Simulation und Modellierung: Textstatistik

Intelligent Editor Environment

Marvin Beese, mabeese@uni-potsdam.de, 786300

Universität Potsdam

Abstract. Das Programm "Intelligent Editor Environment" bietet eine Möglichkeit zur Textanalyse hinsichtlich der Zipf-Verteilung der Wörter des Textes. Unter anderem sind ein Texteditor, ein Tool zur Textgenerierung sowie die Berechnung der Probability-Mass-Function des Zipfschen Gesetzes implementiert. Dieses Programm entstand als Abschlussprojekt für das Physikmodul "Simulation und Modellierung" bei Dr. Ralf Toenjes.

1 Einleitung

Im Rahmen meiner Projektarbeit der Veranstaltung Modulation und Simulation" im Wintersemester 2019/20 soll programmatisch eine statistische Analyse einer Textdatei, eine mathematische Auswertung der Zipf-verteilten Wörter und eine zufällige Generierung von Text durchgeführt werden.

Anwendungsfelder sind bei der Textstatistik Texteditoren, die die Anzahl an Wörtern und Sätzen im Text analysieren können. Die Zipf-Verteilung findet neben der Linguistik bei der Verteilung von Wörtern in einem Text oder in einer Sprache auch viele andere Anwendungen. Beispielsweise ist die Verteilung der Einwohner auf Städte sowie die Wohlstandsverteilung als auch die Dateigrößen oder Webseite-Aufrufe im Internet Zipf-verteilt.

Alle erforderlichen Funktionen und produktive Erweiterungen sind in meinem Textverarbeitungs- und Textanalyseprogramm "Intelligent Editor Environment" beinhaltet. Die Erstellung und Manipulation von Textdateien mit den wichtigsten Datei-Operationen ist implementiert, als auch die Analyse von Textdateien bezüglich ihrer Worthäufigkeiten in tabellarischer oder graphischer Form mit der Analyse der Zipf-Verteilung des Textes und die Generierung von Textdateien mit zufälligen Buchstaben- und Satzzeichen und Zeilenumbrüchen mit gegebener Wahrscheinlichkeit und variabler Größe. Alle mathematischen relevanten Funktionen sowie die Textgenerierungs-Funktionen sind auf Korrektheit mit *Pytest* getestet.

Das Programm wurde mit *Python 3.8* und *Windows 10 x64-based processor* entwickelt und unter *Ubuntu Gnome 18.04* getestet. Eine Auflistung der verwendeten Packages ist unter Kapitel *5 Entwicklungsumgebung und zukünftige Versionen* zu finden.

2 Intelligent Editor Environment

2.1 Programmteile

Das Programm ist unterteilt in mehrere Bestandteile. Die Benutzeroberfläche und elementare Programmteile sind im Package *Basic_Gui* beinhaltet. Die Analysen zur Texteingabe, Dateiinformation und mathematischen Metriken sind im Package *Statistics* und die Textgenerierung im Package *Generation* implementiert.
Dieses Programm als öffentliche Github-Repository einsehbar [1].

2.2 Basic_Gui

Aus der Main-Funktion wird eine Instanz von *WindowInstance* erstellt, in dieser Instanz werden Dateiinformationen wie Dateiname, Speicherpfad und Speichergröße der Datei gespeichert. Aus dieser Instanz von *WindowInstance* wird eine Instanz von *WindowTkinter*, der grafischen Oberfläche für die Textverarbeitung, gestartet. Die wichtigsten Editor-Operationen sind in der Menüleiste eingebunden, wo unter File ebenfalls eine neue Datei erstellt und die aktuelle Datei gespeichert oder eine neue Speicherstelle für diese Datei erstellt werden kann, bereits vorhandene Textdateien können geöffnet werden. Die Dateioperationen sind in der Klasse *Fileoperations* definiert. Zusätzlich zu den vorig genannten Operationen wird in einer Funktion auch die Dateiintegrität geprüft, wenn ein Speicherbefehl gestartet wird oder die Datei geschlossen wird. Bei dieser Prüfung wird der Inhalt der gespeicherten Datei ausgelesen und abgeglichen mit dem Inhalt des Texteditors.

