

Skript zur Vorlesung Knowledge Discovery in Databases II im Wintersemester 2013/2014

Kapitel 8: Parallel, Distributed und Privacy Preserving Data Mining

Skript KDD II © 2012 Matthias Schubert

[http://www.dbs.ifi.lmu.de/cms/Knowledge_Discovery_in_Databases_II_\(KDD_II\)](http://www.dbs.ifi.lmu.de/cms/Knowledge_Discovery_in_Databases_II_(KDD_II))

Bisher: Alle Daten sind in einer Datenbank gespeichert und der Data Mining Algorithmus darf auf alle Objekte beliebig häufig Daten zugreifen.

Aber:

- bei sehr großen Datenmengen ist Verarbeitung zeitaufwendig
- Bei verteilten Datenquellen:
Integration der Daten oft mit hohen Transferkosten verbunden
- Bestimmte Datenobjekte/Ergebnisse können Personen diskreditieren
 - Globaler Data Mining Algorithmus darf nicht einzelne Objekte zugreifen
 - Ergebnisse dürfen dann nicht erlaubt werden, wenn sie Rückschlüsse auf Objektwerte erlauben

Ziele:

- mehrere Rechner für Verbesserung der Effizienz nutzen
⇒ Parallelisierung von Data Mining Algorithmen
- Bei verteilter Datenhaltung:
 - Minimale Transferkosten
 - Effiziente Verarbeitung
 - Abweichung vom Ergebnis eines zentralen Data Mining Algorithmus soll minimal sein.
- bei vertraulichen Daten:
 - Informationen über einzelne Datenobjekte dürfen nicht weiter gegeben werden
 - auch Diskreditierung durch abgeleitete Informationen muss vermieden werden

- Parallelisierung von DBSCAN zur Beschleunigung großer Datenmengen
- Endkundengruppierung bei Großhändlern:
 - Einzelhändler geben einzelne Daten nicht raus.
 - Großhändler brauchen verteilte und „*privacy-preserving*“ Verfahren, um dennoch Kundengruppen zu bestimmen.
- Bestimmung von nicht-kompromitierenden Assoziationsregeln auf Webanfragen

Bsp:

Fridoline M. und Informatik Uni München

=> *Schwangerschaftsabbruch*

Kompromitierung über sogenannte „Vanity-Searches“.

=> Unterdrücken zu spezieller und kompromitierender Regeln

1. Einführung

paralleles, verteiltes Mining, Privacy Preservation

2. Parallele und Verteilte Algorithmen

Unterschied: parallele und verteilte Algorithmen, Aufteilungsunabhängigkeit, verteiltes Clustering.

