Typology - Von n-Grams, Graphdatenbanken und schnellem Tippen

Paul Georg Wagner
Edith-Stein-Schule Darmstadt
Darmstadt
mail@paul-georg-wagner.de

Till Speicher
Otto Hahn-Gymnasium Saarbrücken
Saarbrücken
till.speicher@gmx.de

ABSTRACT

Mit Computern verfassen wir heute Mails, erledigen Büroarbeit, chatten und speichern unsere Gedanken. Ohne die Eingabe von Text wären diese Tätigkeiten nicht denkbar. Texte schnell und effizient einzugeben, bereitet jedoch vielen Menschen Schwierigkeiten. Dies wird durch die zunehmend weiter verbreiteten kleinen Touchscreen-Tastaturen von Tabletts und Smartphones noch verschärft.

Deshalb stellen wir ein System bereit, das in der Lage ist, den angefangenen Satz des Benutzers auszuwerten und ihm weitere Eingaben vorzuschlagen, die er in seinen Text übernehmen kann. Statistiken über die Häufigkeit verschiedener Satzfragmente in deutschen Texten erhalten wir in Form von n-Grams von Google Books und übertragen sie in eine Graphdatenbank. Datenbank und Abfragen haben wir auf verschiedene Arten modelliert und anschließend mithilfe von künstlicher Intelligenz die Parameter der Modelle bestimmt.

Die Güte der verschiedenen Methoden testeten wir unter Verwendung von natürlichen Texten der Seiten blogger.de und Wikipedia.

Mit dem von uns entwickelten System sind wir inzwischen in der Lage, bei zwei vorgegebenen Buchstaben innerhalb von 50 Millisekunden das gesuchte Wort in etwa 2 von 3 Fällen auf einem der ersten fünf Plätze vorzuschlagen.

1. PROBLEMATIK

Die Interaktion mit Computern oder computerbasierten System ist heute integraler Bestandteil unseres täglichen Lebens. Zwar funktioniert ein großer Teil dieser Interaktion über Eingabemedien wie Maus oder Touchscreen, aber auch die Eingabe von Text ist weiterhin eine der wichtigsten Interaktionsformen. Sie wird benötigt um Dokumente zu erstellen, Emails zu schreiben, per Suchfunktion zu navigieren, zu Chatten und vieles mehr.

Text wird hauptsächlich über physische oder virtuelle Tastaturen (On-Screen Tastaturen) eingegeben. Bei vielen Benutzern ist die Eingabe von Text jedoch meist langsam, da sie des 10-Finger-Tippens nicht mächtig sind oder Probleme mit den sehr kleinen Tasten von Touchscreen-Tastaturen haben. Es besteht also ein Bedarf, diese Eingabe zu beschleunigen und zu erleichtern.

Laut der Statistiken der Seite *10fastfingers.com* [21] tippen Computerbenutzer durchschnittlich mit 208 Anschlägen pro Minute, was ca. 300 ms pro Anschlag, bzw. 42 Wörtern in der Minute entspricht. Demnach besteht ein deutsches Wort im Schnitt aus ca. 5 Zeichen. Wäre man nun in der Lage, ein angefangenes Wort nach dem dritten Buchstaben zuverlässig vorherzusagen, so könnte man die Eingabe der letzten 2 Buchstaben sparen, was eine Zeitersparnis von 40% bedeuten würde.

Unser Ziel ist es, innerhalb der Zeit von einem Anschlag, also ca. 300 ms, sinnvolle Vorhersagen für das nächste zu tippende Wort treffen zu können.

Im nächsten Kapitel werden wir zunächst auf den aktuellen Stand der Technik eingehen und verschiedene bereits existierende Methoden zur Beschleunigung von Texteingabe, vor allem im Bereich der Mobiltelefone, darlegen.

Danach werden wir in Kapitel 3 die eigentlich zu lösende Problematik formalisieren und unseren Lösungsansatz in Kapitel 4 mathematisch formulieren.

Im Anschluss daran werden in Kapitel 5 verschiedene Methoden zur Verbesserung der zu machenden Vorhersagen vorgestellt, wobei wir uns unter anderem die künstliche Intelligenz, sowie statistische Überlegungen zu Nutze machen.

Nach dieser theoretischen Darlegung folgt dann in Kapitel 6 eine kurze Beschreibung der praktischen Umsetzung unserer Ergebnisse, sowie diverse Tests und statistische Auswertungen der Qualität unserer Vorhersagen unter verschiedenen Bedingungen, wobei wir uns der deutschen Wikipedia als Testumgebung bedient haben.

Zum Abschluss dieser Arbeit geben wir dann in Kapitel 7 einen Ausblick auf zukünftige Möglichkeiten, die sich mit und durch unsere Vorhersagemethode ergeben.

2. STAND DER TECHNIK

Es existieren aktuell verschiedene Ansätze mit der Zielsetzung, die Eingabe von Text zu beschleunigen und zu verbessern. Ihre Herangehensweisen an das Problem unterscheiden sich dabei mitunter stark.

2.1 Textvorschläge

Verschiedene Dienste und Umgebungen bieten die Möglichkeit, Text vorzuschlagen, bevor er eingeben wird. Dazu wird die vorherige Eingabe analysiert und mit zukünftigen Eingabemöglichkeiten bzw. deren Häufigkeit verglichen, woraus sich die Wahrscheinlichkeit für den weiteren Text ableiten lässt.

Google unterstützt dieses System bei seinen Suchvorschlägen [1] und hatte auch das mittlerweile eingestellte Projekt „Scribe“ [2] ins Leben gerufen, das beim Tippen von Sätzen mögliche Folgewörter oder sogar Satzabschnitte vorschlagen sollte. Auch die Linux Shell und viele Entwicklungsumgebungen sind in der Lage, Vorschläge für die weitere Eingabe zu machen (Tab-Completion).

Viele touchbasierte Tastaturen bieten inzwischen das Vorschlagen von Wörtern oder deren Vervollständigungsmöglichkeiten an. Ein Beispiel hierfür ist die App „SwiftKey X“ [3] für Android, die eine Tastatur zur Verfügung stellt, die in der Lage ist, zutreffende Vorschläge für weitere Wörter zu liefern bzw. den bereits eingetippten Text bei Fehlerhaftigkeit zu korrigieren.

2.2 Autokorrektur

Eine weitere Methode zur Verbesserung der Schreibgeschwindigkeit ist die Korrektur des bereits eingegebenen Textes. Mithilfe einer Rechtschreibprüfung können Schreibfehler eliminiert werden. Diese Methode wird in vielen Office Anwendungen wie z.B. Microsoft Word verwendet, dabei werden oft begangene Fehler sofort korrigiert, während andere nur markiert werden. Auch viele Tastaturen für Touchscreen Geräte korrigieren die Nutzereingabe.

2.3 Eingabehilfen bei Mobilgeräten

Klassische Mobiltelefontastaturen bestehen üblicherweise aus zwölf Tasten, wovon neun mit jeweils drei bis vier Buchstaben belegt sind. Durch Mehrfachbetätigung einer Taste kann der richtige Buchstabe ausgewählt werden.

T9 [4] ist ein System zur Vereinfachung des Schreiben von Texten auf Handys, wobei eine Taste jeweils nur einmal betätigt wird. Das wahrscheinlichste Wort wird durch einen Vergleich mit einem Wörterbuch ermittelt, das die Häufigkeiten einzelner Wörter enthält.

T9 kommt heute auch bei Touchscreen-Tastaturen, z.B. als Bestandteil von Swype [5], zum Einsatz. Dabei zieht man ohne abzusetzen mit dem Finger über die Tasten, wobei die möglichen Wörter wieder per Wörterbuchvergleich ermittelt werden.

2.4 Schrifterkennung

Das Hauptwerkzeug, mit dem viele Menschen ihre Texte verfassen, ist trotz der stetig steigenden Zahl an Computern immer noch der Stift. Er funktioniert in vielen Situationen besser als die häufig unsichere Technik. Außerdem sind Menschen an das Schreiben auf Papier von ihrer Schulzeit her gewöhnt.

Um das Schreiben per Stift mit den Vorteilen der digitalen Textverarbeitung zu kombinieren, gibt es Systeme zur Handschrifterkennung. Man kann mit einem Stift oder dem Finger auf speziellen Tablets schreiben, eine Software erkennt dabei die Buchstaben und wandelt sie in digitalen Text um. Das Betriebssystem Windows [6], sowie viele Notizprogramme wie z.B. Evernote [7] bieten eine solche Erkennung.

2.5 Spracherkennung

Eine ebenfalls zunehmend wichtige Eingabemethode ist die Spracherkennung. Immer mehr Programme bieten die Möglichkeit an, gesprochene Sätze in Text umzuwandeln.

Mit der Funktion Siri [8] lassen sich auf einem iPhone Kurznachrichten diktieren, Nummern wählen und sogar Unterhaltungen führen. Auch die Steuerung von Anwendungen per Sprache ist möglich, sie wird beispielsweise von Windows unterstützt.

2.6 Motivation

Aufgrund der Vielzahl von Lösungsmöglichkeiten für das Problem der verbesserten Texteingabe, lässt sich schließen, dass dieses Thema sehr ernst genommen wird und Lösungen dringend notwendig sind.

Aus diesem Grund haben auch wir uns entschieden auf diesem Gebiet zu forschen und einen Ansatz gewählt, mit dem wir zukünftig getippte Wörter vorhersagen können.

3. DETAILS

Sei W die Menge aller deutschen Wörter.

Der Nutzer tippt einen Satz s ein, bestehend aus den Wörtern $(w_1, w_2, \ldots, w_n) \in W$. Um eine Vorhersage für das nächste Wort w_{n+1} zu erhalten, betrachten wir die Vorgängerwörter w_n, w_{n-1}, w_{n-2} und w_{n-3}. Falls bereits vorhanden, wird auch der erste Buchstabe des nächsten Wortes w_{n+1} ausgewertet.

Betrachten wir den Teilsatz *Ich gehe über die* :

Ich	gehe	über	die
w_{n-3}	w_{n-2}	w_{n-1}	w_n

Auf w_n könnten jetzt verschiedene Wörter folgen. Denkbar für w_{n+1} sind beispielsweise *Straße*, *Brücke* und *Fliesen*.

