Vorlesung Wissensentdeckung

Datenströme

Katharina Morik, Claus Weihs

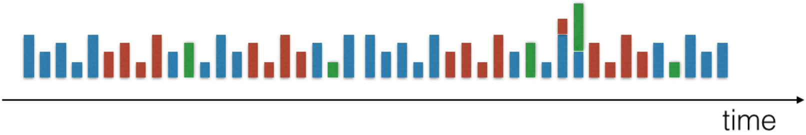
LS 8 Informatik
Computergestützte Statistik
Technische Universität Dortmund

09.07.2015

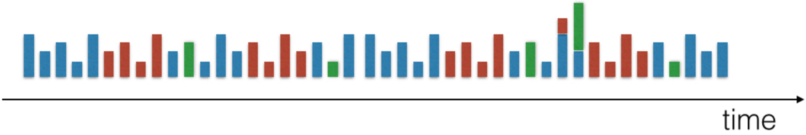
Warum Datenströme

- Die Menge an gesammelten Daten nimmt immer weiter zu.
- Die Betrachtung von Ressourcen, insbesondere der Zeit, wird immer wichtiger.
- Optimale Losgröße = 1? $\Rightarrow$ Jedes Event, jeder User, jedes Produkt, jeder Prozess, jeder Service soll lokal (räumlich und zeitlich) optimal behandelt werden.

Datenströme



Datenströme



Erweitertes Datenmodell

- Die Erhebung (Sampling) von Daten wird explizit modelliert.
- Das nächste Element des Datenstroms ist nur nach erneutem Sampling verfügbar.

Erweitertes Zustandsmodell

- Verteilter + lokaler Zustandsraum der Verarbeitung
- Der Zustandsraum der Ressourcen (CPU + Speicher (+ Energie)) wird um die Kommunikation erweitert

Beispiele

Finanzielle- und Logistiktransaktionen

Beispiele

Sensordatenströme

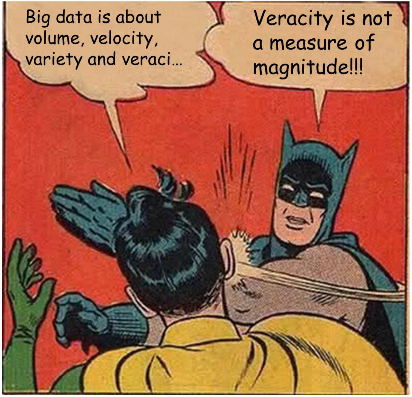
- Punktsensoren (Pyrometer...)
- Flächensensor (Kamera...)
- Durchflusssensoren
- ...

Beispiele

- BufferedInput/OutputStream (Java)
  - Line bzw. Token (String)
  - Byte
- Protokolle
  - File (FileSystem)
  - TCP/IP (Sockets)
  - HTTP(/2) (SPDY `https://www.chromium.org/spdy/spdy-whitepaper`
  - Protocol Buffers
    `https://github.com/google/protobuf`
  - JavaScript Object Notation (JSON)
  - ...

Anwendungsfälle

- Zwei Anwendungsfälle:
  - Die Daten sind zu groß um alle gleichzeitig in-Memory verarbeitet zu werden (Speicherbeschränkung)
  - Nach dem Eintreffen neuer Daten müssen diese möglichst schnell verarbeitet werden (Realzeitliche Verarbeitung)

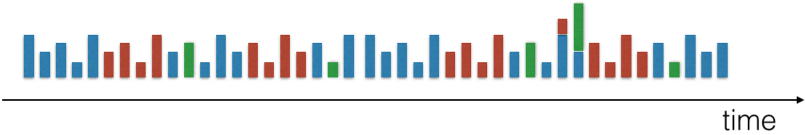


Aufgaben

- Aggregation (Erzeugen von Meta-Daten):
  - Merkmalsextraktion
  - Datenbereinigung
  - Ranking
  - Zählen
  - Concept Drift Detection
  - Reporting (BusinessIntelligence)
  - ...
- Modellierung und Handlung:
  - Ableiten von Schwellwerten oder Regeln aus Meta-Daten und daraus folgenden Handlungen (Benachrichtigung, Steuerbefehl, Warnung,...)
  - Modelllernen, Vorhersage und daraus folgende Handlungen (Benachrichtigung, Steuerbefehl, Warnung,...)
  - ...