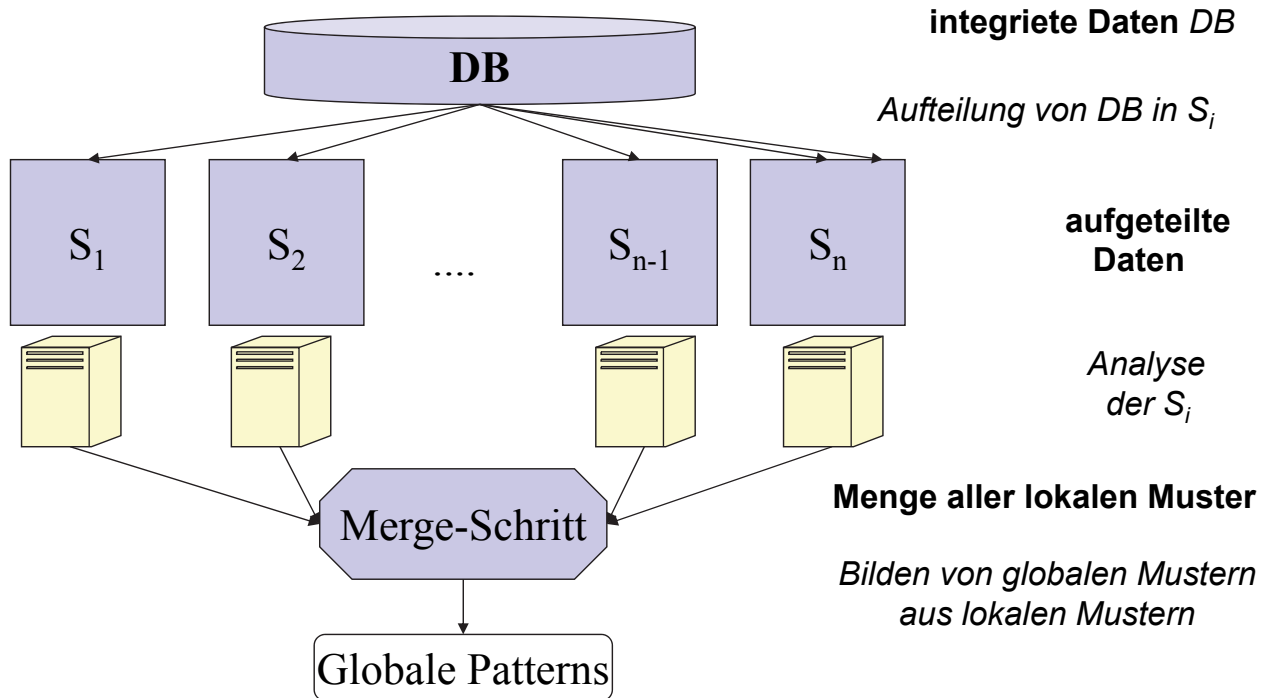
3. Privacy Preservation

Peturbation, Aggregation, Swapping,

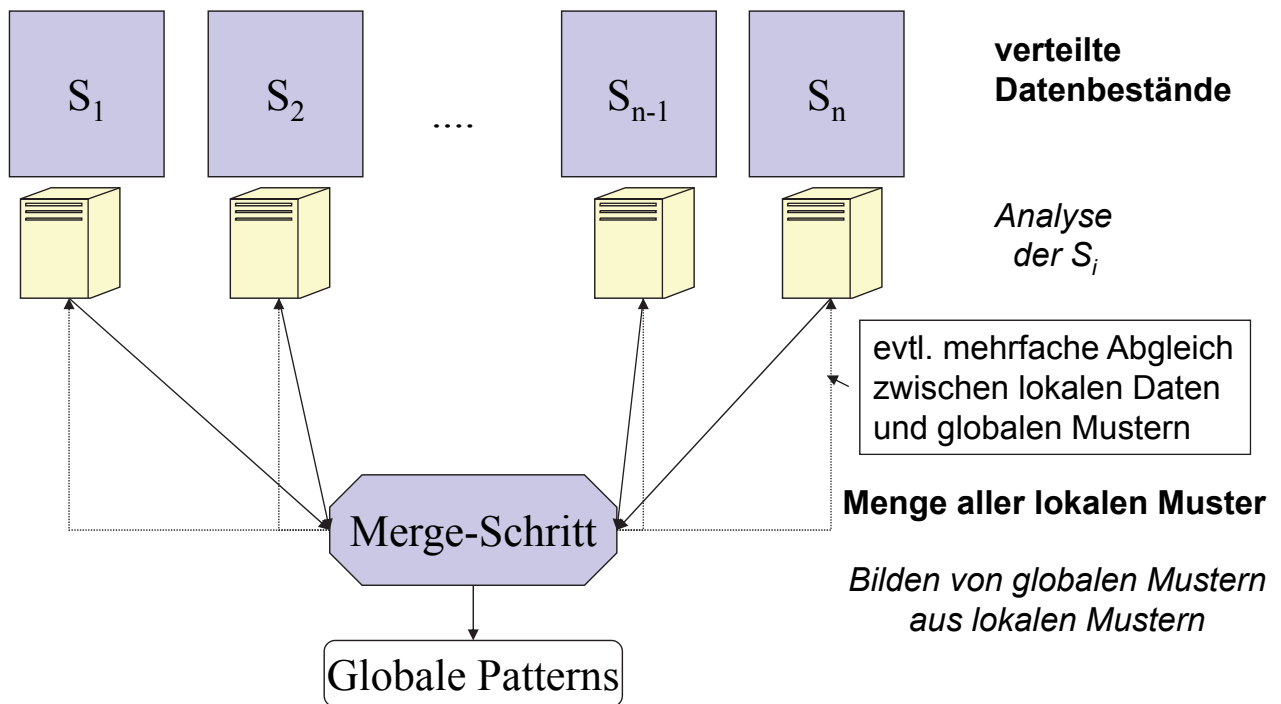
- Paralleles Data Mining:
- Datenbank liegt bereits integriert vor, soll aber auf k Rechnern parallel analysiert werden.
- Performanzgewinn durch „*Divide and Conquer*“ Strategie:
 - ⇒ Teile Datenmenge auf
 - ⇒ Analysiere die Teilmengen auf getrennten Rechnern nebenläufig.
 - ⇒ Fasse lokale Ergebnisse zusammen zu globalen Mustern zusammen.

Entscheidend hierbei:

- Wie teile ich eine Datenmenge so auf, dass lokale Muster mit möglichst wenig Aufwand in ein integriertes globales Model integriert werden können?
- Wie kann ich die teure Kommunikation zwischen den Einzelrechnern so niedrig wie möglich halten.



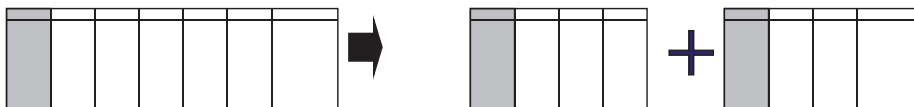
- Verteilung der Daten auf verschiedene Rechner ist vorgegeben
 - ⇒ Kein zusätzlicher Aufteilungsaufwand.
 - ⇒ Zusammenfügen lokaler Muster kann teuer oder unpräzise werden.
- ungünstige Verteilung kann folgende Auswirkungen haben:
 - Abweichung des verteilt errechneten Modells vom zentral errechneten Modell
 - hohe Kommunikations- und Berechnungszeiten zwischen den einzelnen Rechnern.
- **Fazit:** Parallele Data Mining Algorithmen können als Spezialfall von verteilten Algorithmen aufgefasst werden, bei denen man die Aufteilung der Daten auf verschiedene Sites S_i selber bestimmen darf, aber dafür Kosten für die Verteilung der Daten anfallen können.



