# Übungsblatt: TFIDF Berechnung für Textdokumente

Software Architekturen, 6. Semester Bachelor: Wirtschaftsinformatik

FH CAMPUS 02
Informationstechnologien & Wirtschaftsinformatik

Grahsl Hans-Peter, Steyer Manfred

**1) Einführung**

TFIDF *(=> Term Frequency Inverse Document Frequency)* ist ein statistisches Maß, um die Relevanz einzelner Wörter in einer Sammlung von Dokumenten (=Korpus) zu bewerten.

TFIDF hilft u.a. die Frage zu beantworten, wie wichtig bzw. relevant z.B. ein Suchwort in Zusammenhang mit dem jeweiligen Dokument einer Trefferliste ist. Die Wichtigkeit respektive Relevanz eines Wortes steigt dabei proportional mit der Anzahl seiner Vorkommnisse innerhalb eines Dokument, wird aber im gleichen Zug je nach seiner Häufigkeit in der gesamten Dokumentmenge abgeschwächt.

Ein Anwendungsgebiet ist es, die TFIDF Metrik im Rahmen von Suchmaschinen Rankings als eine Komponente zur Bewertung der Relevanz (Ranking) von Treffern für Suchanfragen mit einzubeziehen. Dazu werden verschiedene Ausprägungen der TFIDF Metrik verwendet.

Details siehe z.B. http://en.wikipedia.org/wiki/Tf-idf

**2) TFIDF Formeln**

*a) Term Frequency:*

Term Frequency (Worthäufigkeit) vom Wort i im Dokument j = Anzahl der Vorkommnisse des Wortes i in Dokument j.

$$tf_{i,j} = n_{i,j}$$

Um die Worthäufigkeit sehr langer Dokumente auszugleichen erfolgt häufig eine Normalisierung in Bezug auf die gesamte Wortanzahl N des jeweiligen Dokuments j.

$$tf_{i,j} = \frac{n_{i,j}}{N_j}$$

*b) Inverse Document Frequency:*

Inverse Document Frequency (inverse Dokumenthäufigkeit) ergibt sich durch die Division der Anzahl aller Dokumente D mit der Anzahl jener Dokumente d, die das Wort i beinhalten. Von diesem Quotient wird dann noch der Logarithmus berechnet.

$$idf_i = \log \frac{|D|}{|\{d \in D : t_i \in d\}|}$$

*c) TFIDF:*

TFIDF von Wort i im Dokument j ist dann einfach das **Produkt aus a) und b) => tf * idf**

$$tfidf_{i,j} = tf_{i,j} \cdot idf_i$$

**3) Laden Sie das Hadoop Starter-Kit aus Moodle herunter**

Importieren Sie das StarterKit als existierendes Projekt direkt auf Basis des ZIP in Eclipse und machen Sie sich mit der Code Basis im **Package edu.campus02.iwi.hadoop.tfidf** vertraut.

**4) TFIDF Berechnung mit Hadoop Map Reduce**

Versuchen Sie die TFIDF-Berechnung für alle Wörter und alle Dokumente des Übungsdatensatzes (siehe data/input/tfidf/*.txt) mittels mehreren Map Reduce Schritten durchzuführen.

**Schritt 1:**

Schreiben Sie einen Map Reduce Job (in Anlehnung an das Word Count Beispiel), der zunächst die Häufigkeiten sämtlicher Wörter innerhalb des jeweiligen Dokuments berechnet.

Job Output: Wort \t DokumentID \t Frequenz
\t = TAB

TIPPS:

Der Map Output Key besteht in diesem Fall aus Wort + DokumentID, der Map Output Value ist die Frequenz.