Um nun das wahrscheinlichste Wort vorzuschlagen zu können, benötigen wir eine Liste $L(w_1 : q_1, w_1 : q_2, ..., w_n : q_n)$ der möglichen Worte w mit dazugehörigen Wahrscheinlichkeiten q, bei der das relevanteste Wort w_1 an erster Stelle steht.

Um diese Liste zu erhalten darf jedoch kein Stemming [16], d.h. eine Reduktion auf den Wortstamm, angewendet werden, da Wörter vorgeschlagen werden sollen, die auch grammatikalisch passend sind.

4. ANSATZ

Wir haben uns entschieden, das Problem mit einem statistischen Ansatz zu lösen. Hierfür müssen wir möglichst umfangreiche statistische Daten über Häufigkeiten von Wörtern und deren Verbindungen untereinander evaluieren und sie in einer Weise speichern, die ein möglichst schnelles Auslesen dieser Daten ermöglicht.

4.1 Multigramme

Als Grundlage benötigen wir eine Menge an deutschen Teilsätzen.

Ein n-Gram (Multigramm) m ist ein Element einer beliebigen Sprache über dem Eingabealphabet Σ mit der Länge n, d.h $m = (w_1, w_2, \ldots, w_{n-1}, w_n) \in \Sigma^n$.
Nimmt man als Sprache deutsche Sätze und als Eingabealphabet Σ die Menge aller deutschen Wörter, so sind n-Grams der Länge $n = 4$ (4-Grams) alle Teilsätze mit genau vier Wörtern.

Damit führen wir eine n-Gram-Analyse auf Wortebene statt auf Zeichenebene, wie sie zum Beispiel bei Kryptoanalysen [20] häufig eingesetzt wird, durch.

Ein n-Gram mit $n > 1$ lässt sich auch in kleinere Einheiten zerlegen. Betrachten wir beispielsweise das 5-Gram *Ich gehe über die Straße*. Dieses lässt sich, wie in Abbildung 1 gezeigt, noch weiter aufteilen.

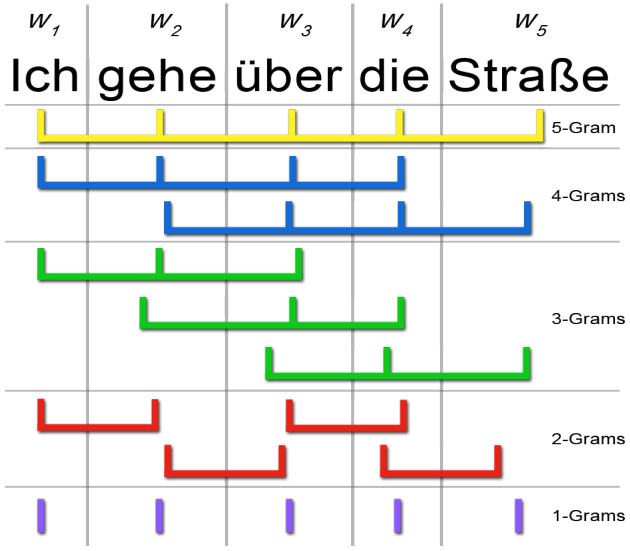

Abbildung 1: Im 5-Gram enthaltene n-Grams

Aus einem 5-Gram lassen sich demnach zwei 4-Grams, drei 3-Grams, vier 2-Grams und fünf 1-Grams bilden.

4.2 Google Books

Bei der Suche nach einer passenden Datengrundlage stießen wir auf Google n-Grams[12].

Diese beinhalten alle n-Grams mit den Häufigkeiten ihres Auftretens auf Grundlage der Google Books in einem spezifizierten Format und sind damit für uns ideal als Datengrundlage verwendbar. Allein im Jahr 2008 liefern über 23.000 Bücher fast 950 Millionen Wörter[13]. Insgesamt beläuft sich die Datenmenge der von uns analysierten 5-Grams auf ca. 80 Gigabyte.

4.3 Datenstruktur Graph

Die durch die Analyse der Google 5-Grams bestimmten Beziehungen zwischen verschiedenen Wörtern muss nun in einer Datenstruktur gespeichert werden. Hierfür eignet sich besonders der Graph. In ihm können Wörter als Knoten, sowie Beziehungen zwischen den Knoten als Kanten gespeichert werden.

Da in einem Satz die Reihenfolge der auftretenden Wörter von Bedeutung ist, muss auch Anfang und Ende einer Beziehung zwischen zwei Wörtern im Graph festgelegt werden. Der Graph ist **gerichtet**.

Außerdem muss auf den Kanten zusätzlich die Häufigkeit des Auftretens der entsprechenden Beziehung in den analysierten n-Grams gespeichert werden. Diese Eigenschaft ermöglicht die spätere Sortierung der möglichen Beziehungen nach Wahrscheinlichkeit.
Der Graph ist **gewichtet**.

Die Kanten selbst müssen zudem Auskunft darüber geben, von welcher Art die Beziehung der durch sie verbundenen Wörter sind. Verschiedene Kantentypen geben in unserem Fall an, wie weit die Wörter im analysierten Datensatz auseinander standen.
Die Kanten des Graphs sind **typisiert**.

Um unsere Datensätze in einem solchen Graph zu speichern benötigten wir eine Graphdatenbank. Dies ist eine Datenbank, welche anstatt Tabellen (relationale Datenbanken) die Datenstruktur des Graphen nutzt, um Datensätze zu speichern. Wir entschieden uns für die in Java implementierte Graphdatenbank Neo4J[14], die alle oben gestellten Anforderungen erfüllt.

5. VORGEHENSWEISE

5.1 Datenbankmodellierung

5.1.1 Aufbau

Nach der im vorigen Kapitel beschriebenen Methode lässt sich aus den Google 5-Grams ein gerichteter, gewichteter Eigenschaftsgraph konstruieren. Dazu müssen wir jeweils die Beziehung zwischen w_1, w_2, w_3, w_4 und w_5 in jedem 5-Gram m_5, untersuchen und als Datensatz in der bereits beschriebenen Graphdatenbank speichern.

Wir zerlegen das 5-Gram kombinatorisch weiter in 4-, 3-, und 2-Grams, um alle möglichen Beziehungen zwischen genau zwei Wörtern im 5-Gram in eine Kante des entsprechenden Typs umzuwandeln. Hierbei entspricht ein 2-Gram einer Kante des Typs 1, ein 3-Gram einer Kante des Typs 2, ein 4-Gram einer Kante des Typs 3 und ein 5-Gram einer Kante des Typs 4. Diese Zerlegung des 5-Grams ist beispielhaft in Abbildung 2 dargestellt.

Nach dieser Vorarbeit können die einzelnen Wörter des 5-Grams als Knoten, sowie die verschiedenen Beziehungen als Kanten in der Datenbank gespeichert werden, falls sie noch

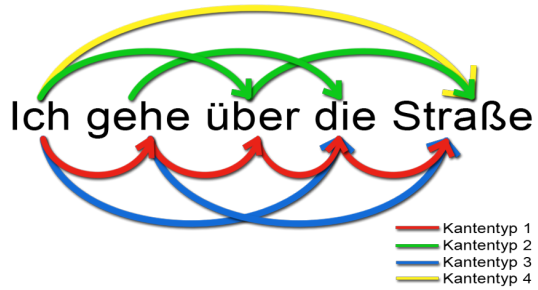

Abbildung 2: Aus einem 5-Gram resultierende Kanten

nicht existieren. Wenn eine solche Kante des entsprechenden Typs bereits existiert, wird einfach ein Zähler, der auf der Kante gespeichert wird, inkrementiert. Auf diese Weise sind die Gewichte der Kanten zunächst die Häufigkeiten des Vorkommens dieser Wortverbindung.

Ein Ausschnitt des so erstellten Graphen ist schematisch in Abb. 3 dargestellt.

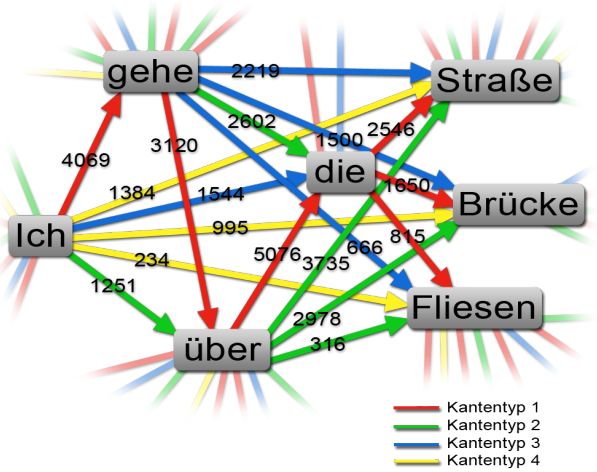

Abbildung 3: Schematischer Ausschnitt des Graphen

Wort	Ich	gehe	über	die	Ranking
Straße	1384	2219	3735	2546	9884
Brücke	995	1500	2978	1650	7123
Fließen	234	666	316	815	2031

5.1.2 Formalisierung

Die Zusammenhänge in einer so erstellten Datenbank lassen sich formalisiert ausdrücken, wobei wir im Folgenden mit einem Wort w auch den entsprechenden Knoten bezeichnen wollen.

Sei K die Menge aller Kanten.
Sei $k = (w, v) \in K$ eine Kante vom Wort w zum Wort v.

Sei $K_n \subseteq K$ die Menge aller Kanten des Typs $n \in \mathbb{N}$ mit $1 \leq n \leq 4$.

Dann gilt: $K_n = \{k \in K | k = (w_u, w_{u+n})\}$, wobei $u \in \mathbb{N}$ und $u + n \leq 5$

Auf diese Weise lassen sich auch Nachbarn eines bestimmten Typs angeben:

Sei $Nachbar(w, n)$ die Menge aller Nachbarn des Wortes w vom Typ n.
Dann gilt: $Nachbar(w, n) = \{v \in W | \exists k \in K_n \text{ mit } k = (w, v)\}$

Durch diese Vereinbarungen sind wir in der Lage, mit Hilfe der Kantengewichte die Güte eines Wortvorschlages anzugeben. Dabei soll zunächst gelten: je höher die Güte, desto wahrscheinlicher ist das Wort.

Sei $G(k)$ das Gewicht der Kante $k \in K$.
Sei $P(w, s)$ die Güte eines Wortes w als Nachfolger des bereits vorhandenen Teilsatzes s.

