Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Human-Centered Computing (HCC)

Entwicklung eines Klassifizierers für die Aufgaben von Bots auf Wikidata

Ronald Scheffler

Matrikelnummer: 3788886

rscheffle@zedat.fu-berlin.de

Betreuerin und Erstgutachterin: Prof. Dr. C. Müller-Birn

Zweitgutachter: Prof. Dr. Lutz Prechelt

Berlin, 23.10.2018

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, den 23.10.2018

Ronald Scheffler

Zusammenfassung

Gemeinschaftliches Arbeiten an Projekten war schon immer ein Kernpunkt für die Nutzung des Internets. Der Umfang du die Popularität des Internets wuchsen rasant und ermöglichten nicht nur Institutionen, sondern auch Individuen den Austausch und Speicherung elektronischer Daten und die Kollaboration an gemeinsamen Projekten. Seit Projekte wie Wikipedia immer populärer und von einer breiteren Öffentlichkeit genutzt wurden, wurde auch verstärkt der Einsatz von Hilfssoftware in Form von Bots notwendig, um die inhaltliche und strukturelle Integrität der Gemeinschaftsprojekte zu gewährleisten. Das Wikidata Projekt dient als Speicherstelle für strukturierte Daten aller zugehörigen Wiki Projekte und nutzt Bots für die automatisierte Moderation seiner Inhalte und als Unterstützung der menschlichen Nutzer.

Bots sind Softwareprogramme, die eigenständig Änderungen entsprechend ihres eigenen vordefinierten Ablaufplans und ohne menschliche Einmischung durchführen. Da sie als Teil der Infrastruktur der Wikis, auf denen sie arbeiten, angesehen werden können müssen sie entsprechenden Protokollen folgen und ihre geplanten Aktionen bewilligt bekommen. Informationen darüber, zu welchen Aufgaben diese Bots heutzutage berechtigt sind, in welchem Umfang sie diese Erledigen und wie korrekt sie dabei sind, gibt es jedoch wenige. Der Fokus dieser Arbeit liegt auf der Erstellung eines Klassifizierers für die von Bots geleisteten Aufgaben im Wikidata Projekt. Der Klassifizierer beschreibt welche Aufgaben und in welchem Umfang von Bots geleistet werden und soll als Grundlage für weitere Untersuchungen im Bereich automatisierter Moderation dienen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation und relevante wissenschaftliche Arbeiten

Das Internet als Informationsquelle zu nutzen ist heutzutage eine Selbstverständlichkeit. Es gibt kaum eine Information, die man nicht auf einer entsprechenden Webseite nachschlagen oder einfach „googlen“ kann. Während die Informationsdichte auf den meisten Webseiten vergleichsweise gering ist gibt es jedoch auch Webseiten und Plattformen wie Wikidata [1], die speziell für das sammeln und verwalten von Informationen angelegt sind.

Wikidata ist eine kostenlose, von seinen Nutzern gemeinschaftlich gepflegte Datenbank für strukturierte Daten. Hierbei werden nicht nur die Informationen selbst in strukturierter Art und Weise und in vielen verschiedenen Sprachen gespeichert, sondern auch deren Verknüpfungen unter einander und zu anderen Datenbanken. Die Offenheit der Wikidata Plattform sorgte dafür, dass im Laufe der Jahre die Menge der gesammelten Informationen enorme Ausmaße annahm und ständig weiter anwächst. Die Strukturiertheit der Daten erlaubt es zudem nicht nur menschlichen Nutzern, mit Hilfe vielfältiger Werkzeuge [2], sondern auch Bots, neue Daten hinzuzufügen, abzufragen oder zu verändern. Mit Steigerung der Menge der Informationen und der Komplexität ihrer Struktur steigen auch die Anforderungen für die Verfügbarkeit, Korrektheit, Vollständigkeit und Nutzbarkeit dieser Informationen. Die Menge an Daten übersteigt schon lange ein Maß, das allein von Menschen gehandhabt werden kann. Daher kommen bei Wikidata auch Bots [3], Softwareprogramme, die selbstsändig nach fest definierten Vorgaben Änderungen an Daten vornehmen können, zum Einsatz.

Nicht nur Wikidata selbst betreibt Bots für ihre Plattform. Jeder Interessierte kann selbst einen Bot erstellen und auf Wikidata operieren lassen, so lange der Bot die vorausgesetzten Anforderungen erfüllt und von Wikidata akzeptiert wird. Die Aufgaben, die von einem Bot übernommen werden können daher sehr vielfältig sein, von Änderungen an ausgewählten Daten bis hin zur Pflege des gesamten Datenbestandes. Um einen Überblick darüber zu erhalten welche Aufgaben von Bots übernommen werden, in welchem Umfang und in welchen Kombinationen, soll ein Klassifizierer erstellt werden. Als Datengrundlage stellt

[1] `www.wikidata.org`

[2] `https://tools.wmflabs.org/hay/directory/#/search/wikidata`

[3] `https://tools.wmflabs.org/hgztools/botstatistics/?lang=www&project=wikidata&dir=desc&sort=ec`

Wikidata bereits Datenlogs in verschiedenen Formaten zur Verfügung. [4]

In der Vergangenheit wurden bereits Klassifizierungen von Nutzerverhalten auf öffentlichen Datenbanken vorgenommen. Der Anstoß für diese Arbeit ist eine Klassifizierung von menschlichen Nutzern und Bots von Prof. Müller-Birn [3]. Auch wenn das ursprüngliche Augenmerk dieser Arbeit nicht auf einer reinen Analyse des Verhaltens der Bots lag, sondern auf dem Verhalten der menschlichen Nutzer, wurde hier eine sehr umfangreiche Klassifizierung der Wikidata Nutzer erreicht. Sowohl der Gesamtumfang der Arbeit der Bots als auch deren Aufgabenverteilung zeigt auf, dass die Arbeit der Bots auf Wikidata relevant ist und in Zukunft wohl noch an Relevanz dazugewinnen wird.
Denny Vrandecic and Markus Krötsch [2] diskutieren in ihrem Paper die Strukturiertheit der Daten in Wikidata und geben Ausblicke auf softwaregesteuerte Anwendungmöglichkeiten, die sich daraus ergeben. Thomas Steiner vergleicht in seiner Arbeit [4] ebenfalls menschliche Nutzer von Wikidata und Bots und entwickelte eine Anwendung, die absolvierte Edits live streamen kann und die Anzahl der jeweils getätigten Edits vergleicht. John Riedl und Aaron Halfaker [1] diskutieren in ihrer Arbeit wie wichtig Bots für Wikipedia und andere Datenbanken sind um diese vor böswilligen Nutzern zu schützen. Cristina Sarasua [5] untersucht in ihrer Arbeit das Verhalten von Nutzern auf Wikidata. Viele verwandte Arbeiten, die sich ebenfalls mit dem Nutzen von Bots auf Wissensdatenbanken befassen, begreifen jedoch auch menschliche Nutzer mit ein oder beinhalteten auch andere Datenquellen, die über Wikidata hinausreichten. Der Fokus dieser Arbeit soll ausschließlich auf Bots und Wikidata liegen.

1.2 Zielsetzung der Arbeit

Ziel dieser Arbeit ist die Entwicklung eines Klassifizierers für die Aufgaben der Bots, die auf dem Wikidata Graph operieren. Dieser soll Aufzeigen können welche Aufgaben die Bots übernehmen, in welchem Umfang sie Einfluss auf die Daten nehmen sowie die Veränderung der Aufgaben im Laufe der Zeit.

Das Hauptaugenmerk dieser Arbeit soll auf der Klassifizierung der Aufgaben der Bots liegen. Dafür muss festgestellt werden welche Nutzer menschlich und welche tatsächlich Bots sind. Darüber hinaus muss analysiert werden in welcher Struktur die Informationen über Änderungen, die an Wikidata vorgenommen wurden, gespeichert werden und wie auf diese zugegriffen werden kann. Aus diesen Änderungen müssen dann diejenigen ausgewählt werden, die von Bots ausgeführt werden. Nachdem die verfügbaren Daten analysiert wurden muss ein dafür passender Klassifizierungsalgorithmus gewählt werden. Anschließend werden die Daten so aufbereitet, dass der Algorithmus sie verarbetien kann. Letztlich wird anhand dieser Informationen ein Klassifizierer für die Aufgaben

[4] `https://www.wikidata.org/wiki/Wikidata:Database_download`

der Bots erstellt.
Zusätzlich sollte die Klassifizierung auch die Veränderung der Aufgaben und Aufgabenverteilung der Bots im Laufe der Zeit mit einbeziehen. Während die verfügbaren Daten eine solche Analyse ermöglichen war es mir jedoch nicht möglich große Datenmengen, die einen längeren Zeitraum von Änderungen an Wikidata abdecken, zu analysieren. Trotz ausreichender Hardwareressourcen produzierte mein Arbeitsrechner, unabhängig von der Implementierung des Codes, gravierende Fehler.
Daher beschränkt sich die Klassifizierung nur auf einen kurzen Zeitraum mit den aktuellsten verfügbaren Daten im Zeitraum vom 06.09.2018 bis 17.20.2018.

1.3 Aufbau der Arbeit

Die folgenden Kapitel beschäftigen sich mit den notwendigen Schritten für die Klassifizierung. Dabei werden in der auch englische Begriffe für Bezeichnungen verwendet um diese nicht durch eine Übersetzung zu verfälschen.
Kapitel 2 beschreibt die Struktur der Daten in Wikidata, und wie Bots damit interagieren können. In Kapitel 3 wird das Parsen der zur Klassifizierung notwendigen Daten beschrieben. Kapitel 4 erläutert die Auswahl einer geeignete Klassifizierungsmethode und die Durchführung der Klassifizierung der Bots anhand der verfügbaren Daten. In Kapitel 5 werden die Ergebnisse und Einschränkungen der Klassifizierung diskutiert und eine Ausblick auf mögliche zukünftige Arbeiten zu diesem Thema gegeben.

