

Der Algorithmus von Luhn und $tf*idf$

Inhalt

- * Wiederholung des Algorithmus von Luhn
- * $tf \cdot idf$
- * Modifizierter Algorithmus von Luhn
- * Implementierung

Algorithmus von Luhn

- * Auswahl der Sätze, die den Text am besten repräsentieren
- * Ansatz: Beste Repräsentanten sind die Sätze, welche die meisten für den Text relevanten Terme enthalten; also: Auswahl der relevanten Terme
- * Dann: Berechnen eines Maßes, das die Relevanz eines Satzes in Abhängigkeit von den darin enthaltenen Termen ausdrückt

Relevante Terme

- * Ansätze zur Filterung:
 - * Stopwords: Sprachabhängig, müssen weitestgehend manuell erstellt und gepflegt werden
 - * Anhand statistischer Eigenschaften: Rang der Termfrequenz, häufigste (Stopwords) und seltenste Worte werden entfernt, diese Variante nutzt Zipf-Verteilung

Relevanz von Sätzen

- * Idee: Segmente, die zwischen zwei relevanten Termen liegen; relevante Terme dürfen höchstens durch 4 bis 5 nicht-relevante Terme getrennt sein
- * Relevanzberechnung: $R_s = |T_r|^2 / |T|$ mit $R_s =$ Relevanz eines Satzes, T_r , T Mengen der relevanten bzw. aller Terme in einem Segment, bei mehreren Segmenten zählt der höchste Wert (Anmerkung: Die Anzahl der Segmente bietet sich eigentlich auch als Parameter an.)

Der Ursprüngliche Algorithmus

- * Zerlege Text in Sätze
- * Zerlege Sätze in Token
- * Finde und filtere Types

Der Ursprüngliche Algorithmus

- * Für jeden Satz: Starte das aktuelle Segment bei Index 0
- * Für jedes Token in einem Satz: Falls das Token zu den relevanten Termen gehört, füge es zur Liste der relevanten Terme für diesen Satz hinzu

Der Ursprüngliche Algorithmus

- * Berechne Relevanz des aktuellen Segments anhand der genannten Formel
- * Falls die Differenz zwischen Laufindex und dem Index des letzten relevanten Terms > 4 : Beginne neues Segment
- * Bei mehreren möglichen Segmenten: Wähle das mit der höchsten Relevanz
- * Abstract: Alle Sätze über Schwellenwert

Der Ursprüngliche Algorithmus

* Abstraktes Beispiel:

* (_ _ [* _ _ _ * _ _ * *] _ _ _ _ _ [* _ *] _ _ _ _ _)

* *: Relevant

* _: Nicht relevant

tf*idf

- * Gewichtete Termfrequenz
- * $tf = \text{Frequenz des Terms im Dokument} / \text{Summe der Frequenzen aller Terme im Dokument}$
- * $idf = \text{Anzahl der Dokumente im Korpus} / \text{Anzahl der Dokumente im Korpus, die den gesuchten Term enthalten (vgl. Inverted Index)}$
- * $tf*idf = tf * \log(idf)$

Da fehlt doch was?

- * Richtig, für $tf*idf$ brauchen wir ein Korpus
- * Ansatz
 - * Nutze eine Suchmaschine, um für das Thema des Ursprungsdokuments relevante Dokumente zu finden
 - * Verwende diese Dokumente zur Erstellung eines $tf*idf$ Modells für genau dieses eine Ursprungsdokument

Der Modifizierte Algorithmus

- * die ersten sechs ursprünglich relevante Terme werden als Suchterme genutzt, um ein individuelles Korpus zusammenzustellen
- * dieses Korpus dient zur Erstellung eines $tf*idf$ Modells
- * durch $tf*idf$ Gewichtung ergibt sich eine Verschiebung bei den relevanten Termen

Implementierung

- * Grails
- * Yahoo Search API
- * Wikipedia
- * NekoHTML

Zusammenfassung

- * Auswahl der relevanten Terme bedurfte einiges an Fine-Tuning und war teilweise abhängig vom Dokumenttyp
- * Alternativer Ansatz: Auswahl relevanter Terme mittels $tf \cdot idf$ Gewichtung
- * $tf \cdot idf$ Gewichtung zeigt deutliche Effekte

FRAGEN?