Ludwig-Maximilians-Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme

Skript zur Vorlesung
Knowledge Discovery in Databases II
im Wintersemester 2012/2013

Kapitel 6: Parallel, Distributed und Privacy Preserving Data Mining

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

4.1 Einleitung

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

4.1 Einleitung

Ziele:

- mehrere Rechner für Verbesserung der Effizienz nutzen

  $\Rightarrow$ 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

Anwendungsbeispiele

- Parallelisierung von DBSCAN zur Beschleuningung 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

Übersicht

4.1 Einführung

paralleles, verteiltes Mining, Privacy Preservation

4.2 Parallele und Verteilte Algorithmen

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

4.3 Privacy Preservation

Peturbation, Aggregation, Swapping,

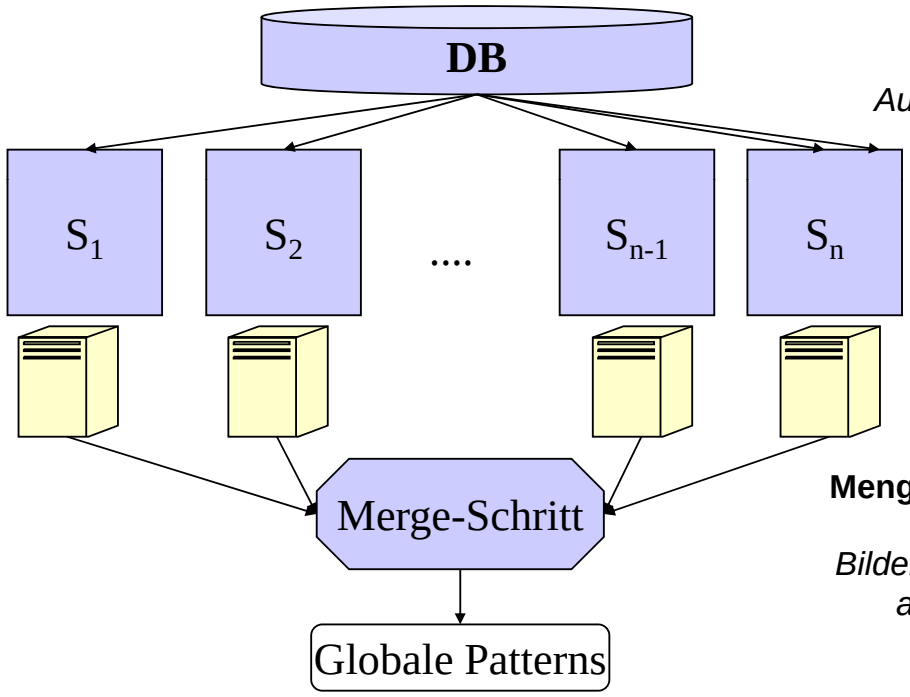
4.2 Parallels und Verteiltes Data Mining

- 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.