2 Wikidata und Bots

Wikidata ist ein eine öffentliche Datenbank in der Daten in strukturierter Form gespeichert werden. Dies ermöglicht nicht nur Menschen, sondern vor allem auch Maschinen auf Daten zuzugreifen und diese zu verarbeiten. Softwaregestützter Zugriff auf Wikidata Daten steht dabei allen offen, Privatpersonen wie auch Organisationen oder Regierungen.
Die Daten selbst sind in Wikidata in „Items“ organisiert. Jedes Item verfügt über einen einzigartigen „Identifier“ und eine Reihe von zusätzlichen Informationen. Diese umfassen eine „Description“, „Labels“ und „Aliases“. Diese bilden die „Terms“ eines Items. Zusätzliche Informationen finden sich in den „Statements“ eines Items. Diese beschreiben durch „Properties“ und dazugehörigen „Values“ die Eigenschaften des Items. Zusätzlich können „References“ und „Sitelink“ angegeben werden, die auf andere relevante Items beziehungsweise externe Quellen verweisen.

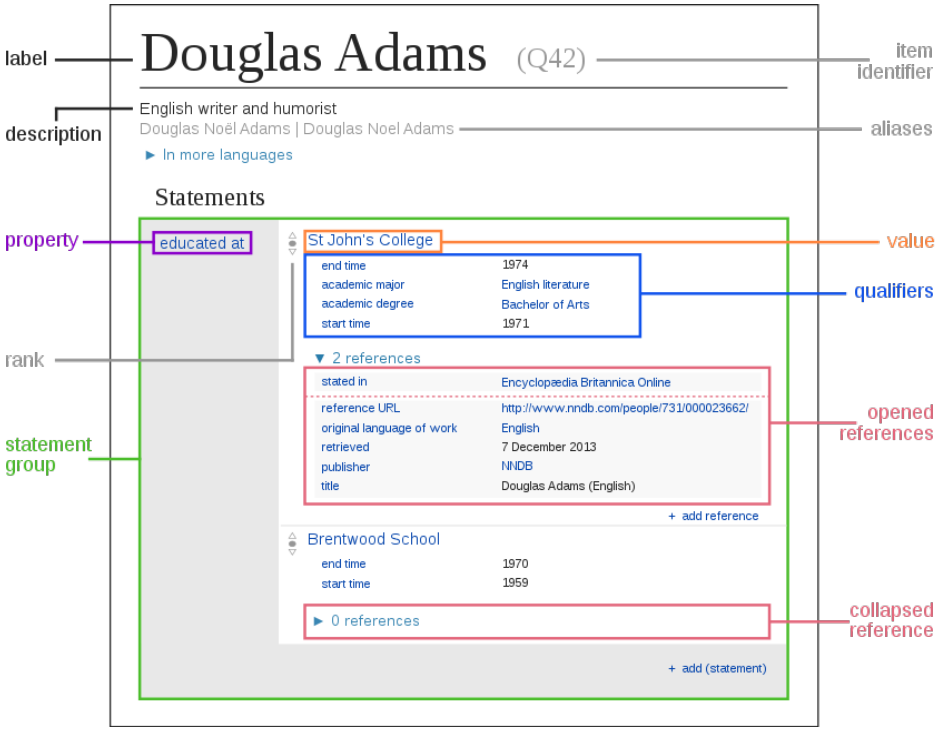

Abbildung 2.1: Aufbau eines Items am Beispiel von Douglas Adams

Wikidata stellt eine API [1] bereit über die Maschinen auf die Datenbank zugreifen können. Items sind jedoch nicht die einzigen Dinge mit denen Bots

[1] `https://www.mediawiki.org/wiki/API:Main_page`

interagieren können. In Namespaces [2] ist festgelegt auf welche Bereiche von Wikidata zugegriffen werden kann. Dazu gehören zum Beispiel die Definitionen von Templates oder Properties, sowie die dazugehörigen Diskussionseiten, auf denen Änderungen am jeweiligen Modul von menschlichen Nutzern diskutiert werden können.

2.1 Bots

„Bots“ sind Programme, die autonom auf Wikidata operieren können und Aufgaben ausführen, die für Menschen oft zu langwierig oder umfangreich sind. Über die Wikidata API erhalten sie Zugriff auf die Datenbank. Um auf Wikidata operieren zu können müssen Bots aktiv über einen Account eingeloggt sein. Somit sind all ihre Edits über ihren Nutzernamen auf sie zurückzuführen. Dieser Umstand wird in dieser Arbeit später genutzt um aus der Edit-History diejenigen Änderungen herauszufiltern, die von Bots ausgeführt wurden. Um durch ein Bot-Flag in der Datenbank gekennzeichnet zu werden müssen Bots zunächst einen Genehmigungsprozess durchlaufen. Sie können jedoch auch schon vor dieser Genehmigung auf Wikidata operieren. Um die Bots zu identifizieren, die in dieser Arbeit klassifiziert werden sollen müssen Informationen aus verschiedenen Quellen ausgewertet werden.
Über die Mediawiki API [3] können diejenigen Bots abgefragt werden, die schon als Bot geflagt wurden. Bots, die sich gerade im Genehmigungsprozess befinden können auf der entsprechenden Webseite [4] geparsed werden. Zusätzlich stellt Wikidata Listen für Bots mit Bot-Flag[5], ohne Bot-Flag[6] und Extension Bots[7], bereit, die ebenfalls geparsed werden können.
Die in dieser Arbeit verwendete Liste aller Bots wurde von Marisa Nest, Mitarbeiterin der FU-Berlin, erstellt. Insgesamt wurden 482 Bots identifiziert.

2.2 Wikidata Edit-History

Die Edit-History für alle vergangenen Änderungen an Wikidata wird in einem XML Format zum Download bereitgestellt [8]. Die Edits sind dabei als „Revisions“ in „Pages“ organisiert, welche alle Änderungen an einem bestimmten Bereich, zum Beispiel einem Item, im entsprechenden Namespace enthalten.

[2] https://www.wikidata.org/wiki/Help:Namespaces
[3] https://www.mediawiki.org/wiki/API:Main_page
[4] Botflaghttps://www.wikidata.org/wiki/Wikidata:Requests_for_permissions/Bot
[5] https://www.wikidata.org/wiki/Category:Bots_with_botflag
[6] https://www.wikidata.org/wiki/Category:Bots_without_botflag
[7] https://www.wikidata.org/wiki/Category:Extension_bots
[8] https://www.wikidata.org/wiki/Wikidata:Database_download

3 Parsing der Daten

Für das Parsen der Daten wird aktuellste Anaconda Distribution für Python 3 genutzt. Sie enthält bereits viele Bibliotheken für das einfache Parsen von XML Daten (mittels Elementtree) sowie für die spätere Klassifizierung der Daten.

Edits, die in den XML Dumps gespeichert sind haben folgende Form:

```
<page>
  <title>Q23</title>
  <ns>0</ns>
  <id>136</id>
  <revision>
    <id>583397567</id>
    <parentid>579540697</parentid>
    <timestamp>2017-10-24T03:46:30Z</timestamp>
    <contributor>
      <username>Reinheitsgebot</username>
      <id>940976</id>
    </contributor>
    <comment>/* wbsetreference-add:2| */ [[Property:P569]]: 22 February
        1732, #quickstatements; invoked by Mix&amp;apos;n&amp;apos;match:
        References</comment>
    <model>wikibase-item</model>
    <format>application/json</format>
    <text id="586766906" bytes="162826" />
    <sha1>j9ywk7yzscooec6l8e7id1tkbvqb8pu</sha1>
  </revision>
</page>
```

Listing 3.1: K-Means.

Die für die Klassifizierung relevanten Daten sind hierbei:

- revision: Der Edit, der untersucht werden soll
- ns: der Namespace auf dem der Edit ausgeführt wurde
- username: der Name des Nutzers, der den Edit ausgeführt hat
- comment: Informationen darüber welche Änderungen durchgeführt wurden

Edits können sowohl von registrierten als auch unregistrierten Nutzern vorgenommen werden. Im Falle eines unregistrierten Nutzers wird als „contributor“ an Stelle von Nutzername und Nutzer-ID nur eine IP-Adresse gespeichert. Da Bots nur Änderungen ausführen können, wenn sie auch eingeloggt sind, können diese Änderungen ignoriert werden.
Die Edit-History enthält keine Informationen darüber ob der Nutzer, der den Edit durchgeführt hat, ein Mensch oder ein Bot ist. Informationen über vorhandene Bot-Flags werden nicht im XML Log gespeichert.
Während des Parsings wird die gesamte XML Struktur nach jeder Page durchsucht. Für diese Page wird wiederum jede Revision daraufhin untersucht ob sie von einem Bot ausgeführt wurde. Ist das der Fall werden die Daten der Revision und der zugehörigen Page gespeichert.

3.1 Definition der Actions

Informationen über die tatsächliche Aufgabe, die mit diesem Edits ausgeführt wurde, werden im „comment“ gespeichert. Während des Parsing Prozesses wurden zunächst alle einzigartigen Comments für jeden Namespace gespeichert und anschließend auf existierende Muster untersucht. Dabei stellte sich heraus, dass sich die Informationen in den Comments, je nachdem welcher Bereich bzw. welcher Namespace von Wikidata editiert wurde, stark unterscheiden können und unterschiedlich stark strukturiert und aussagekräftig sind. Für die Namespaces 0 und 120, die Namespaces für Änderungen an Items und Properties, folgen die Comments einem festen Schema, durch das die ausgeführte „Action“ identifiziert werden kann. Zur Definition ein Action, einem Label für einen konkreten Edit, den ein Bot auf Wikidata ausführt, wurde der Teilstring zwischen dem ersten Leerzeichen und dem ersten Doppelpunk als Action ausgewählt. So wird zum Beispiel aus dem Edit

- /* wbsetreference-add:2| */ [[Property:P569]]: 22 February 1732, #quickstatements;

die Action „wbsetreference-add“. Die Zusatzinformationen, die nach dem Bindestrich folgen (in diesem Beispiel „add“ werden mit in die Action aufgenommen. Dies resultiert in den meisten Fällen in unterschiedlichen Actions, die zwar das gleiche Element eines Items oder einer Property verändern, aber mit einem unterschiedlichem Resultat. Zum Beispiel kann die Action „wbeditentity“ ein Entity Object update, überschreiben, oder komplett neu anlegen.

