FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Political Polarity in US Twitter

Seminar BigData

at the
Friedrisch Schiller University of Jena
Faculty of Mathematics and Computer Science
Graduate Degree Computer Science

submitted to
Prof. Dr. Bücker,
Dr. rer. nat. Bosse and
Herrn Schoder

submitted by
Kenny Gozali,
Chris Gerlach and
Walter Ehrenberger

Jena, January 29, 2023

Abstract

In der vorliegenden Arbeit behandeln wir eine Sentimentalitätsanalyse von US amerikanischen Politikern aus dem *House of Representatives*. Dazu haben wir Daten von Twitter der letzten 12 Jahren zu den genannten Repräsentanten *gescrapt* und mithilfe des Big Data Frameworks Spark verarbeitet. Ziel der Sentimentalitätsanalyse war es, Unterschiede der beiden Parteien (Republikaner und Demokraten) zu bestimmten politischen und auch allgemeinen Themen herauszufiltern. Jedoch haben sich in den gegebenen Daten weniger Diskrepanzen zwischen den beiden Parteien erkennen lassen, als zu Beginn erwartet, wie im Laufe dieser Arbeit deutlich wird.

Contents

List of Figures

1. Introduction

Als mächtigste Weltmacht beeinflussen die Vereinigten Staaten nahezu jeden Teil des Globus. Mitunter deshalb und aufgrund der enormen Präsenz in den Medien sowie des Einflusses auf diese fallen Diskrepanzen in der Bevölkerung schneller auf als in anderen Ländern. Aufgrund dieser Stellung wirkt sich die dortige Sentimentalität somit auch auf das Leben in anderen Ländern aus. Der Kapitolsanschlag sowie die Black Lives Matter Proteste der letzten Jahre sind ein Zeichen für die zunehmende Polarität und Unzufriedenheit in der Bevölkerung, wie sich auch in folgender Grafik erkennen lässt [al.14].

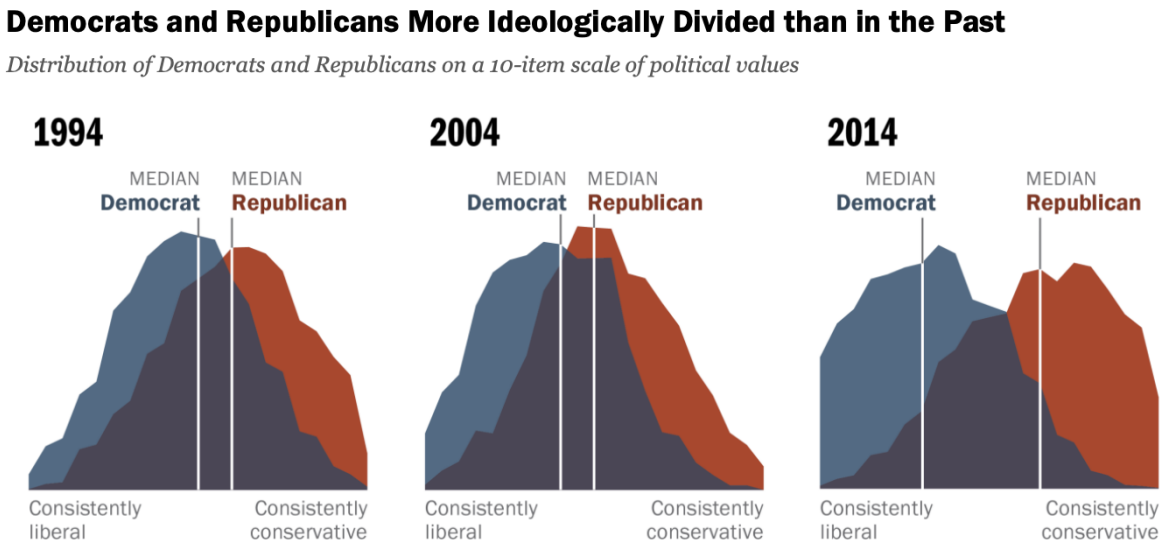

Figure 1.1.: Entwicklung der Polarität politisch engagierter Amerikaner. Basierend auf 10 politischen Metriken werden Demokraten (blau) und Republikaner (rot) hier verglichen. Wie zu erkennen bewegen sich die zu Beginn teils noch überlappenden Ideologien in den letzten 10 Jahren auseinander [al.14].

