Projektarbeit bei Prof. Dr. habil. Ruedi Stoop im Rahmen der Veranstaltung “Theory, programming and simulation of neural networks” ; FS2012, Institut für Neuroinformatik, Universität Zürich / ETH Zürich.

Einsatz einer Hauptkomponentenanalyse mit hebbscher Lernregel in einem Programmierprojekt zum E-Mail-Clustering

Hernani Marques <h2m@access.uzh.ch>; Universität Zürich, August 2012
Hauptfach: Computerlinguistik und Sprachtechnologie; Nebenfächer: Soziologie, Neuroinformatik

1 Einleitung

Im Rahmen eines (eigenen) laufenden Programmierprojekts unter dem Namen **wahatttt** [1], das im Quellcode frei verfügbar ist, sollen E-Mails (im Sinne eines *Clusterings*) gruppiert werden. Mangels näherer Informationen zur Textkollektion ist nicht bekannt wieviele Textkategorien (oder *Cluster*) *real* existieren. Es ist das Ziel des Experiments demnach herauszuarbeiten, ob unter Einsatz eines selbstlernenden *Künstlichen Neuronalen Netzwerks* (*KNN*) die verfügbaren E-Mails *sinnvoll* klassifiziert werden können und unter welchen Bedingungen das (möglicherweise) gelingt.

Einer ähnlichen Aufgabe haben sich Niederberger et al. [1] bereits angenommen, deren Methode einer Einbettung der *hebbschen Lernregel* im Rahmen einer *Hauptkomponentenanalyse* [2] gleichkommt. Nach einer Präzisierung der eigenen Aufgabenstellung in Abschnitt 2 wird auf diese Methode (in Abgrenzung anderer denkbarer Ansätze) näher eingegangen (3) und deren Wahl begründet.

Danach wird auf eine verfügbare Programmierung des o. g. *KNN* der Wahl eingegangen und die prinzipielle Funktionsweise demonstriert (Abschnitte 4 und 5). Da die Ergebnisse *fundamental* von der Trennbarkeit der Input-Daten abhängen, ist deren Operationalisierung ein eigener Abschnitt (6) gewidmet.

Schliesslich werden die gemachten Erfahrungen präsentiert (7), auf ungelöste Probleme und damit verbundenen möglichen Verbesserungschancen eingegangen (8).

2 Aufgabenstellung / Datenmaterial

Das **wahatttt**-Projekt entspringt einem Wunsch der **WHS** [3], welches den Nachlass des 2001 verstorbenen Gründungsvaters des **CCC** [4] “Wau Holland” oder “Wau” [5] (geb. 1951) verwaltet, nach der Verfügbarmachung eines elektronischen (der Öffentlichkeit zugänglichen) Archivs. Dieses soll dem Ideal nach die Themenfelder leicht zugänglich machen, mit denen sich Wau beschäftigt hat und es erlauben die entsprechenden Dokumente zu visieren. Mehr Details dazu finden sich im Projekt-`README`. [6]

Als Datengrundlage dienen (zunächst) freizugängliche E-Mails einer Mailingliste [7], worauf bis vor wenigen Jahren aktiv technische und gesellschaftliche Aspekte von Informationstechnologien debattiert wurden. Das entspricht dem Themenspektrum mit dem sich Wau hauptsächlich auseinandergesetzt hat. Wau selber hat zwischen den Jahren 1996 und 2000 E-Mails an diese Liste abgesetzt.

[1] URL: `https://github.com/2mh/wahatttt/` (13.08.2012)
[2] Engl. *PCA* or *Principal Component Analysis*
[3] **W**au-**H**olland-**S**tiftung. URL: `https://www.wauland.de/` (13.08.2012)
[4] **C**haos **C**omputer **C**lub. URL: `https://www.ccc.de/` (13.08.2012)
[5] Bürg. Name: Herwart Holland-Moritz
[6] URL: `https://github.com/2mh/wahatttt/blob/master/README` (13.08.2012)
[7] URL: `http://www.fitug.de/debate/index.html` (13.08.2012)