- wbeditentity-update
- wbeditentity-override
- wbeditentity-create

Dabei muss auch beachtet werden, dass diese Zusätze bei einigen Edits optional sind. Manche Edits, zum Beispiel „wbsetclaimvalue“ , kommen nur ohne diese Zusatzinformation vor.

3.2 Technischer Ablauf

Zum Parsen wird sowohl der XML Dump als auf die Liste der Botnamen im XML Format eingelesen. Für jede Edit wird zunächst geprüft ob er von einem Bot durchgeführt wurde. Falls der Edit von einem Bot durchgeführt wurde werden die relevanten Daten aus diesem Edit gespeichert. Nachdem alle Edits geparsed wurden werden den gespeicherten Daten die notwendigen Informationen für den Klassifizierungsprozess berechnet und in einer CSV-Datei gespeichert. Da als Datenquelle viele einzelne XML Dateien genutzt wurden, die unabhängig von einander geparst werden, werden auch dementsprechend viele CSV-Dateien generiert. Diese CSV-Dateien werden anschließend zu einer einzelnen Datei zusammengeführt, die dem Klassifizierungsalgorithmus als Datengrundlage dient.

3.3 Ergebnisse

Durch das Parsen der Edit-Historie ergab sich eine Gesamtzahl von 24240362 Edits, 16445943 davon (67,85) wurden von Bots ausgeführt. Für die Bot-Edits in den verschiedenen Namespaces ergab sich folgende Verteilung:

NS	Name	# Edits
0	Items	16755251
1	Talk	107
2	User	44033
3	User Talk	878
4	Wikidata	89569
5	Wikidata Talk	269
8	Mediawiki	18
9	Mediawiki Talk	2
10	Template	640
12	Help	645
13	Help Talk	3
120	Property	14
121	Property Talk	191
1198	Translation	171
2600	Topic	432

Tabelle 3.1: Verteilung der Bot-Edits bezogen auf Namespaces

In nicht aufgeführten Namespaces fanden im beobachteten Zeitraum keine

Edits statt. Von den 482 Bots, die derzeit auf Wikidata operieren können waren lediglich 96 im beobachteten Zeitraum aktiv. 4 dieser Bots haben dabei jedoch nur fünf oder weniger Edits durchgeführt. Sie werden daher bei der Klassifizierung ignoriert. Die Anzahl der Actions, die jeder einzelne Bot durchgeführt hat schwankt extrem. Die Aktivsten Bots haben mehrere Millionen Actions durchgeführt, während wenig aktive Bots bei unter 100 Actions verbleiben. QuickStatementsBot allein ist für 27,6% aller registrierten Bot-Edits verantwortlich. Um diese Streuung zu mindern werden die diese Werte vor dem Clustern normalisiert. Auch wenn die überwältigende Mehrheit aller Edits auf dem Item Namespace ausgeführt wird, so existieren doch Bots, die auch andere Bereiche von Wikidata überwachen und editieren können. Im beobachteten Zeitrahmen wurden 273477 (1,6% aller Bot-Edits) Änderungen an Namespaces außerhalb von Item und Properties durchgeführt.

4 Klassifizierung

Ziel der Klassifizierung ist eine Klassifizierung anhand der Aufgaben die die Bots auf Wikidata erfüllen. Da hierbei neue Daten betrachtet werden und kein „Ground Truth“, also keine Vergleichsgrundlage von einer vorangegangenen Klassifizierung existiert, aufgrund derer man die neuen Daten klassifizieren kann, soll hierfür ein Algorithmus für „Unsupervised Learning“ genutzt werden.

4.1 Clustering

Wie auch bei der vorangegangenen Arbeit von Prof. Müller-Birn [3] soll zur Gruppierung der Daten ein Clustering-Algorithmus angewendet werden. Dabei werden die Daten anhand einer Metrik zu Clustern zusammengefasst. Bots, die ähnliches Verhalten beim editieren von Wikidata aufweisen, werden zusammen gruppiert.
Als Clustering-Algorithmus wird K-Means [7] verwendet. Dieser Algorithmus wandelt jeden Wert eines der n Features in einen Vektor um, der in einem n-dimensionalen Raum verteilt wird. Danach unterteilt der Algorithmus diese Punkte in k Cluster. Die Zugehörigkeit eines Punktes wird von seinem Abstand zum nächsten Centroid oder Mean, dem zentralen Punkt eines jeden Clusters, bestimmt. Die Bestimmung neuer Centroids und die Zuweisung der Punkte zu den Centroids bzw. Clustern wird so lange wiederholt bis kein signifikanter Abstand zwischen den neuen Centroids und denen der letzten Iteration besteht.
Die Vorteile dieses Algorithmus sind seine Einfachheit und die Tatsache, dass er auch mit großen Datenmengen umgehen kann. Der Nachteil besteht darin, dass die Berechnungen mehrfach durchgeführt werden müssen, jedes Mal mit neuen Centroids. Zudem muss vor der Ausführung des Algorithmus festgelegt werden wie viele Cluster Erzeugt werden sollen.

4.1.1 Implementierung

Die Implementierung der Klassifizierung wurde in Python mit Hilfe der sciKit-learn [1] Bibliothek durchgeführt. Diese erlaubt eine einfache Berechnung der Cluster und die Auswahl der benötigten Parameter.

```
1 def cluster(csv, k):
2
3     data = pd.read_csv(csv)
```

[1] http://scikit-learn.org/stable/index.html

```
    # X Features
    X = np.array(data.drop(['botname'], 1))
    X = scale(X.data)
    clustering = KMeans(n_clusters = k,init ='k-means++',n_init= 10,
        random_state = 6)

    clustering.fit(X)
    X_scaled = X
    result = clustering.fit_predict(X)

    data['Cluster'] = result
    data = data.sort_values(['Cluster'])
    data.to_csv(r"C:\Users\Ronald Scheffler\.spyder-py3\clusterresult"+
        str(k)+".csv")

    print(silhouette_score(X_scaled, result))

cluster(r"C:\Users\Ronald Scheffler\.spyder-py3\clustermergemin5.csv",
    5)
```

Listing 4.1: K-Means.

Mit init=k-means++ werden die Startpunkte der Centroids weit von einander entfernt festgelegt. Dadurch sollten sie sich schneller einander annähern als bei einer zufälligen Verteilung und somit schneller zu einem Ergebnis kommen. Mit n_init kann eine feste Anzahl an Iterationen festgelegt werden.
Bei der Klassifizierung der Botdaten hat sich eine Änderung dieser Daten jedoch nicht signifikant ausgewirkt.
Die Anzahl der Cluster, die der Algorithmus erzeugen soll, kann durch die Berechnung der „Elbow Curve“ [2] eingegrenzt werden. Dabei wird der Algorithmus mehrfach ausgeführt, jeweils mit einer anderen Menge von Clustern. Jedem Ergebnis wird ein Score zugewiesen. Der Plot der Ergebnisse zeigt eine charakteristische Ellbogenkurve. Der Ellbogen ist hierbei der Punkt ab dem eine Erhöhung der Anzahl der Cluster keine signifikante Verbesserung des Clusteringergebnisses mehr liefert.
Abbildung 4.1 zeigt, dass ab 5 Clustern keine starke Verbesserung des Ergebnisses mehr auftritt.

4.1.2 Auswahl der Features

Um eine Klassifizierung der Actions zu ermöglichen müssen zunächst die Features bestimmt werden, anhand derer die Bots in Gruppen aufgeteilt werden sollen. Ziel ist es eine Gruppierung anhand der Actions der Bots vorzunehmen. Das Problem dabei ist, die Action anhand der Comments in der Edit-History festgelegt werden. Je nach Namespace, auf dem der Edit durchgeführt wurde,

[2] https://www.datasciencecentral.com/profiles/blogs/python-implementing-a-k-means-algorithm-with-sklearn

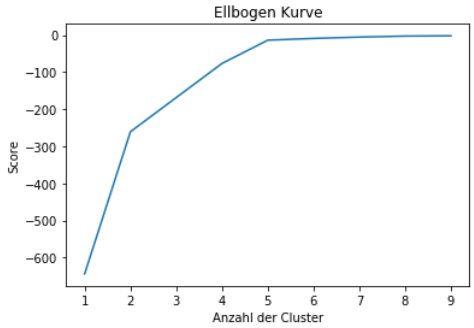

Abbildung 4.1: Elbow Curve für KMeans

unterscheiden sich Comments stark an Aussagekraft und Struktur. In einigen Fällen existiert überhaupt kein Comment anhand dessen eine Action definiert werden könnte. Wie zuvor erwähnt lassen sich jedoch zumindest für die potentiell interessantesten Namespaces einfach geeignete Actions aufstellen.

- Namespace 0: Items
- Namespace 120: Property
- Namespace 121: Property Talk

Wie auch in der Arbeit von Prof. Müller-Birn soll hier der Property Talk Namespace mitbehandelt werden. Dies soll eine gewisse Vergleichbarkeit der Ergenisse gewährleisten. Auch wenn nicht davon auszugehen ist, dass sich Bots aktiv an Diskussionen beteiligen, so können sie auch in diesen Bereichen Aufgaben wie Vandalismuskontrolle oder Updaten von Informationen übernehmen. Die Comments, die von Bots in diesem Namespace zu ihren Edits geschrieben werden verfügen nicht über die gleiche Struktur wie die Comments aus den Namespaces Item und Property. Bei der Analyse der Edit-History ist jedoch nur ein einzelner distinkter Comment aufgetreten, wodurch es kein Problem darstellte, diesen für das Clustering einzubinden.

- 'Wikidata list updated'

Nach Analyse aller Edits aus den genutzten XML Dumps stellte heraus, dass nur 1,6% aller Edits auf anderen Namespaces als Item, Property und Property Talk stattfinden. Diese Arbeit wird sich daher auf die Klassifizierung der Bots anhand ihrer Edits in den Namespaces für Items und Properties beschränken. Innerhalb dieser Namespaces wurden 37 unterschiedliche Actions identifiziert. Es ist unklar ob ein Clustering mit dieser Menge von Features ein brauchbares Ergebnis liefern würde. Daher werden die identifizierten Actions zu Action-Set zusammengefasst. Ein Action-Set umfasst eine Reihe von Actions, die sich auf die Änderung eines bestimmten Aspektes von Items oder Properties beziehen.

