1. Implementierung ausgewählter Ansätze

Dieses Kapitel befasst sich mit der Implementierung von zwei ausgewählten Ansätze zur Dokumentähnlichkeitsbestimmung, nämlich BayesLSHLite. Diese Ansätze, deren Pseudocode schon im vorherigen Kapitel dargestellt wurden, werden in der Programmiersprache R implementiert.

Die Firma dibuco GmbH entwickelt normalerweise Softwarelösungen in den Programmiersprachen Java oder Scala aber die verwendete Programmiersprache für diese Thesis ist R, und die integrated development environment (IDE) dazu ist Rstudio. Ich habe mich für die Programmiersprache R entschieden, da R Statistik mit Datenvisualisierung kombiniert und dies optimal zum zentralen Thema dieser Arbeit Bayessche Statistik, passt. Dies hat unter anderem unterschiedliche Gründe. Es bestehen schon in R Bibliotheken und vorgefertigte Algorithmen, die zur Anwendung bereitsteht. R kann unter anderem sowohl mit Spark (Big Data Framework für Analytik) als auch mit Konstanz Information Miner Analytik Plattform (KNIME) betrieben werden und fast alle Datenbanken können von der ausgelesen werden. Das Textreuse Paket, das später benutzt wird für die Implementierung des Ansatzes kommt vom Massachusetts Institute of Technology (MIT). Hinter R steht eine sehr große, weltweite Community, welche die Pakete frei zur Verfügung stellt, weiterentwickelt und pflegt. Ein weiterer Vorteil von R wie schon im Abschnitt (1.1) erwähnt, ist dass, sie eine Open Source ist. Außerdem viele wissenschaftliche Arbeiten können in R erstellt werden.

* 1. Die Algorithmen

Der für diesen Implementierungsteil angewendete Algorithmus ist der BayesLSH-Lite Algorithmus und das paarweise Ähnlichkeitsalgorithmus. Der BayesLSH-Lite Pseudocode beschrieben wurde im Kapitel 2. Es ist hinzuweisen, dass diesen Algorithmus als Leitfaden für die Implementierung des Ansatzes genommen wird, bzw. für das Schreiben des Codes. Weitere Details werden im Abschnitt (Code noch hinzufügen) erläutert.

* 1. Wichtige Pakete

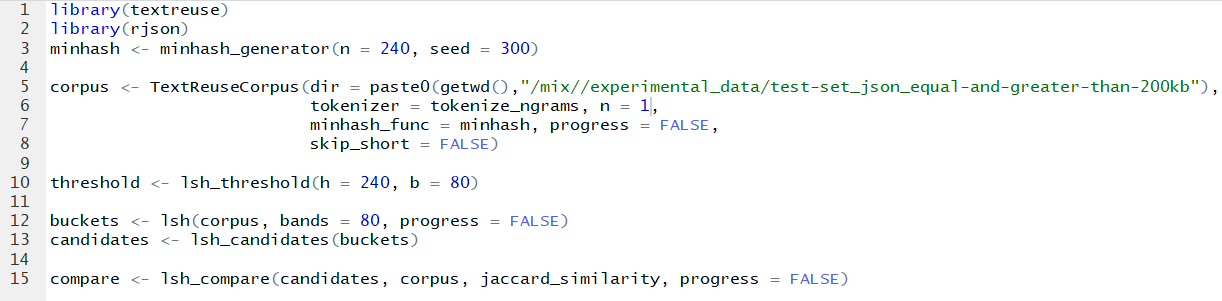
Wichtige Pakete für die Implementierung dieser Arbeit sind die Pakete textreuse und rjson

* + 1. Das Paket textreuse

Das Paket textreuse ist verfügbar auf die R Plattform und enthält eine Menge von Funktionen, zur Ähnlichkeitsmessung zwischen Dokumenten sowie Erkennung von wiederverwendeten Textpassage in Dokumente. Das Paket textreuse implementiert Minhash, Locality Sensitive Hashing Funktionen sowie Ähnlichkeitsfunktionen.

* + 1. Der Code

Der Code dieser Implementierung wurde in R geschrieben und anhand vom existierenden Paket textreuse durchgeführt. Zusätzlich zu dem Paket textreuse wurde auch das Paket rjson zum Auslesen von JSON Dokumente in diesem Code angewendet wie im Abbildung 1 gezeigt wird.