3 Das KNN der Wahl / Theorie

Beim *Künstlichen Neuronalen Netzwerk* der Wahl handelt es sich zunächst im Grundsatz um eine *Hauptkomponentenanalyse*, die - vereinfacht gesagt - darauf abzielt Zusammenhänge in Input-Matrizen zu finden, anhand darin enthaltener Werte, die bei ähnlichen Dokumenten (oder Zeilen) korrelieren sollten. Damit wären Hauptkomponenten ausgemacht, die dazu genutzt werden können, um zwei oder mehr Dokumente zueinander zugehörig zu erklären; oder, sehr einfach ausgedrückt: Das Ziel müsste sein, dass wenn zwei Zeilen in einer Input-Matrize (sehr) ähnlich aussehen, diese als zusammenhängend betrachtet werden.

Die vorgeschlagene Input-Matrize etwa stellt eine *TF*IDF*-Matrize dar. Dabei handelt es sich beim *TF*-Wert - in trivialer Ausführung - um die Anzahl Vorkommnisse eines Wortes in einem Text, gegebenenfalls geteilt durch die Anzahl verfügbarer Wörter. Es sind hiervon Abwandlungen denkbar, die z. B. dann wichtig sind, wenn Dokumente (stark) unterschiedlicher Wortzahl sind. Eine solche Variante wird in Abschnitt 6 diskutiert.

Beim IDF-Wert handelt es sich um die *Inverse Document Frequency*, die im Allgemeinen so definiert wird:

$$idf(i) = log(N/n_i)$$

Dabei ist N die Anzahl aller Dokumente der Kollektion und n_i stellt die Anzahl der Dokumente dar, die den Term i enthalten. Der *IDF*-Wert fällt für einen Term gering aus, welcher sehr häufig auftritt - wie etwa Stoppwörter einer Sprache. Üblicherweise handelt es sich hierbei um Artikel, Präpositionen, Konjunktionen oder andere Partikel. [8] Im Zusammenhang betrachtet stellt nun der *TF*IDF*-Wert (eines Terms in einem Dokument) sicher, dass insgesamt (in der gesamten Kollektion) selten vorkommenden Wörter ein höherer *Score* zukommt, hingegen sehr häufig vorkommende Wörter mit einem geringeren *Score* "bestraft" werden.

Die erwähnte *TF*IDF*-Matrize wäre bei einem einem beliebigen Korpus $N \times l$ gross, wobei N die Anzahl der zu trennenden Dokumente darstellt und l die Anzahl der *Features* (oder Terme / Wörter) repräsentiert, anhand welcher Zusammenhänge gefunden werden sollen.

Das von Niederberger et al. [1] präsentierte System geht nun aber einen Schritt weiter. Das System soll nicht mit einer *feststehenden* Zahl von Dokumenten und *Feature-Termen* alleine arbeiten können, sondern es soll möglich sein dem System zusätzliche Dokumentvektoren (Zeilen) hinzuzufügen, welche gegebenenfalls auch neue *Feature-Terme* (Spalten) einführen. Das System soll sich daraufhin adaptieren.

Das bedeutet: Anstatt bei jedem zusätzlichen Dokument aufs Neue die *Hauptkomponentenanalyse* komplett durchlaufen zu lassen, wird das System lernfähig, das Dokument einem (bestehenden) Cluster hinzuzufügen.

Die gewünschte Anzahl zu bildender Cluster wird anhand des k-Werts definiert - ein Systemparameter, der beliebig gesetzt werden kann. Wie dieser Wert optimal bestimmt wird, ist nicht klar, allerdings scheint es - wie im Abschnitt 5 zur Simulation des neuronalen Netzes illustriert wird - durch häufiges Testen möglich zu sein einen *Schwellenwert* (abhängig der Zahl N) zu finden, um einen adäquaten Mindestwert k zu setzen, was zumindest bei wenig komplexen (gut trennbaren) Datensätzen funktioniert.

[8] Im Deutschen z. B. "der", "die", "das", "und", "oder" usw.

Bzgl. der *hebbschen Lernregel* und wie diese genau definiert ist, gibt die Arbeit von Niederberger et al. [1] auf Seite 3 genau Aufschluss, so dass darauf verzichtet wird die Formeln hier 1:1 wieder zu geben.