1. Nach Aufteilung:

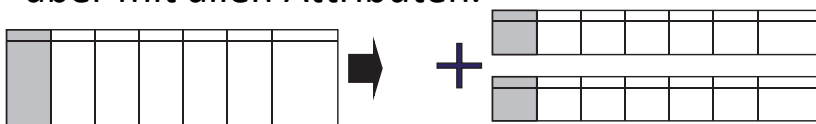
- *Vertikal verteilte Daten*

Alle Datenobjekte auf allen Rechnern, aber nur jeweils ein Teil der Attribute auf jeden Rechner



- *Horizontal verteilte Daten*

Datenobjekte kommen i.d.R. nur auf einem Rechner vor, dafür aber mit allen Attributen.



(Daten können auch vertikal und horizontalverteilt sein.)

2. **Nach Aufgabe:** Klassifikation, Clustering, Assoziationsregeln
3. **Verteilungsabhängigkeit:** Ist das Ergebnis, von der Verteilung auf die Sites abhängig. (Ergebnis verteiltes Verfahren = Ergebnis globales Verfahrens ?)
4. **Art der Teilergebnisse:** Approximationen, Datenpunkte, Verteilungen...
Beispiele: Gauss-Kurven, Hyperrechtecke, Centroide...
5. **Nach Organisation der verteilten Berechnung:**
1 Masterprozeß und viele Slaves, mehrere gleichberechtigte Prozeße

- Verteilungsunabhängigkeit wird meist gefordert
(Egal wie aufgeteilt wird, das Ergebnis bleibt gleich)
- Effizienzsteigerung steht im Vordergrund
- Aufteilung der Daten auf Sites ist entscheidend:
 - Probleme bei der Kombination der Teilergebnisse sollen minimiert werden
 - ⇒ lokale Muster sollten unabhängig voneinander sein
 - ⇒ bei Abhängigkeiten: mehrfache Kommunikation oder inexakte Muster
 - Gesamtlaufzeit ist von der längsten Laufzeit eines Teilschritts abhängig
 - ⇒ alle parallelen Schritte sollten möglichst gleich viel Zeit in Anspruch nehmen
 - ⇒ alle Sites sollten die gleiche Menge an Daten zugeteilt bekommen

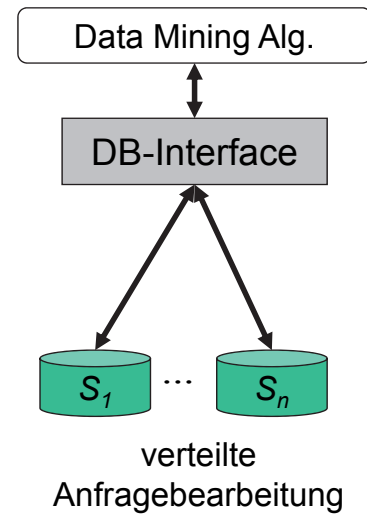
- Prinzipiell lassen sich Data Mining Algorithmen durch
- parallele Basisoperationen parallelisieren und
- beschleunigen:

Beispiele:

- Parallele Bearbeitung von ε -Range Queries erlaubt Performanzsteigerung für dichte-basiertes Clustering.
- Paralleles Bestimmen des nächsten Nachbarn, beschleunigt k NN-Klassifikation

Aber:

- auch hier Aufteilung entscheidend
- Korrektheit der Anfrage wird vorausgesetzt



Grundidee:

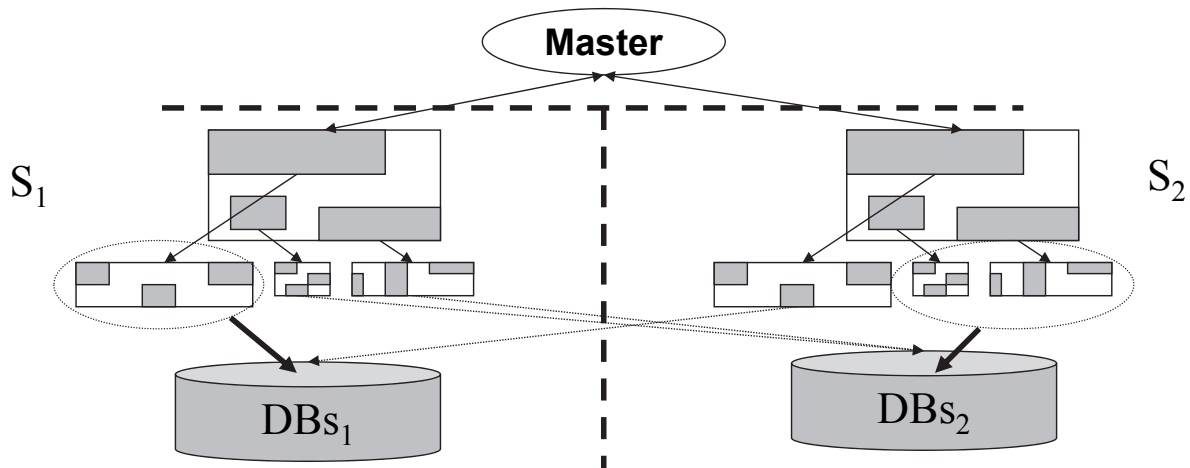
- Horizontale und kompakte Aufteilung des Datenraums
- Bestimmung lokaler Kernpunkte und Cluster
- Verbinden von lokalen Clustern mit:
 - Clustern aus anderen Sites
 - Rauschpunkten aus anderen Sites
- Grundproblem:
 - Was passiert mit Objekten deren ε -Umgebung aus dem Bereich der Site hinaus stehen ?
- Spiegeln der Ränder auf alle Sites
- Kommunikation mit der Datenbank einer anderen Site

PDBSCAN [Xu, Jäger, Kriegel99]

Grundidee:

- Abspeichern aller Featurevektoren in einer parallelisierten Indexstruktur (dR*-Tree)
 - Directory der R-Baums auf jeder Site
 - Datenseiten sind gemäß räumlicher Nähe auf Site verteilt
- Bestimmen lokaler Cluster auf den Sites
- Bei Punkten am Rand benötigt Anfragebearbeitung Zugriff auf die Blätter einer anderen Site.
- Mergen der Cluster mit gemeinsamen Punkten

14



Der dR*-Baum:

- erlaubt Range-Queries auf der gesamten Datenbank
- Anfragen auf S_i , die auf lokaler Partition DBs_i beantwortet werden kann lassen sich vollständig nebenläufig bearbeiten
- bei Zugriff auf Blätter einer anderen Seite sinkt die Nebenläufigkeit und die Kommunikationskosten steigen.

=> Algorithmus soll soweit wie möglich lokale Anfragen benutzen

15

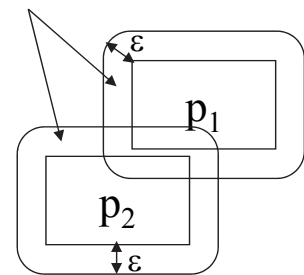
Aufteilung der Daten gemäß der Seiten im dR^* -Baum
(gleich Anzahl an Datenseiten auf jede Site)

Idee:

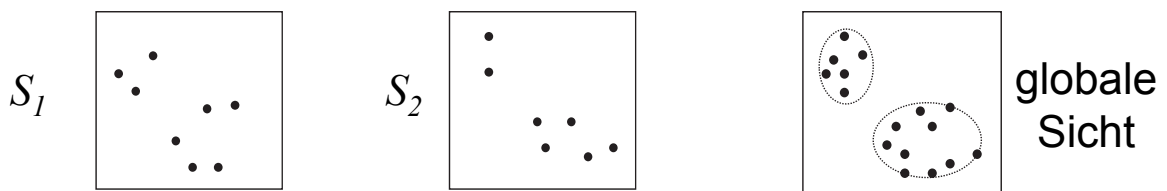
- DBSCAN läuft auf allen Punkten in p_1 und p_2
- Falls sich ε -Umgebung und Randbereich schneiden:
 1. Randbereiche müssen evtl. von anderer Site geladen werden um Kernpunkteigenschaft zu bestimmen
 2. Expansion der Punkte in den Randbereichen würde zu hohen Kommunikationskosten führen.
=> Punkte werden in Mergeliste gespeichert
- Zusammenführen der lokalen Cluster über gemeinsame Mergepunkte:
Mergepunkt muss Kernpunkt in einer Partition sein
=> Verschmelze Cluster

lokale pages p_1 und p_2

Randbereich



- Bei beliebiger Verteilung der Daten auf lokale Sites S_i :
- Ausdehnung der Daten auf jeder Site S_i kann maximal überlappen
 - jede Site S_i kann Elemente der ε -Umgebung des Punkts P enthalten
 - P kann Kernpunkt sein, auch wenn P kein Kernpunkt auf einer beliebigen Teilmenge $S' \subset S$ (=Menge aller Sites) ist.