Definitionen

- An eine Datenquelle (Sensor, Datenbank, Disk,...) $\mathcal{S}$ können parametrisierte Anfragen $\sigma_p(\mathcal{S}) = d, d \in \mathbb{D}^p$ gestellt werden mit $\mathbb{D} \in \{\mathbb{S}^n, \mathbb{B}^n, \mathbb{R}^n, \mathcal{D}, \ldots\}$.
- Ein Datenstrom $\mathcal{D} = (\sigma_{p,i}(\mathcal{S}) = d_i)_{i \in \mathbb{N}}$ ist das Ergebnis einer endlichen oder abzählbar unendlichen Folge von Anfragen an eine Datenquelle $\mathcal{S}$.
- Die Anfragen werden durch einen Trigger $\mathcal{T}$ ausgelöst:
  - Event-basiert
  - Synchron (Alle x ms)
  - So schnell wie möglich



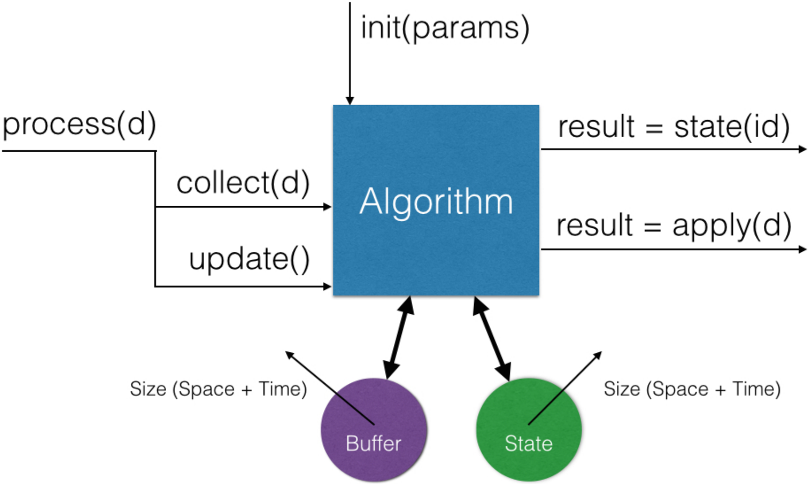
Problem

- Mit dieser Definition ist keine Unterscheidung zu Zeitreihen möglich!
- Zeitreihe $\hat{=}$ Abgespeicherte Rohdaten einer endlichen Teilfolge ($\subseteq$) des Datenstroms
- $\Rightarrow$ Betrachtung der Algorithmen notwendig!

Obacht!

Die Definition könnte leicht um die Einschränkung der Abfragen erweitert werden, hier soll aber auf die Unterschiede in der Verarbeitung eingegangen werden.

Algorithmus



Algorithmus

- Ein Algorithmus a ist eine eindeutige Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen.
- Eine Algorithmeninstanz hat folgende Funktionen und Eigenschaften:
  - a.state: der Zustand des Algorithmus
  - a.state.size$_s$: Die Größe des Zustands (Bytes)
  - a.state = a.init($\mathbb{D}^m$ parameter): Initialer Zustand
  - d= a.process($\mathbb{D}^p$ rawdata): Verarbeiten der Daten
    - a.buffer: Zwischenspeicher für Rohdaten (Random-Access möglich)
    - a.buffer.size$_s$: Die Größe des Zwischenspeichers (Bytes)
    - a.buffer = a.collect($\mathbb{D}^p$ rawdata): Übergeben von Rohdaten an den Algorithmus
    - und/oder: a.state = a.update(a.state, a.buffer): Aktualisieren des Zustands mit den gesammelten Daten
  - a.apply($\mathbb{D}^p$ data): Anwenden des Zustands/Ergebnisses auf andere Daten