Zur Vereinfachung: Im Zusammenhang mit *KNN* ausgedrückt, handelt es sich bei k um die Anzahl Output-Neuronen des Netzwerks und bei der Zahl l um die Anzahl Input-Neuronen. Anders ausgedrückt: Es wird versucht eine funktionale Abbildung von einem Vektor x (oder einer Zeile der *TF*IDF*-Matrize) mit l Komponenten zu einem der k verfügbaren Cluster zu finden.

Ein hebbsches neuronales Netzwerk kennt keinen *Hidden Layer*, womit damit nur (klar) linear-trennbare Probleme lösbar sind, was heisst, dass die Input-Neuronen unmittelbar (ohne undurchsichtige Zwischenschicht) direkt zu den Output-Neuronen durchgeschalten werden.

Wie dies genau passiert, bestimmt ein Gewicht w, das zwischen jeder Input- und Output-Neuronkombination besteht. Diese werden anfangs (pseudo-)zufällig [9] gesetzt und in der Folge entsprechend den hinzukommenden Dokumenten so angepasst, dass die Durchschaltung (zuletzt im Ergebnis) zu einer *Hauptkomponentenanalyse* konvergiert. Ist diese abgeschlossen, können weitere Dokumente hinzugefügt werden, die weitere *Feature-Terme* einführen - womit das Netzwerk sich der neuen Situation *dynamisch* anpasst. Als Nachteil dieses Ansatzes erscheint, dass das (fortlaufende) *Clustering* verschieden ausfallen kann, je nach Reihenfolge der (neu) hinzugefügten Dokumente.

Theoretisch wäre auch der Einsatz anderer *KNN*-Typen denkbar gewesen - z. B. solche, die *supervised* (bzw. *überwacht*) funktionieren und *Hidden Layer-Neuronen* einsetzen, um auch (komplexere) nicht klar trennbare Datensätze zu bewältigen. Dafür allerdings wären (von Hand) vorgeclusterte Dokumente zum Training notwendig, die im Rahmen des **wahatttt**-Projekts aber (noch) nicht vorliegen.

"Altbewährte" *Clustering*-Verfahren wie *Wards-* oder *K-means*-Clustering sind nicht weiter verfolgt worden, da diese in der Arbeit von Niederberger et al. [1] nur fähig waren einzelne Cluster korrekt zu erkennen - im Gegensatz zum vorliegenden Ansatz mit *Hauptkomponentenanalyse* und *hebbscher Lernregel*, welcher gemäss der in der Arbeit präsentierten *Konfusionsmatrix* [10] im Grundsatz fähig war die (bekannten) Evaluationscluster zu rekonstruieren. Das war Motivation diesen Ansatz für ein *real world*-Problem zu testen.

4 Programmierung des KNN

Die Arbeit von Niederberger et al. [1] liegt nicht nur theoretisch vor, sondern auch praktisch. Alle Vorgänge wurden in der Programmiersprache **Python** [11] implementiert und sind öffentlich unter einer BSD-Lizenz [12] in einem **Git**-Repository [13] verfügbar, so dass eine Einbettung in das bestehende **wahatttt**-Projekt besonders leicht möglich war, da es selber in **Python** angesetzt ist.

5 Simulation des KNN

Um die Funktionsfähigkeit des neuronalen Klassifizierers grundsätzlich zu testen, kann eine einfache Input-Matrize erstellt werden, die wie folgt ausschaut:

[9] Computer nach der prävalenten Architektur können keinen "echten" Zufall produzieren.
[10] Eine Matrize zur Bewertung eines Klassifikators anhand von Relevanzkriterien und Trefferquoten
[11] URL: `https://www.python.org/` (13.08.2012)
[12] URL: `http://opensource.org/licenses/alphabetical` (13.08.2012)
[13] URL: `https://github.com/IAS-ZHAW/machine_learning_scripts` (13.08.2012)

```
0.0 0.0 0.0 0.0
0.1 0.0 0.0 0.0
0.0 0.1 0.0 0.0
0.0 0.0 0.1 0.0
0.0 0.0 0.0 0.1
0.0 0.0 0.1 0.0
0.0 0.1 0.0 0.0
0.1 0.0 0.0 0.0
```

Eine solche Matrize wäre bei acht Dokumenten [14] erfüllt, die anhand von vier *Feature-Termen* [15] zu clustern wären und die über (zeilenweise sichtbar) eindeutige Zusammengehörigkeiten aufweisen.