- Dichtebasiertes Clustering verfügt über kein Clustermodell
=> Transfer lokaler Punkte, um ein globales Muster zu erzeugen

Idee:

- wenn die Menge der zu übertragenden Daten gering ist, mehrfacher Abgleich zwischen lokalen und globalen Modellen kein Problem
- Bei k -Means oder verwandten Verfahren lässt sich Zielfunktion und Centroid auch verteilt berechnen.

$$TD^2 = \sum_{o \in DB} \left(\sum_{C_i \in C} (\min\{d(o, C_i)\})^2 \right) = \sum_{S_j \in DB} \left(\sum_{o \in S_j} \left(\sum_{C_i \in C} (\min\{d(o, C_i)\})^2 \right) \right)$$

- globaler Centroid c_j zusammengesetzt aus den lokal zugeordneten Punkte $C_{i,j}$:

$$c_j = \frac{1}{\sum_{C_{i,j} \in DB} |C_{i,j}|} \cdot \sum_{C_{i,j} \in DB} \sum_{o \in C_{i,j}} o$$

- **Fazit:** In jedem Schritt kann das globale Clustering mit wenig Kommunikations-Overhead optimiert werden.

Verteiltes Clustering durch Varianzminimierung (Master-Slave):

Bestimme initiale Aufteilung und Anfangs-Centroide
Schleife:

Übermittle Centroide an die Sites

Ordne lokale Punkte den Centroiden zu

=> berechne lokale Centroide und lokale TD^2 -Werte

Nach Rückübertragung von lokalen Centroiden, Cluster-Kardinalitäten und TD^2 -Werten

⇒ Kombination lokaler Centroide zu globalen durch Aufaddieren der Centroide und Kardinalitäten. (dynamische Berechnung des Centroids)

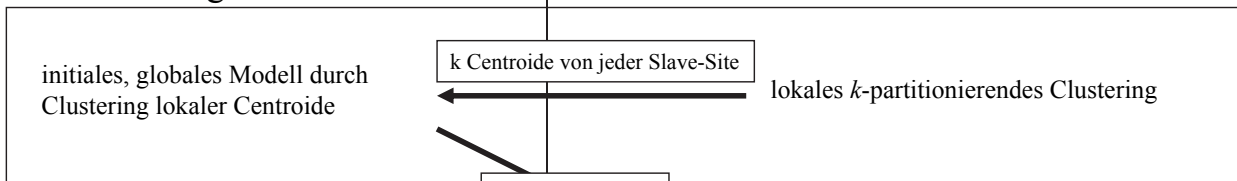
⇒ Bestimmung des globalen TD^2 -Werts

Falls TD^2 -Werte nicht verbessert wurde => Abbruch

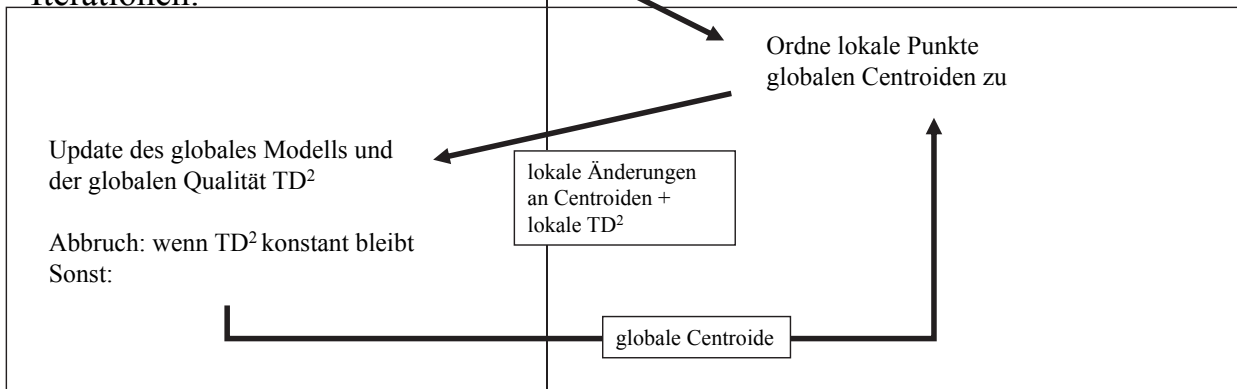
Master-Site

Slave-Sites

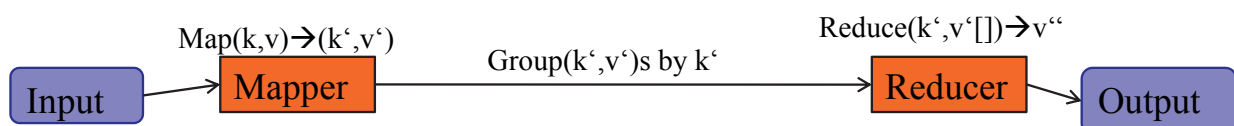
Initialisierung:



Iterationen:



20



- basiert auf den funktionalen Konstrukten: Map und Reduce
- Ursprünglich von Google entwickelt um Daten mit wenig Implementierungsaufwand und hoch parallelisiert zu verarbeiten (Berechnung des PageRank)
- Open Source Version: Hadoop (z.B. bei Facebook im Einsatz)
- Daten werden als <key, value> Paare beschrieben
Beispiel: <DokID3, „Heute ist ein schöner Tag..“>
- Sowohl Mapper als auch Reducer können parallel auf unterschiedlichen Rechnern berechnet werden (Worker)
- Optimierung erfolgt zum Teil automatisch über dynamische Zuschaltung von Worker-Tasks
- Trade-Off: Parallelität und Datentransfer über das verbindende Netzwerk

21

Verarbeitung findet in 3 Schritten statt die parallelisiert verarbeitet werden:

1. Map-Step: Input: <key1, value1> Output:<key2,value2>

Jedes Input-Paar wird dabei in 0 bis n Outputpaare umgewandelt.

