Spezielle Kapitel Sozialer Web – **Technologien** – Entity Resolution

Seminaristischer Unterricht

Prof. Dr.-Ing. Hendrik Gärtner

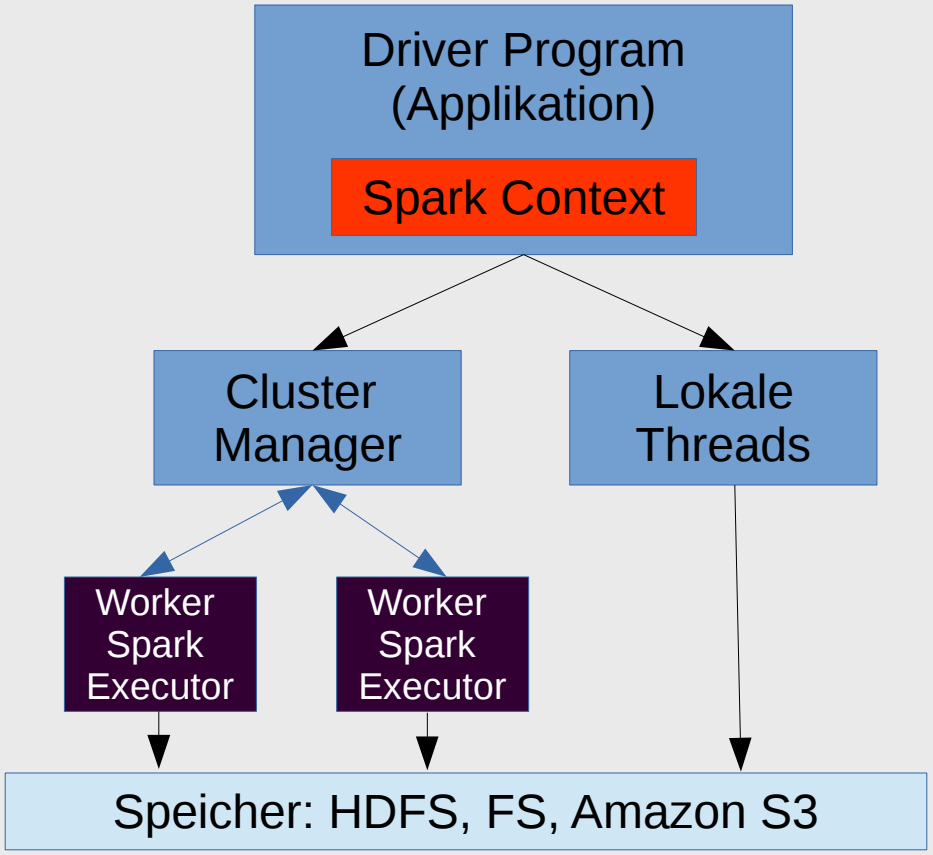
Gliederung

- Wiederholung Spark
  - Serialisierung
  - Akkumulatoren/Broadcast-Variablen
  - Caching
- EntityResolution
  - Motivation
  - Problemstellungen
  - Similarity Analyse
  - Umgang mit Skalierung
  - Bewertung der Ergebnisse

Spark Architektur

Eine Spark-Applikation besteht aus zwei Programmen:

- Ein sogenannter Treiber (Driver Program) sowie die
- Worker, die die eigentliche Arbeit verrichten



Ein Worker läuft entweder:

- innerhalb eines Clusters oder
- in Localen Threads

Die RDDs sind verteilt

Wie kommen Code und Daten vom Treiber zu den Workern?

Vorgehensweise von Spark

Wenn Spark eine Transformation ausführt wird automatisch ein sogenanntes **Closure** erzeugt und dieses

- auf dem Treiber Knoten **serialisiert**,
- zum entsprechden Knoten im Cluster transferiert,
- **deserialisiert** und
- und am Ende auf dem Knoten ausgeführt.

