Lösungsskizze

Seltene gemeinsame Wörter – semantisch ähnliche Sätze

1 Projektbeschreibung

Das Ziel des Praktikums war es ein Programm zu entwickeln, welches ähnliche Sätze in einem Korpus identifiziert. Hierbei wurden zwei Ansätze verfolgt. Beim ersten Ansatz wurde davon ausgegangen, dass die Ähnlichkeit durch die Anzahl gemeinsamer seltene Wörter beschrieben wird. Beim zweiten Ansatz hingegen wurden die 5 seltensten Wörter jedes Satzes untersucht und miteinander verglichen. Hierbei wurden zum einen mehr ähnliche Sätze gefunden, zum anderen wurde die zu verarbeitende Datenmenge vergrößert wodurch, was einen größeren Rechenaufwand erforderte.

Ein besonderes Augenmerk der Praktikumsaufgabe lag auf der Verarbeitung von großen Korpora mit bis zu 100 Millionen Sätzen. Deshalb war eine hocheffiziente Programmierung notwendig. Zusätzlich zu den deutschen Korpora wurden noch englische Korpora verwendet und die Korrektheit des Programms getestet.

2 Lösungsansatz

Zur Lösung des Problems wurde der Algorithmus *Satzähnlichkeit fein* aus der Vorlesung Fortgeschrittene Methoden des Information Retrieval verwendet. Der Algorithmus soll in folgenden Schritten ablaufen.

1. Inverse Liste der Quelldatenbank exportieren
2. Für jede Satz-ID eine Liste erstellen, die entsprechend zugehörige Wort-IDs enthalten
3. Für jede gesammelte Wort-ID aus dem vorherigen Schritt, eine Liste von Satz-IDs erzeugen, die diese Wort-ID gemeinsam enthalten
4. Liste der Satzpaare ermitteln, die gemeinsame Wörter haben und sortieren
5. Doppelte Satzpaare zählen und bei Überschreitung einer vorher festgelegten Schwelle persistieren

Hierbei wurden einmal die häufigsten 87.839 Wörter ignoriert, um nur die seltensten zu betrachten. Beim zweiten Ansatz wurden für jeden Satz die häufigsten 5 Wörter ausgewählt. Dadurch das die Wörter in der Datenbank nach Häufigkeit sortiert vorliegen, handelte es sich hierbei um die fünf Wörter mit der höchsten ID.

Der Vergleich der Sätze lief bei beiden Ansätzen gleich ab. Zuerst wurde aus jedem Satz ein Vektor erzeugt, der die IDs der in ihm vorkommenden Wörter enthält. Diese Vektoren wurden in einer Datei gespeichert. Aus diesen Satzvektoren wurden wiederum Wortvektoren gebildet. Die Wortvektoren wurden ebenfalls in einer Datei zwischengespeichert. Ein Wortvektor enthält die IDs aller Sätze in denen das Wort vorkommt. Anschließend wurden aus den Wortvektoren Satzpaare gebildet. Ein Satzpaar bedeutet, dass diese beiden Sätze, ein gemeinsames Wort besitzen. Die Satzpaare wurden in eine Datei geschrieben und sortiert. Nun werden die Paare gezählt, bei X Paaren, das heißt X gemeinsame Wörter, wird davon ausgegangen das es sich um ähnliche Sätze handelt. Die so gefundenen Satzpaare werden in eine Textdatei geschrieben und gespeichert.

Anfangs wurde versucht alle Schritte im RAM auszuführen. Das beschleunigte zwar den Algorithmus enorm, allerdings wird bei größeren Datenmengen schnell die Maximale Kapazität des RAMs erreicht und das Skript bricht ab.

3 Einrichtung und Ausführung

Damit das Skript reibungslos laufen kann, muss ein Linuxbetriebssystem mit einem installiertem Python-2.7-Paket vorhanden sein. Die Dateien `run.sh` und `main.py` gemeinsam in einen beliebigen Ordner (z.B. das Home-Verzeichnis des aktuellen Benutzers) ablegen. Für den Export der inversen Liste werden Zugangsdaten für die MySQL-Zieldatenbank benötigt. Mit einem Texteditor `main.py` öffnen und in die Zeilen

```
# DB connection information
# DB host
host = 'localhost'
# DB user
user = 'dbuser'
# DB password
passwd = 'password'
# Name of used DB
dbName = 'deu_mixed_2011'
```

die Daten der benutzten Datenbankverbindung eintragen. Optional können die Konfigurationsvariablen verändert werden.