Beispiel: <ID, Text> => <Heute, 1>,<ist,1>, <ein,1>....

2. Shuffle-Step: Map-Output wird sortiert und nach Key zusammengefasst.

Beispiel: <Heute, <1,1,1,1,1,1,1,1,1,1,1>>

3. Reduce-Step: Verarbeitung der Liste für jedes Paar aus Schritt 2.

Beispiel: <Heute,12>

Für kompliziertere Programme können mehrere MapReduce Iterationen durchgeführt werden oder unterschiedliche MapReduce-Programme hintereinander geschaltet werden.

- **Partitioner:** Regelt die Aufteilung der Inputdaten auf die Mapper
Default: HashPartitioner
- **Combiner:** Aggregationsschritt, der **lokal** nach einem Mapper ausgeführt wird.
Verarbeitung zwischen Map-Step und Shuffle-Step.
=> Transfervolumen vom Rechner des Mappers kann reduziert werden
Beispiel: <Heute, <1,1,1,1,>> -> <Heute,4>

Gegeben: Eine Datemenge D aus n Featurevektoren, k Anzahl der suchten Cluster.

Gesucht: k Centroide die TD^2 minimieren: $TD^2(D, C) = \sum_{x \in D} \left(\min_{c \in C} (dist(c, x)) \right)^2$

Schritte:

- Zuordnung der Vektoren zum nächsten Cluster
- Berechnung der Centroide für eine Menge von Vektoren
- Berechnung der TD^2

⇒ alle Schritte sind linear mit einem Scan über D durchführbar

⇒ Teilergebnisse sind additiv. Cluster und TD^2 sind Summen und damit in beliebigen Teilmengen berechebar. (**Assoziativgesetz**)

Mastermethode:

Sample k initiale Centroide C .

WHILE $TD^2 < oldValue$

$oldValue = TD^2$

 Bilde Zuordnung Objekte D zu Centroiden C (Mapper)

 Bilde neues C und TD^2 (Reducer)

RETURN C

Bemerkung:

- nur die teuren Schritte werden parallel bearbeitet
- pro Iteration ein MapReduce Aufruf
- C muss sowohl dem Mapper als auch dem Reducer bekannt sein

Input: D:Vektoremenge,C: Menge der Centroide , $k=|C|$: Anzahl der Centroide

Output: <centroid_id, vector>

```
FOR EACH v in D DO
    bestCluster = null; minDist = ∞
    FOR EACH C_i in C DO
        IF minDist > dist(C_i, v) THEN
            minDist = dist(C_i, v)
            bestCluster = C_i
        ENDIF
    END FOR
    OUTPUT<bestCluster, v>
END FOR
```

Anschließend erzeugt Shuffle-Schritt <Cluster,<v1,..,vl>> Paare

26

Input: <Cluster,<v1,..,vl>>

Output: <newC, TD2> neue Centroide und Parts of TD2

```
FOR EACH <C, <v1,..,vl>> DO
    linearSum = 0;
    count = 0;
    TD2 = 0;
    FOR EACH v_j in <v1,..,vl> DO
        linearSum += v_j
        count = count+1
        TD2 +=dist(vj,C)
    END FOR
    newC = linearSum/count
    OUTPUT< newC, TD2>
END FOR
```

Anschließend erzeugt Shuffle-Schritt <Cluster,<v1,..,vl>> Paare

27

- Anzahl der Aufrufe entspricht Anzahl der Iterationen
- Optimierung durch lokale Combiner, die Teile der linearen Summe bereits auf einem Rechner aggregieren.
- Algorithmus klärt noch nicht die Frage eines guten initiale Clusterings
- neue Methoden versuchen hier durch Sampling-Techniken noch größere Datenmengen zu verarbeiten

Zusammenhang zum verteilten Data Mining:

⇒ Schutz nur bei mehreren Parteien notwendig:

Data-Owner: Hat Einblick in (einen Teil) der exakten Daten

Data-User: Möchte Muster aus den Daten ableiten.

Privacy Preserving Data Mining:

Erlaubt dem *Data-User* globale Muster aus den von den *Data-Ownern* zur Verfügung gestellten Daten abzuleiten, ohne dabei einzelne Datenobjekte zu diskreditieren:

1. *Data-User* darf keine Rückschlüsse auf Werte einzelner Datenobjekte ziehen können.
2. Die abgeleiteten Daten dürfen kein Wissen ableiten, dass Rückschlüsse über einzelne Datenobjekte erlaubt.

