# Beleg 3 – Aufgabenstellung

Thema: Text Analyse und Entity Resolution

Entity Resolution ist ein typisches Problem der Datenbereinigung und Datenintegration. Dabei wird untersucht, welche einzelnen Verlinkungen zwischen Datensätze existieren. Das Anwendungsfeld dieser Problemstellung ist vielfältig und reicht vom Mapping von Nutzern bis hin zu Finden von Datenduplikaten.

In dieser Belegarbeit wird ein Algorithmus zum Finden von Produktduplikaten in unterschiedlichen Datenbasen implementiert. Hierzu werden die folgenden Beispieldaten verwendet. Sie stammen aus dem metric-learning Projekt (https://code.google.com/p/metric-learning/):

- **Google.csv**, Google Produkte
- **Amazon.csv**, Amazon Produkte
- **Google_small.csv**, 200 Records gesampled aus der Google Datei
- **Amazon_small.csv**, 200 Records gesampled aus der Amazon Datei
- **Amazon_Google_perfectMapping.csv**, das sogenannte"Gold Standard" Mapping
- **stopwords.txt**, ein Liste von allgemeinen Englischen Wörtern.

Die Dateien befinden sich im Test- Ressourcen-Ordnern des Projekts.

# Aufgabe 1: Entwurf von grundlegenden Funktionen zur Textanalyse und Entity Resolution

## Teil 0: Aufbereitung der erforderlichen Daten

Im ersten Teil, der vorgegeben ist, werden die Daten geladen und zur Verarbeitung vorbereitet.

**Das Fileformat von Amazon ist:**

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

**Das Fileformat von Google ist:**

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

**Das Fileformat des Goldstandards ist:**

“idAmazon”,”idGoogle”

**Das Fileformat der Stopwords ist:**

Stopword

In der Klasse Utils.scala befinden sich Methoden zum Auslesen der einzelnen Dateien.

## Teil 1: Analyse der Texte – Anwenden von “Bag of Words”-Techniken

Für diesen Teil der Aufgabenstellung werden die Dateien EntityResolution.scala sowie EntityResolutionTest (Unit Tests) benötigt. Die zu implementierenden Funktionen verteilen sich im File EntityResolution.scala auf die Klasse EntityResolution sowie das zugehörige Companion-Objekt. Die Aufteilung erfolgt, da die Klasse EntityResolution nicht von Serializable erbt und ohne das Companion-Objekt die Methoden nicht serialisiert werden könnten. Es ist verboten, die Klasse EntityResolution von Serializable erben zu lassen! Im ersten Teil der Aufgabenstellung werden die Fließtexte der Produkte so aufbereitet, dass sie für das Finden von Produktduplikaten verwendet werden können. Dazu werden im ersten Schritt die Texte Tokenisiert und alle informationslosen Wörter (die sogenannten Stopwords) entfernt.

Implementieren Sie zu diesem Zweck die Funktion ***tokenize*** und ***getTokens***. Die Funktion tokenize soll eine Produktbeschreibung (String) in einzelne Wörter aufsplitten und die Stopwords herausfiltern. Verwenden Sie für das Aufsplitten der Strings die Funktion tokenizeString aus der Klasse Utils. Ergebnis ist eine Liste von Wörtern, die keine Stopwords mehr enthalten. Diese Methode soll dann für getTokens verwendet werden, um aus den gesamten Produkt-RDDs die einzelnen Tokens herauszufiltern.

Zur Überprüfung der Funktionsfähigkeit der getTokens-Methode ist die countTokens zu implementieren. Sie soll alle sich im resultierenden RDD befindlichen Tokens zählen (Duplikate sollen dabei nicht eliminiert werden – “Bag of Words”). Implementieren Sie weiterhin die Methode findBiggestRecord, die den Datensatz mit den meisten Tokens finden soll.

## Teil 2: Implementierung von TF-IDF

Um beurteilen zu können, in wie weit die Produktbeschreibungen sich ähneln, muss die Relevanz der einzelnen Wörter innerhalb der Produktbeschreibungen ermittelt werden. Dies erfolgt mittels der Berechnung der TF-IDF-Werte. Diese setzen sich zusammen aus:

TF: Term Frequency: Relative Häufigkeit eines Wortes innnerhalb eines Dokuments

IDF: Inverse Document Frequency: Allgemeine Bedeutung des Terms für die Gesamtmenge der betrachteten Dokumente

Der TF-IDF-Wert lässt sich dann über die Multiplikation der Term Frequency mit der Inverse Document Frequency ermitteln.

**TF-IDF= TF * IDF**

Implementieren Sie die Funktion ***getTermFrequencies***. Die Funktion soll die einzelnen TF-Werte für ein Dokument (String) ermitteln.

Im nächsten Schritt müssen die IDF-Werte berechnet werden. Dazu wird der gesamte Korpus (alle Wörter der Dokumente) benötigt. Ermitteln Sie diesen in der Funktion ***createCorpus***, in dem Sie die beiden ProduktRDDs vereinigen.

Implementieren Sie die idf-Funktion, die für jedes Wort den entsprechenden idf-Wert ermittelt. Gehen Sie dabei folgendermaßen vor:

- Bestimmen Sie die Anzahl der Dokumente
- Ermitteln Sie die Menge aller Wörter (streichen der Duplikate)
- Ermitteln Sie für jedes Wort, in wie vielen Dokumenten es vorkommt
- Berechnen Sie für jedes Wort das Verhältnis zwischen der Anzahl der Vorkommen und der

Anzahl der Dokumente (idf-Wert)

- 

Um den Algorithmus zu vervollständigen ist noch die Funktion calculateTF_IDF erforderlich. Sie berechnet für eine Liste von Terms die jeweiligen TF-IDF Werte. Ergebnis ist ein Dokumentenvektor, der für jedes vorhandene Wort, den TF-IDF-Wert bereit hält (Sparse-Representation eines Document-Vectors).

