Jugend forscht

in Mathematik/Informatik

Crypsor: Automatische Verschleierung sensibler Suchanfragen

Eric Oliver Schmidt

Georg-Cantor-Gymnasium

Klasse 12

Alter: 18 Jahre

Betreuungslehrer: Dr. Andreas Koch

Betreuer: Prof. Dr. Matthias Hagen

Martin-Luther-Universität Halle-Wittenberg

Institut für Informatik

Schuljahr 2020/2021

Kurzfassung

In dieser Arbeit beschäftige ich mich mit der Fragestellung des Datenschutzes und einer Möglichkeit, der Datensammlung und Nutzerprofilbildung großer Suchmaschinen entgegenzuwirken. Mein Ziel ist es, Suchanfragen eines Nutzers so umzuformulieren, dass sensible Informationsbedürfnisse verschleiert und nicht direkt Teil des von der Suchmaschine erstellten Nutzerprofils werden. Mein Ansatz berechnet so genannte Keyqueries (engl.: Schlüsselanfragen), die ähnliche Treffer wie die Originalanfrage liefern, aber deutlich weniger sensibel sind. Am Ende soll der Nutzer nach Verschleierung seiner Suchanfrage, durch welche die Suchmaschine personenbezogene Interessen und Daten nur erschwert feststellen kann, trotzdem ein möglichst gutes und den Treffern seiner Originalanfrage recht ähnliches Suchergebnis erhalten.

Dieses Projekt baut auf meiner Jugend forscht-Arbeit des letzten Jahres auf. Ich habe dabei verschiedene Änderungen vorgenommen und das Verfahren um- und ausgestaltet. Startet der Nutzer meine App von diesem Jahr, so fällt ihm (oder ihr) zunächst das nutzerfreundliche Design auf. Er kann seine Sucherlebnis direkt anpassen oder gleich im Web nach seiner sensiblen Anfrage suchen. Hat er die Suche gestartet, stellt er fest, dass das Verfahren zur Berechnung von Nominalpharsen und Keyqueries deutlich zügiger abläuft als noch im letzten Jahr. In der Nutzerauswahl werden ihm die berechneten Nominalphrasen angezeigt. Sie erscheinen qualitativ hochwertiger und – noch viel wichtiger – enthalten weniger sensible Terme: Der Nutzer spart weitere Zeit, die er sonst zum Abwählen sensibler Phrasen gebraucht hätte. Das Suchergebnis, das am Ende angezeigt wird, wirkt deutlich informativer und relevanter zum Thema, weil das Ranking der Ergebnisse mithilfe der Bilbiothek Lucene überarbeitet ist. Die Implementierung von JSoup und JUnit-Tests machen mein Verfahren flexibler für etwaige Veränderungen der Struktur von Googles Seiten im Web und ermöglichen eine fortwährende Überprüfung der Korrektheit des Programms.

Die empirische Bewertung erfolgt durch Versuche mit verschiedenen Suchanfragen und beinhaltet auch eine Nutzerstudie. Eine Auswertung mit der Evaluierungsmethode eines Verfahrens von Arampatzis et al. [1] zeigt vielversprechende Ergebnisse. Dazu habe ich dieses Verfahren von Arampatzis et al. [1] in diesem Jahr implementiert, um mein Verfahren mit diesem zu vergleichen. Meine Planung zur Durchführung einer größeren Nutzerstudie zu verwirklichen, bietet die Möglichkeit einer weiteren Evaluierung.

Inhaltsverzeichnis

1 Einleitung

Ich finde es erschreckend, wie viel eine einzige Suchanfrage über einen Menschen aussagt – oder wie eine kleinere Sammlung von Suchanfragen das eigene Leben beschreiben kann, obwohl diese im Allgemeinen als eher „unsensibel“ angesehen werden. So auch infolge des AOL-Suchskandals, als Suchanfragen der Nutzer aus drei Monaten geleakt wurden. Unter ihnen sind die Suchanfragen von Ms. Thelma Arnold aus den USA. Als AOL-Nutzer #4417749 konnte der 62-Jährigen ein sehr ausführliches Profil zugeordnet und sie sogar zuhause gefunden werden. Nachdem ich darüber gelesen hatte, stellte sich mir die Frage: Sind wir alle gläserne Menschen? Kann unser gesamtes, komplexes Leben durch ein paar Suchanfragen, die wir an Google stellen, nachvollzogen werden? Die Antwort, die ich mir selbst geben musste, fand ich bestürzend. Denn der Informationsgehalt über uns im Netz ist nicht nur auf Skandale beschränkt. Tag für Tag stellen wir Milliarden Suchanfragen an Google. Allein in der Zeit, in der Sie dieses kleine Stück meiner Einleitung gelesen haben, wurden die detaillierten Datensammlungen von Google um ca. 3,8 Millionen Suchanfragen erweitert. Mein Ziel ist es, zu helfen, einige dieser Informationen zu verschleiern. Ms. Arnold wird uns durch diese Arbeit begleiten. Anhand einer ihrer Suchanfragen, mit der sie identifiziert werden konnte, möchte ich mein Verfahren erklären.

Datenschutz ist heutzutage so aktuell wie noch nie. In Zeiten der DSGVO und zahlreichen Vorwürfen an Großunternehmen, Daten der Nutzer zu erheben, zu sammeln und zu verkaufen, nimmt der Datenschutz einen immer höheren Stellenwert in der Mentalität der Bevölkerung ein. Der Konzern Google zum Beispiel, der zweifelsohne mit einem globalen Marktanteil von 92 % [6] eine Quasi-Monopolstellung im weltweiten Suchmaschinenmarkt besitzt, gerät immer wieder in die Kritik, Nutzerdaten unerwünscht zu erheben und sehr detaillierte Profile mit den Suchanfragen seiner Nutzer zu erstellen. Googles Cookies und Werbenetzwerk helfen, das Verhalten von Menschen im Web sehr genau zu erfassen und daraus ihre Interessen abzuleiten [11]. Dabei sind Gebiete wie Krankheiten, persönliche Probleme und Finanzen zutiefst private Informationen eines jeden Menschen, welche auch geschützt werden sollten. Nicht umsonst ist Datenschutz zu einem Grundrecht in Deutschland geworden.
Damit persönliche Daten nicht nur auf dem Papier geschützt und Verbraucher in ihrem alltäglichen Umgang mit digitalen Anwendungsmöglichkeiten nicht in ihrem Verhalten beeinflusst werden, habe ich eine Möglichkeit kreiert und evaluiert, welche verhindern kann, dass zu viele persönliche Informationen über Suchanfragen preisgegeben werden.

Zielstellung meiner Arbeit ist es, Suchanfragen so zu verändern und zu obfuskieren („verschleiern“), dass für eine Suchmaschine am Ende nicht mehr klar erkennbar ist, welches Interesse der Nutzer wirklich hat. Trotzdem sollen die gleichen oder sehr ähnliche Ergebnisse angezeigt werden können. Die dazu in dieser Arbeit untersuchte Lösungsmöglichkeit ist die Verwendung von s. g. Keyqueries, die Wissenschaftler der Bauhaus-Universität Weimar eingeführt und mein Betreuer von der Martin-Luther-Universität Halle-Wittenberg, Herr Prof. Dr. Matthias Hagen, in seinen Forschungen und wissenschaftlichen Arbeiten erforscht haben.