Das zu erwartende Ergebnis ist offensichtlich: Es müssen fünf Cluster entstehen, wobei das erste Dokument und das Dokument Nr. 5 [16] jeweils ein Cluster für sich zu bilden haben. Bei den anderen sechs Dokumenten finden sich jeweils paarweise Dokumente, die zusammen ein Cluster bilden [17]

Testet man diese Vermutung mit einem Python-Programm von Niederberger [18] bestätigt sie sich praktisch ausnahmslos, in einer Art, wie sie in Abbildung 1 sichtbar ist.

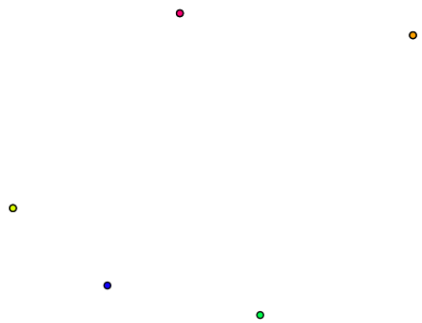

Abbildung 1: Bildung von 5 Clustern

Getestet wurde dies mit den Standardeinstellungen von 100 Lerniterationen und 100 Testdurchläufen. Die Dokumentvektoren werden dem System dabei in einer (pseudo-)zufälligen Reihenfolge zugespielt. Das *Clustering* gelingt nur dann korrekt, wenn dem System genug Platz (ein genug grosses k) gegeben wird. D. h. versucht man das System zu "zwingen" *genau* fünf Cluster zu bilden, kommt es zu Fehlzuordnungen. Bei systematischen Versuchen mit der o. g. Input-Matrize kommt es bei $k \geq 8$ zu einem stabilen *und* korrekten *Clustering*, wobei natürlich bei $k = 8$ drei der Cluster ständig leer ausgehen. Das ist ein interessanter Schwellenwert, denn im vorliegenden Beispiel liegen acht Dokumente vor. Sprich: Bleiben auch bei häufigen Lern- und Testdurchläufen eine oder (besser) mehrere der Clusterpositionen leer, so kann vermutet werden, dass das System einen deutlichen Zusammenhang in den Daten ausmachen kann und die Wahrscheinlichkeit da ist, dass das *Clustering* auch korrekt ist - entsprechend der Güte des Inputmaterials.

Gelingt es dem System im Gegenzug praktisch für jedes Dokument einen eigenen Cluster zu bilden (vorausgesetzt das k entspricht der Dokumentenzahl), obwohl aus dem Textmaterial hervorgeht, dass einige Dokumente einen Zusammenhang aufweisen müssten, so ist die Operationalisierung des Inputs nicht gelungen.

Doch noch eine weitere *wichtige* Beobachtung kann gemacht werden, die sehr dazu beiträgt das *Clustering* mit weitaus unübersichtlicheren Matrizen besser zu deuten, wie sie bei realen Dokumenten auftreten können: Falls in der Input-Matrize zu grosse Werte auftreten, beginnen die

[14] Jedes Dokument wird durch eine Spalte repräsentiert.
[15] Wo der Feature-Term auftritt, steht in der Spalte eine `0.1`
[16] Das Dokument mit der Spalte `0.0 0.0 0.0 0.1`
[17] Dokumentenpaare 2 und 8, 3 und 7 sowie 4 und 6
[18] URL: `https://github.com/IAS-ZHAW/machine_learning_scripts/blob/master/src/ml/atizo/atizo_clustering_sim.py` (13.08.2012)

Output-Werte des Systems sich anzugleichen bzw. die Cluster kommen sich (visualisiert) näher. Das kann bis zu einem Punkt gesteigert werden, wo gehäuft nur noch ein Cluster entsteht, obschon die Dokumente gemäss Input-Repräsentation klar zuordenbar erscheinen.