Online Algorithmus

- Die Erhebung von Daten wird explizit modelliert
  - a.trigger($\mathcal{T}$ trigger, process($\sigma_p(\mathcal{S})$)) (Parameter aus a.state)
  - Steigt a.state.size$_s$ im Durchschnitt über die Zeit?
  - Wie lange dauert die Aktualisierung des Zustandes?
  - Wie kann mit Veränderungen der Verteilung der Datenquelle über die Zeit umgegangen werden?
- realzeitliche Betrachtung:
  - a.state.size$_t$: Wie groß ist das durch den Zustand abgebildete Zeitintervall?
  - a.buffer.size$_t$: Wie groß ist das durch den Buffer abgebildete Zeitintervall?
  - Wie lange muss der Datenstrom beobachtet werden, um sinnvolle Ergebnisse zu erhalten?
  - Mit welchem Delay wird der Zustand des Algorithmus' aktualisiert?

Zählen kann schwierig sein ...

- Eingabe: ein Strom $(d_i)_{i \in N}$ von Tweets
- Ausgabe: 10 häufigste #-tags
- Naiver Ansatz:
  - Richte für jeden #-tag einen Zähler (4 Byte) ein.
  - Grosser Speicherbedarf!
- Approximationsalgorithmus
  - Liefert ein Ergebnis und den möglichen Fehler.
  - Fenstergrösse und Speicherbedarf vs. Genauigkeit!

- Lossy Counting
  - Man teilt den Strom $S = s_1, s_2, \ldots$ in Fenster von w Elementen und zählt das Vorkommen von Beobachtungen e_i. Die Häufigkeit $D(e)$ wird angegeben als f, Δ.
  - Nach einem Fenster wirft man alle Zählungen weg, die nicht häufig genug sind, übernimmt nur die anderen.
  - Der Parameter Δ zählt mit, wie viel verlorengegangen sein kann.

LossyCounting

```
function INIT(ℕ windowSize)
    state.w = windowSize
function PROCESS(d_i)
    if state.count < state.w then
        state.count++
        collect(d_i)
    else
        state.count = 0
        state.window++
        update()
```

```
function UPDATE
    for e ∈ buffer do
        if state.contains(e) then
            (f, Δ) =state.e
            state.e = ( f++, Δ)
        else
            Δ = state.window − 1
            state.e = (1, Δ)
    buffer =[]
    prune()
function PRUNE
    for e ∈ state do
        (f, Δ) = state.e
        if f + Δ ≤ state.window then
            state.remove(e)
```

Beispiel Lossy Counting

- Strom: ABBC ACBD ADBB AADC ABAA BDDC
- Zustand: (e, f, Δ)
- Real: (A,8,0), (B,7,0), (C,4,0), (D,5,0)
- Lossy: (A,5,4), (B,7,0), (C,1,5),(D,2,5)

#window	S	D	Prune(D)
1	ABBC	$(A,1,0),(B,2,0),(C,1,0)$	$(B,2,0)$
2	ACBD	$(A,1,1),(C,1,1),(B,3,0),(D,1,1)$	$(B,3,0)$
3	ADBB	$(A,1,2),(D,1,2),(B,5,0)$	(B,5,0)
4	AADC	$(B,5,0),(A,2,4),(D,1,4),(C,1,4)$	$(B,5,0),(A,2,4),(D,1,4),(C,1,4)$
5	ABAA	$(B,6,0),(A,5,4),(D,1,4),(C,1,4)$	$(B,6,0),(A,5,4)$
6	BDDC	$(B,7,0),(A,5,4),(D,2,5),(C,1,5)$	$(B,7,0),(A,5,4),(D,2,5)$

Vorhersage mit Naive Bayes

- Wahrscheinlichkeit für der Kunde kauft (A), der Kunde kauft nicht ($\bar{A}$) bei Beobachtungen x
- Mit dem Satz von Bayes bestimmen wir die bedingte Wahrscheinlichkeit.
- Wir schreiben das um als Zählen:
  - Wie oft kommt x vor?
  - Wie oft kommt A vor?
  - Wir oft kommt $\bar{A}$ vor?

