Projektdokumentation

Spell-Checker

Vergleich drei verschiedener Algorithmen

Vorgelegt von:

Valerie Claessen

Matrikelnummer: 2179913

Abgabedatum: 23.09.2016

E-Mail: valerie.claessen@hhu.de

Inhalt

1. Einleitung: 3
2. Definitionen 4
   N-Gramm Vergleich 4
   Jaccard-Koeffizient 5
   Damerau-Lesenshtein Distanz 5
3. Hunspell Spell-Checker 6
4. Elemente 7
5. Funktionen 8
   a) Lexikon erstellen: 8
   b) Text einspeisen: 8
   c) Lexikoneinträge hinzufügen: 9
   d) Korrekturalgorithmus wählen: 9
   e) Algorithmen vergleichen: 9
   f) Geschwindigkeit vergleichen: 9
   g) Genauigkeit vergleichen: 10
   h) Fehler anzeigen lassen: 10
   i) Text korrigieren: 10
6. Werkzeuge und Hilfsmittel 10
7. Vorraussetzungen: 11
8. Anleitung zur Ausführung und Benutzung: 12
9. Reflexion und Fazit 13

1. Einleitung:

Sprache ist in allen Bereichen des Lebens von großer Bedeutung. Im Zeitalter der Digitalisierung und Automatisierung ist es daher besonders wichtig, diese in korrekter Form aufzunehmen, um so eine adäquate Weiterverarbeitung zu ermöglichen.

Befindet man sich in Bereich der geschriebenen Sprache, ist die größte Fehlerquelle zunächst ein falsch niedergeschriebenes Wort. Danach können noch Fehler auf grammatikalischer oder semantischer Ebene folgen. Diese zu korrigieren stellen jedoch eine deutlich größere Herausforderung dar, als die von einfachen Tippfehlern. Um Fehler auf Wortebene abzufangen, wird eine Autokorrektur (im weiteren Verlauf der Arbeit Spell-Checker) verwendet.

Ein Spell-Checker erkennt falsch geschrieben Worte, gibt Korrekturvorschläge oder ersetzt das Wort direkt durch eine passende Korrektur. Die wohl bekanntesten Spell-Checker findet man im Bereich der einfachen Textverarbeitungsprogramme wie Word, LibreOffice oder OpenOffice. Aber auch Kommunikationsanwendungen wie Smartphone Messenger oder E-Mail-Programme, nutzen sie. Ein weiteres Anwendungsgebiet sind Suchmaschinen wie Google.

Doch auch in weniger sichtbaren Bereichen spielen Spell-Checker eine große Rolle. Der heute sehr interessante Bereich des maschinellen Lernens (Machine Learning) ist sehr auf die Korrektheit der zu verarbeitenden Sprache angewiesen. Überall dort, wo eine Mensch-Maschine Interaktion stattfindet, muss die menschliche Sprache von einem Programm erkannt werden, das Sprache nicht intuitiv erkennt und verarbeitet.

All diese Gebiete machen eine stetige Weiterentwicklung und Verbesserung von Spell-Checkern unabdinglich für eine erfolgreich voranschreitende Digitalisierung.

Das hier vorgestellte Programm zeigt drei verschiedene Ansätze, die genutzt werden können, um einen Spell-Checker zu implementieren. Dabei liegt der Fokus besonders auf dem Vergleich von Effizienz und Genauigkeit. Dies sind zwei Aspekte, die angesichts der Masse an Worten in einer Sprache von großer Bedeutung sind.

Die drei genutzten Algorithmen können auf jeglichen Text angewandt werden, da das Programm ebenfalls ein eigenes Lexikon erstellt. Die einzige Einschränkung stellt die Verwendung von Sonderzeichen dar. Daher wurde die Sprache Englisch für erste Tests gewählt.

2. Definitionen

Bei den Mechanismen, die in dem Programm angewendet werden, handelt es sich um Algorithmen, die dazu genutzt werden, einen Spell-Checker zu implementieren. Bei einem Spell-Checker handelt es sich um eine Funktion, die dazu genutzt wird, falsch geschriebene Worte zu identifizieren und sie entweder direkt zu korrigieren oder eine Auswahl an Korrekturvorschlägen zu geben. Dies wird besonders in Schreibprogrammen wie Word oder Open Office genutzt, aber auch abgewandelt in Apps wie WhatsApp. Generell können Spell-Checker überall sinnvoll eingesetzt werden, wo der Nutzer selber Texte produziert.