Die Idee meines Ansatzes ist, die Originalanfrage an eine Suchmaschine des Vertrauens, z. B. ChatNoir,[1] Startpage,[2] DuckDuckGo[3] oder gar eine eigene kleine lokale Suchmaschine zu stellen, die weniger oder keine Nutzerdaten sammelt, jedoch erwartungsgemäß nicht immer qualitativ so hochwertige Suchergebnisse wie Google liefert. Für die am höchsten gerankten 10, 20 oder 50 Ergebnisse der Originalanfrage an die vertrauenswürdigere Suchmaschine werden lokal andere Anfragen berechnet (Keyqueries), die jeweils einen Teil der Originalergebnisse liefern, aber möglichst weniger sensibel als die Originalanfrage sind. Aus diesen Keyqueries kann der Benutzer diejenigen ausschließen, die ihm zu privat sind, oder eben auswählen, welche beispielsweise an Google abgeschickt werden können, da sie „unsensibel“ genug sind. So erhält er die oft qualitativ als sehr gut eingeschätzten Google-Ergebnisse für diese unbedenklichen Anfragen. Die ausgewählten Anfragen werden dann an Google oder eine andere gewünschte Suchmaschine gestellt, die Ergebnisse heruntergeladen und lokal bezüglich ihrer Relevanz für die Originalanfrage umsortiert und dem Nutzer angezeigt. Aus den obfuskierten Keyqueries erkennt Google bzw. eine andere Suchmaschine im Idealfall nicht mehr so leicht, welches eigentliche Interesse der Nutzer hat. Das Ziel ist natürlich trotzdem noch, sehr konkrete Anfragen zu senden, um kombiniert in etwa die ursprünglichen Ergebnisse lokal ausrechnen zu können ohne (zu viel) vom Interesse des Nutzers preiszugeben. Hätte Ms. Arnold beispielsweise ihre Suchanfragen nicht im Klartext abgesendet, wäre sie um einiges schwieriger zu identifizieren gewesen – oder gar überhaupt nicht.

Ich konnte letztes Jahr schon mit diesem Thema an Jugend forscht teilnehmen. Mit jedem Hinweis habe ich mich intensiv auseinandergesetzt und dadurch meinen Algorithmus weiter verbessert. So standen Nutzerfreundlichkeit, Geschwindigkeit und auch Qualitätssteigerung ganz oben auf meiner Agenda. Genau dies konnte ich nun umsetzen und in meinen Ansatz integrieren.

Eine empirische Evaluation des Keyquery-Ansatzes zeigt durchaus vielversprechende Ergebnisse. Fast zwei Drittel der getesteten Anfragen können erfolgreich verschleiert werden, ohne dass so viele der Originalergebnisse wegfallen, dass das Sucherlebnis signifikant beeinflusst wird.

2 Vorüberlegungen

Schon im letzten Jahr habe ich mich im Rahmen meiner Jugend forscht-Arbeit mit dem spannenden Thema der Anfragenverschleierung beschäftigt. Datenschutz ist ein wichtiger Bereich des Lebens und im Zuge der Digitalisierung eine Problematik unaufhaltbar wachsender Bedeutsamkeit. Ich selbst habe an mich und mein Verfahren den Anspruch, es stetig zu verbessern, daher entschloss ich mich, dieses Jahr noch einmal mit der Thematik bei Jugend forscht teilzunehmen und die über das Jahr gesammelten Rückmeldungen zu beachten und mit einzubinden.

[1] https://www.chatnoir.eu/
[2] https://www.startpage.com/
[3] https://www.duckduckgo.com/

Heutzutage gelten personenbezogene Daten als ungemein wichtiges und wertvolles Gut großer Internetkonzerne. Spezifische Werbung, Suchanfragenvorschläge und anderes werden durch Sammeln von Daten in Suchmaschinen und im Web integriert. Doch wie weit möchte jeder als Privatperson – oder auch als Forscher oder Unternehmer – seine Daten preisgeben? Einige Browser, wie z. B. Google Chrome[4] bieten einen Inkognito-Modus an. Doch dieser, wie nicht flächendeckend bekannt ist, löscht einzig beim Schließen den lokalen Verlauf und Cache des Geräts, überträgt aber trotzdem Daten und Nutzungsverhalten (wie etwa Suchanfragen) an Dritte.

Viele Nutzer stellen Anfragen, die beispielsweise gesundheitliche oder finanzielle Probleme betreffen. Auch Ms. Arnold suchte nach `hand tremors`. Diese Anfrage ist sensibel, da sie ziemlich private Informationen preisgeben kann – nicht „nur“, dass Ms. Arnold wohl dieses Symptom hat, sondern u. a. auch Auskunft über ihr Alter gibt. Anhand dieses Beispiels möchte ich meine Arbeit veranschaulichen.
Aufgrund der hohen Bedeutung des Themas Datenschutz gerade auch für Suchanfragenprofile, haben sich einige Wissenschaftler schon mit Fragestellungen der Anfragenverschleierung beschäftigt. Petit et al. [7] konzentrierten sich dabei aber vor allem auf den Schutz der Daten beim Routing. Der Ansatz von Arampatzis et al. [1] ist schon ähnlicher zu meiner Themenstellung. Die dort vorgeschlagene Verschleierung habe ich jedoch durch die Nutzung des Konzeptes der Keyqueries und durch die Nutzereinschätzung noch (deutlich) verbessert.

3 Lösungsidee und algorithmische Umsetzung

In diesem Kapitel werden die grundlegende Konzeption und Details des Ansatzes beschrieben.

3.1 Begriffsklärung und allgemeine Lösungsidee

Im Kontext meiner Arbeit werden als Nominalphrasen Wortgruppen aus zwei bis fünf Wörtern bezeichnet, die mindestens ein Substantiv und sonst nur Adjektive in direkter Aufeinanderfolge beinhalten. Ein Beispiel für eine zu `hand tremors` verwandte Nominalphrase ist *medical symptom*. Solche Phrasen sollen als Bausteine der Keyqueries fungieren (Abschnitt 3.2). Dabei sind Keyqueries folgendermaßen definiert (vgl. [4]): Eine Anfrage q ist eine Keyquery für eine Dokumentenmenge D gegen eine Suchmaschine S, wenn sie folgende Bedingungen erfüllt:
(1) Jedes $d \in D$ ist unter den top-k Suchergebnissen für q gegen S, (2) q liefert mindestens l Suchergebnisse und (3) keine Teilanfrage $q' \subset q$ mit weniger Anfragetermen liefert jedes $d \in D$ in ihren top-k Suchergebnissen zurück.
Sagen wir, Dokument d ist das zehnte Dokument, das Google anzeigt, wenn Ms. Arnold heute nach `hand tremors` gesucht hätte. Die Anfrage `hand tremors` ist also eine Keyquery für d gegen Google, wenn wir die top-20 Suchergebnisse betrachten.