Serialisierbarkeit von Funktionen

```
class SearchFunctions(val query:String){
    def isMatch(s:String):Boolean ={s.contains(query)}

     def getMatchesFunctionReference(rdd:RDD[String]):RDD[Boolean]={
        // problem: isMatch means this.isMatch), so we pass all of this
        rdd.map(isMatch)}

     def getMatchesFieldReference(rdd:RDD[String]):RDD[Array[String]]={
        // problem: isMatch means this.isMatch), so we pass all of this
        rdd.map(x=>x.split(query))}

   def getMatchesNoReference(rdd:RDD[String]):RDD[Array[String]]={
       // Safe: Extracts just the field we need into a local variable
       val query_ = this.query
       rdd.map(_.split(query_))}
}
```

Beispiel aus: Learning Spark – Lightning Fast Data Analysis, O'Reilley, 2015, page 32

Problemstellungen der Spark-Closures

Beispiel Sentimentanalyse:

dataRDD.map(x=> calculateSentimentValues(x,sentimentVals))

- Wie wird damit umgegangen, wenn große, statische Datenmengen (z.B. das sentimentDict) an die Worker gesendet werden müssen? Was ist, wenn diese immer wieder benötigt werden?

- Was ist, wenn bestimmte Ereignisse an den Driver zurückgemeldet werden sollen ? (z.B. wenn eine Textpassage unter einen bestimmten Sentiment-Wert rutscht)

➢ **Broadcast-Variablen**

➢ **Akkumulatoren**

Broadcast-Variablen

- Senden sehr effizient „Read-Only“-Daten zu allen Workern
- Werden auf allen Workern gespeichert, um sie für mehrere Operationen verwenden zu können
- Sind z.B. für das Versenden großer „Lookup“-Tabellen geeignet

Anwendung von Broadcast-Variablen

Beispiel Sentimentanalyse:

```
val sentimentVals:Map[String,Double]= loadDictionary(...)

// Definition einer Broadcast-Variablen:

val sentimentValsBroadcast= sc.broadcast(sentimentVals)

// Übergabe der Broadcast-Variable

dataRDD.map(x=> calculateSentimentValues(x,sentimentValsBroadcast))


def calculateSentimentValues(line:String,
                             sentimentDict:Broadcast[Map[String,Double]]):Double={

...

val dictionary= sentimentDict.value

}
```

Richtung von Closures

Beispiel in Scala:

```
val l=List(1,2,3,4,5,6,7,8,9,10)

var counter=0

l.foreach(x=>counter=counter+x)
```

→ Counter ist 55

Beispiel mit Spark

```
val rdd= sc.parallelize(l,8)

var counter=0

rdd.foreach(x=>counter=counter+x)
```

→ Counter ist 0

Mit dem Closure geht die Verbindung zum Driver verloren

Anwendung von Akkumulatoren

```
val l=List(1,2,3,4,5,6,7,8,9,10)
val rdd= sc.parallelize(l,8)
val counteraccu= sc.accumulator(0)
def count(element:Int, counter:Accumulator[Int]):Unit={
   counter += element
}
rdd.foreach(count(_,counteraccu))
```

→ counteraccu ist 55!

- += ist ein spezieller Operator, der definiert sein muss
- Definition eigener Akkumulatorentypen möglich

Eigenschaften Akkumulatoren

- Akkumulatoren können nur mit einer assoziativen Operation zusammengefasst werden
- Wird verwendet um effizientz count und sum durchzuführen
- Nur der Driver kann die Werte des Akkumulators sehen – die Tasks dürfen sie nur schreiben
- Accumulators können in Actions und Transformations verwendt werden
  - Actions: jedes update des Akkus wird nur einmal durchgeführt
  - Transformations: keine Garantie (nur fürs debugging)
- Types: integers, double, long, float
- Custom type möglich

Caching

```
val set=Range(1,1000000)

val rdd= sc.parallelize(set,8)

val res= rdd.map(x=>Math.sqrt(x)).filter(x=>(x %2)==0).filter(x=>(x%3)==0).cache

res.count

res.collect
```