Für die jeweiligen Namespaces wurden alle Actions zusammengefasst, die Items neu erstellen, die Terms beziehungsweise Statements editieren. Actions

Action-Set	Namespace	Action Ziele	# Edits
AS1	Item	Merge, Create, Redirect	682676
AS2	Item	Claims, References, Qualifier	13906315
AS3	Item	Label, Description, Aliases	1800731
AS4	Item	Sitelinks	55913
AS5	Property	Claims, References, Qualifier	14
AS6	Property	Label, Descriptions, Aliases	0
AS7	Property Talk	Wikidata list	191
AS8	Item, Property	Revert, Protect	103

Tabelle 4.1: Actionsets

zum Editieren von Sitelinks und Actions in Property Talk werden ebenfalls einem eigenen Actionset zugewiesen. Reverts und Protections können zwar in allen betrachteten Namespaces auftreten, führen jedoch zum gleichen Ergebnis und können damit zusammengefasst werden.
Beim Parsen der Edit-Historie werden jeweils alle Actions aller Namespaces gezählt. Die Vorkommen aller Actions, die zu einem Action-Set gehören werden addiert und zusammen mit dem verantwortlichem Bot gespeichert. Die so erhaltenen Zahlen für jedes Action-Set dienen als Features für das Clustering. Die Anzahl der zu untersuchenden Features konnte auf acht reduziert werden. An der Verteilung der Anzahl der Action über die einzelnen Action-Set lässt sich jedoch schon hier erkennen, dass die zu erwartenden Cluster sehr unausgeglichen sein werden.

4.1.3 Ergebnisse des Clusterings

Zum Clustern Bots bezüglich ihrer Actions auf Wikidata wird der K-Means Algorithmus verwendet. Geclustert werden 92 Bots mit nach je 8 Features. Es sollen 5 Cluster erstellt werden.

```
1    clustering = KMeans(n_clusters = k,init ='k-means++',n_init= 10,
         random_state = 6)
```

Änderungen an den Startpositionen oder an der Anzahl der Iterationen bewirkten keine Änderung des Ergebnisses Die Verteilung der Actions und Bots auf die ertellten Cluster sieht wie folgt aus:

Wie bei der ungleichen Verteilung der Action über die Action-Sets zu erwarten war sind die Bots auch extrem ungleichmäßig über die Cluster verteilt. Fast alle Bots sammeln sich in Cluster 1. Die restlichen Cluster werden von jeweils nur einem Bot belegt. Eine Anpassung der Clusteranzahl für den Algorithmus würde dabei keine Verbesserung erzielen, sondern nur dafür sorgen, dass entsprechend mehr oder weniger Bots in den großen Cluster fallen. Die

	Item-Erstellung	Item-Statements	Item-Terms	Edit Sitelinks
Cluster 1	32473	9379405	335247	2
Cluster 2	605000	2614512	35919	0
Cluster 3	45092	1890214	136	12
Cluster 4	111	22184	219	0
Cluster 5	0	0	0	0

Tabelle 4.2: Actionsets 1 - 4

	Edit Properties	Property Talk	Reverts	# Bots
Cluster 1	0	0	2	88
Cluster 2	0	0	0	1
Cluster 3	0	0	0	1
Cluster 4	0	0	101	1
Cluster 5	0	191	0	1

Tabelle 4.3: Actionsets 5 - 8

Anzahl der Bots in den kleinen Clustern würde erhalten bleiben.

# Cluster	Silhouette Koeffizient
2	0.8924706620214305
3	0.8954953984555398
4	0.9026295198478599
5	0.883180696317682
6	0.8309204845325773

Tabelle 4.4: Silhouette Koeffizient

Der Silhouette Koeffizient, wie diskutiert in [6], gibt Auskunft über die Qualität des Clusterings. Die Berechnung des Koeffizienten (siehe Listing 4.1) für mehrere Werte um das gewählte k=5 für die Anzahl der Cluster zeigt, für die gegebenen Daten k=5 ein guter Wert ist. Je weiter das Ergebnis von 0 entfernt ist, um so weniger überlappen die Cluster.

Die extremen Werte bei der Verteilung der Actions in den Cluster machen es schwer die Cluster eindeutig zu differenzieren. In Abbildung 4.2 sieht man eindeutig die Unterschiede bei der Verteilung der Anzahl der Edits. Abbildung 4.3 zeigt die anteilige Verteilung der Actions aus den Action-Sets in den Klassen. Schon Tabelle 3.1 zeigte auf, dass sich die Mehrzahl der Action auf den Namepace der Items beziehen. Der Property Namespace wurde gar nicht genutzt.

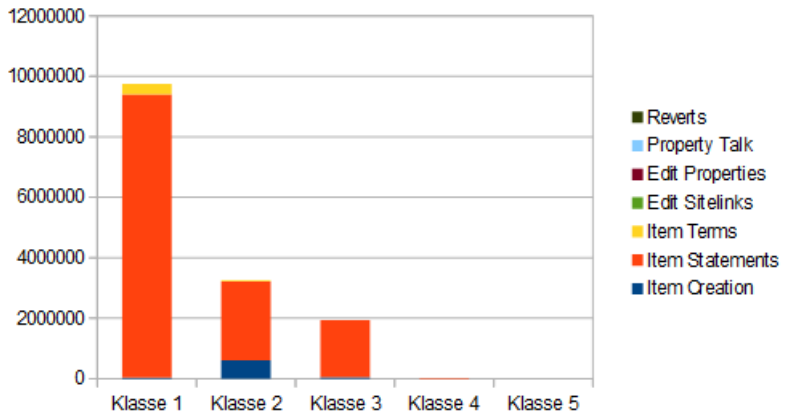

Abbildung 4.2: Anzahl der Actions pro Klasse

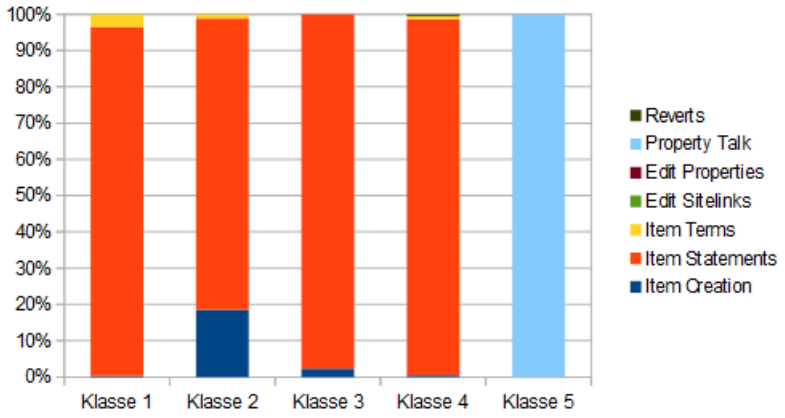

Abbildung 4.3: Anteil der Action-Set pro Klasse

Klasse 1

Die Bots in Klasse 1 haben den Großteil aller Actions absolviert. Dieser Klasse wurde auch der Großteil aller Bots zugeordnet. Zudem wurden hier vor allem Edits an den Statements von Items vorgenommen. Schon bei der Zuordnung der Actions zu den Action-Sets (siehe Tabelle 4.1) waren diese Actions offensichtlich dominant. Aber auch die Erstellung von Items und die Veränderung von Item-Terms werden von Bots in dieser Klasse absolviert. In geringem Umfang können diese Bots auch Sitelinks editieren. Vergleicht man diese Klasse mit der Klassifizierung, die Prof. Müller-Birn [3] in ihrem Paper aufgestellt hat, so lässt sie sich am ehesten als „Item Expert“bezeichen.
Die Folgenden Klassen werden jeweils nur durch einen einzigen Bot repräsentiert.

Klasse 2

Repräsentant der Klasse 2 ist QuickStatementsBot. Dieser Bot allein leistete schon über 25% aller Edits im beobachteten Zeitraum. Sollten noch weitere Bots dieser Art existieren sind sie vermutlich aufgrund eines wesentlich geringeren Arbeitspensums einer der anderen Klassen zugeteilt worden. Die Hauptaufgabe von Bots dieser Klasse ist wiederum das Editieren von Item Statements. Die Erstellung von Items nimmt hier jedoch einen wesentlich höheren Stellenwert ein als bei Klasse 1, dem Item Expert. Das Editieren von Sitelinks kam in dieser Klasse dafür gar nicht vor.

Klasse 3

Repräsentant der Klasse 3 ist KrBot. Abgesehen von der geringen Anzahl an durchgeführten Action ist die Klasse 3 der Klasse 1 ähnlich. Auch hier werden die meisten Änderungen an Item Statements durchgeführt, während Item-Erstellung und Änderungen an Item-Terms in geringerem Maße auch vorkommen. Der größte Unterschied zu den anderen Klassen ist, dass diese Bots anteilsmäßig die meisten Sitelinks editieren. Leider ist die Gesamtzahl dieser Edits in den genutzten Daten noch immer sehr gering, wodurch es fraglich ist, ob dies ein effektives Herausstellungsmerkmal dieser Klasse ist.

Klasse 4

Klasse 4 wird von Florentyna repräsentiert. Wiederum liegt die Gesamtzahl der ausgeführten Actions unter denen der vorangegangenen Klassen. Und auch in dieser Klasse bilden Änderungen an Item-Statements den Hauptanteil aller Actions. Item-Erstellung und Änderungen an Item-Terms kommen fast gar nicht vor, sind aber möglich. Herausstellungsmerkmal dieser Klasse ist, dass auch Reverts von Edits anderer Nutzer vorgenommen werden. Dies kam bisher nur in Klasse 1 vor und das, vor allem gemessen am wesentlich höheren Gesamtvolumen der dort ausgeführten Actions, nur in sehr geringem Maße.