- Als Key-Type verwenden Sie die im Package bereits vordefinierte Tuple-Klasse **TextTextTuple**.
- Als Value-Type können Sie direkt die in Hadoop verfügbare Klasse **LongWritable** verwenden.
- Als DokumentID dient der Dateiname, den Sie in der Mapper Instanz vom Context-Objekt abfragen können.
  String filename = ((FileSplit)context.getInputSplit()).getPath().getName();

**Schritt 2:**

Schreiben Sie einen weiteren Map Reduce Job der den Output aus Schritt 1 weiterverarbeitet. Dieser Job übernimmt das Abzählen aller Wörter eines Dokuments. Dazu wird die Mapper Instanz als Key die DokumentID und den zusammengesetzten Value Wort+Häufigkeit ausgeben. Die Reducer Instanz erhält somit die Liste mit allen eindeutigen Wörtern samt Häufigkeiten zu einer DokumentID. Diese Häufigkeiten werden aufsummiert, womit sich gesamte Wortanzahl zur DokumentID ergibt.

Job Output: Wort \t DokumentID \t Frequenz \t AnzahlWörter
\t = TAB

TIPPS:

Der Map Output Key ist die DokumentID, der Map Output Value ist das Wort samt Häufigkeit:

- Als Key-Type können Sie direkt die in Hadoop verfügbare Klasse **Text** verwenden
- Als Value-Type verwenden Sie die im Package bereits vordefinierte Tuple-Klasse **TextLongTuple** um das Wort samt Frequenz zu kapseln.

Der Reduce Output Key ist die Kombination aus DokumentID + Wort, der Reduce Output Value ist die Kombination aus Frequenz + gesamter Wortanzahl:

- Als Key-Type verwenden Sie die im Package bereits vordefinierte Tuple-Klasse **TextTextTuple**.
- Als Value-Type verwenden Sie die im Package bereits vordefinierte Tuple-Klasse **LongLongTuple**.

*In der Reducer Instanz müssten Sie 2x durch die Value-List iterieren, was leider nicht möglich ist. Daher ist es nötig, die Value-Liste per Key zu cachen. Erstellen Sie dazu in der reduce Methode einfach eine ArrayList<TextLongTuple> und cachen Sie während der Berechnung der Frequenzsumme alle Value-List Einträge des aktuellen Keys. Danach geben Sie alle Key-Value Paare samt gesamter Wortanzahl basierend auf den gecachten Einträgen aus.*

**Schritt 3:**

Schreiben Sie einen letzten Map Reduce Job der basierend auf dem Output von Schritt 2 alle finalen TFIDF-Werte berechnet. Dazu wird die Mapper Instanz als Key das Wort und als Value die gesamte restliche Information (der Einfachheit halber als TAB separierten String) ausgeben. Die Reducer Instanz erhält somit pro Wort die Value-Liste mit den allen benötigten Informationen zur TFIDF Berechnung bezogen auf das Dokument.

Sie haben die folgenden Informationen berechnet (siehe Schritten 1 & 2):

- die Frequenz des Wortes pro Dokument
- die gesamte Wortanzahl des Dokuments

Es „fehlt“ also noch:

- die Anzahl der Dokumente in denen das entsprechende Wort vorkommt: diese errechnen Sie beim 1. Durchlauf durch die Value-List pro Key (=Wort) im Reducer
- die gesamte Dokumentanzahl: diese können Sie der Einfachheit halber *hardcoded* im Reducer hinterlegen

Job Output: Wort \t DokumentID \t tfidfWert
\t = TAB

TIPPS:

Der Map Output Key ist das Wort, der Map Output Value ist die gesamte unveränderte Information zur Berechnung der TFIDF Werte.

- Als Key-Type und Value-Type können Sie direkt die in Hadoop verfügbare Klasse **Text** verwenden.

Der Reduce Output Key ist die Kombination aus Wort+DokumentID, der Reduce Output Value ist der endgültige Double-Wert:

- Als Key-Type verwenden Sie die im Package bereits vordefinierte Tuple-Klasse **TextTextTuple**.
- Als Value-Type können Sie direkt die in Hadoop verfügbare Klasse **DoubleWritable** verwenden.

Auch hier müssen Sie analog zu Schritt 2 die Value-List in der reduce Methode des Reducers mittels ArrayList cachen, damit Sie die Möglichkeit haben die Values 2x zu durchlaufen.