1.00	0.75	1.00	1.00	1.00
20	0.98	0.98	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00
50	0.99	0.99	1.00	1.00	1.00	1.00	0.92	0.98	1.00	1.00
100	0.98	0.98	1.00	1.00	1.00	1.00	0.95	1.00	1.00	1.00

Fig. 1 Real Data Set 1: Varying *K*

- Unterschiedliche ε





	Chernoff		Lossy		Zipfian		Space		BOMO	
ϵ	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**
0.00100	0.98	0.98	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00
0.00060	0.98	0.98	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00
0.00020	0.98	0.98	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00
0.00010	0.98	0.98	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00
0.00005	0.98	0.98	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00

Fig. 4 Real Data Set 1: Varying ϵ

Empirischer Vergleich (V)

- Speicherbedarf des naiven (BOMO) Ansatzes war im Schnitt mindestens 11 mal so hoch wie bei allen anderen Algorithmen
  - spezielle Algorithmen für Data-Streams sind also wirklich notwendig
- Chernoff-Algorithmus hat, wie erwartet, kleinere Probleme mit abhängigen Daten
- top-K lossy Counting Algorithmus arbeitet sehr genau, braucht aber (deutlich) mehr Speicher als Chernoff
- Zipfverteilungsbasierter Algorithmus liefert ungenaue Ergebnisse, wenn sich die Datenreihenfolge ändert

Gliederung

1. Einleitung
2. Chernoff-basierter Algorithmus
3. top-K lossy counting Algorithmus
4. Empirischer Vergleich
5. **Mining top-K itemsets in a sliding window**
6. Zusammenfassung
7. Fragen / Diskussion

Gründe für sliding window Ansatz

- In vielen Anwendungen sind alte Daten nicht mehr relevant oder nicht mehr von Interesse
- Anpassung der Algorithmen, damit nur die letzten m Transaktionen betrachtet werden

Anpassung der Algorithmen

- Alle Batches die zum aktuellen window gehören müssen gespeichert werde:
  - „$Q_{l,0} \leftarrow Q_{l,1} \leftarrow Q_{l,2} \leftarrow Q_{l,3} \leftarrow \ldots \leftarrow Q_{l,n-1} \leftarrow Q_{l,n}$“ (bei jedem neuen Batch der bearbeitet wird)
- Veränderungen die im aktuellen Batch stattgefunden haben werden in $Q_{l,n}$ gespeichert
- Wenn ein Batch das window verlässt:
  - globale Pool F_l wird um $Q_{l,0}$ gesenkt