[4] https://www.google.de/chrome/

Algorithmus 1 Verschleiern einer Suchanfrage

Eingabe:

Anfrage q des Nutzer
Keyquery-Parameter k, Parameter t der besten Suchergebnisse von q
Anteil a der mit Keyqueries mindestens zu findenden top-Ergebnisse der Anfragen Q

Ausgabe:

Vorschläge für verschleierte Anfragen + Suchergebnisse

1: $P \leftarrow \varnothing, \quad Q \leftarrow \varnothing$
2: $D \leftarrow$ top-t Treffer von q gegen ChatNoir (oder Startpage) //REST-API
3: **for** jedes $d \in D$ **do**
4: $\quad P \leftarrow P \cup \{$ 10 Schlüsselphrasen aus d extrahiert $\}$ //Apache OpenNLP POSTagger, Version 1.7.0
5: **for** jedes $D' \subseteq D$ **do**
6: $\quad Q \leftarrow Q \cup \{$ Keyqueries für D' mit jeweils top-k Treffern gegen ChatNoir $\}$ //Details in Algorithmus 2
7: Wähle $Q' \subseteq Q$, sodass für die kombinierte Ergebnismenge E der $q' \in Q'$ gilt: $|D \cap E| \geq |D| \cdot a$
8: Nutzereinschätzung: Sind bestimmte Keyqueries aus Q' zu sensibel?
9: **while** Nutzer verwirft Keyquery q' mit Ergebnismenge $D^* \leftarrow E' \cap D$ **do**
10: $\quad Q' \leftarrow Q' \setminus \{ q' \}$
11: $\quad Q' \leftarrow Q' \cup \{$ alternative Keyquery(ies) für $D^* \}$ //Details in Algorithmus 3
12: $F \leftarrow$ top-k Treffer jedes $q' \in Q'$ gegen Google
13: BM25-Index für F lokal aufbauen //Lucene
14: Stellen von q an lokalen F-Index + Ausgabe des Ergebnisrankings

Die Parameter k und l werden in der Regel im Bereich 10, 50 oder 100 festgelegt. Zu Keyqueries werden in dieser Arbeit nur Nominalphrasen aus den Dokumenten einer vorgegebenen Dokumentmenge zusammengesetzt, da in Suchanfragen nur selten Verben, Artikel und andere Wortarten zu genaueren Suchergebnissen führen. Um die Ergebnisse einer Originalanfrage zu berechnen, wird die datenschutztechnisch vertrauenswürdige Suchmaschine ChatNoir verwendet [2].

Im Algorithmus 1 ist der prinzipielle Ablauf des Keyquery-basierten Verschleierungsverfahrens als Pseudocode angegeben. Es sollen mittels Berechnung unsensiblerer Keyqueries möglichst viele der Originalergebnisdokumente gefunden werden. Dabei kann der Ablauf grob in sechs Schritte unterteilt werden:

1. Stellen der Suchanfrage q des Nutzers an ChatNoir und Speichern der top-t Suchergebnisse in D (Zeile 2).
2. Extrahieren der zehn häufigsten Nominalphrasen aus jedem $d \in D$ (Zeile 3–4).
3. Berechnen von Keyqueries für jede Teilmenge $D' \subseteq D$, wobei nur die extrahierten Nominalphrasen verwendet werden (Zeile 5–6).

4. Ranking der Keyqueries, um die Quote a (Anteil mindestens zu findender Originalergebnisdokumente) zu erreichen oder möglichst die Differenz zu a gering zu halten (Zeile 7).
5. Nutzereinschätzung ggf. mit Nachberechnung von Keyqueries (Zeile 8–11).
6. Stellen der vom Nutzer freigegebenen (und gegebenenfalls nachberechneten) Keyqueries an Google, lokales Umsortieren der erhaltenen Suchergebnisse. BM25 wird als Retrieval-Modell verwendet (Zeile 12–14).

3.2 Berechnung der Keyqueries

Um möglichst spezifische Suchergebnisse für den Benutzer zu erhalten, ohne die komplette ursprüngliche Anfrage q preiszugeben, werden unsensiblere Keyqueries zur Ermittlung der Suchergebnisse genutzt (vgl. Pseudocode in Algorithmus 1).

Die REST-API von ChatNoir liefert im ersten Schritt die top-t Ergebnisse für die eigentliche, aber zu verschleiernde, Suchanfrage q des Nutzers (in unserem Beispiel `hand tremors`). Durch mitgelieferte IDs der Ergebnisse kann über die API von ChatNoir der Textinhalt der Ergebnisse ausgelesen und gespeichert werden.

3.2.1 Extrahieren von Nominalphrasen

Aus dem Textinhalt der Ergebnisdokumente werden HTML-Tags unter Verwendung der Bibliothek JSoup[5] entfernt und anschließend die Wortarten jedes einzelnen Wortes mithilfe des OpenNLP Part-of-Speech-Taggers[6] bestimmt. Auf diese Weise werden alle Nominalphrasen extrahiert und die zehn häufigsten pro Dokument gewählt (Algorithmus 1, Zeile 4). Zu beachten bei der Sortierung von Nominalphrasen nach Häufigkeit ihres Vorkommens ist, dass in einigen Fällen Teile dieser Nominalphrasen deutlich öfter auftreten können als die gesamte Phrase.

In einigen Vorexperimenten hat sich herausgestellt, dass Nominalphrasen, die in möglichst vielen Dokumenten aus D vorkommen, am wichtigsten sind und bei Gleichstand die längeren zu bevorzugen sind. So ist vor allem die Wahrscheinlichkeit höher, dass eine Phrase Teil einer Keyquery für mehr Dokumente werden kann. Ein letztes nachgeschaltetes Wertungskriterium bei Gleichstand in Dokumentenzahl und Länge ist das Bevorzugen der absolut häufigeren Phrasen. Häufige Phrasen sind weniger spezifisch und dadurch ggf. weniger sensibel. Dokumente, die eine Phrase vielfach enthalten, werden wohl eher weiter oben in den Suchergebnissen gerankt. So lassen sich die Nominalphrasen sortieren und bewerten. Im Algorithmus werden die top-10 Nominalphrasen für jedes Dokument in einer Textdatei gespeichert und Suffixe wie Plural „-s“ im Englischen etc. mithilfe der Java-Bibliothek[7] des Porter-Stemming-Algorithmus [8] entfernt.

[5] https://jsoup.org/
[6] opennlp.apache.org/docs/1.7.0/apidocs/opennlp-tools/opennlp/tools/postag/POSTaggerME.html
[7] https://tartarus.org/martin/PorterStemmer/index.html

Nominalphrasen mit gleichen Stämmen werden zusammengefasst und die Häufigkeiten addiert (zum Erstellen von Keyqueries werden aber die ungestemmten Nominalphrasen genutzt).

Durch vorausgegangene Pilot-Studien kam ich zu dem Entschluss, die Qualität der Nominalphrasen weiter zu verbessern. Dazu gehört das Aktualisieren und Erweitern der Listen zur Entfernung von Stoppwörtern und das verbesserte Entfernen von häufig vorkommenden, allerdings nicht zum wirklichen Webtext gehörenden Phrasen, wie zum Beispiel „Impressum“. Des Weiteren kann der Nutzer auf Wunsch auch direkt Nominalphrasen für die weitere Berechnung ausschließen.