Aufgabe dieser Arbeit ist es nicht, sich mit den komplexen und vielschichtigen Hintergründen für diese Entwicklung auseinanderzusetzen. Vielmehr wird hiermit ver-

sucht, eben diese Polarität in den oberen Reihen der amerikanischen Politik genauer zu analysieren.

Unsere Zielsetzung bestand darin, mit den Tweets der letzten 12 Jahre von 420 Politikern des Repräsentantenhauses eine Sentimentalitätsanalyse durchzuführen. Dabei handelt es sich um ein Mittel der natürlichen Sprachverarbeitung, bei dem die Ansicht beziehungsweise Gefühlslage eines Textes quantifiziert wird.

2. Our project

2.1. Background

In diesem Kapitel sollen die verwendeten Bibliotheken und der Grund für ihre Verwendung genauer beleuchtet werden. Damit Daten generiert werden konnten, wurde die Bibliothek *GetOldTweets3* verwendet. Mit einer öffentlich zugänglichen Userliste für Politiker aus Amerika wurden mithife dieser Packages die Daten erhoben. Zur Weiterverarbeitung der Daten wurden *NLTK* und *TextBlob* genutzt. Beides sind Tools für die Verarbeitung von Sprache. Um eine Analyse über die verarbeiteten Ausgaben durch ein MapReduce laufen zu lassen, wurde schließlich Spark verwendet, um eine Zeit effiziente Verarbeitung zu gewährleisten.

2.1.1. GetOldTweets3-Pakage

GetOldTweets3 ist ein kostenloses Python 3 Package mit welchem Twitterdaten ohne API-Schlüssel abgerufen werden können. Mit *GetOldTweets3* können Tweets mit einer Vielzahl von Suchparametern wie Start-/Enddatum, Benutzername(n), Textabfrage und Referenzortbereich extrahiert werden. Außerdem können Tweet-Attribute die einbezogen werden sollen berücksichtigt werden. Beispielsattribute sind Folgende: Nutzername, Tweettext, Datum, Retweets und Hashtags [Yos20]. Die offizielle API von Twitter hat ein beschränktes Zeitfenster, weshalb keine Tweets älter als eine Woche abgerufen werden können. Es gibt einige Tools, die Zugang zu älteren Tweets anbieten, diese sind jedoch meistens kostenpflichtig. Das Forschungsteam hat nach einem anderen Tool gesucht um diese Aufgaben zu übernehmen, wodurch die Wahl auf das Package *GetOldTweets3* gefallen ist [Hen19].

Die Analyse des Codes von *GetOldTweets3* und die Funktionsweise des Searchthrough

Browsers von Twitter zeigt, wie das Package auch an alte Tweets kommt. Wenn auf Twitter Seiten oder User gesucht werden, startet ein Scroll-Loader. Das heißt, beim scrollen nach unten, tauchen immer mehr Tweets zu den jeweiligen Suchparametern auf. Diese Tweets bekommen sie durch Abfragen an einen JSON-Provider. *GetOldTweets3* imitiert den Searchthrough Browser von Twitter um den Scroll-Loader zu starten und zieht sich dann anhand der Abfragen an einen JSON-Provider die JSON-Datei und gibt diese decodiert zurück um somit alle Tweets anhand der oben gegebenen Parameter herauszufiltern. Dies kann man in dem Github-Repository gut nachvollziehen [Hen18]. Somit ist es möglich, sowohl aktuelle als auch sehr alte Tweets zu scrapen.

```
1  #!/bin/bash
2  # cat user_list.csv
3
4  while IFS="," read -r rec_column1 rec_column2 rec_column3 rec_column4 rec_column5
5  do
6    echo "Writing to data/$rec_column3"
7    python GetOldTweets3/cli.py --username $rec_column3 > data/$rec_column3
8
9  done < <(tail -n +2 user_list.csv)
```

Figure 2.1.: Code Beispiel für das Scrapen der Tweets
Ist eine Python Bibliothek mit der Twitter Daten durch den Scroll-Loader des Searchthrough Browsers von Twitter als JSON-Datei abgerufen werden können.

So kann durch eine paar Zeilen Code, wie man in der Abbildung erkennen kann, eine Bash-Datei erstellt werden, durch welche die Daten gesucht und abgespeichert werden. Das Scraping kann durch die Größe der JSON-Datei einige Zeit in Anspruch nehmen. Wir haben für 2 Millionen Tweets insgesamt eine Laufzeit von ca. 35 Stunden verbucht.