- Cache speichert das berechnete RDD auf dem JVM Heap
- Egal wie viel Actions auf dem RDD ausgeführt werden, es wird nur einmal berechnet
- Ist der Speicher voll, so wird nach dem Prinzip LRU (Least Recently Used) der Speicher freigegeben
- Fällt ein Knoten aus, so werden die Daten für den Knoten neu berechnet

Operationen: cache und persist

- Die Operation cache entspricht der Operation persist(StorageLevel.MEMORY_ONLY)
- Unpersist löscht das RDD wieder aus dem Speicher

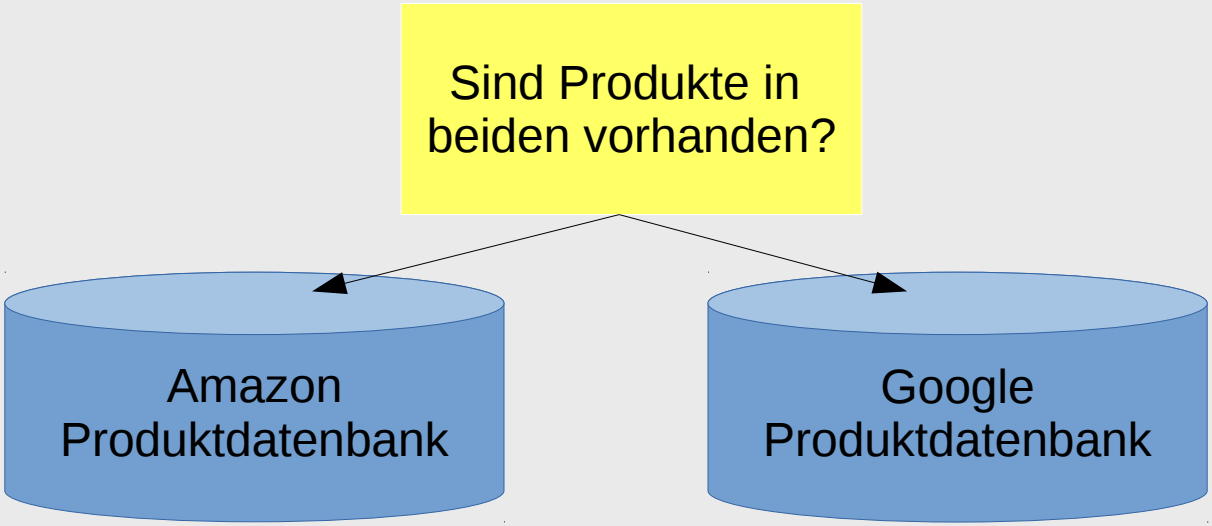
Level	Space Used	CPU time	In Memory	On Disk
MEMORY_ONLY	High	Low	Yes	No
MEMORY_ONLY_SER	Low	High	Yes	No
MEMORY_AND_DISK	High	Medium	Some	Some
MEMORY_AND_DISK_SER	Low	High	Some	Some
DISK_ONLY	Low	High	No	Yes

Entity Resolution

Entity Resolution – Record Linkage

- Record linkage (RL) bezeichnet die Aufgabe, Datensätze aus unterschiedlichen Quellen zu finden, die auf die selbe Entity verweisen
- Record Linkage ist z.B. erforderlich, wenn Datenbanken, die kein Mapping zwischen den Elementen besitzen (z.B. einheitliche Schlüssel), zusammengeführt werden
- Record Linkage kann z.B. auch für das Bereinigen von Datenmengen angewendet werden (z.B. generiert durch Webcrawling)

Beispiel aus der Belegarbeit



Das Fileformat von Amazon ist:

"id","title","description","manufacturer","price"

Das Fileformat von Google ist:

"id","name","description","manufacturer","price"

Welche Schritte sind erforderlich, um die Verlinkungen zu finden?