`wordIdBoundary` Jede Wort-ID, die kleiner oder gleich diesem Wert ist, wird ignoriert. Die Höhe wirkt wirkt sich auf die Endqualität und auf die Gesamtlaufzeit aus. Je höher dieser Wert ist, um so mehr potentielle Kandidaten werden ignoriert, aber die Laufzeit wird verringert.

`countOfWords` Die Anzahl der seltensten Wort-IDs pro Satz, die zum Vergleich benutzt werden. Die Sätze, die weniger Wörter besitzen als dieser Wert, werden ignoriert. Je höher dieser Wert ist, um so größer ist die Qualität der Ergebnisse, aber die Gesamtlaufzeit erhöht sich, da die Anzahl der Vergleiche steigt.

`countOfSentences` Die Anzahl der Sätze, die verarbeitet werden sollen. -1 zeigt an, dass alle Sätze in der Exportdatei verarbeitet werden.

Die Datei speichern und den Editor schließen. Ein Terminal öffnen und zum Verzeichnis navigieren in dem `run.sh` liegt. Das Skript wird mit dem Kommando `./run.sh` gestartet.

Nachdem das Skript erfolgreich beendet ist, liegen in dem selben Verzeichnis drei neue Textdateien. Die Exportdatei (benannt nach dem Namen der Datenbank) beinhaltet alle Satz-ID/Wort-ID-Paare aus der inversen Liste. Wenn ein neuer Export erstellt werden soll, muss diese Datei gelöscht werden. In der Datei `sorted-counted-pairs` liegen aufsteigend sortiert die gezählten Satzpaare im Format

```
AnzahlGemeinsamerWoerter Satz-ID1 Satz-ID2
```

. In der Datei `sentences.txt` sind die Satzpaare als Klartext zur Auswertung aufgelistet.

4 Implementierung

Das Programm wurde für Linuxsysteme konzipiert und wurde in einem Python- und Shellskript implementiert. Dieses Kapitel beschreibt die Implementierung der einzelnen Schritte, die in Kapitel 2 erläutert wurden. Die Herausforderung war die Implementation möglichst unabhängig von der Größe des Arbeitsspeichers durchzuführen.