Im einfachsten Fall ist sie die Summe der Gewichte der entsprechenden Kanten zum Wort w_n:

Sei $s = (w_{n-4}, w_{n-3}, w_{n-2}, w_{n-1})$
und $k_i = (w_{n-i}, w_n) \in K_i$

Dann definieren wir: $P(w_n, s) := \sum_{i=1}^{4} G(k_i)$.

5.2 Normierte Kantengewichte

Obige Methode zur Berechnung der Güte eines Wortvorschlags arbeitet mit der tatsächlichen Anzahl der Wortbeziehungen als Kantengewichte, also absoluten Größen. Zum Vergleichen der Vorschläge eignen sich jedoch besser relative Werte. Diese relativen Werte erhalten wir durch Normierung der Kantengewichte.

5.2.1 Globale Normierung

Bei der globalen Normierung werden alle absoluten Kantengewichte $G(k \in K)$ durch das globale Maximum $\max(G(k \in K))$, also das maximale Gewicht aller Kanten, geteilt.

Dies hat zur Folge, dass sich für häufige Kanten hohe Werte nahe an der 1 ergeben, während seltene Kanten sehr kleine Gewichte von annähernd 0 erhalten.

Die Güte $P(w, s)$ ändert sich bei Verwendung der globalen Normierung relativ gesehen nicht, da sämtliche Gewichte lediglich durch einen festen Wert geteilt wurden.

5.2.2 Lokale Normierung

Bei der lokalen Normierung definieren wir zunächst für jedes Wort $w \in W$ das lokale Maximum
$m_{w,n} = max(G(k) | k = (w, v) \wedge w \in Nachbar(w, n))$, also das maximale Gewicht der von einem Knoten w ausgehenden Kanten des Typs n. Nun können die Kanten des Typs n ausgehend vom Knoten w durch das jeweilige lokale Maximum $m_{w,n}$ geteilt, und so lokal normiert werden.

Dadurch erhält man pro Wort und Typ jeweils eine Kante mit dem Gewicht 1, unabhängig davon, wie oft oder selten

diese Kante gemessen an der Gesamtmenge der Kanten K vorkommt.

5.2.3 Beispiel

Nehmen wir den Teilsatz *Er kapituliert vor dem* , so ist *kapituliert* in diesem Satz das am wenigsten gebräuchliche Wort, das aber gleichzeitig im Vergleich zu häufigeren Wörtern viel bessere Rückschlüsse auf das nächste Wort ermöglicht.

Wären die Kanten nun global normiert, so hätten die Verbindungen zu *kapituliert* eine viel niedrigere Gewichtung, und die mit ihm verbundenen Wörter würden nur schwach berücksichtigt werden. Bei einer lokalen Normierung hängt die Gewichtung dagegen lediglich von den Verbindungen zu dem Wort selbst ab.

Für die Vorhersage des nächsten Wortes bedeutet dies, dass bei der globalen Variante Kanten zu Wörtern, die insgesamt sehr häufig vorkommen, immer stark berücksichtigt und deshalb bevorzugt vorgeschlagen werden. Bei der lokalen Normierung dagegen ist dies auch für seltenere Wörter der Fall. Unsere Vermutung ist daher, dass die Berechnung der Güte aufgrund lokal normierter Gewichtung zu besseren Ergebnissen führt.

5.2.4 Summierte Normierung

Eine besondere Art der lokalen Normierung ist die summierte Normierung. Hierbei werden die Kanten nicht durch ihr lokales Maximum, sondern durch die Summe
$\sum G(k)|k = (w, v) \wedge w \in Nachbar(w, n)$ aller lokalen Kantengewichte geteilt.

Im Gegensatz zur lokalen Normierung erhält man hier nicht zwingend eine Kante mit dem Gewicht von 1 als Maximum, dafür werden bei dieser Normierung sehr häufige Beziehungen stärker gewichtet als bei der lokalen Normierung. Allerdings geht dies durch die Aufsummierung zu Lasten der Vergleichbarkeit einzelner Kantengewichte.

5.3 Parametrisierte Gewichte

Beim bisherigen Verfahren der Berechnung der Güte $P(w, s)$ werden die (normierten) Gewichte aller Kanten $k \in K$ gleich behandelt. Diese Vorgehensweise verfälscht jedoch die Berechnung der Güte, da die Anzahl der Kanten mit steigendem Typ n abnimmt:

$$|K_1| > |K_2| > |K_3| > |K_4|$$

Dies liegt daran, dass die Anzahl der n-Grams pro Teilsatz mit fallendem n zunimmt (siehe Kap. 4.1).

Um dies zu kompensieren, müssen wir zur Berechnung der Güte $P(w, s)$ die Gewichte der einzelnen Kanten noch mit einem entsprechenden Parameter p_1 bis p_4 multiplizieren.

Demnach können wir die gewichtete Güte wie folgt definieren:
$P_{gew}(w_n, s) := \sum_{i=1}^{4} p_{n-i} * G(k_i)$

Allerdings ist nicht klar, wie dominant die einzelnen Kanten für optimale Ergebnisse sein müssen, da beispielsweise direkte Vorgänger, also niedrige Kantentypen, häufig mehr über das Folgewort aussagen können.

5.4 Gelernte Parameter

Um zu ermitteln, wie stark die einzelnen Kanten berücksichtigt werden müssen, berechnen wir die Parameter p_n für den jeweiligen Kantentyp n. Dazu gehen wir auf zwei verschiedene Arten vor.

5.4.1 Genetischer Algorithmus

Genetische Algorithmen[17] sind Methoden, welche auf der evolutionären Entwicklungstheorie nach Darwin basieren. Sie werden bei Optimierungsproblemen eingesetzt, bei denen kein, bzw. kein effizienter Lösungsalgorithmus bekannt ist.

Zu Beginn der Berechnung wird eine *Population*
$A_n = (i_1, i_2, \ldots, i_n)$, bestehend aus n zufällig erzeugten Lösungen des Problems, *Individuen*, erstellt. In unserem Fall sind die Lösungen des Problems verschiedene mögliche Parameter.

Nun wird für alle Individuen mit Hilfe einer *Fitnessfunktion* eine Bewertung erstellt, welche die Qualität der entsprechenden Parameter angibt. Die Individuen mit den besten Bewertungen werden *rekombiniert*, d.h zwei Individuen der Parentalgeneration vererben einen zufällig ausgewählten Teil ihrer Parameter an ein neues Individuum in der Filialgeneration. Durch *Mutation*, also einer zufälligen Veränderung der Parameter mit einer gewissen Wahrscheinlichkeit, entstehen auch ganz neue Parameter. Auf diese Weise wird die nächste Generation an Individuen ermittelt und die Schritte können wiederholt werden.

Im Optimalfall erhält man nach einer gewissen, aber schwer vorherzusagenden Anzahl an Generationen Individuen, die sich auf lange Sicht nicht mehr relevant verbessern.

5.4.2 Monte-Carlo-Simulation

Die Monte-Carlo-Simulation [15] ist eine statistische Methode bei der man Werte mithilfe von sehr oft durchgeführten Zufallsexperimenten bestimmt. Man macht sich das „Gesetz der großen Zahlen“ zunutze, d.h. man geht davon aus, dass der Mittelwert der Menge aller Ergebnisse gegen den gesuchten Wert konvergiert, wenn man das Zufallsexperiment häufig genug durchführt.

In unserem Fall bedeutet dies, dass wir die Parameter $P = (p_1, ..., p_4)$ mit $p \in \mathbb{R}, 0 \leq p \leq 1$ ermitteln, indem wir sehr oft verschiedene Zufallszahlen für jeden Parameter generieren, sie mit den Kanten k_{n-i} multiplizieren und so das Gewicht $G(k_{n-i})$ der jeweiligen Kante erhalten. Wie sich diese Gewichte auf die Qualität der vorgeschlagenen Wörter auswirken, ermitteln wir, indem wir sie an natürlichen Sätzen testen und die Position pos_i des gesuchten Wortes w_n der Ergebnisliste L bestimmen.

Wir betrachten nun x Sätze und wiederholen oben genannte Operation mehrfach für jeden Satz s_i. Dabei schauen wir uns die Länge des Intervalls $I = pos_{i\_min} - pos_{i\_max}$ an. Seine Größe lässt Rückschlüsse darauf zu, wie stark bestimmte Parameter das Ergebnis verbessern bzw. verschlechtern. Je größer das Intervall ist, desto besser sind die Parameter, die zu seiner oberen Grenze pos_{i_max} geführt haben im Vergleich zu denen für die Untergrenze pos_{i_min}. Nun speichern wir die Intervalllänge zusammen mit den Parametern

$P_i = (p_{i,1}, ..., p_{i,4})$, die für diesen Satz s_i das beste Ergebnis liefern.

Die finalen Parameter werden nun auf folgende Weise gebildet:
$P_final = \sum_{i=1}^{x} P_i * Laenge(I)$.

Da wir sehr viele zufällige Werte für jedes P_i generieren, die Qualität der resultierenden Vorschläge bestimmen und bei der Addition zu den finalen Parametern P mit der Intervalllänge $Laenge(I)$ multiplizieren, wir also bessere Werte stärker berücksichtigen, soll das Ergebnis hin zu den idealen Parametern konvergieren, die zu bestmöglichen Vorschlägen führen.

5.5 Vorhersagen

Gegeben sei ein Teilsatz $s = (w_1, w_2, w_3, w_4)$.
Gesucht ist ein möglichst optimales Nachfolgewort w_5.

Allgemein lässt sich das Vorhersagen eines Wortes als Berechnung der Güte $P_{gew}(w_5, s)$ für jedes mögliche Nachfolgewort w_5, das eine Verbindung (Kante) zu mindestens einem der Vorgängerworte in s besitzt, auffassen.

Algorithmus zur Berechnung von $P_{gew}(w_5, s)$:

```
Für n von 1 bis 4
Beginn

  Für jede Kante k mit Typ 5-n ausgehend von n-tem Wort
  Begin
    Folgewort f = Endknoten von k
    Gewicht g = n-ter Parameter * Kantengewicht von k

    Speichere f,g in Datenstruktur
  Ende

Ende
```

Diesen Algorithmus implementierten wir in Java. Als Datenstruktur wählten wir eine TreeMap, da diese die Daten in einem Binären Baum speichert und damit eine logarithmische Laufzeit $\mathcal{O}(n * log(n))$ beim Sortieren garantiert[19].