  
Abbildung 1: Der Code zur Implementierung des BayesLSH mit dem Jaccard Index

**Die Funktion minhash** (Noch zu definieren in Grundlagen Abschnitt mit BayesLSH): diese implementiert die Funktion minhash\_generator () zur Umwandlung von einem Text, der in Token übersetzt wurde, in Hash. Hier ist hinzuweisen, dass die Token zufällig ausgewählt werden. Ein Token entspricht eine Folge alphabetischer und/oder numerischer Zeichen, mit Trennung links und rechts durch Textformatierung und/oder Interpunktionszeichen [Yatsko2014]. Durch Hashing von String in Zeichenvektor auf ganze Zahlen und Auswahl der Minimalwert erfolgt die Berechnung eines minhashes. Unter Hashing wird die Umwandlung eines beliebig großen Datensatz in eine Zeichenkette mit einer festen, kürzeren Länge, die den ursprünglichen Datensatz referenziert, verstanden [TEC18]. Die Definition von n Anzahl an Hashes, die von zufällige selektierte Token zu gehackerte Token werden, ermöglicht es der Funktion bis zu n Hashes zu produzieren. Durch den Parameter seed kann die mehrmalige Generierung von minhashes durch eine gleichen minhash Funktion sichergestellt werden. Diese Funktion wird dann später als Parameter hash\_func in der Funktion TextReuseCorpus () angewendet.

**Die Funktion Corpus:** diese Funktion implementiert die Funktion TextReuseCorpus () und enthält eine Liste von Objekten der Klasse TextReuseDocument (). Die Klasse TextReuseDocument () enthält einen Dokumenttext und ihre Metadaten. Die Funktion Corpus ist anhand eines Dateienordners erstellt. Hier in diesem Code durch den Parameter dir wird die Zuweisung zu einem Dateienordner gemacht. Diese Funktion verhält sich so, dass sobald ein Dokument oder eine Datei geladen wird, der Text davon wird in Token umgewandelt. Das Ganze wird durch den Parameter tokenizer eingestellt für n-gramm. N-gramm entspricht eine Sequenz, die geordnet ist, von n in einem Textkörper extrahierte Wörter. Weiterhin wenn es Dokumente bestehen, die klein sind, wird die Funktion Corpus diese einfach trotzdem berücksichtigen durch den Parameter skip\_short.

**Die Funktion threshold:** implementiert die Funktion lsh\_threshold, die dabei hilft bei der Bestimmung der minimalen Jaccard Ähnlichkeit um zwei Dokument als matches anzunehmen. Unter dem Begriff match wird Übereinstimmung verstanden. Das Threshold ist durch die Folgende Formel gegeben [Leskovec2014]:

r ist die Anzahl an Spalten und wird durch die Formeln

gegeben. Wobei h und b jeweils die Anzahl an minhash-Signaturen und die Anzahl an LSH-Bänder entsprechen.

**Die Funktion buckets:** implementiert die Funktion lsh (), deren Aufgabe die Entdeckung von möglichen Matches eines gegeben corpus von Dokumenten (Sammlung von Dokumenten von der Funktion TextReuseCorpus) zum schnellen Vergleich von nur ähnliche Dokumentenpaaren dieser Sammlung, ist. Dafür werden jede Hash-Signaturen in Bänder geteilt, die eine bestimmte Anzahl an Zeile haben und es erfolgt das Hashing von diesen Bändern. Die Dokumente werden so aussortiert, dass wenn die gleichen Zeilen haben, dann wird das Hashing von beiden in den gleichen Bucket durchgeführt. Buckets können als Cache bezeichnet werden. Diese Funktion ist versehen um später in der Funktion lsh\_candidates () angewendet zu werden um potentialen Kandidatenpaare zu berechnen.

**Die Funktion Candidates:** implementiert die Funktion lsh\_candidates () und aus den Buckets der Funktion LSH () mögliche Kandidatenpaaren zurückgibt. Diese Funktion wird in der

**Die Funktion compare:** implementiert die Funktion lsh\_compare (), deren Aufgabe die Kandidatenpaaren zu Vergleichen und die Entsprechende Jaccard Ähnlichkeit zu bestimmen und auszugeben

[RPR18] <https://www.r-project.org/> (Letzter Abruf:21.06.2018)

[GNU18] <http://www.gnu.org/philosophy/free-sw> (Letzter Abruf:21.06.2018)

[HPS18] HP Support Assistant, integriertes Service in HP Pavilion Notebook

[RDO18] <https://www.rdocumentation.org/packages/textreuse/versions/0.1.4> (Letzter Abruf:24.06.2018)

[RPT18] <https://cran.r-project.org/web/packages/textreuse/textreuse.pdf>

[RMI18] <https://www.rdocumentation.org/packages/textreuse/versions/0.1.4/topics/minhash_generator>

[TEC18] <https://searchsqlserver.techtarget.com/definition/hashing>

[Yatsko2014] Computational linguistics or linguistic informatics?, <https://link.springer.com/article/10.3103/S0005105514030042>, Springer Verlag (schon im BIBtex hinzugefügt)

[RYT18] https://de.ryte.com/wiki/Hashing(Letzter Abruf: 26.06.2018)

[SCH17] Drew Schmidt, Christian Heckendorf, Guide to the ngram Package, November 2017

[Leskovec2014] Leskovec, Jure and Rajaraman, Anand and Ullman, Jeffrey DavidMining of Massive Datasets (in Bibtex schon hinzugefügt)

Abkürzung

IDE integrated development environment

KNIME Konstanz Information Miner

MIT Massachusetts Institute of Technology