Dieses Phänomen beginnt sich ab Werten im Bereich 0.9 zu manifestieren und führt bei grösseren Werten (z. B. 42) praktisch nur noch zu einem Cluster mit acht Dokumenten. Das ist aber nicht nur dann eine Beobachtung, wenn alle Werte im Beispiel von 0.1 auf 0.9 oder 42 geändert werden, sondern wenn alle bis auf nur einen (oder wenige) Wert(e) geändert werden. Geschieht das nur vereinzelt, so kann dem mit einem deutlich grösseren k-Wert beigekommen werden. Treten Werte nahe 1 und höher sehr häufig auf, dann kam es im Versuch praktisch durchwegs zu einem Cluster alleine - scheinbar unabhängig von (in Relation zur Dokumentenzahl) stark überhöhten k-Werten. Z. B. erstellt das System im Falle einer Input-Matrize mit Werten 42 statt 0.1 und $k = 300$ durchwegs nur einen Cluster, den sich alle acht Dokumente teilen müssen. 299 der Clusterpositionen bleiben dann notorisch leer.

Das ist eine Erkenntnis, die im Auge behalten werden muss, wenn am realen Beispiel (z. B. mit den zu kategorisierenden E-Mails) Cluster interpretiert werden sollen. Nicht nur (1) der k-Wert muss bestimmt werden, sondern muss auch zugesehen werden, dass (2) nicht zu grosse Werte in der Matrize vorkommen und darüber hinaus dürfen (3) jene die auftreten nicht von zu grosser (nummerischer) Distanz sein. Alles das kann dazu führen, dass sich das System bei verschiedenen Durchläufen instabil (im Output) verhält und das *Clustering* somit unbrauchbar und mit aller Wahrscheinlichkeit unkorrekt verläuft.

6 Operationalisierung der Inputdaten

Das E-Mail-Korpus weist für den gewünschten Zeitabschnitt [19] folgende Eigenschaften auf: Es sind 1'341 Nachrichten vorhanden, die im Schnitt rund 2KB gross sind - wird bloss der Body-/Körperbereich der E-Mails betrachtet. [20]

Nach einer Vektorisierung der Daten in *Tokens* [21] sind rund 320 derer je Nachricht festzustellen. Diese werden dann weiter verarbeitet, denn: Ein Blick in die Vektoren deutet darauf hin, dass viele "unsaubere" Wörter bestehen. So muss z. B. verhindert werden, dass Wörter mit Bindestrichen, anderen (verbleibenden) Interpunktionszeichen oder etwa Anführungszeichen beibehalten werden. Dies würde dazu führen, dass zwei eigentlich bedeutungsgleiche Wörter als ungleich betrachtet würden.

Nach einer weitergehenden Filterung und Zerlegung von zusammengesetzten Wörtern [22] bestehen im Schnitt 248 Wörter je Dokument. Das darf allerdings nicht darüber hinweg täuschen, dass einige Dokumente sehr kurz, während andere deutlich länger sind. Im Maximum findet sich nach der Weiterverarbeitung ein Dokument mit einer Wortzahl von fast 9'000. Im Gegenzug finden sich drei Dokumente, die eine (brauchbare) Wortzahl von 0 haben, die also überhaupt nicht geclustert werden

[19] Die Jahre 1996 bis 2000 gemäss Abschnitt 2

[20] Die E-Mails weisen weitere Felder auf, etwa: Absender, Empfänger, Betreff; allerdings wird darauf verzichtet diese miteinzubeziehen, um zu verhindern, dass am Schluss die (ohnehin) verfügbaren E-Mail-Threads reproduziert werden bzw. im schlimmsten Fall gar E-Mails schlichtweg nach Absender-Empfänger-Verbindungen, anstatt nach dem eigentlichen Inhalt der Dokumente geclustert werden. Darüber hinaus garantiert eine Thread-Struktur nicht, dass der Body/Körper-Inhalt einer Nachricht gemäss Betreffzeile beibehalten wird. Eine Diskussion kann auch *off-topic* verlaufen.