2.1.2. NLTK-Natural language Toolkit

NLTK ist ein Pythonpackage für die Arbeit mit menschlichen Sprachdaten. Es bietet einfach zu bedienende Schnittstellen zu über 50 Korpora und lexikalischen Ressourcen wie WordNet, zusammen mit einer Reihe von Textverarbeitungsbibliotheken für Klassifizierung, Tokenisierung, Stemming, Tagging, Parsing und semantische Schlussfolgerungen sowie Wrapper für industrielle NLP-Bibliotheken und ein aktives Diskussionsforum [NT23].

Aus diesem Grund bietet *NLTK* sehr viele Möglichkeiten zur Vorverarbeitung und Analyse, benötigt aber auch einen gewissen Zeitrahmen zur Einarbeitung in die Analysen. Aus diesem Grund hat sich das Forscherteam dafür entschieden, NLTK nur zur Vorverarbeitung zu nutzen und TextBlob für die semantische Analyse zu nutzen. Warum sich für TextBlob entschieden wurde, wird genauer in 2.1.3 besprochen. So verwenden wir den Wordkorpus von NLTK für englische Stoppworte, da diese nicht Teil der Analyse sein sollen.

Für eine individuelle und sehr ausführliche semantische Analyse bietet NLTK sehr viele Möglichkeiten durch die Interaktion mit verschiedenen Packages in Python, was aber den oben genannten Zeitrahmen benötigt, zum Einarbeiten. Der Vorteil von NLTK gegenüber TextBlob sind genau diese Interaktionen mit anderen Packages. Für größere Projekte, bei denen man die semantische Analyse auch individuell anpassen möchte, sollte man die NLTK Bibliothek benutzen. NLTK ermöglicht es durch verschiedene Vorverarbeitungsschritte, welche in der Bibliothek eingebaut sind, eine individuelle Pipeline und Analyse zu erstellen.

```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
>>> tagged = nltk.pos_tag(tokens)
>>> tagged[0:6]
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
('Thursday', 'NNP'), ('morning', 'NN')]
```

Figure 2.2.: Code Beispiel für das Arbeiten mit NLTK
Tokenisierung und Tagging von Texten mit NLTK

2.1.3. TextBlob

TextBlob ist eine Python Bibliothek für Python zwei und drei. Diesem Package arbeiten, ähnlich wie NLTK, mit verschiedenen Packages, welche in Python schon verfügbar sind.

In TextBlob sind zwei verschiedene semantische Analysen vorhanden. Da gibt es zum einen die Patternanalyse verwendet, welche die Pattern Bibliothek in Python nutzt, und dann gibt es da noch die Naive-Bayes-Analyse [Lor20a]. Hier stellt NLTK zum Beispiel mehr zur Verfügung, aber benötigt damit auch mehr Einarbeitungszeit. Damit bietet TextBlob eine besser Übersicht, weshalb sich das Forschungsteam aufgrund der wenigen Zeit für diese Bibliothek entschieden hat.

Ein weiter Grund, warum sich schlussendlich für diese Bibliothek entscheiden wurde, sind auch die zwei Outputwerte Polarität und Subjektivität welche eine gute Anwendungsmöglichkeit darstellen [Lor20b]. In der Analyse haben wir die Patternanalyse von TextBlob verwendet, welche uns die Polarität in einem Intervall von [-1, 1] und die Subjektivität im Rahmen von [0, 1] zurück gibt. Ist der Wert bei der Polarität näher an der -1 als an der 1, dann zeigt es einen negative Emotion. Im umgekehrten Fall ist es eine positive Emotion.

Bei dem Wert der Subjektivität beschreibt ein Wert, der gegen null tendiert, einen Fakt oder Faktenwissen und eine Tendenz gegen eins entspricht einer stärkeren subjektive Meinung mit rein [Lor17a] [Lor17b]. Bei der Patternanalyse geht es darum, ein Muster bei negativen und positiven Aussagen zu erkennen und dies dann auf neue Testdaten oder unbekannte Daten anzuwenden. Dabei spielt sowohl die Syntax als auch die Semantik und die Wortwahl eine bedeutende Rolle [boe18]. Mit etwas mehr Zeit hätte man auch noch die Analyse des Naive-Bayes in einem MapReduce verwenden können.