Entity Resolution – Vorgehensweise

- Einlesen der Daten und Aufbereiten
- Implementierung einer Methode zum Vergleich der einzelnen Records:
  - Deterministisches Ergebnis?
  - Bestimmung auf Basis von Wahrscheinlichkeiten
- Skalierung? Muss jedes Element mit jedem verglichen werden?
- Wenn auf Wahrscheinlichkeiten gearbeitet wird: Wir gut ist mein Ergebnis?

Einlesen und Aufbereiten der Datensätze

- Methoden zum Einlesen sind vorgegeben (Utils.scala)
  - getData zum Einlesen
  - ParseLine zum Parsen einer Zeile
  - TokenizeString zum Splitten der Wörter
  - DeleteQuote für das Löschen von Anführungszeichen
- Anwenden der vorgegebenen Funktionen ist Teil der Belegarbeit

Ergebnis: Für jedes Produkt gibt es ein Tupel der Form:

(ProduktID, Text zusammengesetzt aus Titel, Description, Manufacturer)

Wie kann die „Gleicheit“ der Informationen der Produkte bewertet werden?

Abstandsmaße

- Levensteihn-Abstand?
- Anzahl der Wörter? (Produkte haben ungefähr gleich lange Beschreibungen?)
- Anzahl der gleichen Wörter?
  - Unterschiedliche Textlängen in den Beschreibungen
  - Was ist mit Wörtern, die keinen Inhalt beitragen (z.B. to, a, the, from...)
  - Wie häufig kommt ein Wort in der Produktbeschreibung vor?
  - Welche Relevanz haben die Wörter in Bezug auf alle Produktbeschreibungen?

TF-IDF-Vefahren (1/2)

Schritt 1: Herauslöschen der sogenannten stopwords (Wörter ohne Informationsgehalt) aus allen Produktbeschreibungen

Schritt 2: Bestimmung der Term-Frequency für jedes Wort:

Die Term Frequency beschreibt die Relative Häufigkeit eines Wortes innerhalb eines Textes

Beispiel:

„Dies ist ein Text und das ist noch ein Text“

Dieser Text beinhaltet 10 Wörter, d.h. die Term Frequency sind: „dies“: 0.1, „ist“:0.2, „ein“:0.2, „das“:0.1, „noch“:0.1, „text:“ 0.2, „und“:0.1

TF-IDF Verfahren (2/2)

Schritt 3: Bestimmung der Inverse Document Frequency.

Die Inverse Document Frequency bewertet die Relevanz eines Wortes über die gesamte Datenmenge.

<u>Beispiel:</u>
Produkt 1: „Software Microsoft Word Textverarbeitung“

Produkt 2: „Software OpenOffice Textverarbeitung“

Produkt 3: „Software World of Warcraft – echt super super super Game“

Produkt 4: „MS Word Textverarbeitung – auch super“

- Bestimmung aller vorkommenden Wörter (ohne Duplikate):
  Set(Software, Microsoft, Textverarbeitung, Word, super,…)
- Bestimmung der Anzahl der Vorkommen der Wörter in den Produkten (Häufigkeit dabei irrelavent, z.B. super:2, Textverarbeitung: 3, OpenOffice:1,…)
- IDF ist die Anzahl der Dokumente/Anzahl der Vorkommen
- TF-IDF ist die Term Frequency * der IDF-Wert

Anwendung in der Belegarbeit

1) Berechnung der Term Frequency für jedes Produkt:
   Zu jedem Produkt wird ein Vector berechnet, der die TermFrequency beinhaltet
2) Berechnung des sogenannten IDF-Dictionaries:
   1) Zusammenfassen der Amazon und Google Produkt RDDs
   2) Ermittlung aller im Corpus vorhandenen Wörter (Streichen Duplikate)
   3) Bestimmung für jedes Wort, in wie viel Produkten es vorkommt
   4) Berechnung des IDF-Wertes für jedes Wort (Anzahl Dokumente/Anzahl Vorkommen)
3) Berechnung der TF-IDF-Werte für jedes Produkt:
   TF-IDF= TF * IDF