3.2.2 Berechnen von Keyqueries

Das Verfahren zur Berechnung der Keyqueries habe ich im Vergleich zum Vorjahr komplett überarbeitet. Anlass hierfür sind Laufzeit, Ressourcennutzung und Qualität der Suchergebnisse, allerdings auch Übersichtlichkeit und Wiederverwendbarkeit des Quellcodes. Darüber hinaus habe ich für diese Arbeit automatische Tests implementiert, um systematische und auch eher zufällige Fehler auszuschließen.

Der intelligente und vorteilhafte HBC-Algorithmus [5] (Algorithmus von Hébert, Bretto und Crémilleux) setzt sich gegen Backtracking- und Bruteforce-Methoden durch. Das Verfahren ist in mehrere Schritte aufgeteilt und wurde zur Keyquery-Berechnung angepasst.

In einem ersten Schritt werden die extrahierten Nominalphrasen (Menge P in Algorithmus 1) darauf überprüft, ob sie Teil einer Keyquery für mehr als ein Dokument sein können (sie also in mindestens zwei Dokumenten aus D enthalten sind). Verallgemeinert betrachten wir als Keyquery für eine Dokumentmenge D all diejenigen Nominalphrasen $p \in P$, deren ChatNoir-Ergebnismenge E' mindestens m Dokumente aus D enthält (Algorithmus 2, Zeilen 6, 17). Im Idealfall ist $m = |D|$, d. h. es werden alle Originaldokumente gefunden. Je größer der Parameter m gewählt wird, desto höher ist die Qualität der Keyqueries, das heißt, es werden mehr Dokumente d pro Keyquery gefunden. Solche Keyqueries mit einer höheren Anzahl m gefundener Originalergebnisdokumente sind im Vergleich zu anderen Keyqueries mit kleinerem m „stärker“.

Das Bestimmen von Keyqueries gegen ChatNoir für möglichst alle top-t Suchergebnisse der Originalanfrage erfolgt durch Kombination der besten Nominalphrasen miteinander (z. B. *medical symptom best treatment*). Dabei sollte darauf geachtet werden, dass Phrasen nicht mehrfach in einer Keyquery enthalten sind, weil dies den „Sinn“ der Suchanfrage nicht wesentlich ändert. Die Wichtungen zum Ranken der Keyqueries entsprechen der Anzahl m von der Keyquery gefundener Dokumente der Originalanfrage. Durch die Kombinationsstrategie des angepassten HBC-Algorithmus kann es vorkommen, dass ein Kandidat mehrfach generiert wird. Um den Rechenaufwand zu optimieren wird daher überprüft, ob der Keyquery-Kandidat schon einmal berechnet wurde, bevor weiter verfahren wird (Zeile 13).

Algorithmus 2 Berechnen von Keyqueries

Eingabe:
Dokumentmenge D
Keyquery-Parameter k, l und m
Menge P extrahierter Nominalphrasen

Ausgabe:
Menge Q aller für D berechneten Keyqueries

```
1:  Q ← ∅    E' ← ∅    C_n ← ∅ ∀ n ∈ {1,2,3,4,5}
2:  C_1 ← P
3:  for alle p ∈ P do
4:    if p liefert mindestens l Suchergebnisse gegen ChatNoir then
5:      E' ← { top-k Suchergebnisse von p gegen ChatNoir }
6:      if |E' ∩ D| ≥ m then
7:        Q ← Q ∪ {p}                                   //p ist Keyquery
8:        C_1 ← C_1 \ {p}
9:  i ← 1
10: while C_i ≠ ∅ ∧ i ≤ 5 do
11:   for alle c, c' ∈ C_i für die gilt |c ∩ c'| = i − 1 do
12:     ĉ ← c ∪ c'
13:     if ĉ wurde noch nicht generiert then
14:       if ĉ \ {p} ∈ C_i für alle p ∈ ĉ then
15:         if ĉ liefert mindestens l Suchergebnisse gegen ChatNoir then
16:           E' ← { top-k Suchergebnisse von ĉ gegen ChatNoir }
17:           if |E' ∩ D| ≥ m then
18:             Q ← Q ∪ {ĉ}                             //ĉ ist Keyquery
19:           else
20:             C_{i+1} ← C_{i+1} ∪ {ĉ}                 //ĉ ist weiter Kandidat
21:   i ← i + 1
```

Zunächst werden alle vom Nutzer genehmigten Nominalphrasen als Suchanfragen einzeln an ChatNoir gestellt (Zeile 3–8) und die top-k Treffer gespeichert. Die Nominalphrasen, welche nach diesem Vorgang den Bedingungen einer Keyquery entsprechen (gewünschte Dokumente in den top-k Treffern und mindestens l Ergebnisse insgesamt), werden direkt gespeichert. Alle anderen bleiben Kandidaten für die Kombination des zweiten Schrittes des HBC-Verfahrens. In diesem werden alle möglichen Kombinationen der Einzelphrasen gebildet (z. B. `best treatment` und `hand symptom` zu `best treatment hand symptom`). Ab dem dritten Schritt, die Kandidaten bestehen nun aus drei Nominalphrasen, erfolgt die Kombination nur, wenn die Überlappung zweier Kandidaten groß genug ist (Zeile 11). Ein Kandidat besteht im i-ten Schritt aus i Nominalphrasen und zwei Kandidaten werden nur dann kombiniert, wenn sie $i - 1$ Nominalphrasen gemeinsam haben (z. B. `best treatment vitamin b` und `best treatment hand symptom` zu `best treatment vitamin b hand symptom`).

Nach dem Berechnen der Keyqueries wird überprüft, ob jedes der top-t Ergebnisse der ursprünglichen Anfrage q durch mindestens eine Keyquery gefunden wird. Es werden alle Suchanfragen gespeichert, die zwar keine Keyquery sind (etwa weil sie weniger als alle Originalsuchergebnisse finden), die aber dennoch einige der gewünschten Dokumente finden. Auf diese kann zurückgegriffen und untersucht werden, ob eine von diesen Anfragen noch fehlende Dokumente findet. Genügt auch dies nicht, wird der Algorithmus erneut gestartet, diesmal jedoch für eine Teilmenge D' der Dokumente D, für welche noch keine Keyqueries gefunden wurden. Dies erfolgt durch Kombinieren „neuer“, also weiter unten gerankter Nominalphrasen für möglichst viele der Dokumente aus D'. Ist auch dies erfolglos (wenn z. B. diese Dokumente keine starken Nominalphrasen enthalten), bleiben diese Dokumente ohne Keyquery. Das Aufnehmen von sehr schwachen Nominalphrasen als Keyqueries hat in Vorexperimenten die Qualität der Keyquery-Ergebnisse eher verschlechtert als eine geringe Zahl nicht gefundener Dokumente. Stattdessen ist ein Erhöhen der Anzahl von Ergebnissen gegen ChatNoir (also der Parameter k und t) sinnvoll, wenn bei einer Suchanfrage nur schwächere Nominalphrasen berechnet werden. So ist mehr Text vorhanden, aus dem Nominalphrasen extrahiert werden und es lassen sich entweder neue Nominalphrasen ermitteln oder schon zuvor gefundene sind nun in genügend Dokumenten enthalten.