2.1.4. Spark

Spark ist ein Big Data Framework zur Verarbeitung, Filterung und Analyse von großen Datenmengen. Diese Bibliothek vereinfacht die Anwendung eines MapReduce, indem es verschiedene Funktionsweisen und Tools dafür anbietet. So begrenzt sich der Programmcode auf die wesentlichen Funktionen eines MapReduce und verschafft dadurch eine guet Übersicht über den Code. Des Weiteren stellt Spark verschiedene Datenstrukturen zur Verfügung, um die Arbeit mit großen Datenmengen zu erleichtern. Dazu gehört zum Beispiel das Resilient Distributed Datasets (RDD). Ein weiter Vorteil, den Spark bietet, ist die hohe Verarbeitungsgeschwindigkeit der Daten. Aus diesen Gründen hat sich das Forscherteam für das Mapreduce entschieden, welches in dem Forschungsprojekt Anwendung finden soll, diese Bibliothek zu verwenden, um eine einfach und zeit effiziente

Verarbeitung der Twitterdaten zu haben.

2.2. Datenverarbeitung

In diesem Kapitel wird näher auf den Programmcode des vorliegenden Forschungsprojektes eingegangen und erklärt was genau in der Datensammlung und Datenverarbeitung gemacht wurde. Als erstes hat das Forscherteam die Daten wie in Punkt 2.1.1 beschrieben mit der Bibliothek gescrapped und als csv-Datei abgespeichert. Wie das Sracpping in der Bibliothek genau funktioniert ist ebenfalls unter dem Punkt 2.1.1. GetOldTweets beschrieben. Um die gespeicherten Tweets für das Mapreduce vor zu verarbeiten nutzen das Team die zur Verfügung stehenden Bibliotheken NLTK und cleantext.

```
start = time.time()
directory = '/data/'
directory_sanitized = '/data_sanitized/'

path = os.getcwd()+directory
user_file_names = sorted_alphanumeric(os.listdir(path))

current_user_sanitized = []

for user_file in user_file_names:
    with open(path + user_file, 'r') as file:
        for line in file:
            for word in line.split("\n\n"):
                current_user_sanitized.append(tweetDecomposer(word))
        save_sanitized_file(user_file, directory_sanitized,
                            current_user_sanitized)
        current_user_sanitized = []

print(">>>> JOB DONE, it took " + str(round(time.time() - start, 2)) + " seconds")
```

Figure 2.3.: Sanierungs-For-Schleife der Daten

Die Variable "directory" und "directory_ Sanitized" geben den Path an, in welchen Ordner die Daten gespeichert werden sollen. mit der bibliothek os von Python kann man zum Beispiel über "os.listdir(Path)" alle Dateinamen innerhalb dieses Ordners einlesen lassen. Über die Variable user_ file_ names bekommt man eine alphanumerisch sortierte Liste der Usernamen der Politiker zurück, welche dann über eine For- Schleife durchgegangen werden, da der Name der CSV-Dateien mit folgenden Muster den Username entspricht,

"Name_ D" oder "Name_ R". D steht für demokratisch und R für republikanisch. In dieser Datensanierungsschleife wird die Funktion tweetDecomposer verwendet. Diese Funktion übernimmt in der vorliegenden Datensanierung die Hauptaufgabe.

Mit dem Bibliothek cleantext wurden die emojis aus dem Text entfernt, wie man in Abbildung A.1 in den ersten Zeilen der Funktion sehen kann. Dann werden alle Worte innerhalb eines Tweets klein geschrieben und aufgetrennt, damit dann die ID, die Zeitzone und der Username aus dem Tweet entfernt werden kann. Mit NLTK werden dann die Stoppworte durch ein join aus den Tweets entfernt, so das man zu den letzten Datensanierungsschritten kommen kann.

Da die Annotations und Hashtags gespeichert werden sollen, wurde, wie in Abbildung B.1 zu sehen, ist eine extra For-Schleife. Die for-Schleife läuft über den gesamten Input des Tweets, dazu zählt Datum, Zeit, Username, Hashtags,Emoji usw. Als erstes wird überprüft ob es sich um eine URL handelt oder nicht. Tritt der Fall ein das es eine URL ist wird diese einfach übersprungen und nicht mit abgespeichert. Die Annotations können durch eine "@" erkannt werden, während die Hashtags mit einem "# " erkannt werden. Beide Erkennungsmaker werden nicht mit abgespeichert. Der restliche Inhalt des Hastags und der Annotation werden in einer Liste gespeichert. Als letztes haben wir alles Spezielle Charakter, wie Punkte, Kommas, usw. entfernt aus den tweetword. Die Funktion gibt dann alle interessanten Daten für die Analyse zurück.Zum Schluss werden diese Daten in einer CSV-Datei gespeichert Ordner data_ sanitized. Als nächster Schritt folgt die Hauptanalyse unseres Projektes in 2.3.