Wenn in *WindowTkinter* unter Text-Generation ein Menüpunkt angeklickt wird, so wird ein Fenster geöffnet, in dem die Einstellungen für die Zufallsgenerierung von Text konfiguriert werden kann. So wird in der Klasse *ConfigRanGenerationFrame* die Wahrscheinlichkeit für Buchstaben sowie Satz- bzw. Leerzeichen eingestellt, die Anzahl an vorangegangenen Zeichen für einen periodischen Zeilenumbruch und die Größe der zu erstellenden Datei angegeben. Diese Konfigurationsinformationen werden dann in einer Konfigurationsdatei gespeichert, die dann entweder auf ein bereits bestehendes Dokument oder auf ein neues Dokument angewendet wird.

Unter dem Menüpunkt *Mathematical-Analysis* kann die tabellarische und grafische Analyse der Top-100 Wörter gestartet werden, hierbei wird ein Fenster des *AnalysisFrame* gestartet. Im *Analysisframe* gibt es einen Bereich für das Plotten der Daten auf einer doppelt-logarithmischen Skala und einen Bereich für die tabellarische Darstellung der Daten. Die Berechnung der Steigung der geplotteten Daten, des Zipfschen Exponenten, die Suche der Wortvorkommen eines Wortes und die Berechnung der Zipfschen Wahrscheinlichkeit des Auftretens für einen bestimmten Rang kann in diesem Fenster dargestellt werden. Die Berechnungen finden im Package *Statistics* in den Klassen *FileAnalysis* und *MathematicalAnalysis* statt.

2.3 Statistics

Im Package *Statistics* befindet sich die Klasse *FileAnalysis*, in der die Funktionen *splitString* für die Rückgabe von einzelnen Wörtern eines Textes, *textStats* für die Identifikation und Speicherung von einzelnen Wörtern sowie deren Vorkommen in einem Wörterbuch und *sortTextStats* für die absteigende Ordnung dieses Wörterbuches implementiert sind.

In der Klasse *TextinputStatistics* werden mit den Funktionen *countLetters*, *countWords*, *countSentences* und *countLines* die Anzahl der Zeichen des Textes, der Wörter, der Sätze und der Zeilen des Textes berechnet.

Alle mathematischen Analysen erfolgen in der Klasse *MathematicalAnalysis*. Um die Steigung der Geraden durch die geplotten Punkte zu berechnen, muss eine lineare Regressionsanalyse durchgeführt werden. Dafür werden die x- und y-Mittelwerte, die mit *calcXMedian* bzw. *calcYMedian* berechnet werden, die kumulativen Quadrate über die Abweichung der x-Werte mit *calcSxx* sowie das kumulative Produkt der x- und y-Abweichungen mit *calcSxy* programmatisch ermittelt. Die Steigung der Verteilung beträgt durch die lineare Regressionsanalyse mit der Funktion *linRegSlope* demnach Sxy/Sxx. Der Exponent der Zipf-Verteilung wird in der Funktion *exponentZipf* berechnet. Für die Berechnung der Probability-Mass-Function des Zipfschen Gesetzes für einen speziellen Rang wird die Riemannsche Zeta-Funktion, die mit *zetaRiemann* berechnet wird, eingebunden und der Prozentwert mit *pmfZipf* berechnet. Die vorhersagbare Anzahl an Wörtern eines Ranges in einem Text kann dann mit *estOccurrences* und der Abgleich zu der tatsächlichen Anzahl an Wörtern eines Ranges mit *estToRealOccurrences* berechnet werden.

Alle relevanten mathematischen Funktionen wurden in der Klasse *MathAnaTest* im Package *Testing* mithilfe von *Pytest* auf ihre Korrektheit getestet.