- Naive Bayes

$$g(x) = y \quad y \in \{A, \bar{A}\}$$

$$P(A \mid x) = \frac{P(x \mid A)P(A)}{P(x)}$$

- zählen:

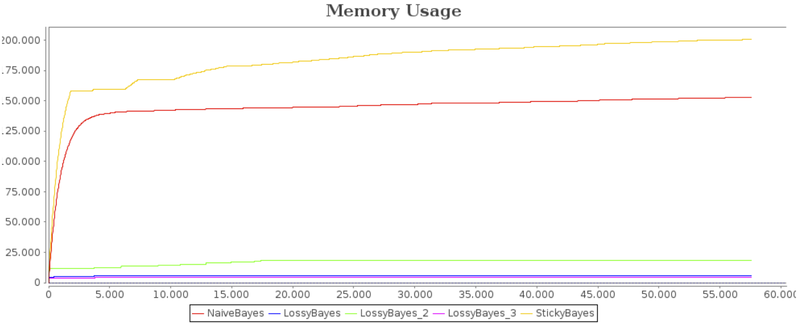
$$Q = \frac{(|x| : |A|)(|A|)}{(|x| : |\bar{A}|)(|\bar{A}|)}$$

$$Q \geq 1 \rightarrow g(x) = A$$

- Wir verwenden lossy counting und untersuchen Speicherbedarf und Genauigkeit.

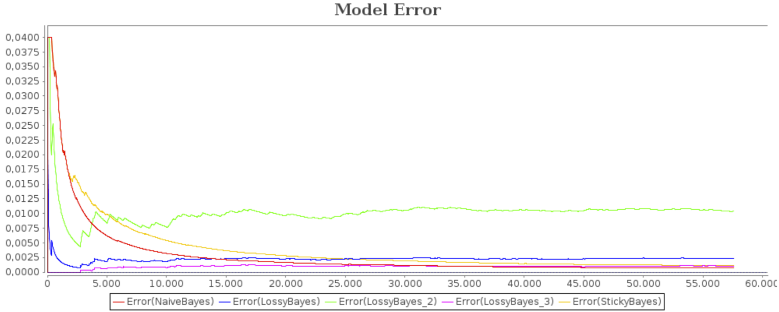
Zählen im Datenstrom

- Datenstrom bis Element 58.000
- 5 Zähler für 5 Attribute mit je 1000 Werten
- Speicherverbrauch bei korrekter Zählung 150.000, bei lossy counting (bei allen Varianten) unter 25.000.



Speicherplatz vs. Genauigkeit des Ergebnisses

- Ab 15.000 Beobachtungen macht Lossy Bayes einen größeren Fehler als Naive Bayes.
- Die optimierte Variante Lossy Bayes 3 ist etwas besser als Naive Bayes.



Vorhersage mit Naive Bayes

- Wahrscheinlichkeit für der Kunde kauft (A), der Kunde kauft nicht ($\bar{A}$) bei Beobachtungen x
- Mit dem Satz von Bayes bestimmen wir die bedingte Wahrscheinlichkeit.
- Wir schreiben das um als Zählen:
  - Wie oft kommt x vor?
  - Wie oft kommt A vor?
  - Wir oft kommt $\bar{A}$ vor?

- Naive Bayes

$$g(x) = y \quad y \in \{A, \bar{A}\}$$

$$P(A \mid x) = \frac{P(x \mid A)P(A)}{P(x)}$$

- zählen:

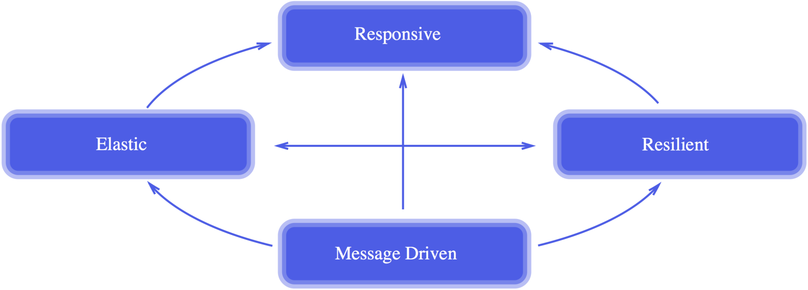
$$Q = \frac{(|x| : |A|)(|A|)}{(|x| : |\bar{A}|)(|\bar{A}|)}$$

$$Q \geq 1 \rightarrow g(x) = A$$

- Wir verwenden lossy counting und untersuchen Speicherbedarf und Genauigkeit.