2.3. MapReduce

Hallo Palmoooooo und Walta

3. Analysing data (good title missing)

3.1. data1 (good title missing)

Hey Kennyyyyy

3.2. data2 (good title missing)

Hey Kennyyyyy

4. Schluss

4.1. Resume

Wie sie sehen sehen sie nichts.

A. Code Example Sanitization 1

```
def tweetDecomposer(tweet):

    # Stop on last line
    if tweet.find("No more data. finished scraping!!") == 0:
        return

    # Save, then remove emojis
    # emojis = adv.extract_emoji(tweet)
    tweet = clean(tweet, no_emoji=True)

    # Seperate by word and stop on too few lines
    tweet = tweet.lower()
    tweetWords = tweet.split()
    if len(tweetWords) < 4:
        return

    # remove tweet ID
    tweetWords = tweetWords[1:]
    date = tweetWords[0]
    time = tweetWords[1]
    timezone = tweetWords[2]
    # remove time/timezone and tweet username
    tweetWords = tweetWords[4:]

    mentions = []
    hashtags = []
    text = ""

    # removing stopwords
    tweetWords = " ".join([word for word in tweetWords
                           if word not in STOP_WORDS]).split()
```

Figure A.1.: Ausschnitt eins der Sanierungsfunktion der Daten

B. Code Example Sanitization 2

```
    for tweetWord in tweetWords:

        if remove_url(tweetWord):
            continue

        # Annotations
        if tweetWord[0] == "@":
            if tweetWord[1:] == " ":
                continue
            annotation = tweetWord[1:].strip()
            annotation = annotation.strip('.')
            mentions.append(annotation)
            continue
        # Hashtags
        if tweetWord[0] == "#":
            if tweetWord[1:] == " ":
                continue
            hashtag = tweetWord[1:].strip()
            hashtag = hashtag.strip('.')
            hashtags.append(hashtag)
            continue

        # remove special characters
        tweetWord = re.sub('[^A-Za-z0-9 ]+', '', tweetWord)

        if len(tweetWord) == 0:
            continue

        text += tweetWord + " "
    if len(text) == 0:
        return None
    return date, time, text, hashtags, mentions
```

Figure B.1.: Ausschnitt eins der Sanierungsfunktion der Daten

Literature

[al.14] al., Michael D.: *Political Polarization in the American Public*. 2014

[boe18] boersennews: *Patternanalyse*. https://www.boersennews.de/lexikon/begriff/pattern/852/. Version: 2018. – Last visited on 28.01.2023

[Hen18] Henrique, Jefferson: *GetOldTweets-python*. https://github.com/Jefferson-Henrique/GetOldTweetspython/blob/master/got3/manager/TweetManager.py. Version: 2018. – Last visited on 27.01.2023

[Hen19] Henrique, Jefferson: *GetOldTweets3 0.0.11*. https://pypi.org/project/GetOldTweets3/. Version: 2019. – Last visited on 27.01.2023

[Lor17a] Loria, Steven: *TextBlob-Sentimentanalysen*. https://github.com/sloria/TextBlob/blob/dev/textblob/en/sentiments.py. Version: 2017. – Last visited on 27.01.2023; Funktion PatternAnalyzer

[Lor17b] Loria, Steven: *TextBlob-Sentimentanalysen Github*. https://github.com/sloria/TextBlob. Version: 2017. – Last visited on 27.01.2023

[Lor20a] Loria, Steven: *TextBlob*. https://github.com/sloria/TextBlob/blob/dev/textblob/blob.py. Version: 2020. – Last visited on 27.01.2023; Class TextBlob

[Lor20b] Loria, Steven: *Tutorial: Quickstart*. https://textblob.readthedocs.io/en/dev/quickstart.html. Version: 2020. – Last visited on 28.01.2023

[NT23] NLTK-Team: *NLTK-Documentation*. https://www.nltk.org/. Version: 2023. – Last visited on 27.01.2023

[Yos20] Yoss, Andrea: *GetOldTweets3*. https://andrea-yoss.medium.com/

`getoldtweets3-830ebb8b2dab`. Version: 2020. – Last visited on 27.01.2023