Data Mining

Praktikum

Wintersemester 2019/20

Abteilung Datenbanken, Universität Leipzig
http://dbs.uni-leipzig.de

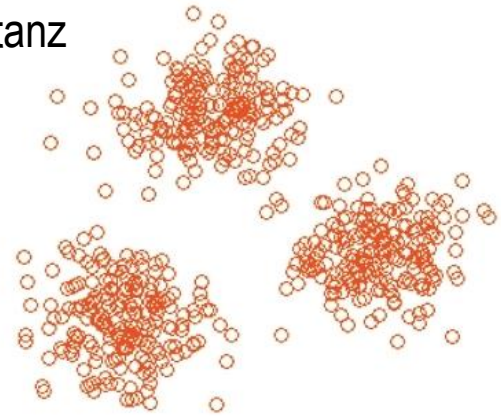
Einführung

- Implementierung von Data-Mining-Algorithmen mit Java und Spark
- Begleitend zur Vorlesung “Data Mining”
- *Einschränkungen:*
  - Verwendung von Apache Spark
  - Schreiben der Programme in **Java** (nicht Scala, Python, R, oder SQL)
  - Umsetzung der Algorithmen mit RDDs (nicht DataFrames oder Datasets)
- Durchführung in Gruppen zu je 2 Studierenden
- Praktikumsleistung: 3 Testate
  - Testat 1: bis 29. November 2019
  - Testat 2: bis 31. Januar 2020
  - Testat 3: bis 31. März 2020

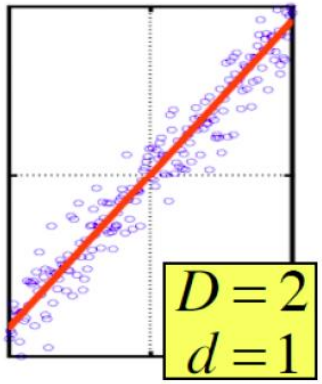
Wichtige Informationen: https://dbs.uni-leipzig.de/stud/2019ws/dwhdmprak

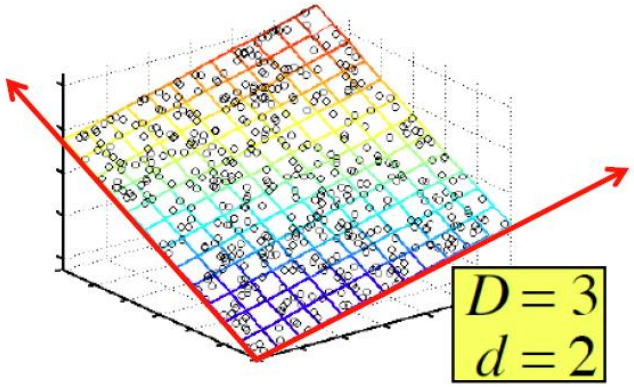
Themen

- **K-Means Clustering**
  - Gruppierung von Datenpunkte in **Cluster**
    - Mitglieder eines Cluster haben geringe paarweise Distanz
    - Mitglieder verschiedener Cluster haben hohe paarweise Distanz
  - Beispiele
    - Gruppierung von Musikalben nach Käufern
    - Gruppierung von Dokumenten nach Thema
    - Segmentierung von Bildern



- **Dimensionsreduktion**
  - Datenpunkte eines D-dimensionalen Raumes liegen vorrangig in der Nähe eines d-dimensionalen *Unterraumes*
  - Reduzierung der Punkte auf deren Projektionen im Unterraum





Themen

- **Empfehlungssysteme**
  - Schätzung unbekannter Bewertungen
  - Verschiedene Herangehensweisen
    - Kollaboratives Filtern
    - Latentes Variablenmodell

	Avatar	LotR	Matrix	Pirates
Alice	10		2	
Bob		5		3
Carol	1		5	
David				4

- **Assoziationsregeln**

Warenkörbe
Brot, Cola, Milch
Milch, Windeln
Bier, Cola, Windeln, Milch
Bier, Brot, Windeln, Milch
Cola, Windeln, Milch, Bier