Ziel ist es, mit den berechneten Keyqueries möglichst viele, oder gar alle $d \in D$ zu finden. Der Anteil gefundener Dokumente aus D soll größer als eine vorgegebene Quote $0 < a \leq 1$ sein (Algorithmus 1, Zeile 7). Je größer a tatsächlich wird, desto besser. Sind möglichst viele Dokumente $d \in D$ von den Keyqueries abgedeckt, sind es spezifischere und stärkere Keyqueries. Die Chance steigt dann, gute Ergebnisse von Google zu erhalten.
Eine Keyquery, die durch Kombination mehrerer Nominalphrasen berechnet wurde, findet oft mehr Dokumente aus D weiter oben im Ranking als eine, welche aus weniger Nominalphrasen besteht. Damit steigt ebenfalls die Chance, dass gegen Google mehr der eigentlichen Treffer der Originalanfrage q gefunden werden.

Speichern der Berechnungen und Daten

Gesendete Suchanfragen und deren Suchergebnisse werden lokal auf dem Gerät gespeichert. Dies ermöglicht, dass zukünftige Zugriffe auf dieselbe Anfrage schneller und auch ohne Internetzugriff auf ChatNoir erfolgen können, wenn bestimmte Anfragen mehrmals gestellt werden. Gespeichert wird die Suchanfrage, die Anzahl und IDs der Treffer. Außerdem werden die ermittelten Nominalphrasen mit ihren Scores gespeichert, genauso wie die schlussendlich berechneten Keyqueries mit ihren Bewertungen und jeweils gefundenen Dokumenten. Die folgenden Schritte, d.h. die Nutzereinschätzung (vgl. Abschnitt 3.2.3), sowie das Stellen der Keyqueries an Google (Abschnitt 3.2.4), werden nicht in der Textdatei dokumentiert und bei jedem Programmaufruf ausgeführt, egal ob online oder „offline“. Die benötigten Informationen werden für die entsprechende Anfrage aus der Logdatei ausgelesen. Außer für Testzwecke wird diese Datei verschlüsselt, sodass andere Nutzer des Gerätes nicht den Suchverlauf einsehen können.

3.2.3 Nutzereinschätzung

Die Keyqueries, welche die meisten Dokumente aus D finden, werden priorisiert. Ist jedes Dokument mit mindestens einer Keyquery abgedeckt, werden dem Nutzer die gefundenen Anfragen angezeigt. Nicht alle berechneten Keyqueries werden an Google gestellt. So findet eine Sortierung und Vorauswahl statt, in der die stärksten in eine Keyquery-Abdeckung Q_C einfließen.
Dabei muss jedoch die Bedingung erfüllt sein, dass durch die Keyquery-Abdeckung keines der zuvor gefundenen Originalergebnisdokumente „verloren geht“, sodass ggf. auch schwächere Keyqueries für die sonst nicht mehr gefundenen Dokumente mit einbezogen werden. Ist beispielsweise *hand symptom* die einzige Keyquery für ein Dokument d' und wird diese Keyquery nun nicht verwendet, kann dieses Dokument nicht mehr gefunden werden.

Die Originalanfrage wird aus den Nominalphrasen ausgeschlossen und es wird sichergestellt, dass extrahierte Nominalphrasen nicht alle einzelnen Wörter der Originalanfrage in anderer Reihenfolge enthalten. Dennoch können die errechneten Keyqueries durch den Algorithmus nicht garantiert komplett unsensibel und „entschärft“ sein. Handtremor kann ein Anzeichen des Parkinson-Syndroms sein, daher ist so ein Beispiel für eine zu sensible Nominalphrase *parkinson disease*. Deshalb kann der Nutzer vor Abschicken an Google überprüfen, ob die vorgesehenen Keyqueries unsensibel „genug“ sind oder verworfen werden müssen. Dann werden ggf. alternative Keyqueries für die verworfenen gesucht (vgl. Algorithmus 2 für den Pseudocode). Um dem Nutzer Zeit zu ersparen, habe ich nun einen automatischen Privacy-Filter zur Vorsortierung eingebaut. Es gibt vier Verschleierungsklassen: keine Verschleierung, die oben beschriebene Entfernung der Originalwörter, das zusätzliche Entfernen von Synonymen mithilfe der JAWS-Bibliothek[8] und darüber hinaus die Verschleierung per Ontologie: Semantisch über- (Hyperonyme) oder untergeordnete Begriffe (Hyponyme) werden vorab aussortiert.

3.2.4 Stellen der Keyqueries an Google

Nach Freigabe durch den Benutzer können die Keyqueries an Google oder eine andere „große“ Suchmaschine gestellt werden. Da die Suchergebnisse der Keyqueries gegen ChatNoir ähnlich denen der Originalanfrage gegen ChatNoir sind, ist zu erwarten, dass mit den Keyqueries auch gut gerankte Dokumente der Originalanfrage gegen Google gefunden werden können. Um auch andere Suchmaschinen, wie z. B. Bing, Ecosia usw. einfach anstelle von Google nutzen zu können und um das Parsen der Webergebnisse zu erleichtern, habe ich die Suchmaschinenanfrage mit der Bibliothek JSoup implementiert.

[8] https://github.com/jaytaylor/jaws

Algorithmus 3 Nutzereinschätzung und Nachberechnung von Keyqueries

Eingabe:
Dokumentmenge D
Menge P der extrahierten Nominalphrasen
Menge Q aller Keyqueries
Menge Q_C einer Keyquery-Abdeckung

Ausgabe:
Menge Q^* zugelassener Keyqueries

$L \leftarrow \varnothing$ (Menge aller „verlorenen“ Dokumente)
while Nutzer genehmigt $q_C \in Q_C$ nicht **do**
  if Nutzer genehmigt q_C nicht **then**
    $Q_C \leftarrow Q_C \setminus \{q_C\}$
    $L(q_C) \leftarrow$ {Dokumente $d \in D$, die nur von q_C gefunden werden }
    $L \leftarrow L \cup L(q_C)$
while es existiert eine Keyquery q_i für $L' \subseteq L$ für die gilt: $q_i \neq q_C$ **do**
  $Q_C \leftarrow Q_C \cup \{q_i\}$
  $L \leftarrow L \setminus L'$
if $|L| > 0$ **then**
  Algorithmus 2 für $D' := L$ mit „neuen“ Nominalphrasen

3.3 Nutzerinterface

Die graphische Benutzeroberfläche der App wurde ebenfalls stark überarbeitet und ist nun nutzerfreundlich, aufgeräumt, übersichtlich und modern ausgestaltet. Der Nutzer kann Einstellungen zur Abwägung von Qualität und Prozessdauer auswählen sowie interaktiv am Verfahren teilnehmen, indem er berechnete Keyqueries und Nominalphrasen genehmigt oder ablehnt. Gleichermaßen kann der Nutzer seine Standard-Suchmaschine (z. B. Google, Bing, Ecosia) und auch seine bevorzugte vertrauenswürdige Suchmaschine zur Berechnung der Keyqueries (z. B. ChatNoir, Startpage, DuckDuckGo, eine eigene kleine private Suchmaschine) auswählen.