Ablauf paralleler Algorithmen



integriete Daten *DB*

Aufteilung von DB in S_i

aufgeteilte Daten

Analyse der S_i

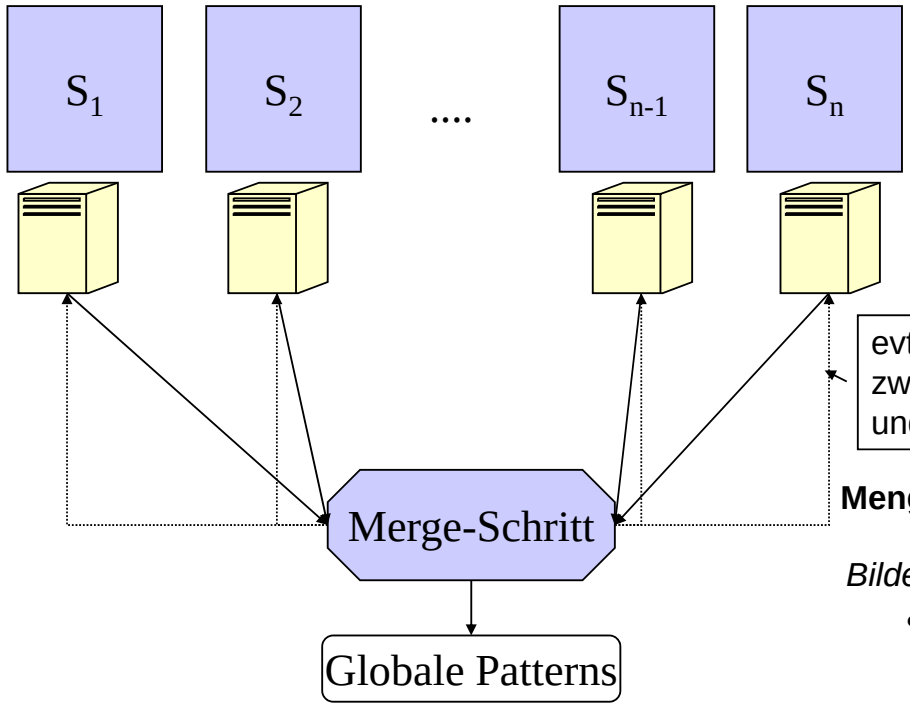
Menge aller lokalen Muster

Bilden von globalen Mustern aus lokalen Mustern

Verteiltes Data Mining

- Verteilung der Daten auf verschiedene Rechner ist vorgegeben
  - $\Rightarrow$ Kein zusätzlicher Aufteilungsaufwand.
  - $\Rightarrow$ 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.

Ablauf verteilter Algorithmen



verteilte Datenbestände

Analyse der S_i

evtl. mehrfache Abgleich zwischen lokalen Daten und globalen Mustern

Menge aller lokalen Muster

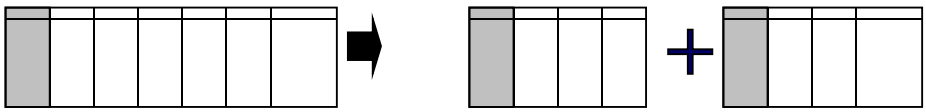
Bilden von globalen Mustern aus lokalen Mustern

Kategorisierung der Algorithmen

1. Nach Aufteilung:

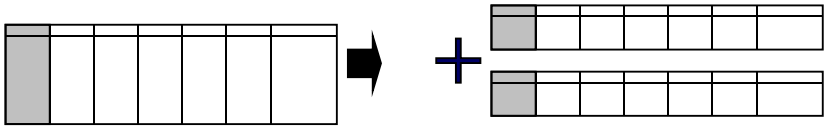
- *Vertikal verteilte Daten*

  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.)

Kategorisierung der Algorithmen

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…

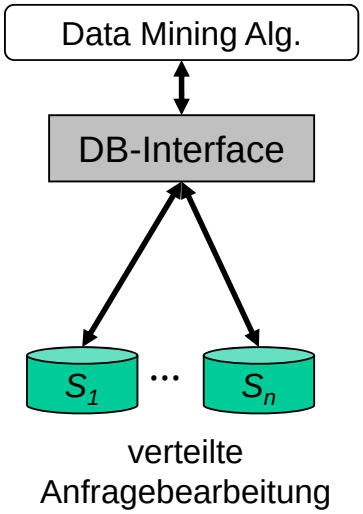
5. **Nach Organisation der verteilten Berechnung:**
   1 Masterprozeß und viele Slaves, mehrere gleichberechtigte Prozeße

Parallele Algorithmen

- 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

Parallelisierung durch Parallele Anfragebearbeitung

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



verteilte Anfragebearbeitung

Beispiele:

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

Aber:

- auch hier Aufteilung entscheidend
- Korrektheit der Anfrage wird vorausgesetzt

Paralleles Dichtebasiertes Clustering

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

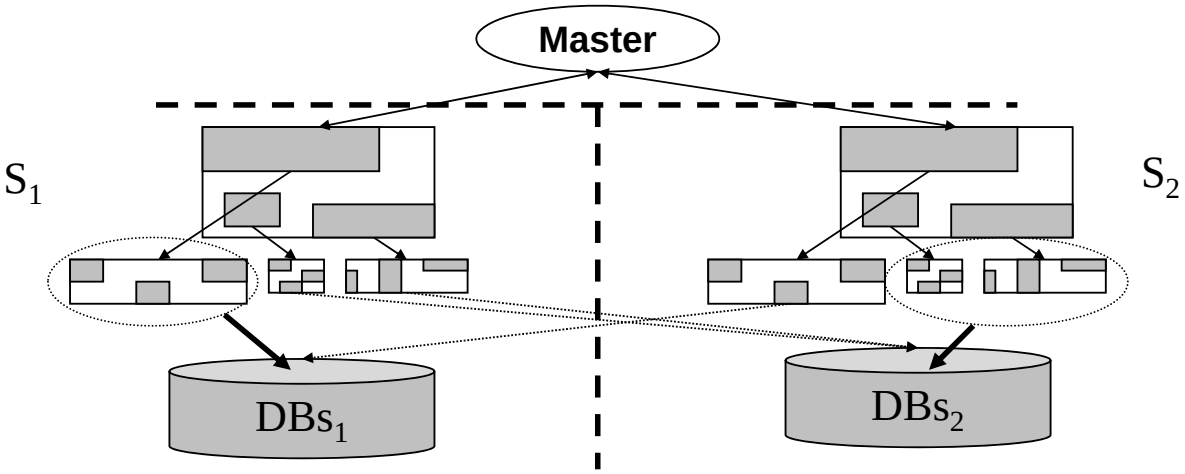
Lösungvorschläge

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

Paralleler DBSCAN



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

PDBSCAN

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.

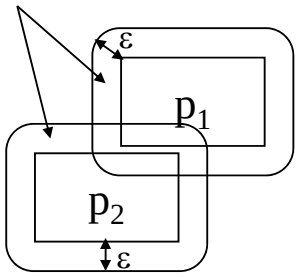
     => Punkte werden in Mergeliste gespeichert
- Zusammenführen der lokalen Cluster über gemeinsame Mergepunkte:

  Mergepunkt muss Kernpunkt in einer Partition sein

  => Verschmelze Cluster

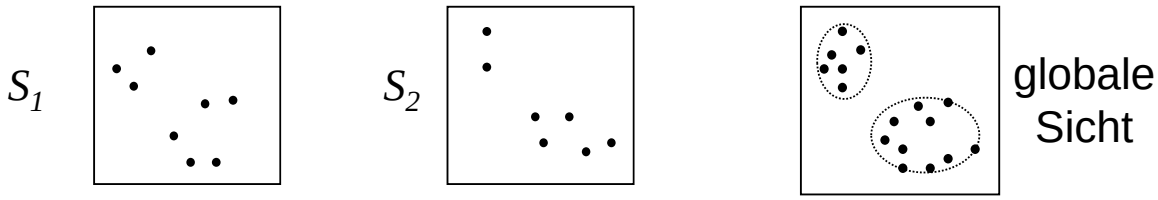
lokale pages p_1 und p_2

Randbereich



Verteiltes Dichtebasiertes Clustering

- 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‘ ⊂ S (=Menge aller Sites) ist.



- Dichtebasiertes Clustering verfügt über kein Clustermodell

  ⇒ Transfer lokaler Punkte, um ein globales Muster zu erzeugen

Verteiltes Partitionierendes Clustering

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.