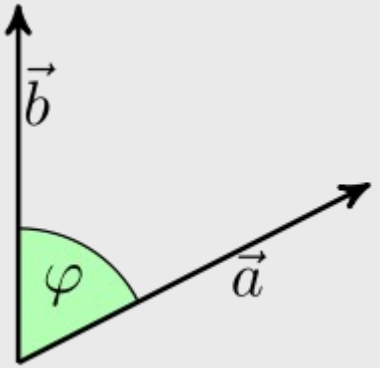
Ergebnis: Jedes Produkt besteht aus einem Dokumentenvector mit den TF-IDF-Werten

Cosinus Ähnlichkeit

$$\text{Kosinus-Ähnlichkeit} = \cos(\theta) = \frac{a \cdot b}{\|a\|\|b\|} = \frac{\sum_{i=1}^{n} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{n} (a_i)^2} \cdot \sqrt{\sum_{i=1}^{n} (b_i)^2}}$$

- a und b sind Dokumentenvektoren
- a*b ist das sogenannte dot-Produkt (Skalarprodukt)

$$\vec{a} \cdot \vec{b} = |\vec{a}|\,|\vec{b}|\,\cos\sphericalangle(\vec{a},\vec{b}).$$



-
- ||x|| ist die L_2-Norm des Vectors, die sich aus: sqrt(Σ a_i^2)
- Der Cosinus ist der Winkel zwischen den Vectoren
- Die Algebraische Definition des Skalarprodukts ist:

$$\mathbf{A} \cdot \mathbf{B} = \sum_{i=1}^{n} A_i B_i = A_1 B_1 + A_2 B_2 + \cdots + A_n B_n$$

Anwendung der Kosinus-Ähnlichkeit

- Berechnung der Vektor-Normen
- Berechnung der Skalarprodukte
- Erzeugen aller möglichen Produktkombinationen aus der Google und Amazon Datenmenge
- Berechnung der Kosinus Ähnlichkeit für alle Produktpaare

Bei welcher Kosinus-Ähnlichkeit handelt es sich voraussichtlich um dasselbe Produkt?

Wie gut ist mein Verfahren?

Auswertung mit dem Gold-Standard

- Gold Standard beinhaltet alle Produktverlinkungen
- Auswertung mit Gold Standard:
  - Wählen eines Thresholds
  - Zählen der True-Positives, False-Positive, True-Negative, False-Negative