N-Gramm Vergleich

Bei dem ersten genutzten Algorithmus handelt es sich um einen einfachen Vergleich der N-Gramme zweier Worte. Das als falsch identifizierte Wort wird in seine N-gramme (hier Tri-Gramme) zerteilt und mit jedem vorhandenen Wort innerhalb des Lexikons (ebenfalls in Tri-Gramme zerteilt) verglichen. Das Wort, welches die meisten gemeinsamen N-Gramme mit dem falsch geschriebenen Wort hat, wird als beste Korrektur identifiziert. Für alle drei Algorithmen gilt dabei, dass alle Korrekturen mit gleichen Wahrscheinlichkeiten angezeigt werden.

Beispiel: Einfacher N-Gramm Vergleich

Falsches Wort: „girrl“	→ **g, *gi, gir, irr, irl, rl*, l**
Korrekturvorschlag 1: „girl“	→ **g, *gi, gir, irl, rl*, l**
Korrekturvorschlag 2: „giraffe“	→ **g, *gi, gir, ira, raf, aff, ffe, fe*, e**

Der erste Korrekturvorschlag hat also sechs gemeinsame Tri-Gramme mit dem zu korrigierenden Wort, der zweite Korrekturvorschlag nur drei. So wird das Wort „girl“ als Korrektur eingesetzt.

Jaccard-Koeffizient

Für den zweiten Korrektur-Algorithmus wird der Jaccard-Koeffizient genutzt (Hamers et al., 1989). Die Wörter werden dafür zunächst auch in Tri-Gramme zerlegt, nur die Berechnung selbst ist deutlich komplexer:

SIM (A-B) = gemeinsame Tri-Gramme / (Tri-Gramme A + Tri-Gramme B – gemeinsame Tri-Gramme)

Beispiel: Jaccard-Koeffizient

Falsches Wort: „girrl“ → **g, *gi, gir, irr, irl, rl*, l**

Korrekturvorschlag 1: „girl“ → **g, *gi, gir, irl, rl*, l**

Korrekturvorschlag 2: „giraffe“ → **g, *gi, gir, ira, raf, aff, ffe, fe*, e**

SIM (A-B) = 6 / (6 + 7 – 6) = 6/7 → deutlich größer

Sim (A-B) = 3 / (9 + 7 – 3) = 3/13

So wird auch bei diesem Algorithmus „girl“ als Korrektur identifiziert.

Damerau-Lesenshtein Distanz

Der dritte und letzte Algorithmus nennt sich Damerau–Levenshtein Distanz (Mitton & Mitton, 2016). Dabei wird gemessen wie groß die Distanz zwischen zwei Worten gemessen in Veränderungsschritten ist. Dabei gibt es verschiedene Arten der Veränderung:

1. Löschen: Ein Buchstabe wird gelöscht
2. Einfügen: Ein Buchstabe wird eingefügt
3. Vertauschen: Zwei benachbarte Buchstaben werden miteinander vertauscht-
4. Trennen: Zwei Worte werden getrennt.
5. Ersetzen: Ein Buchstabe wird durch einen anderen ersetzt.

Nun wird gezählt, wie viele solcher Veränderung nötig sind, um das falsch geschriebene Wort in den Korrekturvorschlag zu verwandeln. Je weniger Veränderungen vorgenommen werden müssen, desto ähnlicher sind sich zwei Worte. Diese Methode habe ich der Anleitung von Peter Norvig (2016) entnommen.

Beispiel: Damerau-Levenshtein Distanz

Falsches Wort: “girrl”
Korrekturvorschlag 1: “girl” → Es muss nur ein “r” gelöscht werden, also nur eine Veränderung
Korrekturvoschlag 2: “house” → Alle Buchstaben sind falsch, es müssen also fünf Ersetzungen getätigt werden, was fünf Veränderungen entspricht.