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

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

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

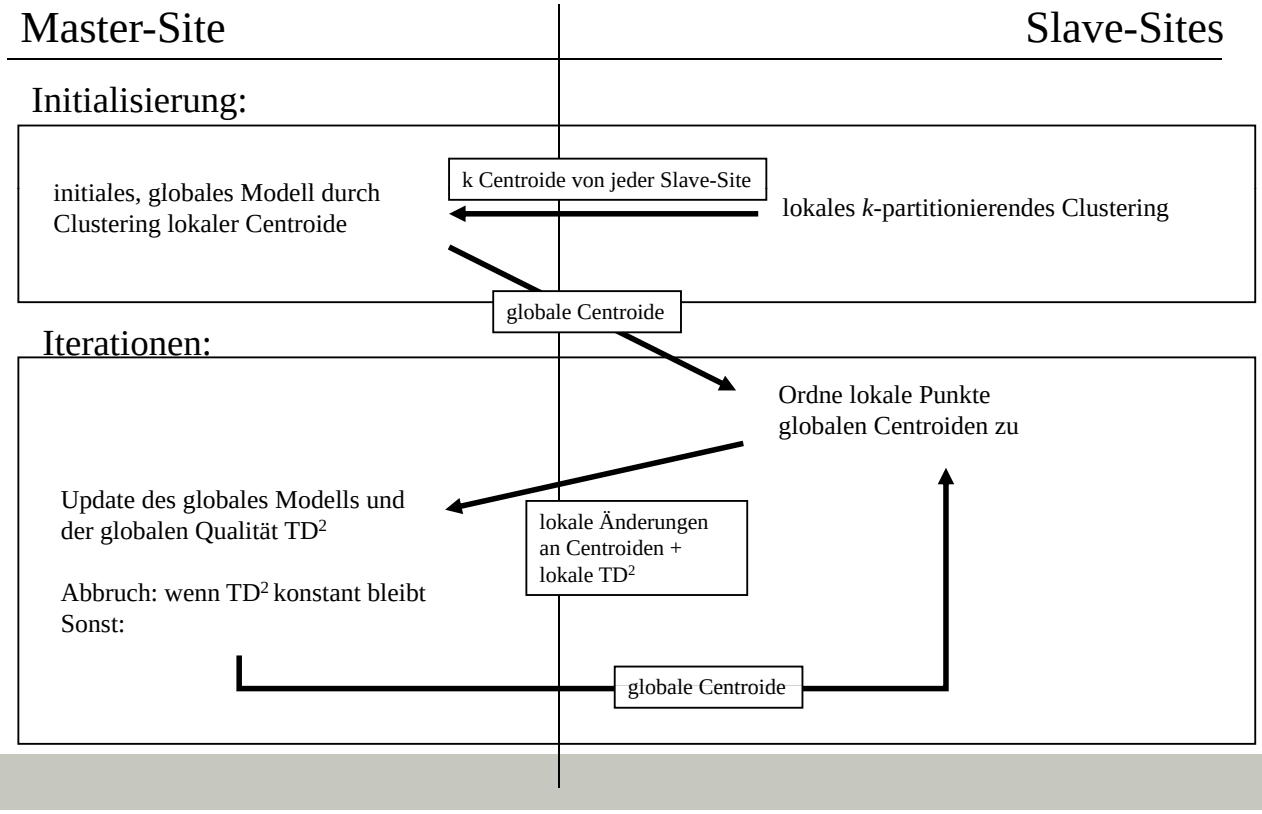
- **Fazit**: In jedem Schritt kann das globale Clustering mit wenig Kommunikations-Overhead optimiert werden.

Verteiltes Partitionierendes Clustering

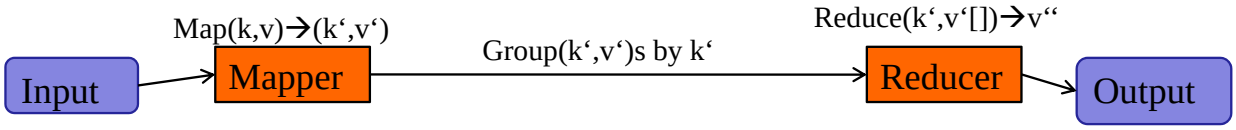
Verteiltes Clustering durch Varianzminimierung (Master-Slave):

```
Bestimme initiale Aufteilung und Anfangs-Centroide
Schleife:
  Übermittle  Centroide an die Sites
  Ordne lokale Punkte den Centroiden zu
      => berechne lokale Centroide und lokale TD²-
  Werte
  Nach Rückübertragung von lokalen Centroiden,
  Cluster-Kardinalitäten und TD²-Werten
   ⇒ Kombination lokaler Centroide zu globalen durch
     Aufaddieren der Centroide und
     Kardinalitäten.(dynamische Berechnung des Centroids)
   ⇒ Bestimmung des globalen TD²-Werts
  Falls TD²-Werte nicht verbessert wurde =>Abbruch
```

Ablauf Verteiltes Partitionierenden Clusterings



Parallele Daten Analyse mit MapReduce



- 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

Aufbau eines MapReduce Programms

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 zusammengefast.

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.

optionale Schritte im Framework

- ***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>

K-Means mit MapReduce

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

$\Rightarrow$ alle Schritte sind linear mit einem Scan über D durchführbar

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

k-Means mit MapReduce

Mastermethode:

Sample k initiale Centroide C.