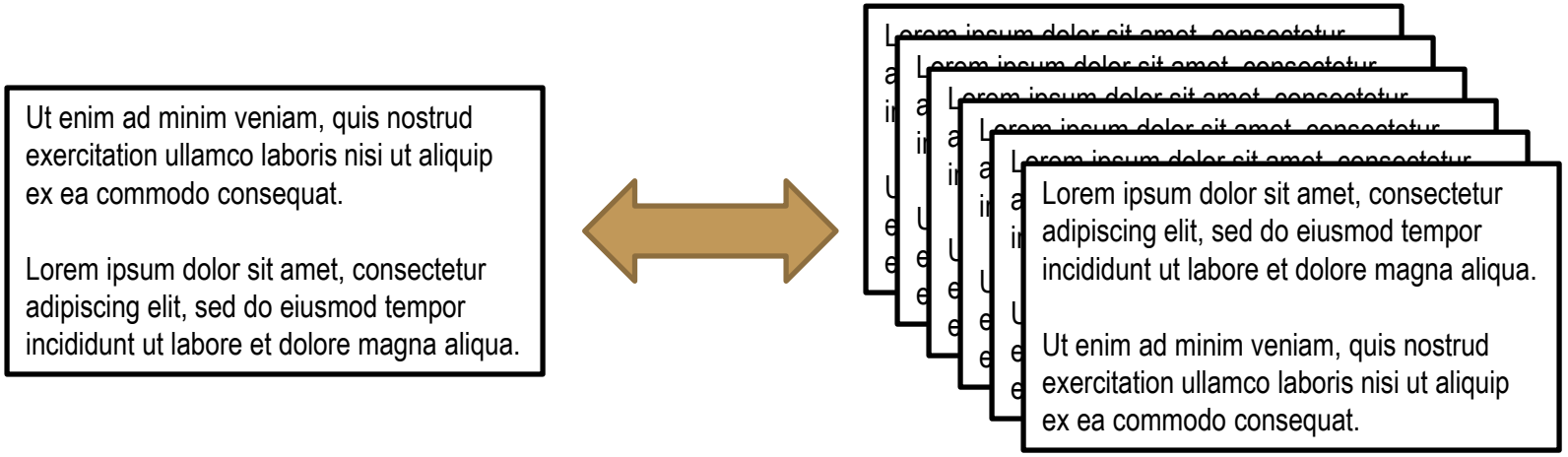
Assoziationsregeln:

{Milch} → {Cola}

{Windeln, Milch} → {Bier}

Themen

- **MinHashing & LSH**
  - Finde ähnliche Paare in einer *riesigen* Menge von hochdimensionalen Daten
  - Berechnung in linearer statt quadratischer Zeit

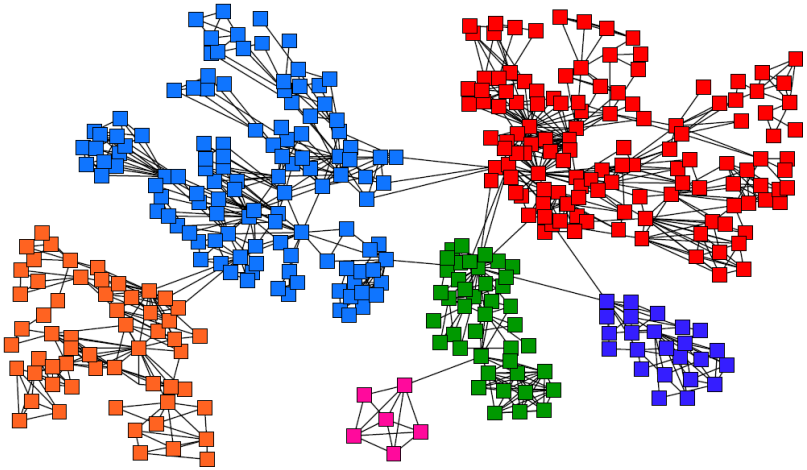


- **Supervised Machine Learning**
  - Auffinden eines möglichst akkuraten Modells zur Vorhersage eines Attributs
  - Logistische Regression, SVM
  - Entscheidungsbäume und Random Forests