Wenn beim Speichern eines möglichen Nachfolgeworts als Index, sowie des dazugehörigen Gewichts als Wert, der Index bereits existiert, wird das zusätzliche Gewicht einfach zum Wert addiert. Auf diese Weise erreichen wir die Summierung der parametrisierten Kantengewichte für $P_{gew}(w, s)$.

Falls für das Nachfolgewort w_5 bereits ein oder mehrere Buchstaben bekannt sind, kann die Speicherung von Wortvorschlägen natürlich auch entsprechend gefiltert werden. Dies verbessert nicht nur die Vorhersagegenauigkeit, sondern auch die Geschwindigkeit (siehe Kap. 6.4).

6. ERGEBNISSE

Zur Überprüfung unserer Vermutungen aus dem vorigen Kapitel erstellen wir zunächst gemäß unserer Modellierung eine Datenbank. Aus großen Mengen an natürlichsprachlichem Text wählen wir Teilsätze aus und erstellen mit unterschiedlichen Verfahren und Methoden Vorschläge für ein Nachfolgewort (Evaluation). Die gemachten Vorschläge können wir dann auswerten, da wir das tatsächliche Nachfolgewort des Teilsatzes kennen.

Der gesamte Quellcode des Projektes ist verfügbar unter unserer Google Code Seite[27].

6.1 Datenbankerstellung

Um die Datenbank zu erstellen, müssen zunächst die nach Jahren getrennt gespeicherten Google 5-Grams zusammengefasst und von Interpunktion und Sonderzeichen befreit werden. Durch diesen Prozess erhalten wir eine ca. 500 MB große Datei, welche die reinen statistischen Daten enthält, die wir dann in unsere Datenbank übertragen.

Eine auf diese Weise erstellte Datenbank enthält ca. 205.000 Knoten und ca. 7,7 Mio. Kanten. Von jedem Knoten gehen dabei im Durchschnitt 37,9 Kanten aus (Knotengrad).

Zum Vergleich: Im Duden (2009) stehen ca. 135.000 Wörter. Diese Diskrepanz liegt darin begründet, dass wir kein Stemming betreiben, also nicht nur den grammatikalischen Wortstamm betrachten, sondern z.B. auch konjugierte und deklinierte Wörter seperat behandeln.

6.2 Testdatensätze

6.2.1 *blogger.de*

Mit Hilfe eines Webcrawlers [11] sammelten wir ca. 150 MB an Textdaten von Blogs der Seite blogger.de. Diese Textdaten bestehen vor allem aus Alltags- und Websprache. Allerdings bedingt dies auch den relativ hohen Anteil an orthographischen und grammatikalischen Fehlern.

6.2.2 *Wikipedia*

Die Datenbank des Wikipedia-Projekts ist als Download[9] frei verfügbar und beinhaltet eine große Menge an sprachlich hochwertigem und mit wenig Fehlern versehenen Text zu vielen verschiedenen Themen.

Allerdings müssen wir die für uns nutzbaren, reinen Textdaten zunächst noch aus dem im XML-Format vorliegenden Datensatz extrahieren. Dazu muss dieser zurück in die Wiki-Formatierung konvertiert werden, wobei wir uns des Tools *wikixmlj*[10] bedient haben. Anschließend werden Umlaute in normale Buchstaben umgewandelt und nicht zu gebrauchende Textteile (z.B. Navigation und Tabellen), sowie alle Sätze mit weniger als fünf Wörter entfernt.

Diese Aufgaben erledigt ein Parser, der den gesamten 12 Gigabyte großen Wikipedia-Datensatz durchläuft, welcher alle Artikel, Beschriftungen und Diskussionsseiten enthält. Er produziert dabei eine 4,4 Gigabyte große Textdatei, die ca. 48 Mio. für uns verwertbare Sätze enthält.

6.3 Parameter Lernen

6.3.1 *Genetischer Algorithmus*

Den Ansatz des Genetischen Algorithmus zum Bestimmen der optimalen Parameter führten wir mit einer Populationsgröße von $n = 50$ Individuen durch. Ein Individuum $i = (p_1, p_2, p_3, p_4)$ besteht dabei aus zufällig ausgewählten

Startparametern p_n für die einzelnen Kantentypen mit $0 \leq p_n \leq 1$.

Die Rekombination der Individuen geschah mittels eines einfachen Crossing-Overs, d.h. die Parameter des Kindes setzen sich aus den *pos* vorderen Parametern des einen Elternteils und den $4-pos$ hinteren Parametern des anderen Elternteils zusammen, wobei $pos \in \mathbb{N}$ mit $1 \leq pos \leq 4$ eine zufällige Position ist. Sind beispielsweise die beiden Elternindividuen $i_{f1} = (0.2, 0.5, 0.3, 0.01)$ und $i_{f2} = (0.9, 0.6, 0.04, 0.7)$, sowie $pos = 3$, so entsteht das Kind-Individuum $i_c = (0.2, 0.5, 0.3, 0.7)$.

Die Fitnessfunktion berechnete die Qualität der Parameter anhand von 1000 Sätzen aus unserem Wikipedia-Datensatz, für die mit den entsprechenden Parametern Vorhersagen getroffen wurden. Die Qualität selbst bestimmten wir mittels eines Cumulative Gains[18], wobei die inverse Position des tatsächlichen Wortes in der Vorschlagsliste für jeden überprüften Satz aufsummiert wird:

$CG = \sum_{i=1}^{1000} \frac{1}{pos_i}$

Je höher der CG, desto besser sind unsere Vorschläge im Mittel.

Allerdings müssen bei diesem Ansatz in der Fitnessfunktion pro Generation 50.000 Sätze evaluiert werden, wobei die Zahl von 1000 getesteten Sätzen pro Individuum bereits so klein wie möglich gehalten wurde. Die aus diesem Grund geringe Geschwindigkeit der Generationsfolge führte dazu, das mit Hilfe der genetischen Algorithmen alleine keine starken Verbesserungen gegenüber einer unparametrisierten Vorhersagemethode.

Wir konnten jedoch beobachten, dass ein Werte nahe 0 für den Parameter p_4, also das Gewicht des Kantentyps 4, keine relevante Beeinflussung des CGs zur Folge hat. Daher vermuten wir, dass die Auswertung des Kantentyps 4 während der Datenbankerstellung auf die Qualität unserer Vorschläge keinen, bzw. nur unwesentlichen Einfluss hat.

6.3.2 Monte-Carlo-Simulation

Bei der Monte-Carlo-Simulation bestimmen wir Näherungswerte der optimalen Parameter, indem wir aus dem Wikipedia Datensatz zufällig Teilsätze s_i mit 5 Wörtern auswählen und anschließend zu jedem Satz mehrfach beliebige Werte für die Parameter P_i generieren. Wir bilden das Intervall I zwischen der besten und der schlechtesten Positionierung pos_i des gesuchten Wortes w_n in der Ergebnisliste L, basierend auf den generierten Parametern.

Dabei werden solange neue Parameter gebildet, bis das Intervall I dreimal hintereinander seine Länge nicht mehr verändert, da weitere Parameter in diesem Fall das Ergebnis offensichtlich nicht mehr beeinflussen. Maximal werden für einen Teilsatz 100 verschiedene Parameterkombinationen generiert.

Anschließend werden die Parameter, die zur besten Vorschlagsliste L geführt haben, mit der Länge des Intervalls multipliziert, aufsummiert und dann durch die Anzahl der betrachteten Teilsätze dividiert. Die daraus entstandenen Parameter werden anschließend normiert, um Werte zwischen 0 und 1 zu erhalten.

Ausgehend von dieser Methode erhielten wir bei einer Evaluation von 140.000 Teilsätzen aus unserem Wikipedia-Datensatz die Werte 1.0, 0.79, 0.73 und 0.71 für die Parameter $P = (p_1, p_2, p_3, p_4)$.

Messungen mit diesen Parametern und einem vorgegebenen Buchstaben führen, wie in Kapitel 6.5.2 gezeigt, in 22,3% der Fällen dazu, dass das gesuchte Wort auf Platz eins vorgeschlagen wird.

6.3.3 Manuelle Parameter

Die vorgestellten Methoden zur Ermittlung der optimalen Parameter für unseren Vorhersagealgorithmus haben verschiedene Anhaltspunkte für das Verhältnis der einzelnen Parameter zueinander geliefert. Es ist jedoch schwierig, mit ihnen ein optimales und praktisch einsetzbares Ergebnis zu erhalten. Daher haben wir basierend auf den obigen Ergebnissen manuell die Qualität verschiedener möglicher Parameterverhältnisse getestet.

Dabei erreichte die Parameterverteilung 5, 2, 2 und 0 für die Parameter der Kantentypen 1 bis 4 das beste beobachtete Ergebnis. Unsere Vermutung aus Kap. 6.3.1, dass der Kantentyp 4 keinen besonderen Einfluss auf die Qualität der Vorhersagen hat, scheint damit zunächst bestätigt.

Wir verwenden daher bei der Parametrisierung diese Parameter.

6.4 Vorhersagegeschwindigkeit

6.4.1 Beeinflussung durch bekannte Buchstaben

Für 250.000 vorhergesagte Wörter mit einem bekannten Buchstaben benötigen wir auf einem aktuellen Desktop-PC (Zweikernprozessor, 3 GHz) ca. 4 Stunden. Daraus ergibt sich ein durchschnittlicher Zeitaufwand von ca. 60 ms für eine Vorhersage.

Wird kein Buchstabe vorgegeben, so steigt die Zeit für eine Vorhersage auf ca. 300-400 ms an. Dies liegt maßgeblich an der steigenden Zeit zur Sortierung der Vorschlagsliste (siehe Kap. 5.5), welche ohne vorgegebenen Buchstaben in den meisten Fällen ca. 10 bis 15 Tausend mögliche Nachfolgeworte beinhaltet.

Mehr als zwei vorgegebene Buchstaben verkürzen die Zeit jedoch nur unwesentlich, da durch die Verkleinerung der Vorschlagsliste nicht mehr die Sortierung, sondern das Filtern der Liste die zeitaufwändige Operation ist. Beim Filtern der Vorschlagsliste muss aber stets durch die ursprüngliche, große Liste iteriert werden, unabhängig von der genauen Zahl der vorgegebenen Buchstaben.

6.4.2 Beeinflussung durch Kantenanzahl

Neben der Erhöhung der Vorhersagegeschwindigkeit durch vorgegebene Buchstaben ist auch die Anzahl der für die Vorhersage überprüften Kanten von Bedeutung.

