Hochschule Karlsruhe
Fakultät für Informatik und Wirschaftsinformatik
Sommersemester 2017
Veranstaltung: Big Data & Advanced Database Concepts
Dozenten: Prof. Dr. Andreas Schmidt
Dr. Jannik Strötgen

Seminararbeit

Big Data - Aufgabenblatt 1

Name:	Jean-Luc Burot, Jean-Pierre Bourhis
Matrikelnummer:	60214 / 55921
Studiengang:	Wirtschaftsinformatik (1. Fachsemester)
Email:	jburot@gmail.com / jp.bourhis@gmail.com
Datum der Abgabe:	1. April 2017

Hiermit versichere ich, **Jean-Luc Burot, Jean-Pierre Bourhis**, dass ich die Hausarbeit mit dem Titel **Big Data - Aufgabenblatt 1** im Seminar **Big Data & Advanced Database Concepts** im **Sommersemester 2017** bei **Prof. Dr. Andreas Schmidt** und **Dr. Jannik Strötgen** selbstständig und nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Zitate sowie der Gebrauch fremder Quellen, Texte und Hilfsmittel habe ich nach den Regeln wissenschaftlicher Praxis eindeutig gekennzeichnet. Mir ist bewusst, dass ich fremde Texte und Textpassagen nicht als meine eigenen ausgeben darf und dass ein Verstoß gegen diese Grundregel des wissenschaftlichen Arbeitens als Täuschungs- und Betrugsversuch gilt, der entsprechende Konsequenzen nach sich zieht. Diese bestehen in der Bewertung der Prüfungsleistung mit „nicht ausreichend“ (5,0) sowie ggf. weiteren Maßnahmen.
Außerdem bestätige ich, dass diese Arbeit in gleicher oder ähnlicher Form noch in keinem anderen Seminar vorgelegt wurde.

Heidelberg, den 1. April 2017 _____

Inhaltsverzeichnis

1 Problem 1

1.1 a)

Probleme mit Binary Matrix als Index

Größe

Eine Binary Matrix kann sehr schnell sehr groß werden, je nach dem was analysiert wird. Die Spalten der Matrix können Dokumente oder auch Zeilen darstellen (document identifier) und je nach Umfang eine große Menge an Daten produzieren.

Angenommen es gäbe 50.000 Zeilen und 20.000 einmalige Wörter, dann hätte die Matrix folgende Anzahl an Zellen:
50.000 x 20.000 = 1.000.000.000

Wenn nun jede Zelle mit entweder einer binären 0 oder 1 belegt wird, dann hat die Matrix folgende Größe:
(1 Bit x 1.000.000.000 Zellen) / 8 Bit = 125.000.000 Bytes = 122.070 KB = 119 MB

Die Größe der resultierenden Matrix hat zur Folge, dass weitere Analysen speicherintensiv und rechenlastig werden können, was wiederum zu einer langsamen Berechnung eines Ergebnisses führt.

Homonyme

Ein weiteres Problem entsteht mit der fehlenden Unterscheidung zwischen Homonymen, da in einer einfachen Analyse nur auf das reine Vorkommen einer Buchstabensequenz geachtet wird.

Wörter wie “can” können interpretiert werden als “können” oder “Dose”. Gleiches gilt für “match”, was als “Streichholz”, “Spiel”, “Wettkampf” oder “übereinstimmen” interpretiert werden kann.

Ranking

Je nach Anwendungsfall kann ein Ranking der Wörter nötig werden. Diese Information fehlt in der Binary Matrix. Hier wird nur das Vorhandensein eines Wortes in einem Dokument bzw. einer Zeile notiert. Es fehlt allerdings

die Anzahl der Vorkommen innerhalb eines Dokumentes bzw. einer Zeile. Ebenfalls fehlt die Anzahl der Vorkommen über alle Dokumente bzw. Zeilen. Damit fällt die Möglichkeit aus, in einem zweiten Schritt eine Gewichtung aller Wörter zu berechnen.

Ein Ranking kann besonders dann interessant werden, wenn es um häufig vorkommende Wörter geht. Wörter wie “is”, “a” und “the” kommen im Englischen sehr häufig vor, tragen u.U. aber zum Kontext nur wenig bei.

Lösungsansätze

Eine Lösung zur Verringerung der Größe des resultierenden Index ist der Inverted Index. Das Resultat dieses Verfahrens entspricht einer Liste der enthaltenen Wörter und der Sammlung der Indexe, bei welchen sie zu finden sind. Wie groß die resultierende Datei wird, hängt letztlich von der Anzahl der Wörter und der Quantität ihrer Vorkommen in unterschiedlichen Dokumenten ab.