WHILE TD2 < oldValue

oldValue = TD2

Bilde Zuordnung Objekte D zu Centroiden C (Mapper)

Bilde neues C und TD2 (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

K-Means: Map-Schritt

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

K-Means: Reduce-Schritt

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

K-Means mit MapReduce

- 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 Datenmegen zu verarbeiten

4.3 Privacy Preservation

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.

Privacy Preservation

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: Veröffentliche 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.

Privacy Preservation

- **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.

  $\Rightarrow$ 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.

$\Rightarrow$ 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 $[-\alpha,..,\alpha]$ $\alpha \in IR^+$
  - 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 Interval $[x_1, x_2]$ liegt, dann entspricht die Breite des Intervals (x_2-x_1) dem „Privacy-Level“ auf dem Konfidenzniveau *c%*.

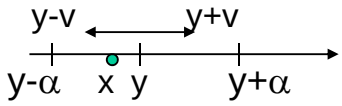
Genauer:

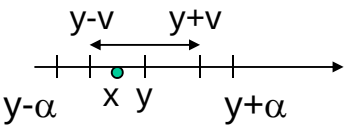
Gegeben: Veränderter Feature-Wert y der durch Addition des Zufallswertes *r* auf den originalen Feature-Wert *x* erzeugt wurde.

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

Beispiel:

- Störeinfluß R ist gleichverteilt in $[-\alpha,..,\alpha]$.
- für 100 % $\Rightarrow \alpha = v \Rightarrow$ Privacy = 2α
  (Wert muss im Interval liege)
- für 50 % $\Rightarrow 2v/2\,\alpha = 0.5$
  $\Rightarrow v = 0.5\,\alpha$
- allgemein: $v = c\% * \alpha$





Rekonstruktion der Originalverteilung

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

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

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

$$= \frac{\int_{-\infty}^{a} f_{X_1+Y_1}(w_1|X_1 = z) f_{X_1}(z)dz}{\int_{-\infty}^{\infty} f_{X_1+Y_1}(w_1|X_1 = z) f_{X_1}(z)dz} = \frac{\int_{-\infty}^{a} f_{Y_1}(w_1 - z) f_{X_1}(z)dz}{\int_{-\infty}^{\infty} f_{Y_1}(w_1 - z) f_{X_1}(z)dz} = \frac{\int_{-\infty}^{a} f_Y(w_1 - z) f_X(z)dz}{\int_{-\infty}^{\infty} f_Y(w_1 - z) f_X(z)dz}$$

Rekonstruktion der Originalverteilung

- Aus $F_{X1}(a)$ 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.

Rekonstruktion der Originalverteilung

Iterativer Näherungs-Algorithmus zur Rekonstruktion der Originalverteilung:

f^0_X:= Gleichverteilung

j:= 0 //Iterationzähler

repeat

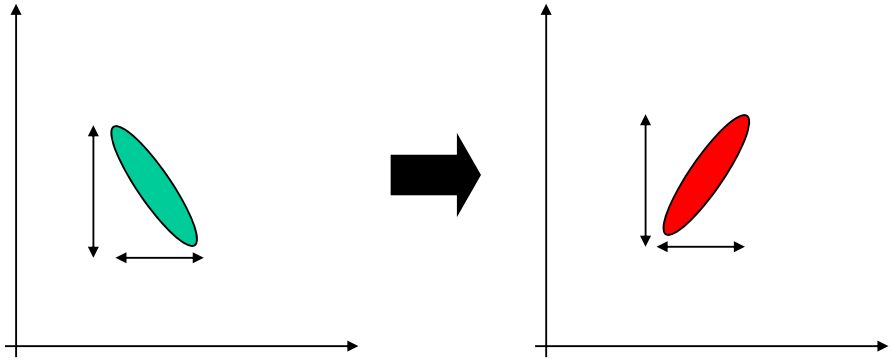
$$f_X^{j+1}(a) := \frac{1}{n}\sum_{i=1}^{n} \frac{f_Y(w_i - a) f_X^j(a)}{\int_{-\infty}^{\infty} f_Y(w_i - z) f_X^j(z) dz}$$

j:=j+1

until (Änderung < ε)

Weitere Techniken Privacy Preservation

- 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.



Privacy Preservation durch Datenapproximation

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

Literatur

- 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