Wie in Kapitel 6.3.1 bereits vermutet, könnte der Kantentyp 4 nur unwesentlichen Einfluss auf die Güte der Ergebnisse haben. Beschränkt man die Überprüfung der Kanten auf

die Typen 1 bis 3, so ergibt sich bei einem vorgegebenen Buchstaben ein durchschnittlicher Zeitaufwand von ca. 40 ms pro Vorhersage, und damit eine Verbesserung von ca. 30%. Diese Verbesserung ist damit zu erklären, dass zum einen weniger Kanten selbst überprüft werden müssen, zum anderen wird auch die Vorhersageliste nochmals kleiner, da Wörter, welche nur über eine Kante des Typs 4 mit einem möglichen Nachfolgewörtern verbunden waren, wegfallen.

Die Messdaten sind übersichtlich in Abb. 4 zu sehen.

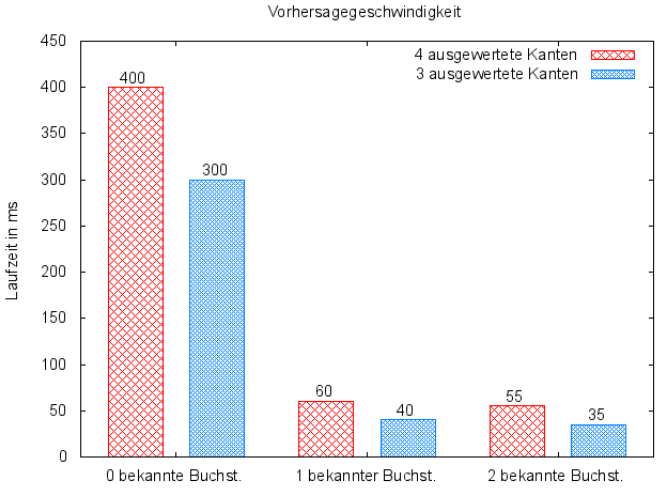

Abbildung 4: Vergleich Vorhersagegeschwindigkeit bei 1, 2 und 3 bekannten Buchstaben, 250.000 getestete Sätze

6.4.3 Bemerkung

Da wir in unserer Testumgebung anfangs nur die Ergebnisse, nicht aber die Laufzeit untersuchen wollten, wurden die Vorhersagen bei den oben dargelegten Zeiten nach zwei Methoden gleichzeitig (lokal und global normiert, siehe 5.2) berechnet.

Eine komplette Änderung des Quelltextes zur korrekten Berechnung von nur einer Methode hätte zu viel Zeit beansprucht, daher sind die ermittelten Zeiten nicht vorbehaltlos zu verwenden. Da beide Methoden vermutlich etwa ähnliche Laufzeiten besitzen, ist in einer optimierten Anwendung theoretisch noch mit einer Steigerung der Geschwindigkeit von ca. 50% zu rechnen.

Dadurch ist insgesamt aber davon auszugehen, dass wir selbst ganz ohne vorgegebenen Buchstaben die Zielsetzung von ca. 300 ms pro Vorhersage (siehe Kap. 1) erreichen werden.

6.5 Genauigkeitsüberprüfung

Zur Überprüfung der Genauigkeit unserer Vorhersage werteten wir die Vorschläge für 250.000 zufällig ausgewählte Teilsätze aus unserem Wikipedia-Datensatz aus, indem wir für jeden Satz die Position des tatsächlichen Nachfolgewortes in unserer Vorschlagsliste evaluierten. Die Messung und Überprüfung der Verbesserung unserer Vorschläge durch die Anwendung unterschiedlicher Verfahren erfolgt nach zwei verschiedenen Methoden.

Zunächst betrachten wir lediglich die Häufigkeit des richtigen Wortes auf den oberen Plätzen (Top 5 und Position 1) der Vorschlagsliste. Dann wird durch die Berechnung der durchschnittlichen Position des tatsächlichen Wortes in der Vorschlagsliste die Genauigkeit der Vorhersagen näher beleuchtet.

6.5.1 Normierung

Durch jeweils eine Evaluation mit global, lokal und summiert normierten Daten werden die Auswirkungen der Normierung auf die Vorhersagen getestet.

Mit nur einem vorgegebenen Buchstaben des tatsächlichen Nachfolgewortes waren ca. 19% der Vorhersagen bei der globalen Normierung, sowie ca. 21% der Vorhersagen bei der lokalen Normierung und ca. 22% bei der summierten Normierung richtig (siehe Abb. 5).

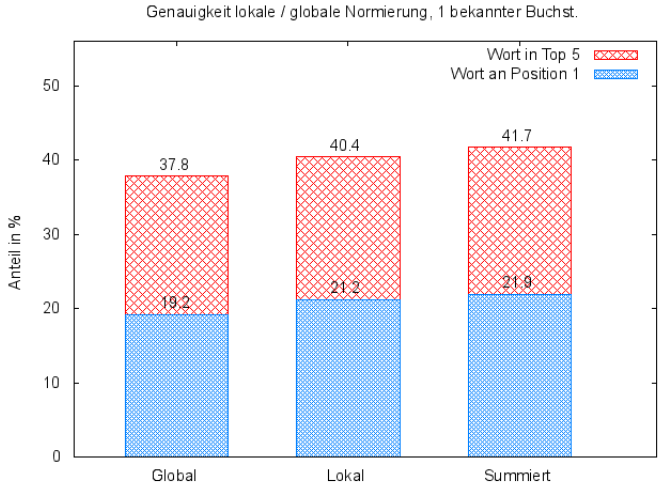

Abbildung 5: Vergleich globale/lokale/summierte Normierung, unparametrisiert, 1 bekannter Buchstabe, 250.000 getestete Sätze

Die lokale Normierung verbessert demnach die Erkennung im Mittel um 2 Prozentpunkte bei einem bekannten Buchstaben, die summierte sogar um 3%.

Stellt man 2 Buchstaben des tatsächlichen Wortes zur Verfügung, steigt die Vorhersagerate auf 27% bzw. 29%, wobei die lokale Normierung wieder die besseren Ergebnisse zeigt (siehe Abb. 6). Die summierte Normierung ist etwa gleich gut wie die lokale.

Betrachten wir bei beiden Versuchen die Häufigkeit der Positionierung des tatsächlichen Wortes bis zum fünften Platz der Vorhersageliste, so stellen wir fest, dass diese Werte im Vergleich der Verfahren um die selben Werte ansteigen, wie bei isolierter Betrachtung der richtigen Vorhersagen auf Position 1. Es ist also kein Einfluss der Normierungen auf die Plätze zwei bis fünf in der Vorhersageliste feststellbar.

Dies liegt daran, dass durch die unterschiedlichen Normierungen die Plätze einzelner Vorschläge insgesamt leicht zum

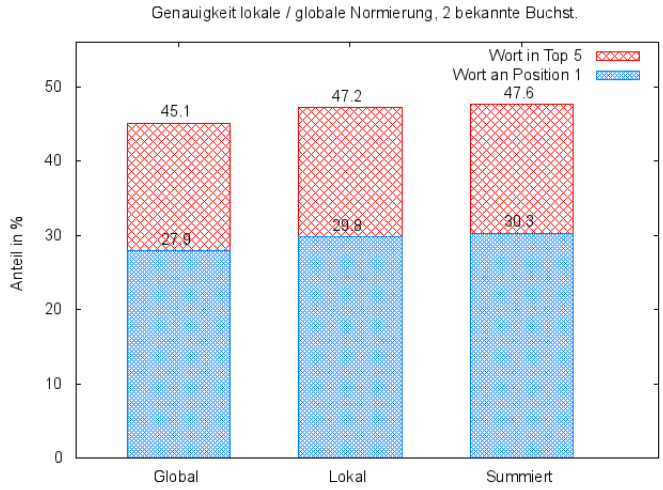

Abbildung 6: Vergleich globale/lokale/summierte Normierung, unparametrisiert, 2 bekannte Buchstaben, 250.000 getestete Sätze

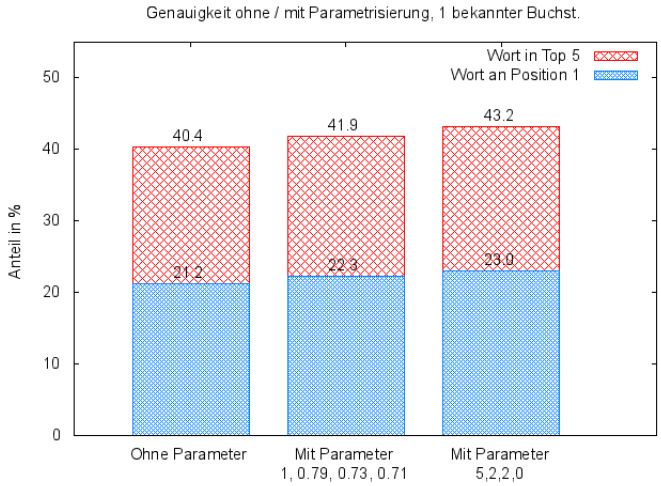

Abbildung 7: Vergleich Genauigkeit ohne / mit Parametrisierung, lokale Normierung, 1 bekannter Buchstabe, 250.000 getestete Sätze

besseren beeinflusst werden, jedoch nur selten Wörter viele Plätze nach oben oder unten rutschen. Über eine Spanne von 4 Plätzen gesehen, nimmt die Beeinflussung durch die lokale bzw. summierte Normierung daher ab.

Dennoch wurde unsere anfängliche Vermutung (siehe Kap. 5.2) durch diesen Versuch bestätigt. Die summierte Normierung ist nur anfänglich bei einem bekannten Buchstaben leicht besser als die lokale Normierung, doch kann sie weniger gut zum Vergleichen einzelner Beziehungen verwendet werden. Wir ziehen daher die lokale Normierung für unser Verfahren vor, und verwenden bei den nachfolgenden Versuchen stets diese.

6.5.2 *Parametrisierung*

Nach dem Test zur Normierung führten wir wiederum zwei Evaluationen durch, nun jeweils einmal mit und ohne Parametrisierung. Als Parameter verwendeten wir zunächst die durch die Monte-Carlo-Simulation (Kap. 6.3.2) erhaltenen Parameter 1, 0.79, 0.73, 0.71, sowie die in Kap. 6.3.3 bestimmten Parameter 5, 2, 2, und 0 für die Kantentypen 1 bis 4.