Klasse 5

ListeriaBot wurde Klasse 5 zugeordnet. Diese Klasse unterscheidet sich drastisch von allen anderen Klassen. Hier finden keinerlei Änderungen an Items, Properties oder Sitelinks statt. Die Arbeit dieses Bots scheint sich einzig und allein auf den Namespace Property Talk zu beschränken. Dieser Namespace wurde von allen anderen Bots vollständig ignoriert. Wäre Property Talk nicht mit in das Clustering aufgenommen worden würde diese Klasse gar nicht existieren und 4 Klassen würden für die Klassifizierung der Bots, die auf den Item und Property Namespaces arbeiten, vermutlich ausreichen.

5 Zusammenfassung und Ausblick

5.1 Diskussion der Ergebnisse

Um Bots, die auf Wikidata operieren, nach ihren Aufgabenfeldern zu klassifizieren wurden Edits aus dem Zeitraum vom 06.09.2018 bis zum 17.10.2018 untersucht. Eine Untersuchung einer Veränderung des Verhaltens von Bots war in diesem kurzen Zeitraum nicht sinnvoll möglich.
Auffälligstes Merkmal der untersuchten Daten war die ungleiche Verteilung der Edits in den Namespaces. Wie zu erwarten war fanden die meisten Änderungen im Bereich der Item, die den Großteil der Daten von Wikidata ausmachen, statt. Meine Vermutung ist, dass das Interesse der menschlichen Nutzer vor allem eben diesen Daten in den Items gilt. Daher werden Bots vorrangig beauftragt eben diese Daten auszulesen oder zu aktualisieren. Aufgaben, die die Überwachung and Pflege anderer Wikidata Bereiche betreffen liegen hauptsächlich im Interesse der Betreiber von Wikidata selbst. Solange diese Bereiche nicht unter massiven Angriffen zu leiden haben, werden die zuständigen Bots nie so viel zu tun haben, wie die Bots, die sich gemeinschaftlich um den eigentlichen Datenbestand von Wikidata kümmern.
Es wurde festgestellt, dass 482 Bots auf Wikidata operieren. Die meisten davon wurden geschaffen um in irgendeiner Form Wikidata Items zu editieren. Ob dabei nur bestimmte Items editiert werden oder eine breite Masse der Daten geht aus dem Datenbestand nicht hervor. Es liegt nahe zu vermuten, dass einige Bots dauerhaft aktiv sind um ihre Aufgaben auszuführen. Andere Bots werden nur für einen bestimmten Zweck erschaffen und nicht mehr benutzt, wenn dieser erfüllt ist. Die Tatsache, dass nur 96 aktive Bots im Beobachtungszeitrahmen registriert wurden, legt nahe, dass viele Bots ihrer sehr spezifischen Aufgabe gerade nicht nachkommen müssen. Die Unterschiede in der Arbeitsleistung der Bots konnten in den Daten nachgewiesen werden. Während QuickStatements-Bot wesentlich mehr Edits ausführt als andere Bots führen spezialisierte Bots wie ResearchBot (1 Edit) nur wenige Edits aus.
Aufgrund der relativ kleinen Vergleichsbasis ist es schwer zu beurteilen wie sinnvoll die Einteilung der Bot-Edits zu Action-Set tatsächlich war. Die Reduktion der Menge der Features für das Clustering Verfahren war notwendig, doch das Ungleichgewicht der Menge der Daten innerhalb der Actionsets sorgten auch für ein unausgewogenes Ergebnis des Clusterings. Zu meiner Überraschung wurden keine Edits im Property Namespace festgestellt. Beim Erstellen der Features für den K-Means Algorithmus hätte man dieses Action-Set ignorieren können.
Das Clustering ergab 5 unterscheidbare Gruppen von Bots, die sich alle verständlich anhand ihrer Merkmale bezüglich der vorgenommenen Actions un-

terscheiden lassen. Von den von Prof. Müller-Birn definierten Klassen ließ sich jedoch nur eine einzige, der Item Expert (Klasse 1), sicher feststellen. Die anderen Klassen zeigten zwar in manchen Punkten auch Übereinstimmungen, aber um eine gesicherte Zuordnung vornehmen zu können waren nicht genügend Daten vorhanden.

5.2 Ausblick

Eine vollständigere Analyse der Daten über einen Zeitraum von mehreren Monaten oder Jahren würde mehr Erkenntnisse darüber bringen, wie sich das Verhalten der Bots im Laufe der Zeit verändert hat.
Rückblickend war die Beschränkung auf die Untersuchung der Actions der Bots allein auch sehr einschränkend für den Erkenntnisgewinn. Bots existieren um Aufgaben für Menschen zu erledigen oder um auf die Taten von Menschen zu reagieren. Das Verhalten menschlicher Nutzer (und deren Zuhilfenahme von Wikidata Tools) und Bots gemeinsam zu wäre vielversprechender.
Die Analyse der Comments von Edits, die nicht auf dem Item Namespace ausgeführt wurden, ist mühselig, kann aber zu neuen Erkenntnissen führen. Wir wissen bereits was Bots mit Items anstellen können und umfassend und korrekt sie dabei vorgehen. Für die anderen Bereiche von Wikidata ist das noch nicht der Fall.

Literaturverzeichnis

[1] J Riedl A Halfaker. Bots and cyborgs: Wikipedia's immune system. 2012

[2] Denny Vrandecic and Markus Krötsch. Wikidata:A Free Collaborative Knowledgebase Communications of the ACM, October 2014, Vol. 57 No. 10, Pages 78-85

[3] C. Müller-Birn, B. Karran, J. Lehmann, M. Luczak-Rösch, Peer-production system or collaborative ontology engineering effort: What is Wikidata?, In OpenSym '15, August 19 - 21, 2015

[4] Thomas Steiner, Bots vs. wikipedians, anons vs. logged-ins, Proceedings of the companion publication of the 23rd international conference on World wide web companion, April 07-11, 2014

[5] Christina Sarasua, Analysing Wikidata Edits Over Time. Short Term Scientific Mission (STSM). University of Koblenz-Landau, Germany. Host: Gianluca Demartini, University of Sheffield, United Kingdom. COST STSM Reference Number: COST-STSM-IC1302-35438. Period: 2016-10-01 to 2016-10-28.

[6] L. Kaufman and P. J. Rousseeuw. Finding Groups in Data: an Introduction to Cluster Analysis. John Wiley & Sons, 1990.

[7] Anil K.Jain, Data clustering: 50 years beyond K-means, Pattern Recognition Letters Volume 31, Issue 8, 1 June 2010, Pages 651-666