Gründe für die Wichtigkeit von Privacy Preservation:

1. Viele Daten werden nur vom Data Owner zur Verfügung gestellt, wenn Privacy Preservation garantiert werden kann.
Beispiel: Analyse des Surf-Verhaltens
2. Data Mining soll nicht als Vorwand dienen an private Daten zu gelangen.
3. Schutz vor Missbrauch zur Verfügung gestellter Daten durch Dritte.
Beispiel: Veröffentlichung Ergebnisse über das Surfverhalten, werden genutzt, um Spam-Mails zu personalisieren.

Fazit:

Data Mining kompromittiert nicht per se die Privatsphäre, aber die für das Data Mining verwendeten Daten häufig schon.

=> Verfahren müssen mit ähnlichen aber nicht originalen Daten funktionieren.

- **Grundidee:** Verändere die Datenmenge so, dass die Muster in den Daten gleich bleiben, aber die einzelnen Datenobjekte nicht mehr erkennbar sind.
- **Ziel abhängig von Generalisierung:**
Muster, die auf wenige Individuen zurückgehen, beschreiben diese oft so genau, dass Rückschlüsse möglich sind.
⇒ Muster müssen allgemein genug sein

Schutz der Privatsphäre durch Weitergabe von:

- Veränderten Datenobjekten (Veränderte Daten)
- allgemeinen Modellen der Daten (Verteilungen)
- Daten die durch allgemeine Modelle generiert wurden (Sampling)

Diskretisierung:

- Disjunktes Aufteilen des Wertebereichs in mehrere diskrete Teilmengen/ Intervalle.
- tatsächlicher Wert wird durch Werte-Klasse ersetzt

Beispiel:

- *Originalfakt*: Person A verdient 42.213 € im Jahr
- *diskretisierte Aussage*:
Person A verdient zwischen 35.000 € und 55.000 € im Jahr

Problem:

- Information zwar schwächer aber immer noch vorhanden
 - Rückschlüsse sind möglich, wenn pro Klasse nicht genügend Objekte vorhanden sind.
- ⇒ gleichmäßige Aufteilung des Wertebereichs nach Anzahl der Objekte
(keine äquidistante Aufteilung des Wertebereichs)

Werte-Verzerrung (Data Perturbation):

- übertrage Summe des Werts w und einer Zufallszahl r : w_i+r_i
- Verteilungen für r :
 - Gleichverteilung $[-\alpha, \dots, \alpha]$ $\alpha \in \mathbb{R}^+$
 - Normalverteilung mit Erwartungswert 0 und Standardabweichung σ
- Da Störverteilung den Erwartungswert 0 hat und sich Erwartungswerte bei der Addition von Zufallsvariablen aufaddierten bleibt Erwartungswert der Werte $E(W)$ erhalten.

„Privacy-Level“ (Qualität der Privacy Preservation)

Idee:

Wenn mit $c\%$ vorhergesagt werden kann, dass der Wert x im Intervall $[x_1, x_2]$ liegt, dann entspricht die Breite des Intervalls $(x_2 - x_1)$ dem „Privacy-Level“ auf dem Konfidenzniveau $c\%$.

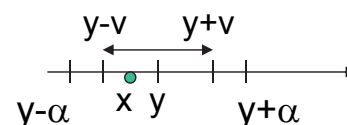
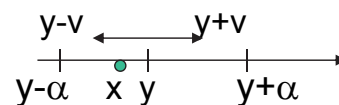
Genauer:

Gegeben: Veränderter Feature-Wert y der durch Addition des Zufallswertes r auf den originalen Feature-Wert x erzeugt wurde.

Gesucht: Breite $2v$ des Intervalls $[y-v, y+v]$, in dem der originale Wert x mit $c\%$ Wahrscheinlichkeit liegt. ($2v \equiv$ Privacy-Level)

Beispiel:

- Störeinfluß R ist gleichverteilt in $[-\alpha, \dots, \alpha]$.
- für 100 % $\Rightarrow \alpha = v \Rightarrow \text{Privacy} = 2\alpha$
(Wert muss im Intervall liegen)
- für 50 % $\Rightarrow 2v/2 \alpha = 0.5$
 $\Rightarrow v = 0.5 \alpha$
- allgemein: $v = c\% * \alpha$



Gegeben: Eine Menge von verzerrten Werten $W = \{w_1, \dots, w_n\}$

- Die Dichtefunktion der Störgröße $R: f_R: \mathbb{R} \rightarrow [0..1]$
- **Gesucht:** Die Verteilung der Originalwerte X mit der Dichtefunktion $f_X: \mathbb{R} \rightarrow [0..1]$
- **Vorgehen :** Abschätzung für einen Wert

$$\begin{aligned}
 F'_{X_1}(a) &= \int_{-\infty}^a f_{X_1}(z | X_1 + Y_2 = w_1) dz = \int_{-\infty}^a \frac{f_{X_1+Y_1}(w_1 | X_1 = z) f_{X_1}(z)}{f_{X_1+Y_1}(w_1)} dz \\
 &= \frac{\int_{-\infty}^a f_{X_1+Y_1}(w_1 | X_1 = z) f_{X_1}(z) dz}{\int_{-\infty}^{\infty} f_{X_1+Y_1}(w_1 | X_1 = z) f_{X_1}(z) dz} = \frac{\int_{-\infty}^a f_{Y_1}(w_1 - z) f_{X_1}(z) dz}{\int_{-\infty}^{\infty} f_{Y_1}(w_1 - z) f_{X_1}(z) dz} = \frac{\int_{-\infty}^a f_Y(w_1 - z) f_X(z) dz}{\int_{-\infty}^{\infty} f_Y(w_1 - z) f_X(z) dz}
 \end{aligned}$$