Aus Abb. 7 ist ersichtlich, dass die Häufigkeit der richtigen Vorhersagen mit einem bekannten Buchstaben durch die Parametrisierung im Vergleich zur unparametrisierten Vorhersage nochmals um ca. 2 bis 3 Prozentpunkte ansteigt. In Bezug auf die richtigen Vorschläge ist die Verbesserung durch die Parametrisierung also etwa so groß wie die lokale Normierung. Die durch die Monte-Carlo-Simulation erhaltenen Parameter sind wie erwartet etwas schlechter als die manuell bestimmten.

Betrachtet man jedoch nicht nur die vorderen Plätze der Vorhersageliste, sondern nun auch die durchschnittliche Position der richtigen Wörter in der Vorhersageliste bei einem bekannten Buchstaben, so wird deutlich, dass die Parametrisierung die durchschnittliche Position des tatsächlichen Wortes in der Vorschlagsliste um ca. 12,5% von 63,9 auf 51,5 senkt (Abb. 8). Die Parametrisierung durch die Parameter der Monte-Carlo-Simulation zeigt hier allerdings keine relevante Verbesserung.

Die bestimmten Parameter verbessern damit die Genauigkeit der gesamten Vorhersage um durchschnittlich etwa 13 Positionen, wobei die Rate der richtigen Vorhersagen um ca. 2 Prozentpunkte ansteigt. Neben der leichten Verbesserung der Genauigkeit hat die Parametrisierung durch Wahl des Parameters 0 für den Kantentyp 4 noch einen weiteren Vorteil, denn die Überprüfung dieser Kante kann somit auch gänzlich unterbleiben. Dadurch erzielen wir neben der leichten Verbesserung der Qualität der Vorschläge noch eine erhebliche Geschwindigkeitssteigerung (siehe Kap. 6.4).

Bei nachfolgenden Versuchen werden wir daher stets die Parameter 5, 2, 2 und 0 verwenden. Allerdings sei darauf hingewiesen, dass diese Parametrisierung nicht das Optimum an Genauigkeit und Geschwindigkeit bedeuten muss. Durch weitere Versuche des Parameterlernens können auch noch bessere Parameter gefunden werden.

6.5.3 *Bekannte Buchstaben*

Nach der Verbesserung der Vorhersage durch Normierung und Parametrisierung verglichen wir die nun erreichte Genauigkeit mit einem, zwei und drei vorgegebenen Buchstaben.

Aus Abb. 9 ist zu erkennen, dass wir nun mit lediglich zwei bekannten Buchstaben des Folgewortes im Mittel in mehr als jedem zweiten Teilsatz den Vorschlag in den Top 5, und in jedem dritten Teilsatz das Wort sogar korrekt vorhersagen können. Die Häufigkeit einer richtigen Vorhersage steigt durch zwei statt einem bekannten Buchstaben um ca. 10 Prozentpunkte.

Es ist ebenfalls ersichtlich, dass wir das Wort häufiger kor-

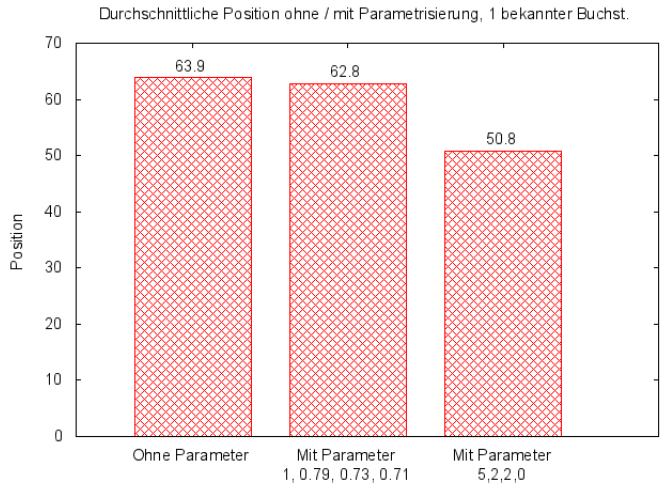

Abbildung 8: Vergleich
Durchschnittliche Position ohne / mit Parametrisierung, lokale Normierung, 1 bekannter Buchstabe, 250.000 getestete Sätze

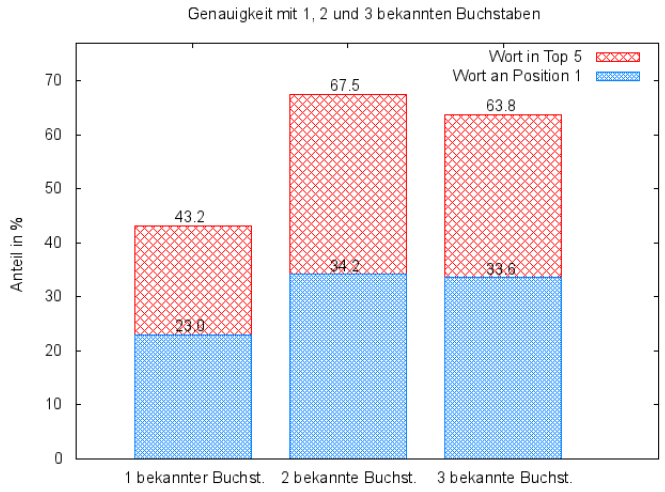

Abbildung 9: Vergleich
Genauigkeit mit 1, 2 und 3 bekannten Buchstaben, lokale Normierung,
Parameter 5,2,2,0, 250.000 getestete Sätze

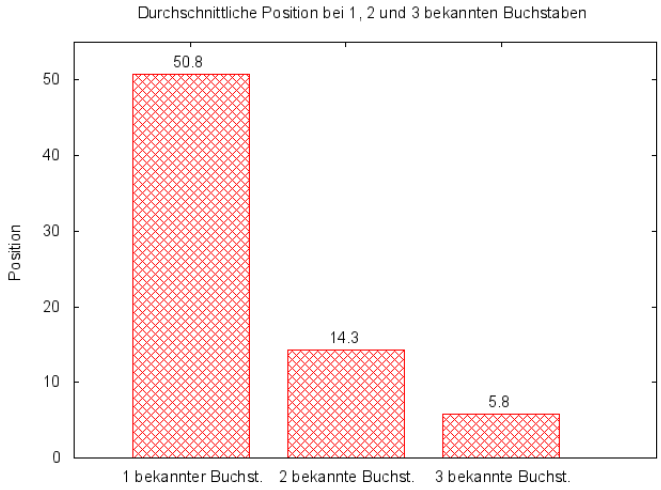

Abbildung 10: Vergleich
Durchschnittliche Position mit 1, 2 und 3 bekannten Buchstaben,
lokale Normierung, Parameter 5,2,2,0,
250.000 getestete Sätze

rekt vorhersagen, als dass es auf Position 2 bis 5 der Vorhersageliste erscheint, vor allem bei nur einem bekannten Buchstaben. Dies kann ein Hinweis darauf sein, dass vor allem kleinere Wörter wie Artikel und Konjunktionen generell häufig vorgeschlagen, und aufgrund ihrer eigenen Häufigkeit in deutschen Sätzen auch häufig richtig vorhergesagt werden. Daher erscheinen sie seltener auf Position 2 bis 5. Auch dass die Genauigkeit bei 3 bekannten Buchstaben wieder zu sinken scheint bekräftigt diese These, da hier Wörter mit weniger als 4 Buchstaben, wie viele Artikel und Bindewörter, nicht mehr in die Wertung eingehen.

Betrachten wir nun die durchschnittliche Position der richtigen Wörter in der Vorschlagsliste für einen, zwei und drei bekannte Buchstaben (siehe Abb. 10), so stellen wir fest, dass die tatsächliche Genauigkeit mit nur einem vorgegebenen Buchstaben deutlich geringer ist, als mit zwei bekannten Buchstaben. Dies bedeutet, dass von einem auf zwei vorgegebene Buchstaben der ohnehin recht große Anteil richtiger Vorhersagen lediglich um ca. 10 Prozentpunkte steigt, während die tatsächliche durchschnittliche Position des richtigen Wortes um über 70% sinkt. Die durchschnittliche Positionierung ist bei einem vorgegebenen Buchstaben also deutlich schlechter, als es die Anzahl der richtigen Vorhersagen bei einem bekannten Buchstaben vermuten lässt.

Aufgrund dieser Ergebnisse vermuten wir, dass vor allem kleinere Wörter bei einem vorgegebenen Buchstaben häufig vorhergesagt werden können. Nun ist zu überprüfen, ob diese kleineren Wörter tatsächlich überproportional oft vorgeschlagen werden, oder ob sie nur aufgrund ihrer Häufigkeit auch häufig (richtig) vorhergesagt werden.

6.6 Statistische Überprüfung

6.6.1 Häufige Wortlänge

Im vorherigen Kapitel wurde beobachtet, dass die Positionen 2 bis 5 in der Vorschlagsliste weniger häufig sind, als richtige

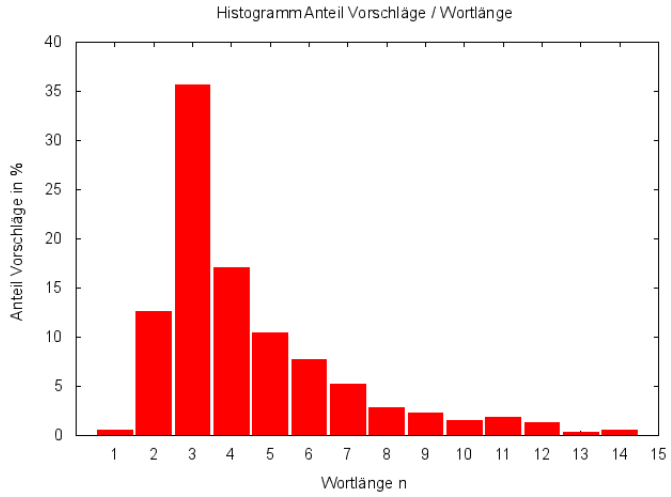

Abbildung 11: Histogramm
Anteil Vorschläge in Abhängigkeit der Wortlänge, lokale Normierung, Parameter 5,2,2,0,
1 bekannter Buchstabe, 250.000 getestete Sätze

Vorhersagen. Zudem sind Vorhersagen bei einem bekannten Buchstaben häufiger richtig, als die durchschnittliche Position der Vorschläge vermuten lässt. Diese Ungleichverteilung könnte bedeuten, dass kleinere, häufig vorkommende Wörter (z.B. Artikel und Pronomen) bevorzugt vorgeschlagen werden.