6 Appendix

6.1 Parsing und Datenaggregation

```
import xml.etree.ElementTree as ET
import csv

#xmlns="http://www.mediawiki.org/xml/export-0.10/"
#xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
#xsi:schemaLocation="http://www.mediawiki.org/xml/export-0.10/
#http://www.mediawiki.org/xml/export-0.10.xsd"
#version="0.10"
#xml:lang="en">

# Methode zum suchen des n-ten Vorkommens eines Characters c in einem
# String s
def findx(string, char, n):
    # Wenn Char in String enthalten
    if string.__contains__(char):
        # Splitte String an Character
        new = string.split(char, n)
        # Finde Position
        # new[-1] gibt letzten Teilstring (Rest) zurück
        posx = len(string) - len(new[-1]) - len(char)
        # Rückgabe
        return posx
    # Wenn Char nicht in String enthalten
    else:
        return 999

def create(data):

    temp = data[findx(data,'-',1)+1:findx(data,'-',2)]
    outputstring = r"C:\Users\Ronald Scheffler\.spyder-py3\output"+temp+"
    .csv"
    outputnormstring = r"C:\Users\Ronald Scheffler\.spyder-py3\outputnorm"
    +temp+".csv"
    outputdb = r"C:\Users\Ronald Scheffler\.spyder-py3\datenbank"+temp+"
    .csv"
```

```
    # Starte CSV Writer
    # "w" - Das für csvfile übergebene Dateiobjekt muss für den
    Schreibzugriff im Textmodus geöffnet werden
    # Python 3, nutze 'w' an Stelle von 'wb'
    # encoding = "utf-8" notwendig für I/O Operationen
    writer2018 = csv.writer(open(outputstring, 'w', encoding = "utf-
8"))
    #Schreibe erste Zeile: Tabellennamen
    writer2018.writerow(['botname','AS1','AS2','AS3','AS4','AS5',
    'AS6','AS7','AS8'])

    # Counter für Gesamtzahl von Edits
    editcounter = 0
    # Counter für Anzahl von Botedits
    boteditcounter = 0
    # Zuteilung der Action für Klassifizierung
    actionnumber = 0
    # Liste aller gefunden Edits
    editarray = []
    # Liste aller gefundenen Bots
    botarray = []
    # Liste aller Edits zusammen mit dem Bot
    bot_edit_array = []
    # Anzahl der Edits für einen Bot
    bot_edit_array = []
    # Counter für Vorkommen jeder einzelnen Action
    edit_count_array = []

    # Action Sets
    actionset1 = ['wbcreateredirect',
                  'wbmergeitems-from',
                  'wbmergeitems-to',
                  'wbeditentity-create-item']
    actionset2 = ['wbcreateclaim',
                  'wbcreateclaim-create',
                  'wbremoveclaims',
                  'wbremoveclaims-remove',
                  'wbsetclaim',
                  'wbsetclaim-update',
                  'wbsetclaim-create',
                  'wbsetclaimvalue',
                  'wbremovereferences-remove',
                  'wbsetreference-add',
                  'wbremovequalifiers-remove',
```

```
                'wbsetqualifier-add',
                'wbsetqualifier-update',
                'wbeditentity-update',
                'wbeditentity-override',
                'wbeditentity-create']
actionset3 = ['wbsetlabel-set',
                'wbsetlabel-add',
                'wbsetdescription',
                'wbsetdescription-set',
                'wbsetdescription-add',
                'wbsetaliases-update',
                'wbsetaliases-add',
                'wbsetaliases-set']
actionset4 = ['wbsetsitelink-set-badges',
                'wbsetsitelink-set',
                'wbsetsitelink-add',
                'clientsitelink-remove',
                'clientsitelink-update']
actionset5 = ['wbremovequalifiers-remove',
                'wbsetqualifier-add',
                'wbsetqualifier-update',
                'wbremoveclaims',
                'wbremoveclaims-remove',
                'wbsetclaimvalue']
actionset6 = ['wbsetlabel-set120',
                'wbsetlabel-add120',
                'wbsetdescription-set120',
                'wbsetdescription-add120',
                'wbsetaliases-update120',
                'wbsetaliases-add120',
                'wbsetaliases-set120']
actionset7 = ['Wikidata list updated']
actionset8 = ['Reverted', 'undo', 'Protected']
# actionset9 = ['Protected'] Zusammengefasst mit Actionset 8

# Counter für Edits pro Actionset
as1counter = 0
as2counter = 0
as3counter = 0
as4counter = 0
as5counter = 0
as6counter = 0
as7counter = 0
as8counter = 0
```

```
as9counter = 0

# Namespace Counter
nsmediacounter = 0
nsspecialcounter = 0
ns0counter = 0
ns0unknowncounter = 0
ns1counter = 0
ns2counter = 0
ns3counter = 0
ns4counter = 0
ns5counter = 0
ns6counter = 0
ns7counter = 0
ns8counter = 0
ns9counter = 0
ns10counter = 0
ns11counter = 0
ns12counter = 0
ns13counter = 0
ns14counter = 0
ns15counter = 0
ns120counter = 0
ns120unknowncounter = 0
ns121counter = 0
ns121unknowncounter = 0
ns122counter = 0
ns123counter = 0
ns828counter = 0
ns829counter = 0
ns1198counter = 0
ns1199counter = 0
ns2300counter = 0
ns2301counter = 0
ns2302counter = 0
ns2303counter = 0
ns2600counter = 0

# Unique Edits in Namespaces
nsmediaarray = []
nsspecialarray = []
ns0array = []
ns0unknownarray = []
ns1array = []
```

```
    ns2array = []
    ns3array = []
    ns4array = []
    ns5array = []
    ns6array = []
    ns7array = []
    ns8array = []
    ns9array = []
    ns10array = []
    ns11array = []
    ns12array = []
    ns13array = []
    ns14array = []
    ns15array = []
    ns120array = []
    ns120unknownarray = []
    ns121array = []
    ns121unknownarray = []
    ns122array = []
    ns123array = []
    ns828array = []
    ns829array = []
    ns1198array = []
    ns1199array = []
    ns2300array = []
    ns2301array = []
    ns2302array = []
    ns2303array = []
    ns2600array = []

    # ElementTree verwendet ein Wörterbuch zum speichern von Attribut-
Werte, so ist es grundsätzlich  ungeordnet
    # Parse Liste mit Botnamen
    treeNameET = ET.ElementTree(file=r'C:\Users\Ronald Scheffler\.spyder-
py3\bot_list_20180716_field.xml')
    rootNameET = treeNameET.getroot()

    tree = ET.parse(data)
    root = tree.getroot()

    # Wenn Elemente leer sind, setze sie auf Siteinfo, damit sie weiter
    #verarbeitet werden können
    # Setze sie danach auf ''
    empty_revision_id = root.find('{http://www.mediawiki.org/xml/export-
```

```
0.10/}siteinfo')
    empty_revision_id.text = ''
    empty_parent_id = root.find('{http://www.mediawiki.org/xml/export-
0.10/}siteinfo')
    empty_parent_id.text = ''
    empty_timestamp = root.find('{http://www.mediawiki.org/xml/export-
0.10/}siteinfo')
    empty_timestamp.text = ''
    empty_username = root.find('{http://www.mediawiki.org/xml/export-
0.10/}siteinfo')
    empty_username.text = ''
    empty_user_id = root.find('{http://www.mediawiki.org/xml/export-
0.10/}siteinfo')
    empty_user_id.text = ''

    empty_user_redirect = root.find('{http://www.mediawiki.org/xml/export-
0.10/}siteinfo')
    empty_user_redirect.text = ''
    empty_user_restrictions = root.find('{http://www.mediawiki.org/xml/export-
0.10/}siteinfo')
    empty_user_restrictions.text = ''
    empty_user_discussion = root.find('{http://www.mediawiki.org/xml/export-
0.10/}siteinfo')
    empty_user_discussion.text = ''
    empty_user_minor = root.find('{http://www.mediawiki.org/xml/export-
0.10/}siteinfo')
    empty_user_minor.text = ''
    empty_user_model = root.find('{http://www.mediawiki.org/xml/export-
0.10/}siteinfo')
    empty_user_model.text = ''
    empty_user_wikiformat = root.find('{http://www.mediawiki.org/xml/export-
0.10/}siteinfo')
    empty_user_wikiformat.text = ''
    empty_user_sha1 = root.find('{http://www.mediawiki.org/xml/export-
0.10/}siteinfo')
    empty_user_sha1.text = ''

    # Sammle Informationen dieser Sonderfälle
    emptyarray = []

    # Durchsuche alle Pages
    for page in root.getiterator('{http://www.mediawiki.org/xml/export-
0.10/}page'):
        # Title - was wurde geändert?
```

```
        title = page.find('{http://www.mediawiki.org/xml/export-
        0.10/}title').text
        # Namespace - Wo wurde etwas geändert?
        ns = int(page.find('{http://www.mediawiki.org/xml/export-

        0.10/}ns').text
        # Page ID
        page_id = page.find('{http://www.mediawiki.org/xml/export-

        0.10/}id').text

        if page.find('{http://www.mediawiki.org/xml/export-0.10/}redirect')
        != None:
            redirect = page.find('{http://www.mediawiki.org/xml/export-
0.10/}redirect')
        else:
            redirect = empty_user_redirect

        if page.find('{http://www.mediawiki.org/xml/export-0.10/}
        restrictions') != None:
            restrictions = page.find('{http://www.mediawiki.org/xml/export-
0.10/}restrictions')
        else:
            restrictions = empty_user_restrictions

        if page.find('{http://www.mediawiki.org/xml/export-0.10/}
        discussionthreadinginfo') != None:
            discussion = page.find('{http://www.mediawiki.org/xml/export-
0.10/} discussionthreadinginfo')
        else:
            discussion = empty_user_discussion

        #Durchsuche alle revisions
        #Halte Revision fest
        for revision in page.getiterator('{http://www.mediawiki.org/xml/export
        -0.10/}revision'):
            #print(revision)
            # Erhöhe Gesamtcounter
            editcounter += 1
            # Suche nach Contributor {http://www.mediawiki.org/xml/export-
0.10/}
            contributor = revision.find('{http://www.mediawiki.org/xml/export
            -0.10/}contributor')
            # Untersuche Namen {http://www.mediawiki.org/xml/export-
```

```
0.10/}
            user_name = contributor.find('{http://www.mediawiki.org/xml/export-

            0.10/}username')

            # Name darf nicht None sein
            # Bei unregistrierten, menschlichen Nutzen steht nur eine IP
            if (user_name != None):
                #print(username.text)
                #Vergleiche mit Botnamenliste
                for botRoot in rootNameET:
                    # Für alle Botnamen in der Botliste
                    for botname in botRoot:
                        # Wenn User = Bot
                        if user_name.text == botname.text or
                        user_name.text == botname.text.strip():

                            # Liste alle gefundenen Bots
                            if user_name.text not in botarray:
                                botarray.append(user_name.text)
                            # Erhöhe Botedit Counter
                            boteditcounter += 1
                            # Weise Elemente zu
                            if revision.find('{http://www.mediawiki.org/xml/
                            export-0.10/}id') != None:
                                revision_id = revision.find('{http://www.
                                mediawiki.org/xml/export-0.10/}id')
                            else:
                                revision_id = empty_revision_id
                            if revision.find('{http://www.mediawiki.org/xml/
                            export-0.10/}parentid') != None:
                                parent_id  = revision.find('{http://www.
                                mediawiki.org/xml/export-0.10/}parentid')
                            else:
                                parent_id = empty_parent_id
                            if revision.find('{http://www.mediawiki.org/xml/
                            export-0.10/}timestamp') != None:
                                timestamp = revision.find('{http://www.
                                mediawiki.org/xml/export-0.10/}'timestamp')
                            else:
                                timestamp = empty_timestamp
                            if contributor.find('{http://www.mediawiki.
                            org/xml/export-0.10/}username') != None:
                                username = contributor.find('{http://www.
```

```
                    mediawiki.org/xml/export-0.10/}username')
                else:
                    username = empty_username
                if contributor.find('{http://www.mediawiki.org/
                xml/export-0.10/}id') != None:
                    user_id = contributor.find('{http://www.
                    mediawiki.org/xml/export-0.10/}id')
                else:
                    user_id = empty_user_id
                #Kompletten Comment zwischenspeichern
                comment = revision.find('{http://www.mediawiki
                .org/xml/export-0.10/}comment')
                if comment != None:
                    # Wandle Element in String um
                    # Schließt b'<ns0:comment xmlns:ns0="
                    http://www.mediawiki.org/xml/export-
0.10/">
                    in Comment mit ein
                    # Länge: Anfang: 67 Ende: 21
                    comment = ET.tostring(comment)
                    # Entferne <comment> </comment> aus comment
                    comment = comment[67:len(comment)-
21]
                    #print('Comment: ' + comment.decode())