3. Hunspell Spell-Checker

Hunspell ist ein weit verbreitetes Programm zur Rechtschreibprüfung vom Entwickler László Németh. Es werden dabei Sprachspezifische Besonderheiten berücksichtigt, ebenso kann Unicode verschlüsselt werden und können komplexe morphologische Analysen durchgeführt werden. Dabei stammt Hunspell von der Vorgängersoftware MySpell ab und arbeitet ebenso mit den MySpell Lexikons. Programmiert wurde das Programm in C++, es ist jedoch auch kompatibel mit gängigen Programmiersprachen wie JAVA, Pearl und .Net.

Besonders interessant ist die Betrachtung der genutzten Mechanismen. Hunspell verwendet zur Prüfung der Rechtschreibung N-Gramme und nutzt zur Datenauswertung Wörterbücher (MySpell Wörterbücher) und Sprachspezifische grammatikalische Regeln.

Verwendet wird das Programm in Textprogrammen wie OpenOffice und LibreOffice aber auch in Internetbrowsern wie Mozilla und Google Chrome. Außerdem nutzten viele LaTeX Versionen die verwendeten Mechanismen.

4. Elemente

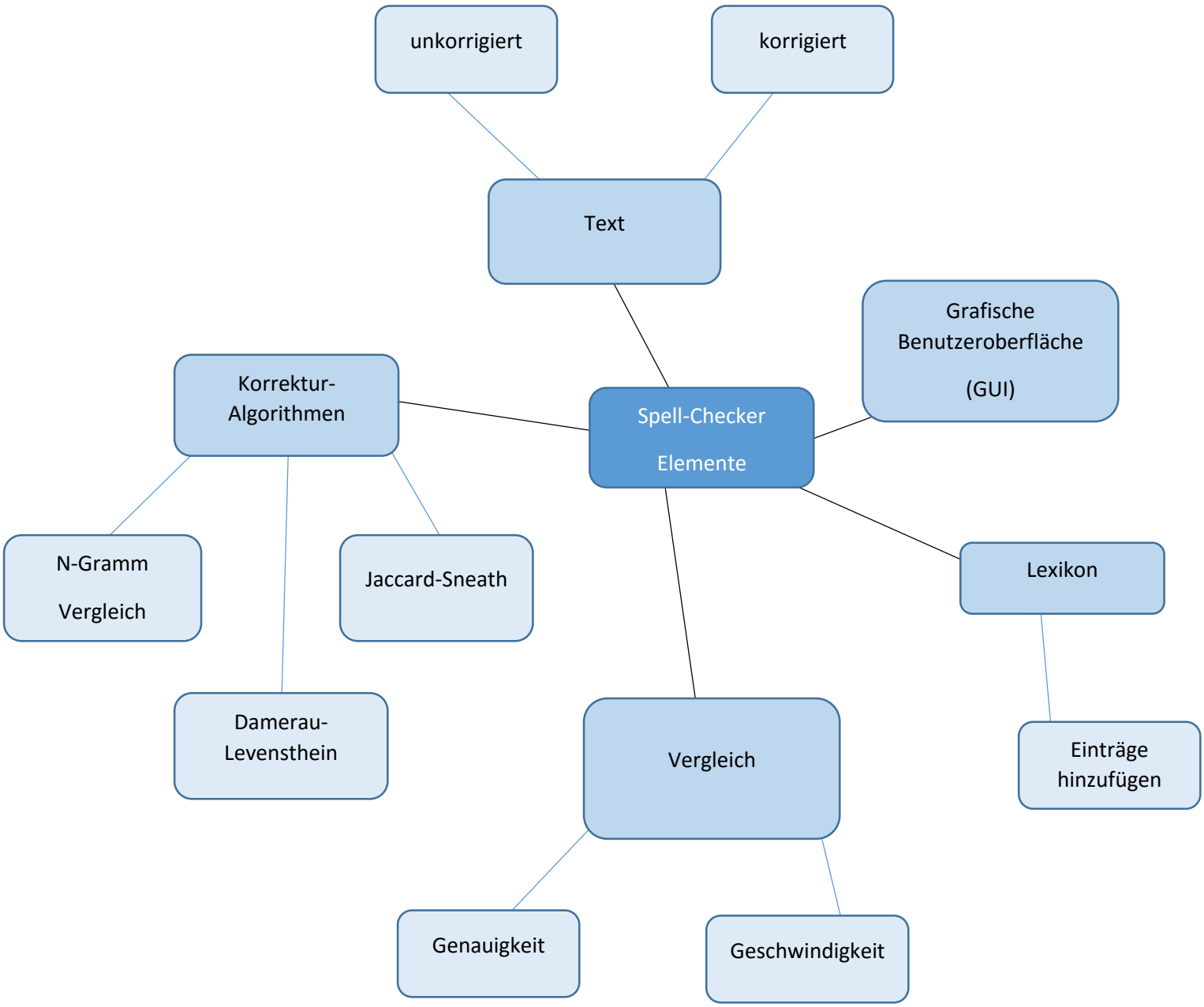

Abbildung 1 Übersicht der Elemente des Spell-Checkers

5. Funktionen

a) Lexikon erstellen:

Ein Lexikon kann aus jeglichem Text erstellt werden (Da im Programm nur auf die Besonderheiten der englischen Sprache eingegangen wird, sollte der Text, der verwendet wird, um ein Lexikon zu erstellen, auch auf Englisch sein.). Es kann ein Dokument in den Projektordner hinterlegt werden, welches „textc.txt“ (text correct) heißen muss. Aus diesem Dokument wird nun das Lexikon erstellt. Vorzugsweise handelt es sich bei dem Text der hierzu benutzt wird um einen Text, der entweder möglichst viele Worte enthält, oder alle Worte abdeckt, die im fehlerhaften Text enthalten sind.

Der nun hinterlegte Text wird, als nächstes von Satzzeichen bereinigt, da diese bei der Korrektur nur stören würden. Jetzt wird der so erhaltene String von Duplikaten bereinigt, die in einem Lexikon nur unnötiger Ballast bedeuten würden und die Ergebnisse verfälschen würden. Zuletzt wird die gesäuberte Wortliste in ein neues Textdokument „lex.txt“ geschrieben. Auf dieses kann nun immer wieder zugegriffen werden, es kann flexibel erweitert oder verändert werden. Diese Funktionen werden im späteren Verlauf noch genauer erläutert.

b) Text einspeisen:

Der Spell-Checker ist darauf ausgelegt, mit englischsprachigen Texten zu arbeiten. Darum empfiehlt es sich, sich bei der Einspeisung von zu korrigierenden Texten nur auf solche zu beschränken.

Ein zu korrigierender Text kann einfach in den Projektordner gelegt werden. Mit der richtigen Benennung (textm.txt (text with mistakes)) wird er automatisch vom Programm erkannt und zur Korrektur eingelesen. Auf der Nutzeroberfläche wird dann der Text in einem Textfeld angezeigt. Dies dient dazu, den „falschen“ Text mit den verschiedenen Korrekturvorschlägen zu vergleichen.

Für die eigentliche Korrektur wird der Text mit Schreibfehlern dann von all seinen Satzzeichen bereinigt und in eine Liste zerteilt, die dann die einzelnen Worte enthält. So können diese leichter von den jeweiligen Algorithmen genutzt werden.

c) Lexikoneinträge hinzufügen:

Wenn dem Nutzer die Einträge, die bereits im Lexikon gespeichert sind, nicht ausreichen, hat er die Möglichkeit über eine eigene Funktion neue Wörter dem Lexikon hinzuzufügen. Über den Hauptmenü-Button „Lexikon“ öffnet sich ein neues Fenster, welches ein Textfeld und einen Button enthält. In das Textfeld können nur neue Einträge erstellt werden, welche sich mittels dem „Hinzufügen“-Button dem bestehenden Lexikon hinzufügen lassen. So getätigte Änderungen am Lexikon sind dauerhaft, es lässt sich also langfristig an einem großen Lexikon arbeiten.

d) Korrekturalgorithmus wählen:

Da es insgesamt drei verschiedene Korrektur-Algorithmen gibt, kann zwischen diesen ausgewählt werden. Dies geschieht über drei Radio-Button. Sobald einer dieser angeklickt wird, erscheint der korrigierte Text (Korrektur erfolgt mit dem gewählten Algorithmus) im rechten Textfeld. Wird ein neuer Button angeklickt, wird der Text im Textfeld automatisch durch die neue Korrektur ersetzt.

e) Algorithmen vergleichen:

Da es drei verschiedene Algorithmen gibt, mit denen ein Text korrigiert werden kann, bietet es sich an, diese in ihrer Genauigkeit und in ihrer Schnelligkeit zu vergleichen.

f) Geschwindigkeit vergleichen:

Dazu wird die Zeit zu Beginn des Methodenaufrufes der jeweiligen Methode gemessen und dann von der Zeit abgezogen, die erreicht wurde, wenn die Korrektur abgeschlossen ist. Um diese Ergebnisse nun zu veranschaulichen und besser zu vergleichen, werden die drei so entstandenen Werte in einem Balkendiagramm dargestellt.

g) Genauigkeit vergleichen:

Um die Genauigkeit der Algorithmen zu prüfen wird der korrekte Text herangezogen. Nun wird für jeden Algorithmus gezählt, wie viele Worte nicht korrekt ersetzt wurden. Diese drei so entstandenen Zahlen werden angezeigt und zusätzlich in einem Balkendiagramm dargestellt, welches optional aufgerufen werden kann.

h) Fehler anzeigen lassen:

Möchte man sich die gefundenen Fehler mit ihren Korrekturvorschlägen anzeigen lassen, kann dazu der „Check“-Button genutzt werden. Ein neues Fenster öffnet sich, und es wird in Sätzen angezeigt, welche Wörter falsch waren und was die entsprechende Korrektur (des jeweiligen Algorithmus) ist. Hier werden, wenn es mehrere gleichwahrscheinliche Wortvorschläge gibt, alle Möglichkeiten angezeigt.

i) Text korrigieren:

Wurde ein Algorithmus ausgewählt, werden falsche Wörter gesucht. Dies geschieht über einen Abgleich mit dem aktuell vorhandenen Lexikon. Existiert ein Wort im Lexikon, wird es als richtig angesehen. Ist ein Wort nicht im Lexikon auffindbar, so wird es als falsch markiert und gelangt so in den Korrekturprozess. Nun wird (je nach Algorithmus) die wahrscheinlichste Korrektur für ein falsches Wort errechnet. Diese wird nun im Fließtext wieder rückwirkend eingefügt und im rechten Textfeld wird der korrigierte Text mit entsprechenden Ersetzungen ausgegeben.

6. Werkzeuge und Hilfsmittel

Bibliotheken:

from __future__ import division:

Dieser Import wird benötigt, um bei der Division Nachkommastellen anzuzeigen. Dies ist besonders beim Jaccard-Sneath Algorithmus von großer Bedeutung, da der Gewichtungsunterschied hier ausschließlich in den Nachkommastellen sichtbar wird.

from Tkinter import *:

Hier wird die Bibliothk zur Erstellung der grafischen Nutzeroberfläche importiert. Alle Elemente, die später für den Nutzer sichtbar sind, werden mit dem Tkinter implementiert.

from collections import Counter:

Diese Bibliothek bietet Alternativen/ Ergänzungen zu den üblichen Python Datentypen (genauer „Containern“ wie dict, list, set und tuple). Der Typ „Counter“ wird in der Damerau Korrektur-Methode zur Verwendung des Lexikons genutzt.

import matplotlib.pyplot as plt:

Diese Bibliothek wird benötigt, um die graphische Darstellung der Algorithmen-Vergleiche zu gewährleisten. Konkret werden damit die Balkendiagramme dargestellt.

7. Vorraussetzungen:

Im Ordner Projekt_PY befindet sich das Python-Programm SpellCheck.py. Nur dieses Programm muss gestartet werden um den Spell-Checker nutzen zu können. Zusätzlich befinden sich noch vier Textdokumente im Ordner. Lex.txt ist ein Textdokument, welches leer ist, wenn das das Programm das erste Mal gestartet wird. Danach wird darin dauerhaft das Lexikon gespeichert und es werden auch neue Lexikoneinträge dauerhaft darin gespeichert. Möchte man ein neues Lexikon erstellen, was eigentlich nicht nötig ist, da zusätzliche Lexikoneinträge förderlich für die Genauigkeit des Programms sind, muss der Inhalt dieses Dokuments manuell gelöscht werden. Des Weiteren befindet sich textc.txt im Ordner. Darin sollte sich (um die Algorithmen zu vergleichen und das Lexikon zu erstellen) eine korrekte Version des zu prüfenden Textes befinden. Das Textdokument textm.txt ist das wichtigste Dokument. In ihm befindet sich der Text, den es zu korrigieren gilt. Möchte man neue „Fehler“ hinzufügen, können diese einfach in das Textdokument geschrieben werden. Hier können auch völlig andere Texte eingefügt werden. Zu beachten ist dabei lediglich, dass das Lexikon immer einen entsprechenden korrekten Text oder eine andere Lexikonquelle benötigt.