Um dies zu untersuchen, ermittelten wir den Anteil der Vorschläge in Abhängigkeit von der Wortlänge des Vorschlags, wie in Abb. 11 zu sehen ist. Aus dem Histogramm geht hervor, dass Wörter der Länge 3 deutlich häufiger vorgeschlagen werden, als Wörter von anderer Länge.

Dieses Resultat muss jedoch nicht zwangsläufig bedeuten, dass Wörter der Länge 3 wirklich bevorzugt vorgeschlagen werden, da solche Wörter ja auch generell häufiger in Sätzen vorkommen. Um einen Vergleich zu erhalten, berechneten wir ein zweites Histogramm (siehe Abb. 12), das den Anteil der tatsächlichen Wörter in Abhängigkeit von der Wortlänge zeigt.

Bei Vergleich beider Histogramme stellen wir fest, dass Wörter der Länge 3 im Schnitt in ca. 35% der Fälle vorgeschlagen werden, während nur ca. 25% der tatsächlichen Wörter diese Länge haben. Bei Wörtern der Länge 2 und 4 ist dies ebenfalls noch ersichtlich, jedoch mit nur noch ca. 4% bis 5% Differenz. Längere Wörter werden dagegen im Durchschnitt weniger häufig vorgeschlagen, als sie vorkommen. Die Differenz zwischen beiden Histogrammen ist in Abb. 13 noch einmal deutlicher zu sehen.

Damit ist bestätigt, dass Wörter mit kurzer Wortlänge überproportional oft vorgeschlagen werden.

6.6.2 Häufige Vorschläge

Neben der allgemeinen Betrachtung der Wortlänge betrachteten wir außerdem die Häufigkeit einzelner vorgeschlagenen Wörter, um zu ermitteln, ob häufig vorkommende, populäre

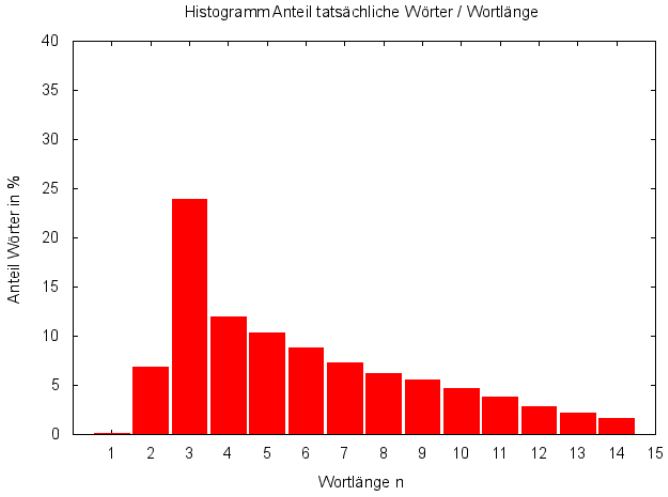

Abbildung 12: Histogramm
Anteil tatsächliche Wörter in Abhängigkeit der Wortlänge, lokale Normierung, Parameter 5,2,2,0,
1 bekannter Buchstabe, 250.000 getestete Sätze

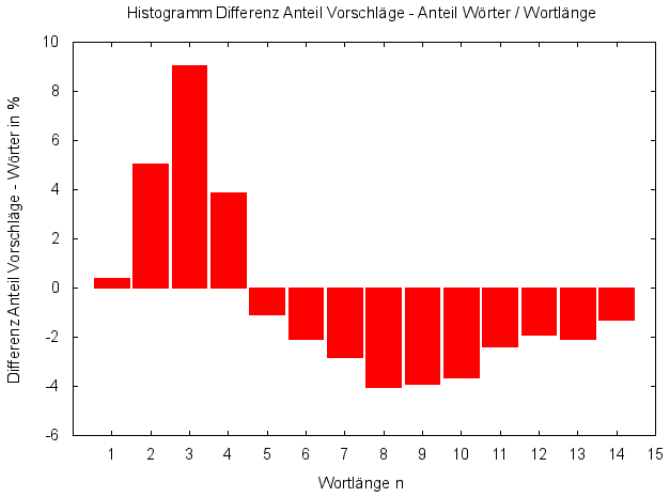

Abbildung 13: Histogramm
Differenz Anteil Vorschläge - Anteil tatsächliche Wörter in Abhängigkeit der Wortlänge,
lokale Normierung, Parameter 5,2,2,0,
1 bekannter Buchstabe, 250.000 getestete Sätze

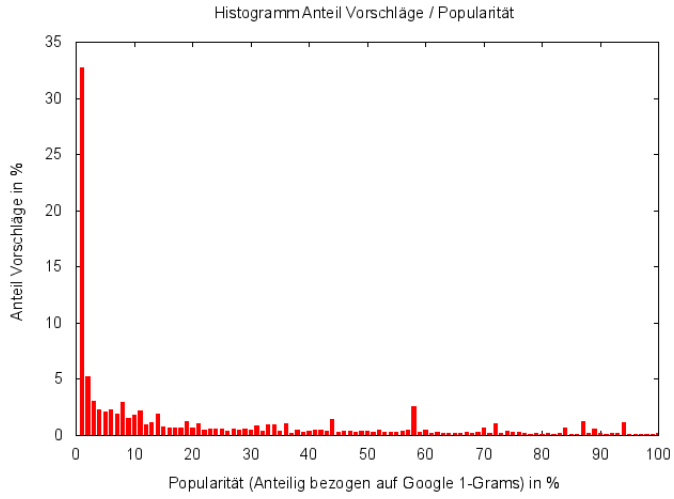

Abbildung 14: Histogramm
Anteil Vorschläge in Abhängigkeit der Popularität, lokale Normierung, Parameter 5,2,2,0,
1 bekannter Buchstabe, 250.000 getestete Sätze

Wörter mit einer überproportional größeren Wahrscheinlichkeit vorgeschlagen werden.

Dazu erstellen wir eine Liste an deutschen Wörtern, sortiert nach ihrer Häufigkeit, die dafür benötigten Daten erhalten wir durch die Google 1-Grams. Anschließend teilen wir diese Liste in 100 Intervalle, um jeweils Abschnitte mit den 1-2 %, 2-3 % ... bis 99-100 % der häufigsten Wörter zu erhalten.

Nun ermitteln wir wieder die Vorschläge für 250.000 Teilsätze der Wikipedia-Datei und sortieren die Vorschläge in die entsprechenden Popularitätsabschnitte ein. Das so erhaltene Histogramm ist in Abb. 14 zu sehen und gibt uns Auskunft über den Anteil der Vorschläge von unterschiedlich populären Wörtern.

Aus dem Diagramm ist ersichtlich, dass das obere Prozent der populären Wörter in über 30% der Fälle vorgeschlagen wird.

Ob dieser Anteil ein überproportional großer Wert ist, kann wieder nur der Vergleich mit dem Anteil an tatsächlichen Wörtern zeigen. Daher berechneten wir auf die gleiche Weise den Anteil der Wörter in den verschiedenen Popularitätsabschnitten in 5 Mio. Sätzen der Wikipedia-Datei. Das Ergebnis ist in Abb. 15 zu sehen.

Beide Histogramme sind sehr ähnlich und haben bei den gleichen Intervallen charakteristische Anstiege, z.B. bei 58%. Im Abb. 16 ist die Differenz beider Histogramme dargestellt. Die Größe der Abweichung ist jedoch, vor allem bei dem ersten Messwert von ca. 1,6%, im Verhältnis zu den tatsächlichen Anteilen von ca. 33,4% und ca. 35% zu sehen und damit insgesamt nur sehr gering.

Populäre Wörter werden demnach nicht bevorzugt vorgeschlagen.

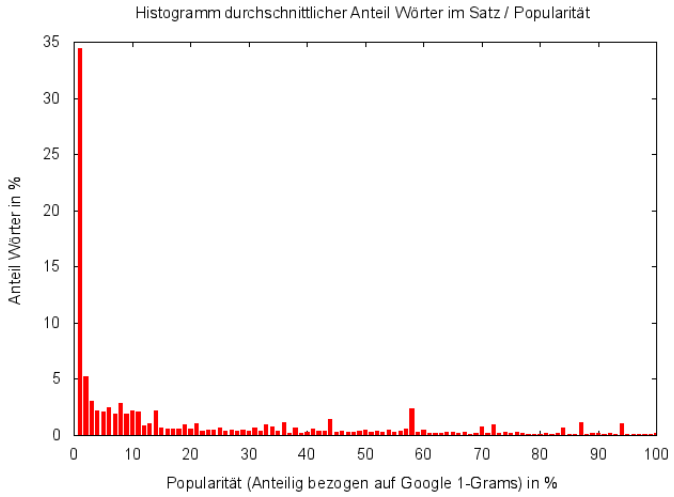

Abbildung 15: Histogramm
Anteil Wörter in Abhängigkeit der Popularität, Grundlage: Wikipedia-Datensatz

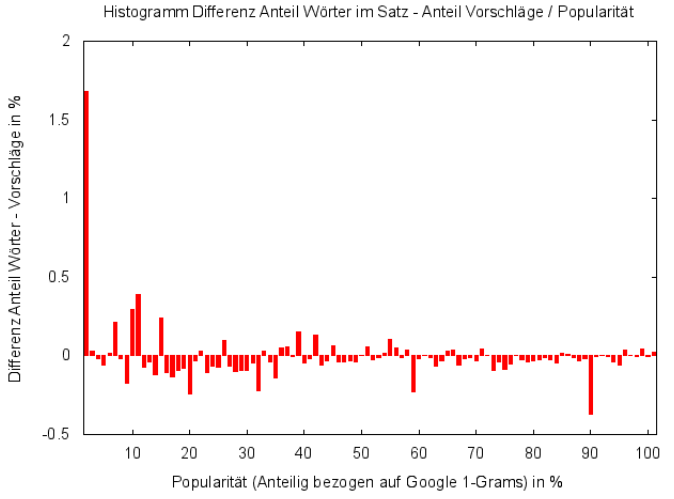

Abbildung 16: Histogramm
Differenz Anteil Wörter - Anteil Vorschläge in Abhängigkeit der Popularität

7. FAZIT UND ZUKÜNFTIGE ARBEIT

7.1 Fazit

Nach diesen soeben dargelegten Ergebnissen lässt sich festhalten, dass wir nun in der Lage sind, bei zwei vorgegebenen Buchstaben innerhalb von ca. 50 Millisekunden das gesuchte Wort in etwa 2 von 3 Fällen auf einem der ersten fünf Plätze vorzuschlagen. Da wir bei unserer Messung, wie in Kap. 6.4, aufgezeigt, zwei Berechnungsmethoden durchgeführt haben, wird diese Zeit sich im Produktiveinsatz sogar noch etwa halbieren.