Age	Male	Class	Survived
Child	True	1	Yes
Adult	True	2	No
Adult	False	1	Yes
Child	True	3	No
…	…	…	…

Themen

- **Analyse von Graphen**
  - Soziales Netzwerk: Empfehlungen von Freunden
  - PageRank: Wichtigkeit von Webseiten für die Reihenfolge von Suchergebnissen



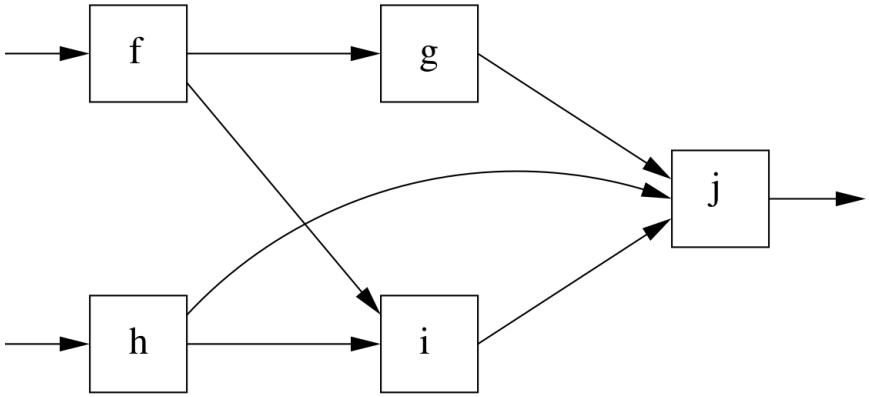
- **Datenströme**
  - Stetiges Erheben von Daten (z.B. Sensoren, Interaktionen im WWW)
  - Beispiel: Zählen von Wörtern
    - Beschränkung auf die neuesten N Elemente
    - Exponentially Decaying Window

q w e r t y u i o p a s d f g h j k l z x c v b n m

Einführung zu Spark

- Einschränkungen von MapReduce:
  - Schwierige Programmierung: Viele Probleme sind nicht (so einfach) als MapReduce-Prozedur beschreibbar
  - Oft müssen mehrere MapReduce-Schritte kombiniert werden
  - Leistungsengpässe: Speicherung auf Festplatte ist wesentlich langsamer als die Arbeit im Hauptspeicher

- Erweiterung: *Workflow Systeme*
  - Verteilte Verarbeitung
  - Weitere Operationen neben Map und Reduce
  - **Azyklisches Datenfluss-Netzwerk**
  - Effiziente Verarbeitung von Daten durch Vermeidung des Speicherns von Zwischenergebnissen auf Festplatte



- Beispiele: **Spark**, Flink, TensorFlow

Spark

- Datenstruktur: *Resilient Distributed Dataset* (RDD)
  - Datenmenge aus Objekten eines Types (z.B. String, Menge, Schlüssel-Wert-Paar)
  - Repliziert und verteilt über Cluster
  - Unveränderbar
- RDDs werden über Hadoop geladen oder aus einer anderen RDD erstellt

- **Transformationen**: Umwandlung eines RDD in eine neue RDD
  - z.B. map, flatMap, filter, join, groupByKey, …
  - Lazy Evaluation: Keine Berechnung

- **Aktionen**: Berechnung eines Wertes oder Exportierung des RDD
  - z.B. count, take, collect, reduce, saveAsTextFile, …
  - Aktionen erzwingen Berechnungen und geben Werte aus bzw. exportieren Daten

Spark: map()