2.4 Generation

Im Package *Generation* befindet sich die Klasse *RandomGeneration*, die mit der Funktion *randomTextGeneration* einen Zufallstext beliebiger Länge zu gegebenen Parametern generieren kann. Hierbei wird für jedes Zeichen entweder ein zufälliger Buchstabe des deutschen Alphabets oder ein Leer- oder Satzzeichen eingefügt wird. Nach einer definierten Anzahl an Zeichen werden Zeilenumbrüche eingefügt. Die relevanten Generierungsfunktionen wurden mithilfe von *Pytest* auf ihre Korrektheit getestet in der Klasse *RandomGenerationTest* im Package *Testing*.

2.5 Programmentwicklung

Die Programmentwicklung erfolgte für jede Unterversion in geplanter Abfolge. Zunächst wurde in Version “0.1 Basic GUI, Texteditor” ein Frame für den Texteditor erstellt, der alle wichtigen Dateioperationen verarbeiten kann. Diese Version beinhaltete die Klassen *WindowInstance, WindowTkinter* und *Fileoperations*. In der Version “0.2 General Analysis” wurden Dateimetainformationen

und Textinhaltinformationen analysiert. Hierfür wurden die Klasse *WindowInstance* erweitert und die Klassen *FileAnalysis* und *TextinputStatistics* erstellt. Alle mathematischen Funktionen, die in der Klasse *MathematicalAnalysis* implementiert wurden und im *AnalysisFrame* erkennbar sind, sind in Version "0.3 Text Analysis" beinhaltet. Dazu gehören neben der Darstellung der Daten in grafischer und tabellarischer Weise ebenso die Berechnung der Steigung der Verteilung und des Exponenten der Zipf-Verteilung. Die Suche nach Vorkommen pro Wort sowie die Zipfsche Probability Mass Function für die Berechnung der vorhergesagten Werte abhängig vom Zipf-Exponenten wurden zusätzlich implementiert. Der grafische Plot und die Tabelle können hier als *.pdf* bzw. *.txt* -Datei exportiert werden. In Version "0.4 Random Generation" wurde ein Frame für die Einstellungen der zufälligen Textgenerierung und die Funktionsweise der Textgenerierung in den Klassen *ConfigRanGenerationFrame* bzw. *RandomGeneration* implementiert.

Die Programmentwicklung mit den einzelnen Updates und Entwicklungspunkten ist unter dem öffentlichen Trello-Board einsehbar [2].

3 Simulation und Modellierung - Eine theoretischeBetrachtung der Berechnungen

3.1 Statistische Analyse und Identifikation der Wörter

Für die statistische Analyse der Wörter im Text wurde die Funktion `L = splitstring(s)` implementiert, die mittels regulärem Ausdruck die Wörter aus dem String s identifiziert und als Liste L zurückgibt. Um eine zufriedenstellende Worttrennung zu ermöglichen, wurden als Trennzeichen neben mehrmaligen Leerzeichen auch Interpunktionen gesetzt.
Somit wird `re.split(r'[\.\,\;\s\:\"\!\?]\s*', str(s))` aufgerufen.
Die Funktion `n,W = textStats(file, ReturnDict=False)` liest die Datei *file* ein und gibt die Gesamtzahl der Wörter n und ein Wörterbuch W zurück. Sollte *ReturnDict=True* sein, so gilt, dass das Wörterbuch die Anzahl der Wortvorkommen im Text dem Wort zuweist. Nachdem der Text von der Datei *file* ausgelesen wurde, wird die Funktion `splitstring(s)` auf diesen Text s angewandt. Für jedes nun ergebende Element der Liste wird zunächst geprüft, ob dies ein alphabetisches Wort und ob es ein Substantiv oder Eigenname mit vorangegangenem Großbuchstaben ist. Für jedes Wort in dieser Liste wird dann geprüft, ob das Wort ungeachtet der Groß- oder Kleinschreibungen schon einmal abgespeichert wurde, ist dies der Fall so wird der Counter für dieses Wort bei jedem Treffer erhöht. Sollte dieses Wort noch keinen Eintrag in Wörterbuch haben, so wird ein neuer Eintrag erstellt. Da bei der alphabetischen Prüfung der Elemente jene Wörter mit Apostroph nicht berücksichtigt werden, wird das nachfolgend geprüft, ebenso wie die Suche nach Wörtern, die mit einem Großbuchstaben anfangen, damit diese einem semantisch gleichem Eintrag mit Kleinschreibung zugeordnet werden können.