In der Version, in der der Spell-Checker vorliegt muss keine Veränderung an den Textdokumenten vorgenommen werden. Für ausgeweitete Tests können diese jedoch sehr flexibel genutzt werden.

8. Anleitung zur Ausführung und Benutzung:

Startet man das Programm, sieht man die Hauptbenutzeroberfläche. Der obere Teil des Fensters besteht aus einer Radiobutton Auswahl. Hier muss zunächst der Algorithmus ausgewählt werden, mit dem die Korrektur durchgeführt werden soll.

Unter dieser Auswahl sind zwei Textfelder. Das linke ist mit „Unkorrigierter Text“ beschriftet. Hier befindet sich der Text, den man zuvor zur Korrektur hinterlegt hat. Wählt man nun einen Korrekturalgorithmus, erscheint im rechten Textfeld der Text, auf dem die Korrektur ausgeführt wurde. Zu beachten ist dabei, dass alle Satzzeichen entfernt wurden. Des Weiteren stehen an Stellen, wo mehrere Korrekturvorschläge berechnet wurden, diese durch „/“ getrennt hintereinander.

Am rechten Rand des Fensters befinden sich vier Butten untereinander. Der erste Button (Lexikon) öffnet beim daraufklicken ein neues Fenster welches die Option bietet, neue Lexikoneinträge hinzuzufügen. Diese müssen einzeln in das Textfeld geschrieben werden und können mit dem „Hinzufügen“ Button dem bereits bestehenden Lexikon angehängt werden. Unter dem Button befindet sich eine kurze Beschreibung welche alles zu Beachtende noch einmal erklärt. Möchte man keine Einträge mehr hinzufügen, kann man das Fenster mit dem „x“ in der oberen rechten Ecke schließen (gilt für alle Fenster).

Der nächste Button auf dem Hauptfenster ist der „Check“ Button. Er öffnet ein neues Fenster, welches ein Textfeld enthält. Darin werden die Fehler mit den dazugehörigen Korrekturen des aktuell ausgewählten Algorithmus angezeigt. Möchte man alle drei Algorithmen so vergleichen, muss man im Hauptfenster nacheinander die Algorithmen auswählen und jedes Mal diesen Button betätigen.

Die unteren beiden Button sind zum Vergleich der Algorithmen gedacht. Klickt man den „Compare Time“ Button, öffnet sich erneut ein Fenster, welches einen Button enthält (Graph), der den grafischen Vergleich der Korrekturzeiten anzeigt. Darunter

befindet sich zur Übersicht die Zeiten in Sekunden, die die jeweiligen Korrekturalgorithmen brauchen, um den angegebenen Text zu korrigieren.

Nach dem gleichen Prinzip funktioniert auch der „Compare Accuracy“ Button. Er öffnet ein gleich angelegtes Fenster. Mit dem darauf angelegten Button wird der Graph angezeigt, der die Algorithmen in ihrer Genauigkeit vergleicht. Es werden darauf die Anzahl der falsch korrigierten Worte verglichen. Unter dem „Graph“ Button befindet sich auch hier wieder zur Übersicht die ausgeschriebene Anzahl an falschen Korrekturen.

Wichtig ist hier, dass die Darstellung von Diagrammen mit matplotlib es nicht erlaubt, zwei Diagramme gleichzeitig zu öffnen. Will man also den zweiten Vergleich machen, muss zunächst das noch offenen Fenster geschlossen werden.

9. Reflexion und Fazit

Vergleicht man nun das fertiggestellte Projekt mit dem Programm, welches im Projektplan angestrebt wurde, lässt sich sagen, dass die grundlegenden Ziele erreicht wurden.

Es wurden wir geplant, drei verschiedenen Algorithmen implementiert, welche alle funktionstüchtig sind. Ihre Effizienz ist dabei sehr unterschiedlich, dies war aber zu erwarten Derartige Unterschiede sind in Bezug auf eine Weiterentwicklung des Programms sehr interessant.