[21] Durch Leerzeichen oder andere bei indoeuropäischen Sprachen (heute) üblichen Trennzeichen separierbare Einheiten. Im Regelfall handelt es sich dabei um Wörter einer natürlichen Sprache.

[22] Z. B. wird "Computer-System" in die zwei Wörter "Computer" und "System" zerlegt.

können bzw. nur unter sich selber als eigenen "Nullcluster".

Um den semantischen Zusammenhalt über Wortarten (wie Verben, Nomen etc.) hinweg zwischen den Dokumenten zu stärken, werden alle Wörter unter Einsatz eines *Porter Stemmers* aus der **NLTK**-Bibliothek [23] auf ihren Wortstamm reduziert und im Zuge dessen auch in Kleinbuchstaben umgewandelt. Wortvorkommnisse wie "Zensur", "zensur", "zensierte" oder "zensieren" fallen damit zusammen zu einem einzigen Stamm - "zens".

Wird die Anzahl einmaliger Stämme betrachtet, die als *Feature-Terme* für die *TF***IDF*-Input-Matrize genutzt werden können, so lautet die Zahl 30'484 oder 22 (im Schnitt) je Dokument.

Diese Zahlen können mit einem **wahatttt**-Programm [24] unabhängig überprüft werden, wobei zuvor das Projekt-`README` [25] zu konsultieren ist, weil auch das Datenrohmaterial vorliegen muss, um diese Daten zu generieren.

Bei der effektiven neuronalen Klassifizierung werden zusätzlich die seltensten Terme, die die höchsten *IDF*-Werte aufweisen sowie jene Stämme, die einen *IDF*-Wert unter 2.0 aufweisen, ebenso gelöscht. [26] Damit reduziert sich die Anzahl *Feature-Terme* auf 13'485. Dieser Schritt kann dadurch begründet werden, dass alle Terme unterhalb 2.0 tendenziell Stoppwörter (in Form von Stämmen) sind oder sich wie solche verhalten [27] und jene Stämme mit den höchsten *IDF*-Werten nur in einzelnen Dokumenten vorkommen, was (bei zu starker Gewichtung der entsprechenden Neuronenverschaltung) zur Bildung von 1-Dokument-Cluster führen könnte, was nicht erwünscht ist.

Auf Grund der Beobachtungen in Abschnitt 5 wird ebenfalls ein eigener *TF*-Begriff verwendet, welcher wie folgt definiert ist:

$$tf(i,d) = \frac{1.0}{avg\_doc\_size} * \frac{i}{|d|}$$

Dabei ist avg_doc_size die durchschnittliche Anzahl Stämme, die je Dokument vorhanden ist; bei i handelt es sich um den Wortstamm und bei d dem Dokumentenvektor mit den Wortstämmen der Betrachtung. Dieser *TF*-Begriff soll (1) sicherstellen, dass die Werte in der *TF***IDF*-Matrize deutlich unter 1.0 sind und (2) dass der (stark unterschiedlichen) Dokumentenlänge Rechnung getragen wird.

Im nun folgenden Abschnitt werden die (bisherigen) Ergebnisse anhand der geschilderten Operationalisierung vorgestellt.

7 Ergebnisse von Experimenten mit der E-Mail-Kollektion

Die bisherigen Ergebnisse deuten (noch) nicht auf ein erfolgreiches (stabiles) *Clustering* hin, wo es passiert, dass häufig die gleichen Cluster erstellt werden. Andererseits fällt auf, dass das System eine gewisse Trennung der Daten vornimmt; das wird deutlicher, umso höher das k gesetzt wird.