3.2 Steigung der Wort-Verteilung

Für die Berechnung der Steigung der Wortverteilung müssen alle relevanten Datenpunkte betrachtet werden, damit eine repräsentative Steigung angegeben werden kann. Hierfür wird eine *Lineare Einfachregression* [3] benötigt. In (1) werden zunächst Sxx und Syy als die kumulativen Quadrate über die Abweichung der x- sowie y-Werte sowie zudem das kumulative Produkt der x- und y-Abweichungen mit Sxy in (2) berechnet. Die Steigung ergibt sich nun aus dem Quotienten aus (2) und (1), siehe (3).

$$Sxx = \sum_{i=0}^{n=max-1} (x_i - x)^2 \quad \text{Dies gilt für } Syy \text{ analog.} \tag{1}$$

$$Sxy = \sum_{i=0}^{n=max-1} (x_i - x) * (y_i - y) \tag{2}$$

$$S = \frac{Sxy}{Sxx} \tag{3}$$

3.3 Exponent der Zipf-Verteilung

Für die qualitative Beschreibung der Zipf-Verteilung muss der entsprechende Exponent e der Verteilungsdichtefunktion berechnet werden. Hierfür wird das vorangehende Ergebnis der Steigung a der Verteilung verwendet und als betragsmäßiger inverser Wert zu 1 dazuaddiert (4). Somit liegt der Exponent nah an der ursprünglichen und somit parameterlosen Form des Zipfschen Gesetzes mit $a = 1$ und somit $e = 2$ [4] [6].

$$e = 1 + (\frac{1}{|b|}) \tag{4}$$

3.4 Probability Mass Function der Zipf-Verteilung

Mithilfe der Zipf-Verteilung kann die Häufigkeit von Wörtern berechnet werden, die im Text vorhanden sind. Hierfür wird der zuvor berechnete Exponent der Zipf-Verteilung sowie die Riemannsche Zeta-Funktion benötigt [7]. Die Riemannsche Zeta-Funktion *H(N,e)* benötigt dafür den Exponenten der Zeta-Verteilung e sowie die Anzahl der verwendeten Ränge *xArr* mit Größe N (5). Schließlich kann die Probability Mass Funktion *f(k,e,N)* berechnet werden, die zum Rang k die Worthäufigkeit abhängig zum Exponenten der Zipf-Verteilung zuordnet (6).

$$H(N, e) = \sum_{n=1}^{N} \frac{1}{xArr(n)^e} \tag{5}$$

$$f(k, e, N) = \frac{1}{k^e \cdot H(N, e)} \tag{6}$$

4 Auswertung der Jupyter-Datei Textstatistik

Für die Auswertung wurde das Buch "Die Leiden des jungen Werthers" in digitaler Fassung verwendet. Nachdem die Textanalyse erfolgt ist und die Worthäufigkeiten des Textes sowohl tabellarisch als auch als Plot vorliegt, ist erkennbar, dass die Wörtverteilung einer Zipf-Verteilung folgt. Dies ist bei der grafischen Darstellung mit doppelt-logarithmischer Skalierung an der linearen Verteilung der Datenpunkte erkennbar. Da die Steigung der Wortverteilung nach linearer Einfachregression $a_{text} = -5,693$ und der sich hieraus ergebende Exponent der Zipf-Verteilung $e_{text} = 1,176$ beträgt, ist die Verteilung auch relativ nah an der ursprünglichen und parameterlosen Form des Zipfschen Gesetzes mit $e_{original} = 2$.