Damit die vermutlich eher geringere Rechenleistung von Mobilgeräten sich nicht auf die Laufzeit und das Verfahren sich nicht auf das Datenvolumen des Nutzers auswirken, ist es möglich, die zu verschleiernde Suchanfrage an einen vertrauenswürdigen externen Server zu übermitteln, damit dieser die Berechnungen durchführt und nur die Ergebnisse dem Nutzer auf sein Gerät sendet.

3.4 Energiebilanz

Nicht nur Datenschutz ist ein zentrales Thema der heutigen Zeit. Auch Umwelt- und Klimaschutz sind wichtige Interessen der Bevölkerung, denn unsere Erde gibt es nur einmal. Nachhaltigkeit ist auch ein wichtiges Grundprinzip meines Verfahrens. Zugegebenermaßen bringen die Berechnungen für den Datenschutz einen Mehraufwand und damit auch einen höheren Stromverbrauch. Dennoch soll eine vertretbare CO_2-Bilanz erreicht werden.

Deshalb wird bei jeder 50. Suchanfrage des Nutzers Ecosia[9] anstelle von Google bzw. einer anderen Suchmaschine verwendet, sofern der Nutzer Ecosia nicht schon standardmäßig eingestellt hat. Dies genügt, um die CO_2-Bilanz des Verfahrens vollständig auszugleichen. Die Suchmaschine Ecosia verspricht, ihre Gewinne zu verwenden, um Bäume zu pflanzen. Laut dem Unternehmen entzieht dabei jede Suche mit Ecosia der Atmosphäre ein Kilogramm Kohlenstoffdioxid.

4 Evaluation

Die Effizienz und Wirksamkeit meines Verfahrens wird mit den bereits von Arampatzis et al. [1] zum Testen genutzten 95 Anfragen[10] evaluiert und es wird eine Nutzerstudie durchgeführt.

4.1 Ergebnisqualität der Keyqueries

Mein Ansatz berechnet für die top-10 und top-20 Suchergebnisse jeder der 95 Testanfragen gegen ChatNoir Keyqueries. Diese werden, ebenso wie die 95 Originalanfragen, an Google gestellt.
Für die Originalanfragen werden die top-100 Suchergebnisse gespeichert, für jede einzelne Keyquery jeweils die top-30. Der Vergleich der Suchergebnisse findet mithilfe ihrer Web-URLs statt.

4.1.1 Near-Duplicates

Zusätzlich zum direkten Vergleich der Original-Suchergebnisse mittels URLs werden auch Dubletten analysiert. Dies sind Dokumente, die zwar nicht dieselbe URL haben, aber nahezu den gleichen Textinhalt besitzen. Bei diesen können zum Beispiel Autor, Datum, Header/Footer, Domain oder auch kleine Veränderungen des Textes, ohne Veränderung des Inhalts, geschehen. Auch wenn Google manchmal solche Dubletten im Ranking für die Keyqueries statt der „Originalergebnisse“ anzeigt, sollen sie trotzdem als gefundene Originaldokumente gewertet werden.

Zur Bestimmung von Dubletten werden Termhäufigkeitsvektoren zu jeweils zwei miteinander verglichenen Dokumenten erstellt. Von diesen Vektoren wird nun das Winkelmaß gebildet (dessen Wert ist stets zwischen -1 und 1, 1 heißt 100 %ige Übereinstimmung). Dies ist ein Standardvorgehen zur Dublettenerkennung. Wann ein Dokument als Dublette eines anderen gilt, ist nirgends eindeutig beschrieben. Durch einige Vorversuche konnte ich eine Schwelle von 0,8 festlegen. Ab einem Winkelmaß von 0,75 sah ich mir die beiden Dokumente jeweils selbst an, um die Ähnlichkeit zu verifizieren.

Damit nur der Webseitentext, allerdings nicht Kopf-, Fußzeilen oder sonstiges miteinander verglichen werden, verwende ich zur Main-Content-Extraction die Bibliothek Boilerpipe. Darüber hinaus wird die Library TIKA genutzt, um Textinhalte z. B. aus pdf-Dateien zu erhalten. Stoppwörter werden entfernt, da diese keine großen Änderungen im Sinn eines Textes vornehmen.

[9] https://www.ecosia.org/
[10] http://lethe.nonrelevant.net/datasets/95-seed-queries-v1.0.txt

4.1.2 Auswertung

Tabelle 1 zeigt einige der 95 sensiblen Suchanfragen. Die Zahlen in den Spalten geben den prozentualen Anteil gefundener top-100 Dokumente (oder Dubletten) der Originalanfrage an.

Tabelle 1: Verschleierung sensibler Suchanfragen gegen Google. Die prozentualen Werte stehen jeweils für den Anteil gefundener Suchergebnisse mit Verschleierung zu den 100 besten Suchergebnissen direkt von Google. Links sind die Ergebnisse von diesem und rechts vom letzten Jahr.

Sensible Suchanfrage	2021	2020
`car radar detectors`	40 %	30 %
`cheating husbands`	20 %	19 %
`hacking yahoo passwords`	54 %	12 %
`how to take optygen`	25 %	19 %
`leukemia symptoms teens`	14 %	0 %
`post traumatic stress`	68 %	28 %
`symptoms of bone infection`	43 %	22 %
`unemployment office`	13 %	8 %

Das Resultat meines Ansatzes ist durchaus vielversprechend und zeigt, dass das Verfahren auf jeden Fall funktioniert und mithilfe der Überarbeitung des HBC-Algorithmus im Vergleich zur Vorversion sogar noch weiter verbessert werden konnte. Betrachtet man den Anteil über Keyqueries gefundener Originalergebnisse, lässt sich erkennen, dass der Ansatz für einige Anfragen schon viele ($\geq$ 60 %, z. B. `post traumatic stress`), für andere Anfragen kaum Originalergebnisse (`unemployment office`) ermittelt. Mein Ansatz fand mit $t = 10$ Ergebnissen gegen ChatNoir für 60 % der sensiblen Anfragen, welche für die Evaluation genutzt wurden, genügend Keyqueries, um einige der top-100 Dokumente der Originalanfrage gegen Google zu erhalten. Mit $t = 20$ waren es sogar für 80 %. Im Durchschnitt findet mein Verfahren etwas mehr als ein Drittel der top-100 Dokumente der Originalanfrage gegen Google. Pilot-Versuche haben gezeigt, dass die restlichen von den Keyqueries gefundenen Dokumente größtenteils auch in das Themengebiet der Originalanfrage einzuordnen sind. Die Originalanfrage konnte verschleiert und so vor Google versteckt werden.