Bei $k = 10$ (Abbildung 2) und $k = 20$ (Abbildung 3) kommt es zwar zu ansehnlichen Cluster, aber sind die Fluktuationen zu hoch. Es ändern i. d. R. 80-90% der Dokumente noch immer ihre Cluster, werden mehrere Testdurchläufe durchgeführt und nach Mengenüberschneidungen zwischen den Clustern verschiedener Testdurchläufe gesucht. Das deutet darauf hin, dass die Trennbarkeit der Daten zu wünschen übrig

[23] **N**atural **L**anguage **T**ool-**K**it. URL: `https://nltk.googlecode.com/svn/trunk/doc/api/nltk.stem.snowball.PorterStemmer-class.html` (13.08.2012)
[24] URL: `https://github.com/2mh/wahatttt/blob/master/src/wh4tExplore.py` (13.08.2012)
[25] URL: `https://github.com/2mh/wahatttt/blob/master/README` (13.08.2012)
[26] URL: `https://github.com/2mh/wahatttt/blob/master/src/wh4tnnClassificator.py` (13.08.2012)
[27] Sie tragen zum adäquaten *Clustering* wenig bei, da sie zu häufig auftreten.

lässt - zumindest bei diesen zwei k-Werten.

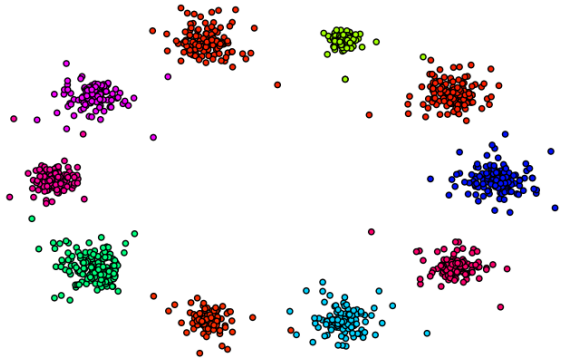

Abbildung 2: Bildung von 10 Clustern

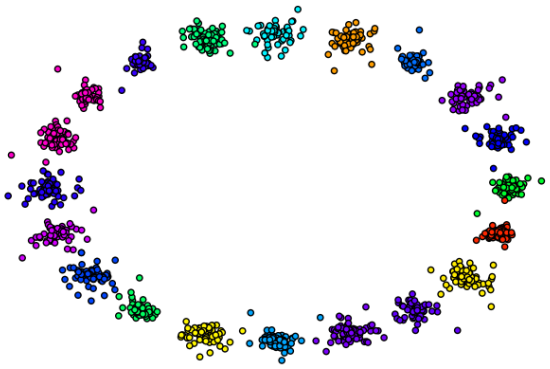

Abbildung 3: Bildung von 20 Clustern

Wird stattdessen k stark überhöht, so z. B. auf 300 gesetzt (Abbildung 4), fällt auf, dass gehäuft einzelne leere Clusterpositionen anfangen zu bestehen. Ebenfalls bleibt eine (blanke) Gleichverteilung nach den verfügbaren k Clusterpositionen aus. Es finden sich Cluster mit 0, häufig 1-2, dann aber auch 11, 12 oder 13 Dokumenten. Weil dies systematisch auftritt, kann angenommen werden, dass bei gewissen Dokumenten eine Zusammengehörigkeit erkannt wird.

Ob es sich allerdings immer um die gleichen Cluster (über Testdurchläufe) hinweg handelt, wurde allerdings (systematisch) noch nicht näher untersucht.

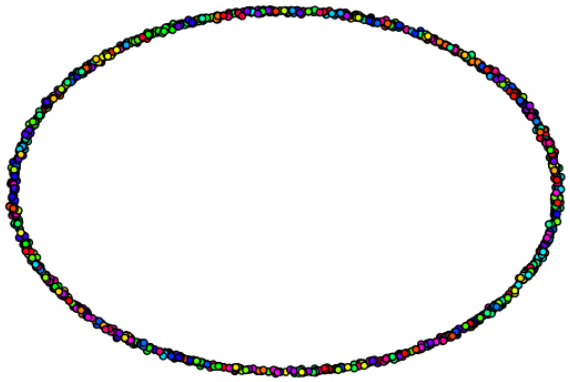

Abbildung 4: Bildung von 300 Clustern

Immerhin: Es kommt zu keinem *worst-case*-Szenario, wonach mit $k = 1341$ praktisch jedes Dokument seinen Platz finden würde. Dies allerdings war bei früheren Versuchen schon anders, wo die *Feature-Terme* mit den höchsten *IDF*-Werten beim *Clustering* mit dabei waren. Diese *IDF*-Werte haben im Zusammenhang mit dem ursprünglich ebenso zu simplizistischen *TF*-Begriff [28] zu einer schieren Gleichverteilung aller Dokumente geführt.