Beim Inverted Index muss zunächst von allen Dokumenten bzw. Zeilen der Text bereinigt werden von Zeichen wie Klammern, Nummern, Ausrufezeichen, Fragezeichen, Komma, usw.. Es sollten nur noch Wörter übrigbleiben, die von Leerzeichen getrennt sind. Der nächste Schritt besteht aus dem Leerzeichen-Split des Textes, um daraus eine Liste von Wörtern zu erstellen. Doppelt vorkommende Wörter müssen herausgefiltert werden.

Je nach Anwendungsfall kann nun ein Verfahren namens Normalizing angewendet werden, in welchem Stemming, Stopword Removal, Tokenization und Case Folding angewandt wird. Beim Stemming werden Wörter mit ihren Wurzelwörtern ausgetauscht, also generalisiert. Aus den Wörtern “walking”, “walked” und “walks” wird “walk”. Beim Stopword Removal werden häufig vorkommende Wörter, die dem Kontext wenig bis kein Zusatznutzen geben entfernt. Solche Wörter sind “the”, “to”, “of”, “and”, usw.. Bei der Tokenization werden Wörter vereint, die spezielle Schreibweisen haben. Beispiele sind “email” oder “e-mail”, “its” oder “it’s”. Beim Case Folding werden Wörter als gleich betrachtet, die in Kapitälchen oder aus einer Mischung von Kapitälchen bestehen. Solche wären “the”, “The”, “THE” und “tHE”.

Das Resultat als Inverted Index besteht aus der Zusammenstellung des Index, also aus einer Kombination eines Wortes und einer Reihe von Indexen, in welchem das Wort vorkommt.

1.2 b)

Um die Nähe zweier Wörter zu bestimmen, sind zwei Varianten zu unterscheiden. Die eine heißt NEAR und betrachtet nur die Nähe zweier Wörter zueinander. Die zweite heißt FBY und steht für followed by, beinhaltet also neben der Nähe noch die Reihenfolge der Wörter. In beiden Fällen misst sich die Entfernung in der Anzahl an Wörtern.

Um die Entfernung zweier Wörter messen zu können, muss der Index nicht nur das Dokument bzw. die Zeile hergeben, sondern auch die Wortposition innerhalb der Zeile. Dazu müssten alle gefundenen Wörter der Reihe nach durchgezählt werden. Die bisherige numerische Information des Dokument- bzw. Zeilen-Index müsste erweitert werden auf ein Informationspärchen aus Dokument- bzw. Zeilen-Index plus Positions-Index.

2 Problem 2

Alle Python-Dateien wurden auf Windows 10 und Ubuntu 16.10 mit Python3 getestet. Die Fehler sollten größtenteils beseitigt worden sein. Werden die Python-Dateien ohne Argumente aufgerufen, dann wird der User darauf hingewiesen. Bei (a) und (b) wird zu Beginn der Index erstellt und dann abgespeichert. Bei der (b) kommt der User als nächstes in eine Endlosschleife, in welcher er zwei Wörter für die Intersection-Berechnung eingeben kann. Gibt der User nicht existierende Wörter ein, dann wird er darauf hingewiesen. Gibt er nichts ein, so verlässt er die Anwendung.

Bei der (c) liefert das Ergebnis maximal drei relevante Zeilen unter Verwendung des folgenden Algorithmus. Zuerst wird die betreffende Zeile gesucht, um daraufhin mit den beiden Wörtern die folgenden Fälle abzudecken:

1. Wenn beide Wörter in derselben Zeile stehen:
   (a) Zuerst erhält man die Zeilen, die beide Wörter enthalten,
   (b) dann die Zeilen ohne a), die das erste Wort enthalten,
   (c) dann die Zeilen ohne a), die das zweite Wort enthalten.
2. Wenn keines der beiden Wörte in der gleichen Zeile stehen:
   (a) Liefert die Zeilen, die das erste Wort enthalten.
   (b) Liefert die Zeilen, die das zweite Wort enthalten.
3. Wenn nur eines der beiden Wörter im ganzen Dokument steht:
   (a) Liefert die Zeilen, die das Wort enthalten.
4. Wenn mit beiden Wörtern nicht gefunden wurde:
   (a) Liefert eine Nachricht zurück.
5. Wenn kein Wort eingegeben wurde:
   (a) Liefert eine Nachricht zurück.
6. Wenn mehr als zwei Wörter eingegeben wurden:
   (a) Liefert eine Nachricht zurück.

Entsprechend dem Punkt (d) wurden alle Python-Dateien auch mit der "documents.txt" erfolgreich getestet.

Es sei allerdings darauf hingewiesen, dass durch die Art der Ausführung bei Python die Performance im Vergleich zu z.B. Java eher niedrig ist. Vereinzelte Tests in Java schafften in weniger als 5min einiges, wofür Python mehr als 30min brauchte und teilweise vorher abgebrochen wurde. Mangels Zeit beschränkten wir uns innerhalb dieser Aufgabenstellung auf die Erstellung eines Python-Clients.