Der Grund dafür, dass ein paar Anfragen schlechtere Quoten haben als der Durchschnitt, liegt darin, dass für diese Anfragen nicht genügend (entspricht i. d. R. weniger als 10) und auch nur zu schwache Keyqueries berechnet werden konnten. Das liegt meistens an der verwendeten Suchmaschine ChatNoir und der Wahl der Parameter k und t. ChatNoir ist statisch und hat deutlich weniger Dokumente im Index als beispielsweise Google. Nimmt man dann für eher unspezifischere oder sehr lange sensible Suchanfragen nur die top-10 Suchergebnisse der Originalanfrage gegen ChatNoir, bleibt qualitativ und quantitativ weniger Text aus den gefundenen Dokumenten, der zu potenziellen Keyqueries verarbeitet werden kann. Erhöht man auf die besten 20 Suchergebnisse, so ist die Textmenge größer, um Nominalphrasen und Keyqueries zu berechnen. Der Anteil getesteter Anfragen, die kaum Originaldokumente fanden, wurde so um ein Fünftel verringert.

Überdies bestätigen Versuche mit den top-50 Dokumenten der Originalanfrage gegen ChatNoir die Vermutung, dass mit noch „mehr“ Text auch noch bessere Ergebnisse erzielt werden können. Hier muss bei der Wahl der Parameter k und t zwischen Qualität und Effizienz abgewägt werden. Ebenso kann die Nutzung von einer anderen vertrauenswürdigen Suchmaschine helfen, wie z. B. Startpage oder DuckDuckGo. Im Gegensatz dazu gibt es natürlich auch zu verschlüsselnde Suchanfragen, für die sehr viele Keyqueries berechnet werden können. Es ist aufgrund der Laufzeit und der Verschleierung jedoch ungünstig, in solchen Fällen alle Keyqueries an Google zu stellen. Daher wurde die Keyquery-Abdeckung eingeführt (vgl. Abschnitt 3.2.3).

Den HBC-Algorithmus habe ich in diesem Jahr so umgestaltet, dass es anwenderfreundlich möglich ist, verschiedene Parameter umzustellen. Außerdem gelang es mir, das Verfahren zum Berechnen der Nominalphrasen weiterzuentwickeln und so die Qualität der schließlich berechneten Keyqueries einen weiteren Schritt nach vorne zu bringen. Durch diese Anpassungen und die Überarbeitung des HBC-Algorithmus, die eine noch optimalere Wahl der Parameter und Abbruchkriterien ermöglicht, hat sich das Ergebnis im Vergleich zum Vorjahr bei gleichzeitiger Verringerung der Laufzeit zugunsten der Effizienz verbessert. Es müssen weniger Anfragen gestellt werden, um eine sensible Suchanfrage zu verschleiern.
Neben direkt messbaren Auswertungsaspekten, wie dem Recall und der Bewertung in der Nutzerstudie, habe ich auch weitere Punkte bearbeitet, die für den Nutzer erkennbar, jedoch nicht direkt quantitativ erfassbar sind. Nachdem der Nutzer seine sensible Suchanfrage abgeschickt hat und die Nominalphrasen berechnet wurden, erleichtert die Neueinführung der Verschleierungsklassen die Auswahl und erspart ihm Zeit, da potenziell sensible Terme nun schon vorher automatisch ausgeschlossen werden. In Pilot-Experimenten zeigte sich erstaunlicherweise, dass das Entfernen solcher zur Originalanfrage semantisch verwandten Begriffe aus den Nominalphrasen zu keiner signifikanten Beeinflussung der Keyqueries und der Qualität der Suchergebnisse führt. Nachdem schließlich die genehmigten Keyqueries an Google gestellt wurden, sieht der Nutzer seine Suchergebnisse. Deren Ranking ist nun deutlich verbessert, da nicht mehr die Sortierung durch Häufigkeiten, sondern mithilfe der Bibliothek Lucene durchgeführt wird, die die Suchergebnisse lokal indiziert. Eine weitere Änderung betrifft direkt den ersten Eindruck, wenn der Nutzer die App startet. Diese habe ich neu gestaltet, damit sie noch nutzerfreundlicher zu bedienen und auch die Suche über einen privaten oder vertrauenswürdigen Server möglich ist. Auch im Hintergrund hat sich einiges getan. Die Nutzung der Bibliothek JSoup zum Parsen von Webtexten macht mein Verfahren jetzt vielseitiger. Im Rahmen der Nutzung meiner App stellte ich fest, dass Google (und andere Suchmaschinen) regelmäßig ihre Suchergebnisseite ändern. Um hierfür schnell und flexibel reagieren zu können, nutze ich JUnit-Tests. Diese überprüfen die Korrektheit des Verfahrens und ermöglichen so eine gezielte Anpassung.
Um Crypsor mit anderen Verfahren zu vergleichen, habe ich das Verschleierungsverfahren von Arampatzis et al. [1] implementiert. Dieses Verfahren erzeugt Verschleierungskandidaten, indem ein 16 Wörter umfassendes Sliding Window über das Dokument wandert und dabei alle Wortkombinationen innerhalb des Sliding Windows prüft. Erste Erkenntnisse aus meinem Ver-

gleich zeigen, dass das Verfahren von Arampatzis et al. durch die Kandidatenerzeugung mit dem Sliding Window den Zeit- und Rechenaufwand im Vergleich zu Crypsor deutlich erhöht. Das Verfahren von Arampatzis et al. erzeugt oft über 100 000 Verschleierungskandidaten, ohne dabei die Qualität der Verschleierung zu verbessern. Aufgrund des hohen Rechenaufwandes ist der Vergleich zu dem Verfahren von Arampatzis noch nicht abgeschlossen und wird fortgeführt.

4.2 Nutzerstudie

Eine empirische Nutzerstudie umfasste 25 Teilnehmer aus meinem Mathematikkurs. Die Nutzer konnten testweise einige der 95 zum Evaluieren genutzten sensiblen Suchanfragen mit einer selbst programmierten Android-App mithilfe meines Verschleierungsansatzes an Google stellen. Dabei konnten sie in drei Kategorien die App mit maximal zehn Punkten (bestmögliche Bewertung) und minimal null Punkten (schlechtestmöglicher Punktewert) einschätzen. Die Ergebnisse dieser kleinen Nutzerstudie sind motivierend: Der Wert für die gefühlte Verschleierungsqualität der Keyqueries zum jeweiligen Thema liegt durchschnittlich bei 7 Punkten, die erzielten Suchergebnisse wurden mit 7,6 und die benötigte Zeit sogar mit 8,4 von 10 Punkten bewertet.
Derzeit plane ich eine weiterem größere Nutzerstudie. Eine sensible Suchanfrage wird an Google und Crypsor gestellt und jeweils die top-10 Suchergebnisse gespeichert. Dem Nutzer werden nun nacheinander diese 20 Dokumente gezeigt – in einer zufälligen Reihenfolge, sodass er nicht weiß, welche Ergebnisse zu welchem Verfahren gehören. Er soll dann einschätzen, wie relevant das jeweilige Dokument im Vergleich zum Informationsbedürfnis der Suchanfrage ist. Die Punkte werden addiert und verglichen. So lässt sich eine Aussage treffen, inwiefern Crypsor trotz Verschleierung sinnvolle und ansprechende Suchergebnisse liefert. Die Nutzerstudie wird direkt in meine App integriert sein. Meinungen der Nutzer, mögliche Fehlermeldungen und durchgeführte Suchanfragen werden anonym erhoben, um die Wirksamkeit des Ansatzes in einem breiteren Umfeld auch praktisch auswerten zu können. Der Keyquery-Ansatz soll, je nach Studienergebnis, weiter angepasst werden.