Für die Analyse eines Zufallstextes wurde ein Text mit einer Buchstabenwarscheinlichkeit von 0.8 und einer Satzzeichenwahrscheinlichkeit von 0.2 generiert, wobei nach 50 Zeichen periodisch ein Zeilenumbruch auftritt. Die Gesamtlänge des Textes beträgt repräsentative 20000 Zeichen. Ich habe für die Textanalyse eine eindeutige Zipf-Verteilung erwartet. Die Auswertung ergab allerdings, dass es eine große Abweichung von der ursprünglichen Form des Zipfschen Gesetzes gibt, da der Exponent dieser Verteilung $e_{zufall} = 4,374$ beträgt und somit deutlich größer ist als 2.
Eine Erklärung hierfür ist, dass es eine deutlich höhere Wahrscheinlichkeit für das Auftreten von Einzelbuchstaben gibt als für Wörter mit mehreren Zeichen. Diese Vermutung stützt der tabellarische Datensatz der Worthäufigkeiten, die belegt, dass Wörter mit einem Zeichen ein Vorkommen haben, das verglichen mit Wörtern mit mehr als einem Zeichen 4 bis 8,5 mal höher ist. Auch der Gradient der Verteilung lässt erkennen, dass die Wörter mit einem Buchstaben überdurchschnittlich häufig und Wörter mit mehr als einem Buchstaben unterdurchschnittlich oft auftreten.

Die Berechnungen und Ergebnisse dieser Auswertung sind aufrufbar in der folgenden Jupyter-Datei: [8].

5 Entwicklungsumgebung und zukünftige Versionen

Dieses Programm wurde in alleiniger Arbeit und ohne Mithilfe anderer Personen oder Programme erstellt. Als IDE wurde *PyCharm 2019.3.1 (Community Edition)* verwendet. Die Berechnung aller mathematischer Funkionen erfolgt ohne Zuhilfenahme zusätzlicher Packages.
Die Packages, die verwendet wurden sind *tkinter, ttkthemes, os, matplotlib, numpy, string, re, getpass* und *unittest*.

In zukünftigen Versionen kann der Fokus hin zu sinnvollen Editor-Erweiterungen gesetzt werden. So kann ein Grammatik-System implementiert werden, das sprachspezifisch die Wörter des Textes analysiert, sie zu Wortarten einordnet und sinnvolle Wortvorschläge für das kommende Wort gibt. Dafür werden bereits geschriebene Texte als Grundlage genommen, damit die Wortvorschläge passend zum Author sind.

6 Fazit

Mithilfe des Programmes "Intelligent Editor Environment" ist berechenbar, dass alle hinreichend großen Texte eine Wortverteilung haben, die einer Zipf-Verteilung entsprechen. Hierbei ist die Sprache des Textes nicht ausschlaggebend, da dies ein physikalisches Phänomen ist, das auch in anderen Gebieten außerhalb der Linguistik Anwendung findet. In diesem Programm kann die Steigung der Verteilung, der Exponent der Verteilung und mit der Probability-Mass-Function die Anzahl der Vorkommen berechnet werden.
Dieses Programm skaliert als Editor problemlos auch für große Dokumente. Bei der mathematischen Analyse gibt es eine deutliche Verzögerung bei der Berechnung und grafischen Darstellung von großen Daten hohen Ranges, weswegen von einer Limit-Setzung von über 1000 dringend abgeraten wird. Die zufällige Textgenerierung skaliert hingegen auch bei einer Limitsetzung von über 20000 Zeichen problemlos und mit einer sehr geringen Verzögerung.

Eine Kurzfassung des Programmes gibt mittels Jupyter-Dokument Auskunft über Programmabschnitte und Ergebnisse der Analyse von "Die Leiden des jungen Werthers" [8].

References

1. https://github.com/bmarv/Intelligent-Editor-Environment
2. https://trello.com/b/j6pdRlIb/intelligent-editor-environment
3. https://de.wikipedia.org/wiki/Lineare_Einfachregression
4. https://de.wikipedia.org/wiki/Zipfsches_Gesetz
5. https://en.wikipedia.org/wiki/Zipfs_law
6. https://www.hpl.hp.com/research/idl/papers/ranking/ranking.html
7. https://en.wikipedia.org/wiki/Riemann_zeta_function
8. https://github.com/bmarv/Intelligent-Editor-Environment/blob/master/Jupyter/IEE%20-%20Textstatistik.ipynb