Eine Abweichung des Projektplans gab es bei der Art der genutzten Algorithmen. Der einfache N-Gramm Vergleich und die Damerau–Levenshtein Distanz wurden wie geplant implementier. Die Nutzung von Vektoren stellte sich jedoch auf Wortebene als nahezu unmöglich dar. Da es auf Wortebene auf die Reihenfolge der Buchstaben ankommt (Anders als bei der Klassifizierung von Texten. Dabei spielt die Reihenfolge der Wörter keine Rolle, nur ihr Vorkommen wird gezählt.), würde ein einfacher doppelter Buchstabe bei der Vektormethode dazu führen, dass alle darauffolgenden Buchstaben auch als falsch erkannt werden würden.

Daher war der Jaccard-Koeffizient eine gute Alternative. Besonders beim Vergleich von Zeit und Genauigkeit brachte dieser Algorithmus interessante Ergebnisse.

Die angestrebten Funktionen der Nutzeroberfläche konnten alle realisiert werden. Die Darstellung der einzelnen Optionen in jeweils eigenen Fenstern ist dabei jedoch noch nicht optimal gelöst. Würden weitere Funktionen hinzukommen, müsste daher aus Gründen der Übersichtlichkeit eine Überarbeitung des Hauptmenüs vorgenommen werden.

Als besonders interessant erwiesen sich die Vergleiche der drei Algorithmen. Es waren deutliche Unterschiede, sowohl in der Ausführungszeit, als auch in der Genauigkeit zu erkennen.

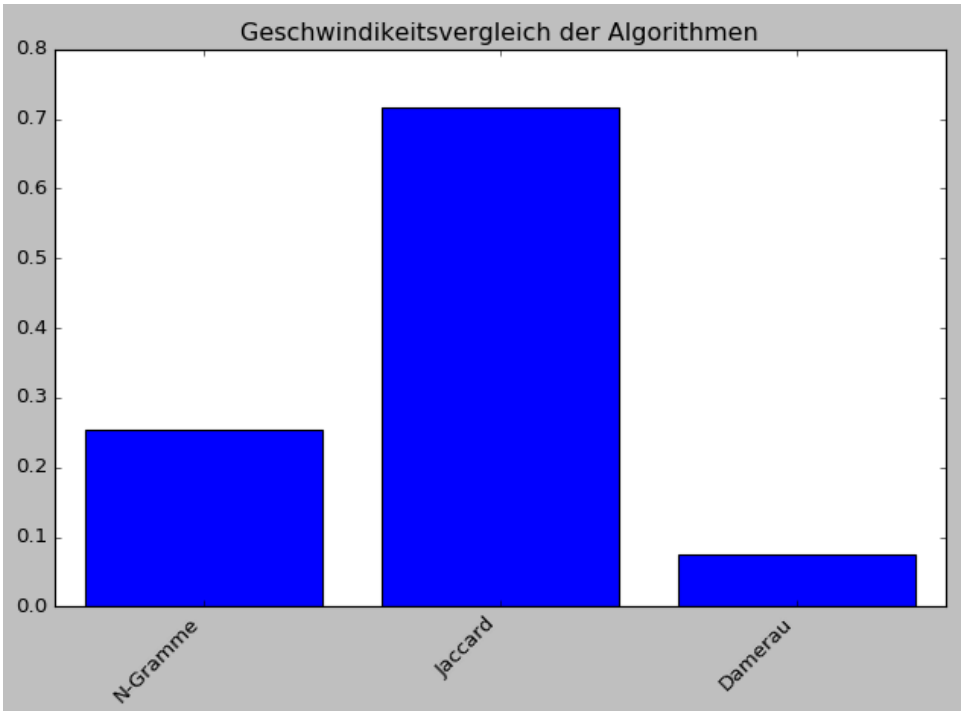

Abbildung 2: Vergleich der implementierten Algorithmen in Bezug auf ihre Geschwindigkeit in Sekunden. (Test-Text: Ausschnitt aus Anna Karenina)