Anwendung einer Funktion auf jedes Element einer RDD und Rückgabe einer neuen RDD mit den Ergebnissen als Elemente

```
JavaRDD<String> lines = sc.textFile(args[0]);

JavaRDD<String[]> wordsArrays = lines
      .map(line -> line.split(" "));

JavaRDD<Integer> numbers = wordsArrays
      .map(wordsArray -> wordsArray.length);

// nur eine Zeile:
JavaRDD<Integer> numbers = sc.textFile(args[0])
      .map(line -> line.split(" "))
      .map(wordsArray -> parts.length);
```

Spark: flatMap()

Anwendung wie map(), doch jedes Element kann auf eine Menge von neuen Elementen abgebildet werden

```
JavaRDD<String> lines = sc.textFile(args[0]);

JavaRDD<String> words = lines
      .flatMap(line ->
      Arrays.asList(line.split(" ")).iterator());

// Action:
int numberOfWords = words.count()
```

Spark: mapToPair()

Anwendung einer Funktion auf jedes Element einer RDD und Rückgabe einer neuen **PairRDD** mit den Ergebnissen als Elemente

```
JavaPairRDD<String, Integer> pairs = words
      .mapToPair(w -> new Tuple2<>(w, 1));
```

- PairRDD ist RDD aus Schlüssel-Wert-Paaren
- Spark bietet für PairRDDs einige zusätzliche Funktionen an:
  - groupByKey()
  - reduceByKey()
  - sortByKey()
  - join()
  - cogroup()
  - mapValues()
  - …

Spark: PairRDD

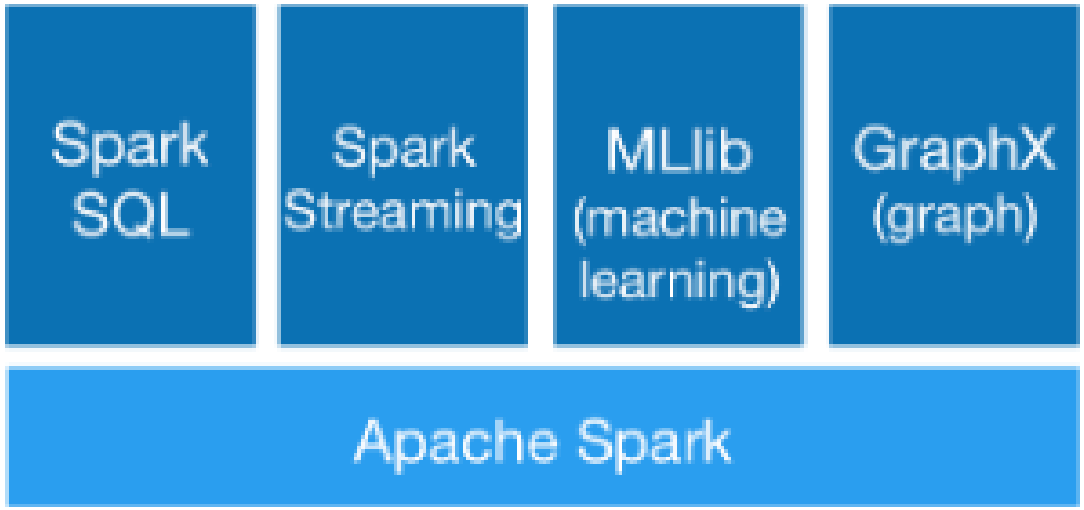
```
JavaPairRDD<String, Iterator<Integer>> grouped =
      pairs.groupByKey();

JavaPairRDD<String, Integer> counts = pairs
      .reduceByKey((n1, n2) -> n1 + n2);

JavaPairRDD<String, Integer> wordLength = words
      .distinct()
      .mapToPair(w -> new Tuple2<>(w, w.length()));

JavaPairRDD<String, Tuple2<Integer,Integer>> res =
      wordLength.join(counts);
```

Spark

- Download und Dokumentation:

  https://spark.apache.org/

- Bibliotheken:



- Praktische Einführung und Aufgaben des Praktikums:

  https://git.informatik.uni-leipzig.de/dbs/dataminingpraktikum