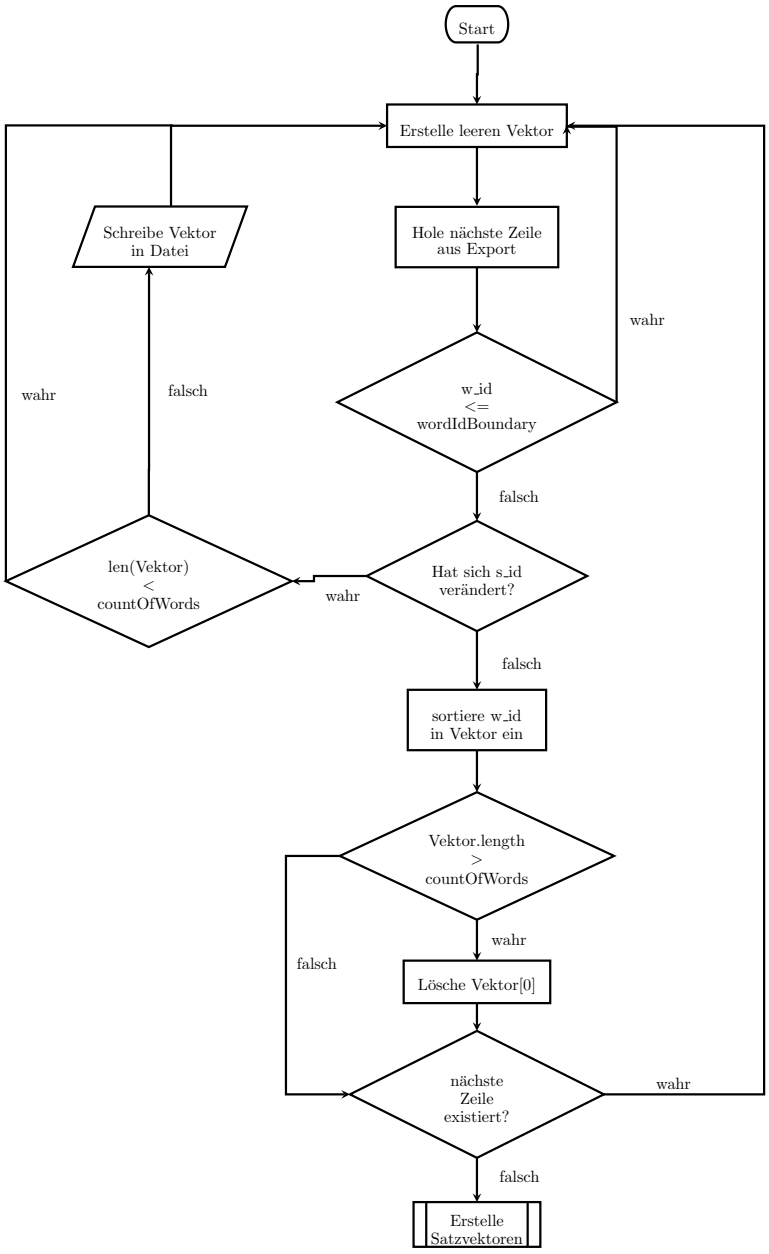
4.1 Export der inversen Liste

Für den Export wird eine SQL-Anfrage

```
SELECT s_id, w_id FROM inv_w;
```

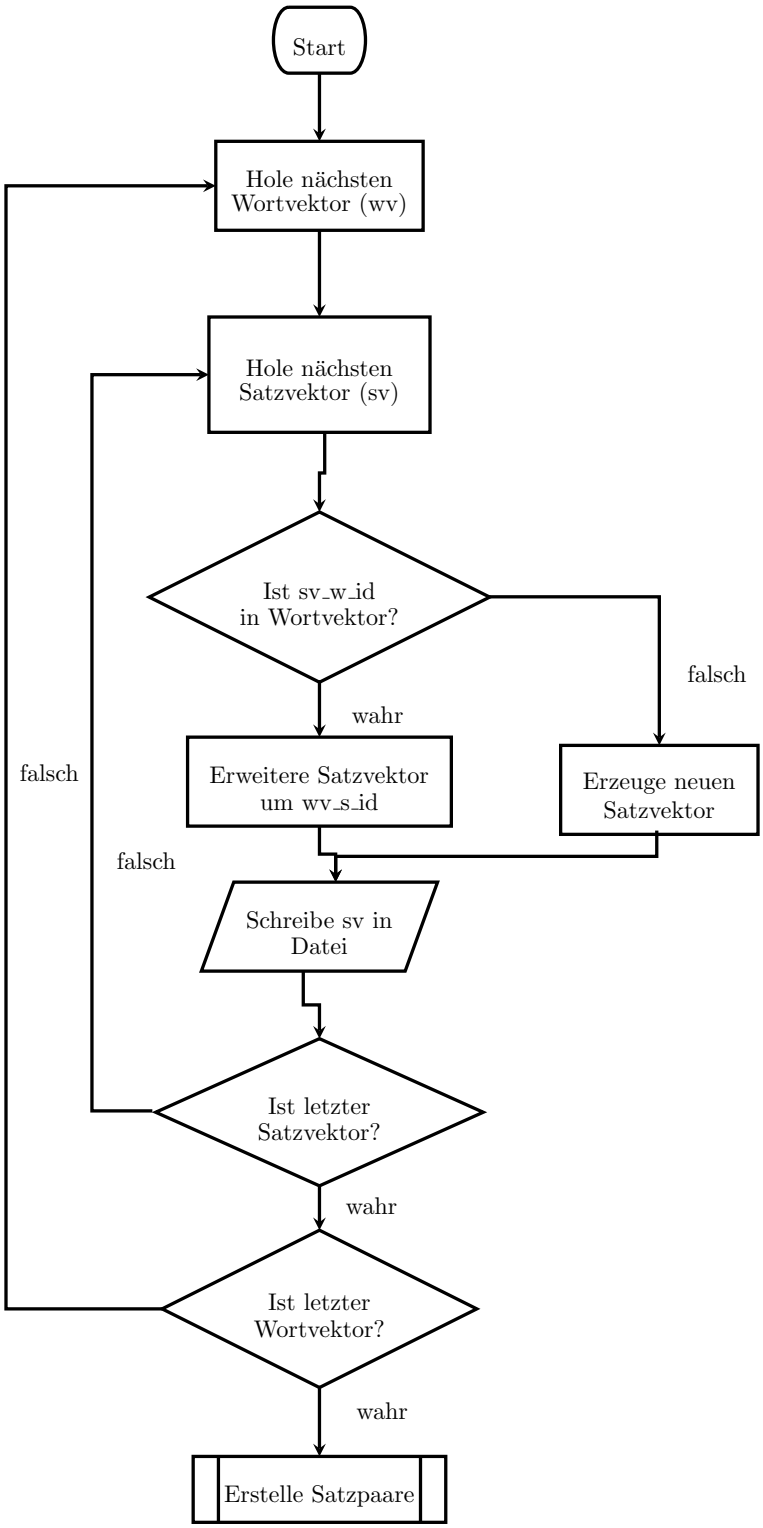
an die Datenbank geschickt. Das Ergebnis wird sortiert im Format `(s_id,w_id)` in eine Ausgabedatei geschrieben. Zur besseren Differenzierung der Exportdateien ist der Name der Ausgabedatei der selbe wie der Name der Datenbank.