                    # decode() weil Byte Object erwartet wird
                    comment = comment.decode()
                    #print(comment)
                else:
                    comment = ''

                # minor
                if revision.find('{http://www.mediawiki.org/xml
                /export-0.10/}minor') != None:
                    minor = revision.find('{http://www.mediawiki
                    .org/xml/export-0.10/}minor')
                else:
                    minor = empty_user_minor
                # Model
                if revision.find('{http://www.mediawiki.org/xml/
                export-0.10/}model') != None:
                    model = revision.find('{http://www.mediawiki
                    .org/xml/export-0.10/}model')
                else:
                    model = empty_user_model
```

```
# Format
if revision.find('{http://www.mediawiki.org/xml/
export-0.10/}format') != None:
    wikiformat = revision.find('{http://www.
    mediawiki.org/xml/export-0.10/}format')
else:
    wikiformat = empty_user_wikiformat
# sha1
if revision.find('{http://www.mediawiki.org/xml/
export-0.10/}sha1') != None:
    sha1 = revision.find('{http://www.mediawiki
    .org/xml/export-0.10/}sha1')
else:
    sha1 = empty_user_sha1

# Reorganisation:
# Gruppiere nach dem was verändert wurde
# Achte auf Namespaces

fullcomment = comment

# Media
if ns == -2:
    nsmediacounter += 1
    if comment not in nsmediaarray:
        nsmediaarray.append(comment)

# Special
if ns == -1:
    nsspecialcounter += 1
    if comment not in nsspecialarray:
        nsspecialarray.append(comment)

# Wikidata Articles / Items
if ns == 0:
    ns0counter += 1
    # createredirect
    # mergeitems
    # betrifft Q
    if findx(comment, 'wbcreateredirect', 1)
     == 3:
        as1counter += 1
        comment = comment[findx(comment,
        ' ' , 1)+1:findx(comment, ':',1)]
```

```
        if comment not in ns0array:
            ns0array.append(comment)
        if comment not in editarray:
            editarray.append(comment)
        bot_edit_array.append([username
        .text,comment])
        break

    if findx(comment, 'wbmergeitems', 1)
    == 3:
        as1counter += 1
        comment = comment[findx(comment,
        ' ' , 1)+1:findx(comment, ':',  1)]
        if comment not in ns0array:
            ns0array.append(comment)
        if comment not in editarray:
            editarray.append(comment)
        bot_edit_array.append([username
        .text,comment])
        break

    if findx(comment, 'wbeditentity-create
    -item', 1) == 3:
        as1counter += 1
        comment = comment[findx(comment,
        ' ' , 1)+1:findx(comment, ':',  1)]
        if comment not in ns0array:
            ns0array.append(comment)
        if comment not in editarray:
            editarray.append(comment)
        bot_edit_array.append([username
        .text,comment])
        break

    # editentity
    # createclaim
    # getclaim
    # removeclaim
    # setclaim
    # setclaimvalue
    # removereferences
    # setreference
    # removequalifiers
    # setqualifier
```

```
if findx(comment, 'wbeditentity', 1)
== 3:
    as1counter += 1
    comment = comment[findx(comment,
    ' ' , 1)+1:findx(comment, ':',  1)]
    if comment not in ns0array:
        ns0array.append(comment)
    if comment not in editarray:
        editarray.append(comment)
    bot_edit_array.append([username
    .text,comment])
    break

if findx(comment, 'wbcreateclaim', 1)
== 3:
    as2counter += 1
    comment = comment[findx(comment,
    ' ' , 1)+1:findx(comment, ':',  1)]
    if comment not in ns0array:
        ns0array.append(comment)
    if comment not in editarray:
        editarray.append(comment)
    bot_edit_array.append([username
    .text,comment])
    break

if findx(comment, 'wbgetclaim', 1)
== 3:
    as2counter += 1
    comment = comment[findx(comment,
    ' ' , 1)+1:findx(comment, ':',  1)]
    if comment not in ns0array:
        ns0array.append(comment)
    if comment not in editarray:
        editarray.append(comment)
    bot_edit_array.append([username
    .text,comment])
    break

if findx(comment, 'wbremoveclaims', 1)
 == 3:
    as2counter += 1
    comment = comment[findx(comment,
```

```
    ' ' , 1)+1:findx(comment, ':',  1)]
    if comment not in ns0array:
        ns0array.append(comment)
    if comment not in editarray:
        editarray.append(comment)
    bot_edit_array.append([username
    .text,comment])
    break

if findx(comment, 'wbsetclaim', 1)
== 3:
    as2counter += 1
    comment = comment[findx(comment,
    ' ' , 1)+1:findx(comment, ':',  1)]
    if comment not in ns0array:
        ns0array.append(comment)
    if comment not in editarray:
        editarray.append(comment)
    bot_edit_array.append([username
    .text,comment])
    break

if findx(comment, 'wbsetclaimvalue', 1)
== 3:
    as2counter += 1
    comment = comment[findx(comment,
    ' ' , 1)+1:findx(comment, ':',  1)]
    if comment not in ns0array:
        ns0array.append(comment)
    if comment not in editarray:
        editarray.append(comment)
    bot_edit_array.append([username
    .text,comment])
    break

if findx(comment, 'wbremovereferences', 1)
== 3:
    as2counter += 1
    comment = comment[findx(comment,
    ' ' , 1)+1:findx(comment, ':',  1)]
    if comment not in ns0array:
        ns0array.append(comment)
    if comment not in editarray:
        editarray.append(comment)
```

```
                    bot_edit_array.append([username
                    .text,comment])
                    break

                if findx(comment, 'wbsetreference', 1)
                == 3:
                    as2counter += 1
                    comment = comment[findx(comment,
                    ' ' , 1)+1:findx(comment, ':',  1)]
                    if comment not in ns0array:
                        ns0array.append(comment)
                    if comment not in editarray:
                        editarray.append(comment)
                    bot_edit_array.append([username
                    .text,comment])
                    break

                if findx(comment, 'wbremovequalifiers', 1)
                == 3:
                    as2counter += 1
                    comment = comment[findx(comment,
                    ' ' , 1)+1:findx(comment, ':',  1)]
                    if comment not in ns0array:
                        ns0array.append(comment)
                    if comment not in editarray:
                        editarray.append(comment)
                    bot_edit_array.append([username
                    .text,comment])
                    break

                if findx(comment, 'wbsetqualifier', 1)
                == 3:
                    as2counter += 1
                    comment = comment[findx(comment,
                    ' ' , 1)+1:findx(comment, ':',  1)]
                    if comment not in ns0array:
                        ns0array.append(comment)
                    if comment not in editarray:
                        editarray.append(comment)
                    bot_edit_array.append([username
                    .text,comment])
                    break

                # setlabel
```

```
if findx(comment, 'wbsetlabel', 1)
== 3:
    as3counter += 1
    comment = comment[findx(comment,
    ' ' , 1)+1:findx(comment, ':',  1)]
    if comment not in ns0array:
        ns0array.append(comment),
    if comment not in editarray:
        editarray.append(comment)
    bot_edit_array.append([username
    .text,comment])
    #break

if findx(comment, 'wbsetdescription', 1)
 == 3:
    as3counter += 1
    comment = comment[findx(comment,
    ' ' , 1)+1:findx(comment, ':',  1)]
    if comment not in ns0array:
        ns0array.append(comment)
    if comment not in editarray:
        editarray.append(comment)
    bot_edit_array.append([username
    .text,comment])
    #break

if findx(comment, 'wbsetaliases', 1)
== 3:
    as3counter += 1
    comment = comment[findx(comment,
    ' ' , 1)+1:findx(comment, ':',  1)]
    if comment not in ns0array:
        ns0array.append(comment)
    if comment not in editarray:
        editarray.append(comment)
    bot_edit_array.append([username
    .text,comment])
    #break

# setsitelink

if findx(comment, 'wbsetsitelink', 1)
== 3:
```

```
                    as4counter += 1
                    comment = comment[findx(comment,
                    ' ' , 1)+1:findx(comment, ':',  1)]
                    if comment not in ns0array:
                        ns0array.append(comment)
                    if comment not in editarray:
                        editarray.append(comment)
                    bot_edit_array.append([username
                    .text,comment])
                    break

                if findx(comment, 'clientsitelink', 1)
                == 3:
                    as4counter += 1
                    comment = comment[findx(comment,
                    ' ' , 1)+1:findx(comment, ':',  1)]
                    #print(username.text)
                    if comment not in ns0array:
                        ns0array.append(comment)
                    if comment not in editarray:
                        editarray.append(comment)
                    bot_edit_array.append([username
                    .text,comment])
                    break

            # revert (item or property):

                if findx(comment, 'Reverted edits', 1)
                 == 0:
                    as8counter += 1
                    comment = comment[:findx(comment,
                     ' ', 2)]
                    if comment not in ns0array:
                        ns0array.append(comment)
                    if comment not in editarray:
                        editarray.append(comment)
                    bot_edit_array.append([username
                    .text,comment])
                    break

                if findx(comment, 'undo', 1) == 3:
                    as8counter += 1
                    comment = comment[findx(comment,
                    ' ' , 1)+1:findx(comment, ':',  1)]
```

```
            if comment not in ns0array:
                ns0array.append(comment)
            if comment not in editarray:
                editarray.append(comment)
            bot_edit_array.append([username
            .text,comment])
            break

    # protect (item or property):

        if findx(comment, 'Protected', 1) == 0:
            as9counter += 1
            comment = comment[:findx(comment,
            ' ', 1)]
            if comment not in ns0array:
                ns0array.append(comment)
            if comment not in editarray:
                editarray.append(comment)
            bot_edit_array.append([username
            .text,comment])
            break

        # Fange unbekannte Fälle ab,
        ns0unknowncounter += 1
        actionnumber = 9999
        if comment not in ns0unknownarray:
            ns0unknownarray.append(comment)

    # Talk
    if ns == 1:
        ns1counter += 1
        if comment not in ns1array:
            ns1array.append(comment)

    # User
    if ns == 2:
        ns2counter += 1
        # Sonderfall
        # Wenn Comment Leer ist
        # Comment wurde initial auf "Leer"
        # gesetzt weil comment = None
        # Comment existiert nicht?
        if comment == '':
            # Annahme: Nur Comment ist leer
```

```
                emptyarray.append([revision_id.text,
                                   parent_id.text,
                                   timestamp.text,
                                   username.text,
                                   user_id.text,
                                   comment])
            else:
                if comment not in ns2array:
                    ns2array.append(comment)

        # User Talk
        if ns == 3:
            ns3counter += 1
            if comment not in ns3array:
                ns3array.append(comment)

        # Wikidata
        if ns == 4:
            ns4counter += 1
            if comment not in ns4array:
                ns4array.append(comment)

        # Wikidata Talk
        if ns == 5:
            ns5counter += 1
            if comment not in ns5array:
                ns5array.append(comment)

        # File
        if ns == 6:
            ns6counter += 1
            if comment not in ns6array:
                ns6array.append(comment)

        # File Talk
        if ns == 7:
            ns7counter += 1
            if comment not in ns7array:
                ns7array.append(comment)

        # Mediawiki
        if ns == 8:
            ns8counter += 1
            if comment not in ns8array:
```

```
            ns8array.append(comment)

    # Mediawiki Talk
    if ns == 9:
        ns9counter += 1
        if comment not in ns9array:
            ns9array.append(comment)

    # Template
    if ns == 10:
        ns10counter += 1
        if comment not in ns10array:
            ns10array.append(comment)

    # Template Talk
    if ns == 11:
        ns11counter += 1
        if comment not in ns11array:
            ns11array.append(comment)

    # Help
    if ns == 12:
        ns12counter += 1
        if comment not in ns12array:
            ns12array.append(comment)

    # Help Talk
    if ns == 13:
        ns13counter += 1
        if comment not in ns13array:
            ns13array.append(comment)

    # Category
    if ns == 14:
        ns14counter += 1
        if comment not in ns14array:
            ns14array.append(comment)

    # Category Talk
    if ns == 15:
        ns15counter += 1
        if comment not in ns15array:
            ns15array.append(comment)
```

```
                            # Property
                            if ns == 120:
                                ns120counter += 1
                                print('120 ', ns120counter)
                                print('120 ', revision_id.text)
                                print('120 ', comment)
                                 # betrifft P
                                if findx(comment, 'wbremovequalifiers', 1)
                                == 3:
                                    as5counter += 1
                                    comment = comment[findx(comment,
                                    ' ' , 1)+1:findx(comment, ':',  1)]
+'120'
                                    if comment not in ns120array:
                                        ns120array.append(comment)
                                    if comment not in editarray:
                                        editarray.append(comment)
                                    bot_edit_array.append([username
                                    .text,comment])
                                    break
                                # betrifft P
                                if findx(comment, 'wbsetqualifier', 1)
                                 == 3:
                                    as5counter += 1
                                    comment = comment[findx(comment,
                                    ' ' , 1)+1:findx(comment, ':',  1)]
+'120'
                                    if comment not in ns120array:
                                        ns120array.append(comment)
                                    if comment not in editarray:
                                        editarray.append(comment)
                                    bot_edit_array.append([username
                                    .text,comment])
                                    break
                                 # betrifft P
                                if findx(comment, 'wbremoveclaims', 1)
                                == 3:
                                    as5counter += 1
                                    comment = comment[findx(comment,
                                    ' ' , 1)+1:findx(comment, ':',  1)]
+'120'
                                    if comment not in ns120array:
                                        ns120array.append(comment)
                                    if comment not in editarray:
```

```
                editarray.append(comment)
            bot_edit_array.append([username
            .text,comment])
            break
         # betrifft P
        if findx(comment, 'wbsetclaimvalue', 1)
        == 3:
            as5counter += 1
            comment = comment[findx(comment,
            ' ' , 1)+1:findx(comment, ':',  1)]
+'120'
            if comment not in ns120array:
                ns120array.append(comment)
            if comment not in editarray:
                editarray.append(comment)
            bot_edit_array.append([username
            .text,comment])
            break

        if findx(comment, 'wbsetlabel', 1)
        == 3:
            as6counter += 1
            comment = comment[findx(comment,
            ' ' , 1)+1:findx(comment, ':',  1)]
+'120'
            print(revision_id.text)
            print(comment)
            print(username.text)
            if comment not in ns120array:
                ns0array.append(comment)
            if comment not in editarray:
                editarray.append(comment)
            bot_edit_array.append([username
            .text,comment])
            break

        if findx(comment, 'wbsetdescription',
        1) == 3:
            as6counter += 1
            comment = comment[findx(comment,
            ' ' , 1)+1:findx(comment, ':',  1)]
+'120'
            print(revision_id.text)
            print(comment)
```

```
                                    print(username.text)
                                    if comment not in ns120array:
                                        ns0array.append(comment)
                                    if comment not in editarray:
                                        editarray.append(comment)
                                    bot_edit_array.append([username
                                    .text,comment])
                                    break

                                if findx(comment, 'wbsetaliases', 1)
                                == 3:
                                    as6counter += 1
                                    comment = comment[findx(comment,
                                    ' ' , 1)+1:findx(comment, ':',  1)]
+'120'
                                    print(revision_id.text)
                                    print(comment)
                                    print(username.text)
                                    if comment not in ns120array:
                                        ns0array.append(comment)
                                    if comment not in editarray:
                                        editarray.append(comment)
                                    bot_edit_array.append([username
                                    .text,comment])
                                    break

                                # Fange unbekannte Fälle ab,
                                ns120unknowncounter += 1
                                actionnumber = 9999
                                if comment not in ns120unknownarray:
                                    ns120unknownarray.append(comment)
                            # Property Talk
                            if ns == 121:
                                as7counter += 1
                                ns121counter += 1
                                # betrifft P
                                if findx(comment, 'Wikidata list
                                updated', 1) == 0:
                                    if comment not in ns121array:
                                        # Übernimm kompletten Comment
                                        ns121array.append(comment)
                                    if comment not in editarray:
                                        editarray.append(comment)
                                    bot_edit_array.append([username
```

```
            .text,comment])
            break
        # Fange unbekannte Fälle ab,
        ns121unknowncounter += 1
        if comment not in ns121unknownarray:
            ns121unknownarray.append(comment)
    # Query
    if ns == 122:
        ns122counter += 1
        if comment not in ns122array:
            ns122array.append(comment)
    # Query Talk
    if ns == 123:
        ns123counter += 1
        ns123array.append(comment)
    # Module
    if ns == 828:
        ns828counter += 1
        if comment not in ns828array:
            ns828array.append(comment)
    # Module Talk
    if ns == 829:
        ns829counter += 1
        if comment not in ns829array:
            ns829array.append(comment)
    # Translation
    if ns == 1198:
        ns1198counter += 1
        if comment not in ns1198array:
            ns1198array.append(comment)
    # Translation Talk
    if ns == 1199:
        ns1199counter += 1
        if comment not in ns1199array:
            ns1199array.append(comment)
    # Gadget
    if ns == 2300:
        ns2300counter += 1
        if comment not in ns2300array:
            ns2300array.append(comment)
    # Gadget talk
    if ns == 2301:
        ns2301counter += 1
        if comment not in ns2301array:
```

```
                            ns2301array.append(comment)
                    # Gadget definition
                    if ns == 2302:
                        ns2302counter += 1
                        if comment not in ns2302array:
                            ns2302array.append(comment)
                    # Gadget definition talk
                    if ns == 2303:
                        ns2303counter += 1
                        if comment not in ns2303array:
                            ns2303array.append(comment)
                    # Topic
                    if ns == 2600:
                        ns2600counter += 1
                        if comment not in ns2600array:
                            ns2600array.append(comment)

                    revisiondaten = ([title.rstrip(),
                                      ns,
                                      page_id.rstrip(),
                                      redirect.text,
                                      restrictions.text,
                                      discussion.text,
                                      revision_id.text.rstrip(),
                                      parent_id.text.rstrip(),
                                      timestamp.text.rstrip(),
                                      username.text.rstrip(),
                                      user_id.text.rstrip(),
                                      minor.text,
                                      fullcomment,  # Comment
                                      comment,      # Action
                                      model.text,
                                      wikiformat.text,
                                      sha1.text])
                    # Schreibe DB Daten für jede Revision
                    writerdb.writerow(revisiondaten)

    # Wieviele Bots haben einen bestimmten Edit durchgeführt
    for edit in editarray:
        counter = 0
        for bot in bot_edit_array:
            if edit == bot[1]:
                counter += 1
        edit_count_array.append([edit,counter])
```

```
for bot in botarray:
    # Zähle Edits in allen Actionsets für diesen einen Bot
    as1 = 0
    as2 = 0
    as3 = 0
    as4 = 0
    as5 = 0
    as6 = 0
    as7 = 0
    as8 = 0
    # Welche Edits hat er durchgeführt?
    for edit in bot_edit_array:
        # Wenn Bot gefunden
        if bot == edit[0]:
            # Prüfe alle Actionsets, setze Counter
            if edit[1] in actionset1:
                as1 += 1
            if edit[1] in actionset2:
                as2 += 1
            if edit[1] in actionset3:
                as3 += 1
            if edit[1] in actionset4:
                as4 += 1
            if edit[1] in actionset5:
                as5 += 1
            if edit[1] in actionset6:
                as6 += 1
            if edit[1] in actionset7:
                as7 += 1
            if edit[1] in actionset8:
                as8 += 1

    #Create CSV here!

    # Nachdem Daten für 1 Bot berechnet wurden:
    # Fülle CSV Daten für den aktuellen Edit
    daten = ([bot,
              as1,
              as2,
              as3,
              as4,
              as5,
              as6,
```

```
                as7,
                as8])

        # Schreibe Daten
        # Wenn Summe aller Actionsets = 0:
        # Action war in einem anderen Namespace
        # Wenn Summe != 0: Schreibe Daten
        if as1+as2+as3+as4+as5+as6+as7+as7 != 0:
            writer2018.writerow(daten)

    print('Edits: ', editcounter)
    print('Botedits: ', boteditcounter)
    print('Distinct Edits: ', len(editarray))
    percent = round(((boteditcounter/editcounter)*100),2)
    print(percent, '%')


create('F:\Wikidata\Stubs\wikidatawiki-20180906-stubs-meta-hist-incr.xml')
```

6.2 Klassifizierung

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import scale
import sklearn.metrics as sm
from sklearn.metrics.cluster import silhouette_score

def cluster(csv, k):

    data = pd.read_csv(csv)
    # X Features
    X = np.array(data.drop(['botname'], 1))
    X = scale(X.data)
    \# Wähle Anzahl der Cluster, Startpunk der Centroids, Iterationen
    \# Random State seed für Reproduktion der Ergebnisse
    clustering = KMeans(n_clusters = k,init ='k-means++',n_init= 10,
     random_state = 6)

    clustering.fit(X)

    X_scaled = X
```

```
    result = clustering.fit_predict(X)

    data['Cluster'] = result
    data = data.sort_values(['Cluster'])

    data.to_csv(r"C:\Users\Ronald Scheffler\.spyder-py3\clusterresult"
    +str(k)+".csv")

    print(silhouette_score(X_scaled, result))

cluster(r"C:\Users\Ronald Scheffler\.spyder-py3\clustermergemin5.csv", 5)
```