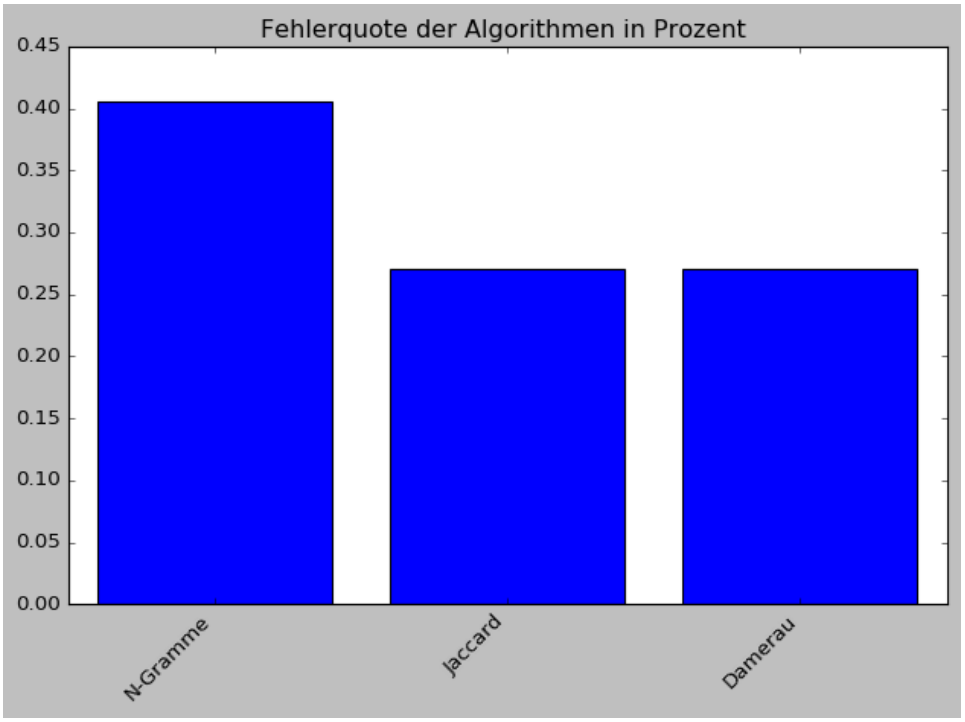

Abbildung 3: Vergleich der implementierten Algorithmen in Bezug auf ihre Genauigkeit (falsch erkannte Wörter). (Test-Text: Ausschnitt aus Anna Karenina)

Nicht wie geplant durchgeführt wurde der Vergleich mit den verwendeten Mechanismen von LibreOffice. Hier stellte mich die Einbindung des Programms vor eine zu große Herausforderung, da das ursprüngliche Programm in C++ programmiert wurde. Zudem ergaben sich immer wieder Probleme mit den Python Versionen (Version 2.7). Daher entschied ich mich dafür, anstatt einer direkten Einbindung des Programms, einen theoretischen Vergleich vorzunehmen. Auch hier war es sehr schwer, ausreichende Informationen über Hunspell zu finden, da die Herstellerseite nur sehr weniger Informationen liefert.

Abschließend lässt sich sagen, dass die Implementierung der Algorithmen sehr gut umzusetzen waren und die Ergebnisse der Korrekturen deutlich besser und genauer als erwartet waren. Des Weiteren ist besonders der Vergleich der Algorithmen und die Flexibilität ein eigenes Lexikon zu erstellen und zu erweitern eine schöne und nützliche Funktion des Programms.

Literatur:

Norvig, P. (2016). *How to write a Spelling Correction.* http://norvig.com/spell-correct.html

Hamers, L., Hemeryck, Y., Herweyers, G., Janssen, M., Keters, H., Rousseau, R., & Vanhoutte, A. (1989). *SIMILARITY MEASURES IN SCIENTOMETRIC SALTON ' S COSINE FORMULA*, *25*(3), 315–318.

Mitton, R., & Mitton, R. (2016). *Language Engineering : Ordering the suggestions of a spellchecker without using context Ordering the suggestions of a spellchecker without using context* ∗, (August 2008), 173–192. http://doi.org/10.1017/S1351324908004804

Abbildungsverzeichnis:

Abbildung 1 Übersicht der Elemente des Spell-Checkers 7
Abbildung 2: Vergleich der implementierten Algorithmen in Bezug auf ihre Geschwindigkeit in Sekunden. (Test-Text: Ausschnitt aus Anna Karenina) 14
Abbildung 3: Vergleich der implementierten Algorithmen in Bezug auf ihre Genauigkeit (falsch erkannte Wörter). (Test-Text: Ausschnitt aus Anna Karenina).. 15