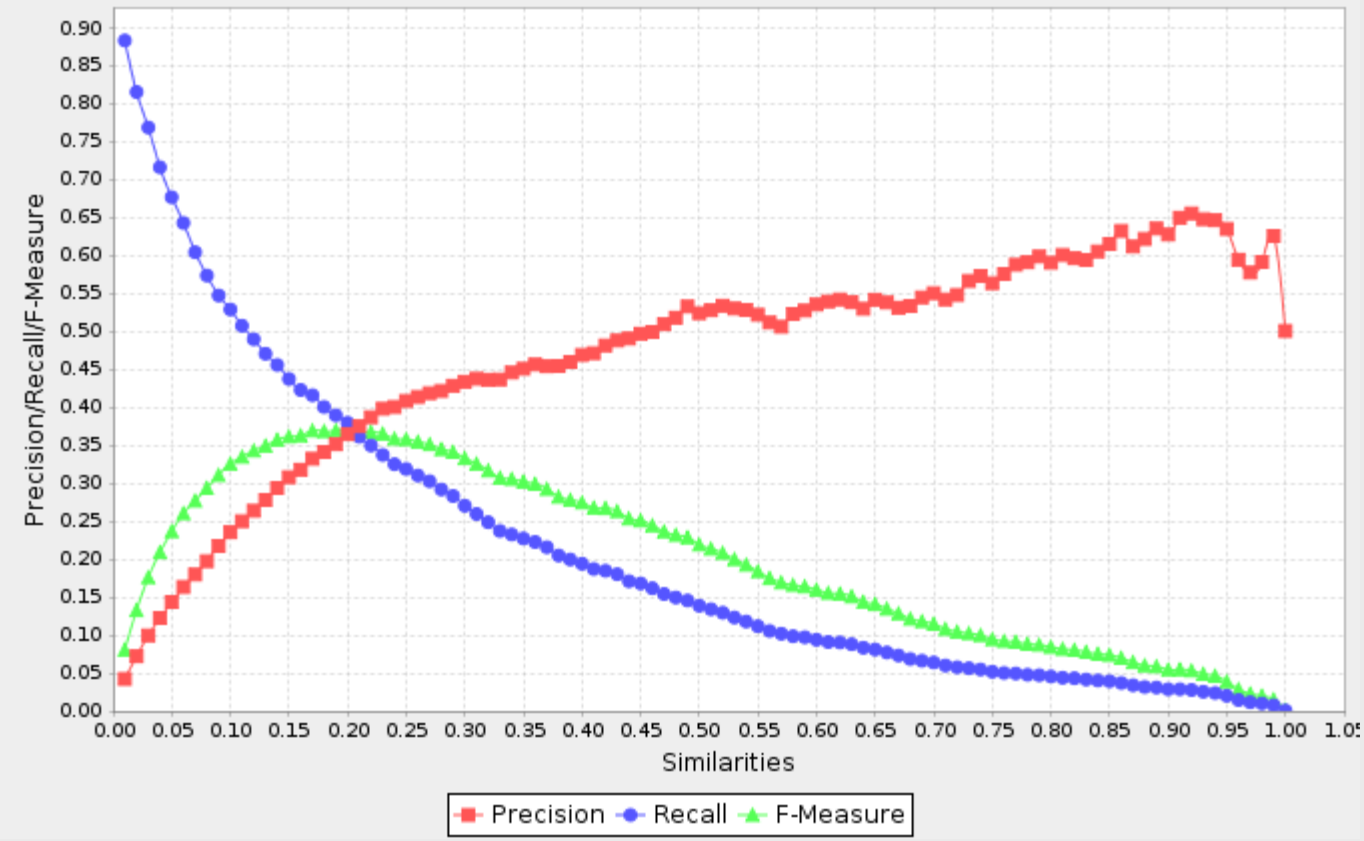
	Tatsache: Selbes Produkt	Tatsache: Unterschiedliches Produkt
Vorhersage: Selbes Produkt	True-Positive	False-Positive
Vorhersage: Unterschiedliches Produkt	False-Negative	True-Negative

Kennzahlen für die Bewertung

- Einfaches Maß:
  Accuracy= Richtige Ergebnisse/Gesamtzahl
- Maß häufig problematisch: Alle Elemente False-Postives und False-Negatives werden gleich gewichtet

  Besser:
- **Precision** = true-positives / (true-positives + false-positives)
- **Recall** = true-positives / (true-positives + false-negatives)
- **F-measure** = 2 x Recall x Precision / (Recall + Precision)

Ergebnis der Analyse – Wenn Funktionen richtig ;-)



Funktionen zur Berechnung und Visualisierung der Daten sind vorgegeben.

Skalierung des Algorithmus

- Was passiert, wenn der die Datenmengen größer werden?
- Anzahl der Produktkombinationen berechnen sich durch die Multiplikation der Anzahl der Produkte beider Datenmengen
- Kann schnell anwachsen

Wie können die Anzahl der Kombinationen reduziert werden?

Wie kann der Algorithmus effizient implementiert werden?

Skalierung des Algorithmus

- Broadcast Variablen
- Erstellung eines Inversen Indexes (Wort → DokID)
- Ergebnis soll eine RDD sein, dass alle Produktkombinationen enthält, die mindestens ein Token gemeinsam besitzen
- Berechnung der Cosinus-Ähnlichkeit nur für die Paare, die mindestens 1 gemeinsames Token haben
- Verwendung nur der gemeinsamen Token für das Skalarprodukt

Vielen Dank für

Ihre Aufmerksamkeit

Prof. Dr.-Ing. Hendrik Gärtner