## Teil 3: Bestimmung der Cosinus-Similarity

Der Abstand von zwei Dokumenten lässt sich über die Cosinus-Similarity bestimmen. Sie lässt sich über die folgende Formel bestimmen:

$$ab = \|a\|\|b\| \cos\theta$$

Der cosinus ist somit der Winkel zwischen den beiden Dokumentenvektoren.

Dabei ist a*b das normale dot-Produkt von a und b:

$$a*b = \Sigma a_i * b_i$$

und $\|a\|$ sowie $\|b\|$ die $L_2$ Norm des Vectors:

$$\|a\| = \text{sqrt}(\Sigma a_i^2) \text{ und } \|b\| = \text{sqrt}(\Sigma b_i^2)$$

Zur Berechnung der Cosinus-Similarität setzen Sie die Funktion ***calculateNorm*** um, die die L2-Norm für einen Vector berechnet, sowie die Funktion ***calculateDotProduct***, die ein Skalar-Produkt für zwei Vektoren bestimmt. Auf dieser Basis können Sie dann die Funktion ***calculateCosinusSimilarity*** umsetzen

## Teil 4: Anwenden der Funktionen auf die Dokumente

Führen Sie das Tokenisieren und die Berechnung in der Funktion c***alculateDocumentSimilarity*** zusammen. Sie bekommt als Paramter zwei Dokumente in Form von Strings sowie das Dictionary mit den TF-IDF-Werten und ermittelt die Cosinus-Similarity für die Dokumente.

Implementieren Sie danach die Funktion ***computeSimilarity***. Sie soll die Funktion calculateDocumentSimilarity aufrufen und die dafür erforderlichen Parameter aus der eigenen Parameterliste extrahieren.

Die Berechnung aller Document Similarities erfolgt in der Funktion ***simpleSimimilarityCalculation***. Auf Basis des AmazonRDD und GoogleRDD werden alle möglichen Produktkombinationen errechnet und dann für jede Kombination den Wert ermittelt. Zum Testen des Ergebnisses ist die Funktion ***findSimilarity*** erforderlich, die für zwei ProduktIDs den Wert ermittelt.

Die Funktion *simpleSimimilarityCalculation soll noch einmal unter der Verwendung einer Broadcast-Variable umgesetzt werden. Dafür sind die Funktionen* ***computeSimilarityWithBroadcast*** *und* ***simpleSimimilarityCalculationWithBroadcast*** zu implementieren.

Die letzte Funktion im ersten Teil ist ***evaluateModel***. Berechnen Sie hier auf Basis des Gold-

Standards folgende Elemente:

- Wie viele Produkt-Duplikate befinden sich im Sample,
- Was ist die durschnittliche Cosinus-Similarität der Duplikate und
- Was ist die durschnittliche Cosinus-Similarität der Nicht-Duplikate.

# Aufgabe 2: Anwenden der Funktionen auf den gesamten Datensatz

Für die Bearbeitung der zweiten Aufgabe sind die Dateien ScalableEntityResolution.scala und ScalableEntityResolutionTest.scala. Auch hier gibt es eine Trennung zwischen der Klasse ScalableEntityResolution und dem Objekt ScalabeEntityResolution, um die Serialisierung der Methoden gewährleisten zu können und nicht Gefahr zu laufen, die gesamte Klasse mit all ihren Membern in die Closures mit einzubeziehen. Es ist verboten, die Klasse ScalableEntityResolution von Serializable erben zu lassen.

Schreiben Sie als erstes die Funktion ***calculateTF_IDFBroadcast***. Sie soll dasselbe tun wie die calculateTF_IDF aus dem ersten Teil. Der einzige Unterschied soll sein, dass sie auf Broadcast-Variablen arbeitet.

Im nächsten Schritt soll mit den Funktionen ***buildInverseIndex*** und ***invert*** ein Inverser Index für die Google und die Amazon Tokens aufgebaut werden. Ein Inverser Index mapped die Wörter auf die Vorkommen, d.h. in diesem Fall soll im Ergebnis eine Menge vom Tupeln der Form (Wort, ProduktID) herauskommen. Nehmen Sie als Basis die Variablen amazonWeightsRDD und googleWeightsRDD und wandeln Sie diese mit der invert-Funktion (im object) um. Schreiben Sie die Ergebnisse in die Variablen amazonInvPairsRDD und googleInvPairsRDD. Cachen Sie die Werte.

In der Funktion ***determineCommonTokens*** sollen für alle Produktkombinationen die gemeinsamen Tokens bestimmt werden. Ergebnis ist ein RDD bestehend aus Tupeln der Form ((amazonID, googleID), Wörter). Sie können dafür die Funktion swap aus dem object verwenden. Speichern Sie das Ergebnis in die Variable commonTokens.

Berechnen Sie zum Schluss Similarity-Werte für alle Paare, die mindestens ein gemeinsames Token besitzen. Dies erfolgt in der Funktion ***calculateSimilaritiesFullDataset***. Verwenden Sie dazu die Variable commonTokens und wenden Sie darauf die noch zu implementierende Methode ***fastConsinusSimilarity*** an. Diese befindet sich im object und berechnet das dot-Produkt nur auf Basis der gemeinsamen Token. Weiterhin soll auch diese Funktion die Broadcast-Variablen verwenden.

Die Belegarbeit kann in Gruppen von 1-2 Personen bearbeitet werden und ist bis zum 27.1.2016 abzugeben. Die Funktionen sollen mindestens die mitgelieferten Tests bestehen.

Die Abnahme der Belegaufgabe erfolgt in der Übung.