- Aus $F_{X_1}(a)$ lässt sich jetzt die Verteilung über alle Werte ermitteln:

$$F'_X(a) = \frac{1}{n} \sum_{i=1}^n F_{X_i}(a)$$

- durch Differenzieren erhält man folgende Dichtefunktion:

$$f'_X(a) = \frac{1}{n} \sum_{i=1}^n \frac{f_Y(w_i - a) f_X(a)}{\int_{-\infty}^{\infty} f_Y(w_i - z) f_X(z) dz}$$

- Da $f_X(a)$ unbekannt, nähert man $f'_X(a)$ iterativ an $f_X(a)$ an.

Iterativer Näherungs-Algorithmus zur Rekonstruktion der Originalverteilung:

$f^0_X :=$ Gleichverteilung

$j := 0$ // Iterationzähler

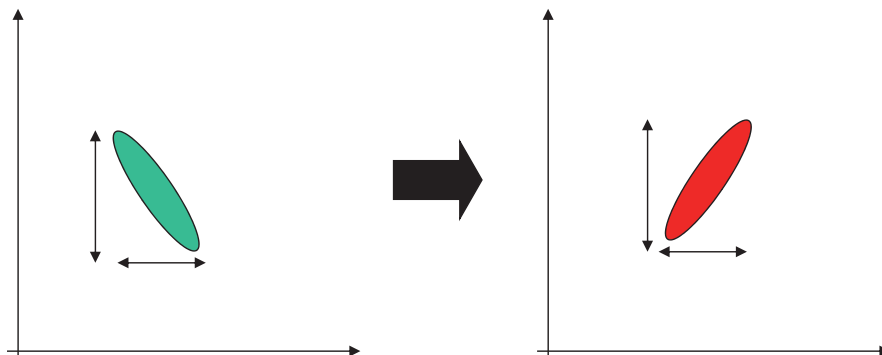
repeat

$$f_X^{j+1}(a) := \frac{1}{n} \sum_{i=1}^n \frac{f_Y(w_i - a) f_X^j(a)}{\int_{-\infty}^{\infty} f_Y(w_i - z) f_X^j(z) dz}$$

$j := j + 1$

until (Änderung $< \varepsilon$)

- Vertausche Attributwerte $X1.a$ mit $X2.a$. (Swapping)
 - ⇒ originale Feature-Vektoren sind nicht mehr nachvollziehbar
 - ⇒ bei Verfahren mit Unabhängigkeit zwischen den Dimensionen kein Unterschied der Verteilungen (Naive Bayes, Entscheidungsbäume)
 - ⇒ bei Mustern die durch Merkmalskorrelation gekennzeichnet sind werden Muster zerstört oder nicht vorhandene Muster zufällig generiert.



Idee:

- Anstatt einzelner Datenobjekte, kann der Data Owner auch lokale Modelle zur Verfügung stellen.
- Diese dürfen dann die lokalen Dataobjekte nicht kompromittieren.

lokale Datenmodelle :

- Verteilungsfunktionen
- explizite Clusterbeschreibungen (Centroide, Verteilungsfunktionen)
- lokal häufige Muster

- X., Jäger J., Kriegel H.-P.: **A Fast Parallel Clustering Algorithm for Large Spatial Databases**, in: Data Mining and Knowledge Discovery, an International Journal, Vol. 3, No. 3, Kluwer Academic Publishers, 1999
- Jagannathan G., Wright R.N. : **Privacy Preserving Distributed k-Means Clustering over Arbitrarily Partitioned Data**, Proc. 11th ACM SIGKDD, 2005
- Kriegel H.-P., Kröger P., Pryakhin A., Schubert M.: **Effective and Efficient Distributed Model-based Clustering**, Proc. 5th IEEE Int. Conf. on Data Mining (ICDM'05), Houston, TX, 2005
- Agrawal R., Srikant R.: **Privacy-preserving data mining**", Proc. of the ACM SIGMOD Conference on Management of Data, Dallas, TX, 2000
- Zhao W., Ma H., He Q.: **Parallel K-Means Clustering Based on MapReduce**, CloudCom, (pp 674-679) 2009
- Lammel R.: **Google's MapReduce Programming Model- Revisited**. Science fo Computer Programming 70, 1-30 , 2008
- Ene A., Im S., Moseley B.: **Fast Clustering using MapReduce**, 17th int. Conf. on Knowledge Discovery and Data Mining (KDD'11), 2011