Zwar ist die Genauigkeit der Vorhersagen durch die aufgezeigte Bevorzugung von Wörtern mit kurzer Länge noch verbesserungswürdig, allerdings hat sich unser gewählter Ansatz, eine rein statistische Betrachtungsweise unter Verwendung einer Graphdatenbank ohne semantische Analyse von Textdaten, bewährt.

Diesem Ansatz verdanken wir auch die bereits recht große Geschwindigkeit unseres Systems, da sämtliche zeitintensiven Berechnungen (Normierung, Parametrisierung) nicht zur Laufzeit gemacht werden müssen. Dadurch sind wir beispielsweise in der Lage, in Echtzeit Teilsätze über das Internet zu empfangen, auszuwerten und die Vorschläge zur Verfügung zu stellen, was die Voraussetzung für einen möglichen Webservice wäre.

7.2 Zukünftige Arbeit

7.2.1 Verbesserung der Vorhersagen

Wir erzielen mit unserer Datenbank momentan schon gute Ergebnisse, allerdings sind weitere Steigerungen möglich.

Nicht alle Messungen und Algorithmen, die wir durchgeführt bzw. verwendet haben, konnten optimal durchgeführt werden (beispielsweise die Zeitmessung in Kap. 6.4). Eine erste, kleine Verbesserung wird daher bereits die Überarbeitung des Quelltextes zur Folge haben. Auch durch weitere Versuche zur Optimierung der Parameter lässt sich die Qualität der Vorhersagen vermutlich noch weiter steigern.

Doch auch neue Vorgehensweisen können deutliche Verbesserungen erzielen. Insbesondere am Anfang eines Satzes können wir bis jetzt noch keine guten Vorschläge berechnen, da wir uns bisher auf die Analyse von Vorgängerworten beschränkten. Ein schnelles Verfahren zum Erraten des Anfangsbuchstabens eines Wortes, insbesondere am Satzanfang, würde die Vorhersagegenauigkeit daher deutlich steigern.

Durch eine variierte Modellierung der Datenbank, beispielsweise durch den Einbezug einzelner Buchstaben statt ganzer Wörter, kann die Geschwindigkeit nochmals gesteigert und gleichzeitig auch die Qualität der Ergebnisse verbessert werden. Eine Modifizierung des Algorithmus zur Ermittelung relevanter Knoten und Kanten könnte die Länge der Ergebnisliste reduzieren und so Zeit bei der aufwändigen Sortierung sparen. Eine weitere mögliche Modifizierung wäre die Zusammenfassung von kürzeren Wörtern (z.B. Artikel, Pronomina...) mit ihren Nachfolgern (z.B. Substantive), um statt einem, gleich zwei Wörter, oder gar ganze Satzfragmente vorschlagen zu können. Auch dies ist durch unsere Modellierung als Graph ohne besondere Probleme zu bewerkstelligen.

Es sollte auch eine andere Datengrundlage für die Datenbank in Erwägung gezogen werden, da die Google Books zum Teil recht alt sind oder einen für die Umgangssprache eher untauglichen Wortschatz liefern. Um dies zu testen, wäre auch ein Test mit dem von uns noch ungenutzten Blogger-Datensatz (Kap. 6.2.1) hilfreich, da dieser, im Gegensatz zu Wikipedia, vor allem Alltagssprache beinhaltet. Hier eröffnen sich auch Möglichkeiten für spezialisierte Datenbanken zu bestimmten Themen, welche z.B. mit der Datengrundlage einer bestimmten Wikipedia-Kategorie erstellt sein könnten. Dadurch passen sich Vorschläge der Art des gewünschten Textes an.

Außerdem könnte die Güte der Ergebnisse gesteigert werden, indem neben der bisherigen Häufigkeitsanalyse auch Grammatik und Semantik des gesamten bisher getippten Textes berücksichtigt werden. Letzteres könnte durch eine stärkere Beachtung seltener aber deshalb sinngebender Wörter geschehen.

Ebenfalls wünschenswert wären personalisierte Wortvorschläge. Diese könnten durch Analyse des individuellen Schreibstils des Benutzers ermittelt werden. Dabei würden häufig verwendete Wörter oder Sätze in einer separaten Datenbank gespeichert und zur Bildung einer individuell angepassten Ergebnisliste herangezogen werden. Auf diese Weise könnte die Datenbank den Schreibstil des Benutzers "lernen".

7.2.2 Anwendungsmöglichkeiten

Damit das von uns entwickelte System keine reine Forschungsarbeit bleibt, sondern auch praktische Anwendung findet, müssen Programme und Plugins entwickelt werden, die sich die Möglichkeiten von Wortvorschlägen zunutze machen und zu einer Verbesserung der Eingabe von Text führen.

Konkret planen wir, eine Android App zu entwickeln, welche die Funktionalität einer Tastatur bietet und dabei in der Lage ist, zukünftige Eingaben durch Vorhersage des Wortes vorwegzunehmen, um so den auf Touchscreens relativ langsamen und schwierigen Tippvorgang zu erleichtern.

Außerdem sind Plugins für Programme wie Mozilla Firefox [22], Mozilla Thunderbird [23], Google Chrome [24] und viele weitere denkbar, möglicherweise sogar die Integration in ein Textverarbeitungsprogramm wie zum Beispiel Open Office [25] oder Libri Office [26], da auch auf PCs eine beschleunigte Texteingabe nützlich wäre.

Durch die Geschwindigkeit unseres Systems kann die Berechnung der Vorschlagsliste auch über das Internet erfolgen und auf einem Server als Webservice angeboten werden. Dann wäre nicht nur die Entwicklung der genannten Applikationen deutlich vereinfacht, sondern es würde außerdem jedem Entwickler die Möglichkeit geben, unseren Service in seine Programme einzubauen, sofern eine Internetverbindung besteht.

Auch ist es möglich, das System auf beliebige andere Sprachen, die grundlegend die selbe Struktur wie Deutsch aufweisen, anzuwenden, also SAE (Standard Average European) Sprachen wie Englisch, Französisch, Italienisch, Spanisch,... Interessant hierbei wäre, inwiefern die von uns erstellten Statistiken sich dabei verändern bzw. ob zum optimalen Einsatz

des Systems andere Parameter gewählt werden müssen.

Alles in Allem wird wohl deutlich, dass die Möglichkeiten, die sich durch unsere Arbeit ergeben, sehr vielfältig sind und ein System zur Textvorhersage nahezu universell einsetzbar ist.

8. DANKSAGUNG

Unser besonderer Dank gilt René Pickhardt, der uns als Betreuer stets mit Rat, Tat und Motivation zur Seite gestanden und uns von Beginn an unterstützt hat.
Deshalb vielen Dank, René!

Außerdem danken wir Herrn Matthias Thimm für seine hilfreichen Korrekturen, sowie Herrn Strang, Betreiber der Seite *10fastfingers.com*, für die freundliche Bereitstellung von Tippstatistiken.

Ein spezieller Dank geht auch an die Deutsche Schülerakademie, auf der die Idee geboren und die Kontakte geknüpft wurden, durch die dieses Projekt ermöglicht wurde.

9. REFERENCES

[1] http://googleblog.blogspot.com/2011/04/more-predictions-in-autocomplete.html, Zugriff 22.12.2011
[2] http://googlesystem.blogspot.com/2010/09/google-scribe.html, Zugriff 22.12.2011
[3] http://www.swiftkey.net/apps, Zugriff 22.12.2011
[4] http://de.wikipedia.org/wiki/Text_on_9_keys, Zugriff 22.12.2011
[5] http://www.swype.com/, Zugriff 22.12.2011
[6] http://sehnenweh.org/2009/12/handschrifterkennung-bei-windows-7-als-tastaturersatz/, Zugriff 22.12.2011
[7] http://www.evernote.com/about/intl/de/, Zugriff 22.12.2011
[8] http://www.apple.com/de/iphone/features/siri.html?cid=mc-de-g-iphone-int-ipn-voicecontrol&sissr=1, Zugriff 22.12.2011
[9] http://dumps.wikimedia.org/dewiki/20111212/, Zugriff 23.12.2011
[10] http://code.google.com/p/wikixmlj/, Zugriff 23.12.2011
[11] http://code.google.com/p/crawler4j/, Zugriff 23.12.2011
[12] http://books.google.com/ngrams, Zugriff 29.12.2011
[13] http://commondatastorage.googleapis.com/books/ngrams/books/googlebooks-ger-all-totalcounts-20090715.txt, Zugriff 29.12.2011
[14] http://neo4j.org, Zugriff 29.12.2011
[15] http://de.wikipedia.org/wiki/Monte-Carlo-Simulation, Zugriff 4.1.2012
[16] http://de.wikipedia.org/wiki/Stemming, Zugriff 4.1.2012
[17] http://de.wikipedia.org/wiki/Genetischer Algorithmus, Zugriff 5.1.2012
[18] http://en.wikipedia.org/wiki/Discounted cumulative gain, Zugriff 5.1.2012
[19] http://docs.oracle.com/javase/1.5.0/docs/api/java/util/TreeMa Zugriff 6.1.2012
[20] http://www.cryptool-online.org/, Zugriff 9.1.2012
[21] http://stats.10fastfingers.com/, Zugriff 9.1.2012
[22] http://www.mozilla.org/de/firefox/new/, Zugriff 12.1.2012
[23] http://www.thunderbird-mail.de/wiki/Hauptseite, Zugriff 12.1.2012
[24] http://www.google.de/chrome/, Zugriff 12.1.2012
[25] http://www.openoffice.org/de/, Zugriff 12.1.2012
[26] http://de.libreoffice.org/, Zugriff 12.1.2012
[27] http://code.google.com/p/complet/, Zugriff 13.1.2012