Chernoff Algo. mit sliding window

$n = 0, P_l = \emptyset, F_l = \emptyset$

for every Batch of R transactions **do**

$n = n + R$

for all l such that $1 <= l <= L$ **do**

switch the batch storages

find potential K-frequent itemsets in the current batch and store them in P_l

$F_l = P_l \cup F_l$ and update the support of each entry in F_l

prune unpromising itemsets from F_l if $|Fl| > n_{0,l}$

endfor

endfor

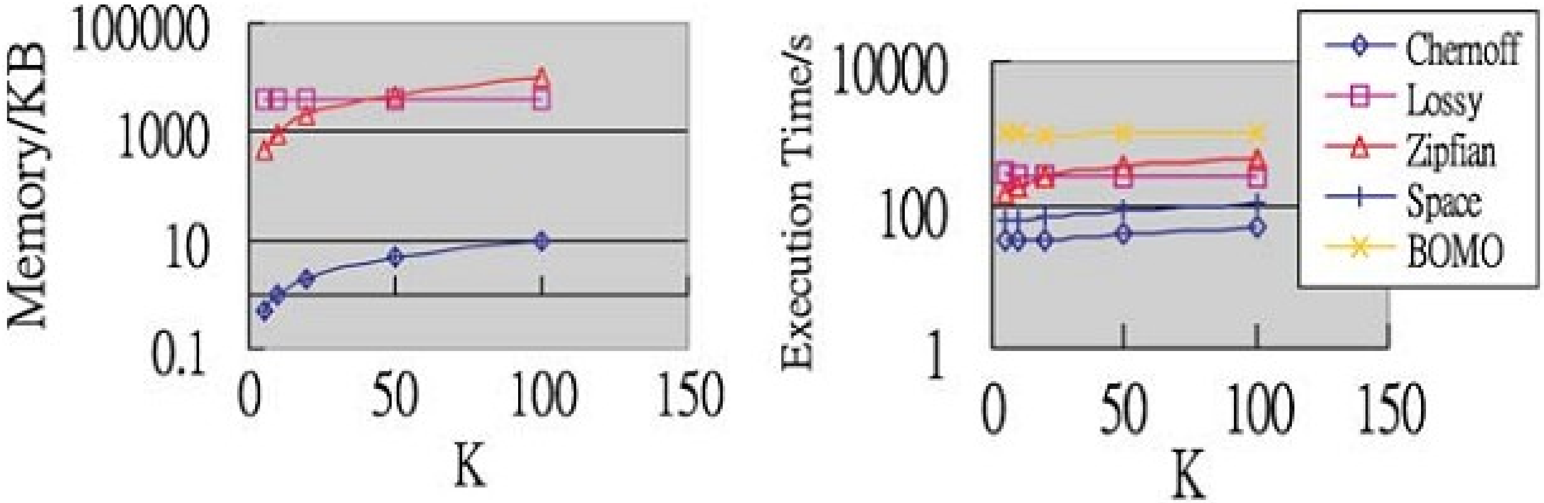
store all updated entries in $Q_{l,n}$

if there is a batch leaving the window

decrement F_l by $Q_{l,0}$

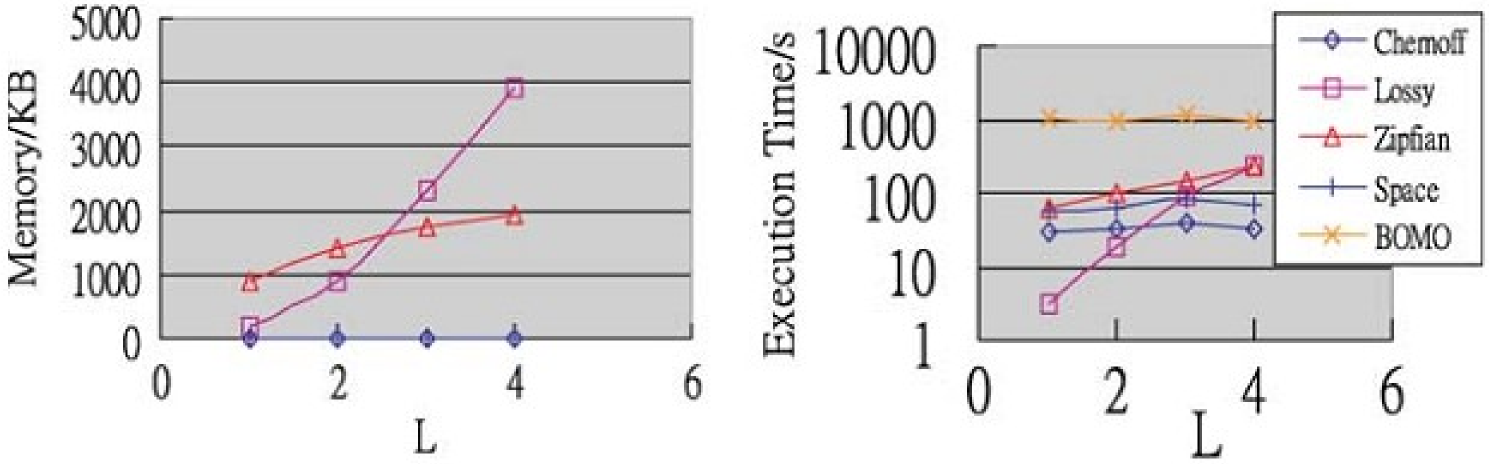
endif

Ergebnisse mit sliding window (I)



	Chernoff		Lossy		Zipfian		Space		BOMO	
K	R	P	R	P	R	P	R	P	R	P
5	1.00	1.00	1.00	1.00	1.00	1.00	0.60	1.00	1.00	1.00
10	1.00	1.00	1.00	1.00	1.00	1.00	0.75	1.00	1.00	1.00
20	1.00	1.00	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00
50	1.00	1.00	1.00	1.00	1.00	1.00	0.92	1.00	1.00	1.00
100	1.00	1.00	1.00	1.00	1.00	1.00	0.95	1.00	1.00	1.00

Ergebnisse mit sliding window (II)



	Chernoff		Lossy		Zipfian		Space		BOMO	
L	R	P	R	P	R	P	R	P	R	P
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
3	1.00	1.00	1.00	1.00	1.00	1.00	0.98	1.00	1.00	1.00
4	1.00	1.00	1.00	1.00	1.00	1.00	0.89	1.00	1.00	1.00

Gliederung

1. Einleitung
2. Chernoff-basierter Algorithmus
3. top-K lossy counting Algorithmus
4. Empirischer Vergleich
5. Mining top-K itemsets in a sliding window
6. **Zusammenfassung**
7. Fragen / Diskussion

Zusammenfassung

- Chernoff-bassierter Algorithmus
  - beweisbare Güte
  - benötigt nur begrenzt Speicherplatz unabhängig von der Stream Länge
  - aber: Schwächen bei abhängigen Daten
- Top-K lossy counting Algorithmus
  - benötigt $O(1/\varepsilon \log (\varepsilon n))$ Speicher
- Beide Algorithmen auch mit einem sliding window Ansatz kombinierbar

Zusammenfassung (II)

- **„Our experiments shows perfect solutions in almost all cases“**

Gliederung

1. Einleitung
2. Chernoff-basierter Algorithmus
3. top-K lossy counting Algorithmus
4. Empirischer Vergleich
5. Mining top-K itemsets in a sliding window
6. Zusammenfassung
7. **Fragen / Diskusion**

Fragen / Diskussion

- Fragen
- Diskussion

- Quellen:
  - „Mining top-K frequent itemsets from data streams“
    (Raymond Chi-Wing Wong, Ada Wai-Chee Fu)
  - „Mining top-K itemsets over a sliding window based on Zipfian Distribution“
    (Raymond Chi-Wing Wong, Ada Wai-Chee Fu)
  - „Approximate Frequency Counts over Streaming Data“
    (G. S. Manku, R. Motwani)