http://www.reactivemanifesto.org/

Reactive Manifesto

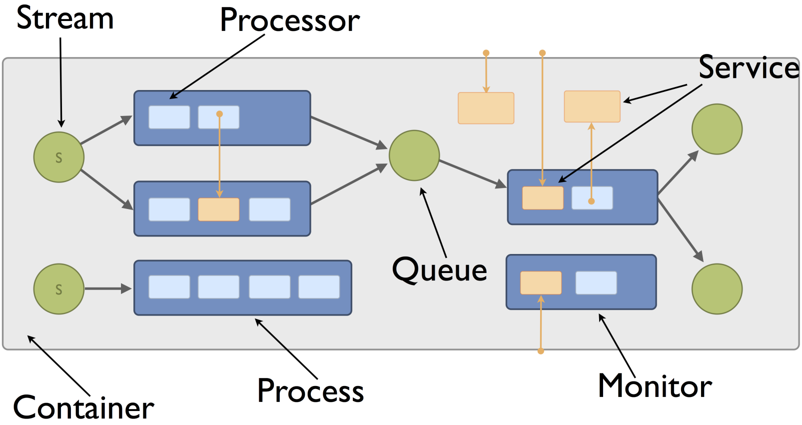


`http://www.reactivemanifesto.org/`

Datenstrom Verarbeitung

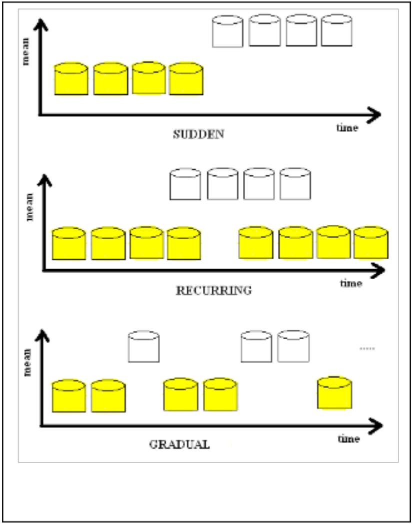
- SQL on Streams
  - StreamSQL
  - PipelineDB
  - ...
- Execution Graphs
  - Apache Spark
  - Apache Flink
  - Apache Storm
  - Apache Samza
  - Apache Samoa
  - Akka Streams
  - Streams Framework
  - ...

Streams Framework



http://www-ai.cs.uni-dortmund.de/SOFTWARE/streams/

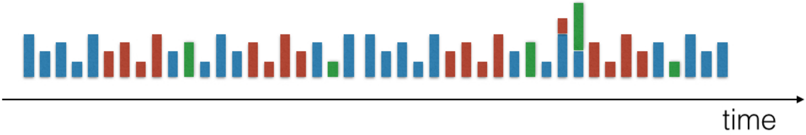
ConceptDrift



ConceptDrift

- Beispiele
  - Defekte Sensoren
  - Saisonale Änderungen
  - Veränderung des Maschinenzustands
  - ...
- Behandlung
  - Naiv: Regelmäßige Aktualisierung des Zustands (Modelle).
  - Fenster/Buffer: Überwachen des Buffers auf Veränderung der Verteilung der Daten.
  - DriftDetection: Ein Algorithmus entscheidet, wann der Zustand aktualisiert werden soll und welche Buffergrö verwendet werden soll.
  - Ensemble: Es wird nicht nur ein Algorithmus verwendet, sondern mehrere. Eine Algorithmus entscheidet über die Gewichtung.

Datenströme



Erweitertes Datenmodell

- Die Erhebung (Sampling) von Daten wird explizit modelliert.
- Das nächste Element des Datenstroms ist nur nach erneutem Sampling verfügbar.

Erweitertes Zustandsmodell

- Verteilter + lokaler Zustandsraum der Verarbeitung
- Der Zustandsraum der Ressourcen (CPU + Speicher (+ Energie)) wird um die Kommunikation erweitert