4.3 Zusammenfassung und Ausblick

Während der Arbeiten an meinem Projekt ergaben sich einige Herausforderungen. Es mussten Performance und Übersichtlichkeit des Programmcodes überarbeitet, Fehler entfernt und passende Bibliotheken zur Verwendung getestet werden. Vor allem die Integration und Anpassung des HBC-Algorithmus bietet enormes Potenzial und gestaltet das Verfahren insgesamt intelligenter.

Dadurch, dass ChatNoir nur englischsprachige Einträge führt, kann die Verschleierung über diese Suchmaschine nicht für andere Sprachen, wie Deutsch, erfolgen. Dies war ein limitierender Faktor bei der Nutzung meiner Anwendung, kann aber seit diesem Jahr unter Verwendung anderer Suchmaschinen (z. B. Startpage oder DuckDuckGo) durch JSoup umgangen werden.

Als weiterer Punkt sind die verwendeten Softwarebibliotheken zu nennen. Für die von mir im

letzten Jahr benutzten Bibliotheken wurden Alternativen gesucht und geprüft. Zuerst testete ich die Library RAKE,[11] die das Extrahieren von Nominalphrasen und zugleich ein Ranking durchführt. Sie basiert auf einem Verfahren von Rose et al. [9] aus dem Jahr 2010. RAKE führte allerdings nicht zu einem befriedigenden Ergebnis. Aus diesem Grund beschloss ich, keine Bibliothek für das Bestimmen von Nominalphrasen und schließlich für das Berechnen von Schlüsselphrasen und deren Wertung zu verwenden und auch diese Programmteile als einfache Heuristiken selbst zu programmieren.
Den zunächst vielversprechend erscheinenden POS-Tagger der Stanford University,[12] beschrieben von Toutanova et al. [10], habe ich durch den Apache OpenNLP POS-Tagger ersetzt, der mit deutlich besseren Laufzeiten überzeugte. Auch KP-Miner [3], verwendet von Hagen et al. [4], schied während meiner Pilot-Experimente aufgrund einiger Ungenauigkeiten bei der Wortartbestimmung aus. Möglicherweise entstanden diese kleineren Probleme, da die Bibliothek ursprünglich für (wissenschaftliche) Abhandlungen und nicht für Webseitentexte entwickelt wurde.

Die Evaluation weite ich gerade noch aus. So lässt sich auch der angepasstere HBC-Algorithmus deutlich besser einschätzen und mit Arampatzis et al. vergleichen. Die Verwendung von VPN-Verschlüsselung sowie eine zufällige Reihenfolge der Suchanfragen bei der Evaluation sorgt für ein noch unabhängigeres Ergebnis. Abhängig von den Ergebnissen der weiteren Evaluation und der Nutzerstudie können die verschiedenen verwendeten Parameter (neu) diskutiert, gewählt und ausgewertet werden, um das Verfahren in Qualität und Effizienz weiterzuentwickeln.

Perspektivisch gesehen ist die Verschleierung von Anfragen an E-Commerce-Plattformen, wie z. B. Amazon, eine Idee, die es wert wäre, weiterzuverfolgen. Denn auch der Informationsgehalt der Suchanfragen dort kann sehr sensibel sein. Perfekt zugeschnittene Werbung durch Analyse des Verhaltens der Kunden und riesige Datenbanken, gefüllt mit persönlichen Informationen, sind nur ein Teil. Dazu kommt das so genannte Dynamic Pricing. Es ist ein Phänomen, das u. a. durch den Sucherverlauf des Nutzers Produktpreise automatisch anpasst – in den meisten Fällen wohl zu Ungunsten der Kunden.

5 Danksagung und Unterstützungsleistungen

Besonderer Dank gilt meinem Betreuer, Herrn Prof. Dr. Hagen und seinem Mitarbeiter Maik Fröbe, für das Korrekturlesen der Arbeit und die Unterstützung bei der Themenfindung 2018. Ihre hilfreichen und vielfältigen Tipps haben mir geholfen, den Fokus zu behalten und die für meine Arbeit relevante wissenschaftliche Literatur einzubeziehen.
Ich möchte mich bei meinem Betreuungslehrer, Herrn Dr. Koch, ebenfalls für das Korrekturlesen bedanken.
Außerdem danke ich meinen Eltern für die mentale Unterstützung und das Korrekturlesen.

[11] https://github.com/Linguistic/rake
[12] https://nlp.stanford.edu/software/tagger.shtml

Literatur

[1] Arampatzis, A.; Drosatos, G.; Efraimidis, P. S.: Versatile Query Scrambling for Private Web Search. In: Information Retrieval Journal 18 (4): Seiten 331–358, 2015.

[2] Bevendorff, J.; Stein, B.; Hagen, M.; Potthast, M.: Elastic ChatNoir: Search Engine for the ClueWeb and the Common Crawl. In: Proceedings of ECIR 2018, Seiten 820–824.

[3] El-Beltagy, S. R. ; Rafea, A. A.: KP-Miner: A Keyphrase Extraction System for English and Arabic Documents. In: Information Systems 34 (1): Seiten 132–144, 2009.

[4] Hagen, M.; Beyer, A.; Gollub, T.; Komlossy, K.; Stein, B.: Supporting Scholarly Search with Keyqueries. In: Proceedings of ECIR 2016. Proceedings, Seiten 507–520.

[5] Hébert, C.; Bretto, A.; Crémilleux, B: A data mining formalization to improve hypergraph minimal transversal computation. In: Fundamenta Informaticae, Seiten 415–433, 2007.

[6] lunapark: Suchmaschinenmarktanteile weltweit 2017. `https://www.luna-park.de/blog/9907-suchmaschinen-marktanteile-weltweit-2014`, Abruf: 27. Dezember 2018.

[7] Petit, A.; Cerqueus, T.; Mokhtar, S. B.; Brunie, L.; Kosch, H.: PEAS: Private, Efficient and Accurate Web Search. In: Proceedings of IEEE 2015, Seiten 571–580.

[8] Porter, M. F.: An Algorithm for Suffix Stripping. In: Program 40 (3): Seiten 211–218, 2006.

[9] Rose, S.; Engel, D.; Cramer, N.; Cowley, W.: Automatic Keyword Extraction from Individual Documents. In: Text Mining: Applications and Theory, Seiten 1–20, 2010.

[10] Toutanova, K.; Klein, D.; Manning, C. D.; Singer, Y.: Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In: Proceedings of HLT-NAACL 2003.

[11] Yu, Z.; Macbeth, S.; Modi, K.; Pujol, J. M.: Tracking the Trackers. In: Proceedings of WWW 2016, Seiten 121–132.