4.2 Erstellung der Wortvektoren



Ein Wortvektor ist eine Liste von Wort-IDs, die in Relation zu einer Satz-ID stehen. Da im vorhergehenden Schritt die Exportdatei sortiert wird, kann in einem Arbeitsschritt sukzessiv jedes Satz-Wort-ID-Paar mit gleicher Satz-ID ausgelesen und formatiert werden. Nachdem der Erstellung eines Wortvektors speichert das Programm diesen im Format {`'key': s_id, 'items':[w_id_1, ..., w_id_n]`} in einer temporäre Auslagerungsdatei ab. Die sortierte Exportdatei bietet den weiteren Vorteil, dass neue Wortvektoren an die Datei angehängt werden können. Das sorgt für eine vergleichsweise schnelle Teillaufzeit.

4.3 Satzvektor erstellen



Um Satzpaare bilden zu können, sind Satzvektoren nötig. Satzvektoren sind Listen von Satz-

IDs, die mit einer gemeinsamen Wort-ID in Relation stehen. Die Wortvektordatei wird zeilenweise ausgelesen und in Satzvektoren formatiert. Die Satzvektoren werden im Format {`'key': w_id, 'items':[s_id_1, ..., s_id_n]`} in eine weitere temporäre Datei ausgelagert. Für jeden neuen Wortvektor ist es nötig, die temporäre Datei erneut durch zu iterieren. Dadurch wird überprüft, ob eine Wort-ID schon vorkam. Für den positiven Fall wird der entsprechenden Satzvektor mit der Satz-ID des Wortvektors erweitert. Im negativen Fall wird ein neuer Satzvektor erzeugt und der Satzvektordatei angefügt. Das wiederholte Iterieren der Auslagerungsdatei ist zeitintensiv.

Um die Laufzeit zu minimieren wäre es denkbar für jeden Satzvektor eine eigene temporäre Datei zu erzeugen. Der Name der Datei enthält die Wort-ID und der Inhalt der Datei sind die entsprechenden Satz-IDs. Dadurch werden die Dateigrößen und die Lese-/Schreibzeiten klein gehalten. Dieser Ansatz könnte jedoch durch die *Inodes* des Linuxsystems limitiert sein, da die Anzahl der Inodes auf einem System begrenzt ist und die Anzahl der Sätze, die verglichen werden sollen, nicht vorhersagbar ist. Wenn dieser Schritt vollendet ist, wird die jetzt obsolete Wortvektordatei gelöscht.

4.4 Satzpaare ermitteln

Für jedes Satzvektor wird jede Satz-ID mit jeder übrigen Satz-ID gepaart und in eine Auslagerungsdatei geschrieben. Dabei wird auf eine aufsteigende Reihenfolge der Paare geachtet. So erzeugt zum Beispiel der Satzvektor `w_id:[s_id_1, s_id_2, s_id_3]` die Paare `s_id_1 s_id_2, s_id_1 s_id_3, s_id_2 s_id_3`. Wenn ein Satzvektor nur eine Satz-ID enthält, wird dieser verworfen. Am Ende des Schrittes wird die Satzvektordatei gelöscht.

4.5 Satzpaare sortieren und Duplikate zählen

Zur Sortierung und Zählung der Paare stellt Linux systemeigene Befehle zur Verfügung. `sort --parallel=3 -o sorted-pairs pairs` sortiert zeilenweise die Satzpaare aufsteigend. Zur schnelleren Sortierung kann mit `--parallel=`n die Sortierung auf n Prozessorkerne (in diesem Fall drei Kerne) verteilt werden. `uniq -d -c` zählt die doppelten Paare und entfernt einzeln vorkommende Satzpaare. Anschließend sortiert ein erneuter `sort`-Aufruf die Zählungen der Aanzahl der Dopplungen nach und gibt alles in die Datei `sorted-counted-pairs` aus. Als letzter Schritt werden für jedes Paar mittels SQL-Abfrage der eigentliche Sätze als Klartext aus der Datenbank gelesen und in der Datei `sentences.txt` abgespeichert.