Es bleibt die Frage offen, ob die Dokumente *tatsächlich* einen zu geringen Zusammenhang (nach der vorliegenden Operationalisierung wie in 6 beschrieben) aufweisen, oder ob nicht an der Operationalisierung mehr gearbeitet werden muss.

8 Ungelöste Probleme / Verbesserungspotenzial

Linguistisch betrachtet können noch einige Schritte vollzogen werden, um den semantischen Bezug zwischen Dokumenten potenziell zu erhöhen. Drei dieser Schritte werden bei Niederberger et al. [1] durchgeführt, die bei **wahatttt** soweit ausbleiben: (1) Die automatische Korrektur von Schreibfehlern, [29] (2) die Übersetzung natürlichsprachlicher Fremdwörter

[28] $tf(i,d) = \frac{i}{|d|}$

[29] Mit der Gefahr, dass Wörter gebildet werden, die eine völlig andere Bedeutung aufweisen. Das ist insbesondere bei kurzen Wörtern problematisch.

[30] Dies ist deshalb sinnvoll, um den semantischen Zusammenhang über Sprachgrenzen hinweg herzustellen.

[31] Eine Menge von bedeutungsgleichen, *synonymen* Begriffen.

ins Deutsche [30] und (3) die Ersetzung von Begriffen, die in einem *Synset* [31] gefunden werden können mit nur einem gemeinsamen Begriff. [32]

Darüber hinaus wäre es noch denkbar sogenannte *Ontologien* oder Begriffshierarchien, wie z. B. **WordNet** [33] fürs Englische existiert einzusetzen, um Unterbegriffe (*Hyponyme*) einem Oberbegriff (*Hypernym*) zuzuordnen. So könnten z. B. Tiere (mit Begriffen wie "Affe" oder "Katze") durch den Begriff "Tier" ersetzt werden, womit sich das *Clustering* stark verändern könnte.

Der Einsatz dieser linguistischen Ressourcen muss aber wohl überlegt geschehen, da sich ansonsten auch falsche Zusammenhänge ergeben können, wenn es z. B. dazu kommt, dass für ein Begriff mehrere *Synsets* in Frage kommen.

Z. B. sind für den Begriff der "Zensur" in **OpenThesaurus**, das Niederberger et al. [1] in seiner Arbeit einsetzt, zwei verschiedene *Synsets* vorhanden; im Folgenden kommasepariert dargestellt:

(1) `Schulnote;Note;Beurteilung;Zensur`
(2) `Zensur;Verbot;Informationssperre;Indizierung`

Danksagungen

Ich danke Herrn Prof. Dr. habil. Ruedi Stoop und Florian Gomez des Instituts für Neuroinformatik (INI) der Universtität Zürich und ETH Zürich mich auf die Arbeit von Niederberger et al. [1] verwiesen und mich bei fundamentalen Fragen im Zusammenhang mit *KNN* mit weiteren Lektürelinks versorgt zu haben. Besonderer Dank gebühren Prof. Dr. Thomas Ott und Thomas Niederberger des Instituts für Angewandte Simulation (IAS) der Zürcher Hochschule für Angewandte Wissenschaften (ZHAW), die rasch auf spezifische Fragen reagiert haben und durch die Verfügbarmachung von Quellcode die komplexeren Arbeitsschritte vorweggenommen haben.

Literatur

[1] T. Niederberger, N. Stoop, M. Christen und T. Ott. Hebbian principal component clustering for information retrieval on a crowdsourcing platform. In *Proceeding of NDES (Nonlinear Dynamics of Electronic Systems)*. 2012.

[32] Kommt in einem Dokument z. B. "Zensur" vor, in einem anderen Dokument dagegen "Informationssperre", so gehören diese Dokumente semantisch potenziell zusammen.
[33] URL: `http://wordnet.princeton.edu/` (13.08.2012)