5 Ergebnisse

Der Algorithmus funktioniert und liefert wie gewünscht sehr ähnliche Sätze. Aufgrund der langen Laufzeit wurden vorerst nur die Ergebnisse der ersten 10.000, 100.000 und 1 Millionen Sätze ausgewertet. Hierbei benötigte der Durchlauf mit 10.000 Sätzen 55 Sekunden, der Durchlauf mit 100.000 Sätzen 44 Minuten und der Durchlauf mit 1 Millionen Sätzen benötigte 4 Tage und 8 Stunden. Die Laufzeit steigt also bei zunehmender Satzmenge exponentiell an. Als Flaschenhals stellte sich hierbei die begrenzte Lese- und Schreibgeschwindigkeit der Festplatte heraus. Die Erstellung der Satzvektoren benötigt die meiste Zeit.

Insgesamt wurden in den 10.000 Sätzen 19 semantisch ähnliche Sätze gefunden, in den 1 Millionen Sätzen wurden 6.532 ähnliche gefunden. Auffällig war hierbei, dass die durchschnittliche Länge der gefundenen Sätze bei 26 Wörtern liegt. Der längste Satz in der Ergebnismenge enthält 34 Wörter, der kürzeste 14 Wörter. Hieraus kann man schlussfolgern, dass der Algorithmus vor allem für lange Sätze gut funktioniert. Ein großer Teil der gefundenen ähnlichen Sätze unterscheiden sich entweder

nur in wenigen Wörtern oder sie sind identisch und einer der beiden Sätze enthält einen weiteren (unterschiedlichen) Teilsatz. So zum Beispiel diese beiden Sätze:

"Die Zivilklage sei an einem Bundesbezirksgericht in Washington im Namen von 144 Hinterbliebenen eingereicht worden, sagte Menschenrechtsanwalt Terry Collingsworth von der Gruppe International Rights Advocates der Nachrichtenagentur Reuters **am Donnerstag**."

„Die Zivilklage sei an einem Bundesbezirksgericht in Washington im Namen von 174 Hinterbliebenen eingereicht worden, sagte Menschenrechtsanwalt Terry Collingsworth von der Gruppe International Rights Advocates der Nachrichtenagentur Reuters."

Man sieht, dass sich beide lediglich durch die zwei Worte „am Donnerstag" am Ende des ersten Satzes unterscheiden. Bei den folgenden Sätzen enthält der zweite Satz einen zusätzlichen Nebensatz am Ende, durch den sie sich unterscheiden.

„Die Zivil-Polizisten verdeckten den 28-jährigen Russlanddeutschen, der am 1.Juli im Landgericht Dresden die Ägypterin Marwa el-Sherbini erstochen hat."

„Die Zivil-Polizisten verdeckten den 28-jährigen Russlanddeutschen, der am 1.Juli im Landgericht Dresden die Ägypterin Marwa el-Sherbini erstochen hat**, mit vollem Körpereinsatz**."

In der aktuellen Ergebnismenge wurden keine Fehler gefunden. Hier ist zu prüfen, ob sich bei größeren Datenmengen Fehler einschleichen.

6 Einschränkungen

Es stellte sich heraus, dass es bei dieser großen Datenmenge nicht möglich ist alle Operationen im RAM auszuführen. Deshalb wurden alle Schritte auf die Festplatte ausgelagert, was zu einer deutlichen Verlangsamung des Skripts führte. Es stellte sich heraus, dass die Festplatte zum Flaschenhals wurde. Dies führt zu Einschränkungen in der Laufzeit.

Eine weitere Einschränkung liegt bezüglich der untersuchten Sätze vor. Es wird nach Sätzen mit mindestens 5 gemeinsamen seltenen Wörtern gesucht. Das setzt zugleich voraus, dass die Sätze mindestens 5 seltene Wörter enthalten. Dadurch werden vor allem kurze Sätze und Sätze mit wenigen seltenen Wörtern nicht beachtet. Setzt man die Grenze mit mindestens 5 gemeinsamen Wörtern herab, führt das zu einer Verschlechterung der Präzision.

7 Erweiterungsmöglichkeiten

Um die benannten Einschränkungen zu beheben gibt es zwei Erweiterungsmöglichkeiten. Die erste Möglichkeit ist den Algorithmus auf einem Cluster von Rechnern laufen zu lassen. Durch die Verteilte Architektur sind deutliche Steigerungen in der Performance möglich. Damit sollte es möglich sein, den Größten Datensatz, mit ca. 250 Mio. Sätzen, in deutlich kürzerer Zeit zu verarbeiten.

Außerdem wäre es möglich, Sätze mit weniger als 5 gemeinsamen Wörtern genauer auszuwerten. Diese Sätze werden geclustert. Man erhält Cluster mit Sätzen die zumindest zum selben Thema gehören. Nun müsste noch ein Algorithmus entwickelt werden um diese Cluster gezielt automatisch auszuwerten.