



Department of Informatics
University of Fribourg
Information Systems Research Group

Bachelorarbeit

2010-2011

Entwicklung einer unscharfen Ontologie für das Semantische Web

Von

Sandro KOLLY

08-212-144

Eichenweg 12

1718 Rechthalten

sandro.kolly@bluewin.ch

Referent: Prof. Dr. Andreas Meier

Betreuer: Edy Portmann

Freiburg, 02. Dezember 2011

Abstract

Diese Arbeit behandelt das Erstellen von unscharfen Ontologien im Social Semantic Web. Mit Hilfe von unscharfen Clustering Algorithmen werden diese berechnet und ausgegeben. Unscharfe Clustering (Fuzzy Clustering) Algorithmen sind Clustering Algorithmen basierend auf unscharfer Logik. Die Ausgabe ist in einer Struktur, die von Computern interpretiert werden kann.

Das Social Semantic Web stellt einen Schnittpunkt zwischen Social Web und Semantic Web dar. Es macht die Informationen aus dem Social Web interpretierbar und somit nützlich für die Computer. Damit wird den Computern ein riesiges Wissen gewährt, das mit bestimmten Technologien jedem Endnutzer weiterhilft in verschiedenen Forschungsrichtungen.

Dieses Wissen wird in unscharfen Clustern berechnet und dargestellt. Mit Hilfe der unscharfen Logik können logische Aussagen natürlicher gemacht werden und somit an ungenaue Aussagen von Menschen angepasst werden.

Dies ergibt zusammengefügt ein natürliches Wissen, welches für Computer interpretierbar und damit weiterverwendbar ist. Der Computer lernt immer mehr dazu, wird immer intelligenter und kann so dem Endnutzer natürlicher, und speziell auf den Endnutzer bezogen, helfen.

Stichworte

Social Web, Semantic Web, Social Semantic Web, RDF, RDFS, OWL, Ontologie, Fuzzy Logic, Clustering, Fuzzy Clustering,

Inhaltsverzeichnis

Abstract	iii
Stichworte.....	iii
Abbildungsverzeichnis	vii
Tabellenverzeichnis	vii
1 Einleitung.....	8
1.1 Problembeschreibung.....	8
1.1.1 Übersicht	8
1.1.2 youReputation	9
1.2 Ziele/Forschungsfragen	9
1.3 Methode	10
2 Social Semantic Web	11
2.1 Social Web.....	11
2.2 Semantic Web.....	11
2.3 Von Web 2.0 zu Web 3.0	12
2.4 Ontologien und ihre Sprachen.....	13
2.4.1 Ontologie in der Theorie	13
2.4.2 Ontologie als Beispiel	14
2.4.3 RDF.....	15
2.4.4 RDF-Schema.....	15
2.4.5 OWL	17
2.4.6 Zusammenfassung der Ontologiesprachen.....	19
3 Unscharfe Logik (Fuzzy Logic).....	20
3.1 Geschichte.....	20
3.2 Formale Erklärung.....	20

3.3	Vergleich unscharfer Logik und scharfer Logik.....	22
3.4	Anwendungsfelder.....	23
4	Clustering.....	24
4.1	Hard Clustering	25
4.2	Fuzzy Clustering	25
5	Vergleich dreier Clustering-Algorithmen.....	26
5.1	Beschreiben der drei Algorithmen.....	26
5.1.1	Erklärung der Auswahl.....	26
5.1.2	Fuzzy C-Means clustering algorithm (FCM).....	27
5.1.3	Pseudocode FCM	28
5.1.4	Gustafson-Kessel (GK)	29
5.1.5	Pseudocode GK.....	30
5.1.6	Fuzzy clustering by Local Approximation of MEMbership (FLAME).....	31
5.1.7	Pseudocode FLAME	32
5.1.8	Abschluss eines Algorithmus	34
5.2	Kriterien für einen Vergleich.....	35
5.2.1	Erklärung der Auswahl.....	35
5.2.2	Komplexität.....	35
5.2.3	Konstanz/Zufallseinfluss	36
5.2.4	Anpassungsfähigkeit.....	36
5.2.5	Erklärung.....	37
5.3	Anwendung der Kriterien auf die Algorithmen	37
5.3.1	FCM.....	37
5.3.2	GK.....	37
5.3.3	FLAME	38

5.4	Vergleich	38
5.4.1	Vergleichsübersicht	39
5.4.2	Visueller Vergleich	40
5.4.3	Zusammenfassung	41
5.4.4	Schlussfolgerung.....	41
5.5	Notiz.....	42
6	Bewertung eines Clusters	42
6.1	Interne Bewertungen.....	42
6.1.1	Interne Bewertungen für Fuzzy Clustering.....	43
6.1.2	Vergleich der Fuzzy Davies-Bouldin Indizes der Ontologien	44
6.2	Informationen zum DBI.....	44
6.3	Erklärung	45
6.4	Berechnung des FDBI für die erstellte Ontologie	45
7	Schlussteil	45
7.1	Zusammenfassung	45
7.2	Ausblick	46
	Literaturverzeichnis	48
	Anhang: FLAME in MatLab für youreputation	50
	flame_calc()	50
	flame_init()	51

Abbildungsverzeichnis

Abbildung 1: Entwicklungspfade zum Social Semantic Web (Blumauer, et al., 2009) ..	12
Abbildung 2: Beispiel einer kleinen Ontologie (Anlehnung an (Thatte, 2008))	14
Abbildung 3: Doppelpes RDF Tripel.....	15
Abbildung 4: Symmetrie in OWL	18
Abbildung 5: Lage von RDF(S) in der Architektur des Semantic Web (Wikipedia)	19
Abbildung 6: Grafische Darstellung der Beispielfunktion zur Körpergrösse	22
Abbildung 7: Einfaches Beispiel dreier Cluster nach einem Clustering (Wikipedia).	24
Abbildung 8: Plot einer GK Ontologie.....	40
Abbildung 9: Plot einer FCM Ontologie.....	40
Abbildung 10: Plot einer FLAME Ontologie	41

Tabellenverzeichnis

Tabelle 1: Die drei Teilsprachen von OWL und ihre wichtigsten Eigenschaften (Hitzler, et al., 2008).....	17
Tabelle 2: Beispiel einer Zugehörigkeitsfunktion aus dem Beispiel der Körpergrösse ..	21
Tabelle 3: Vergleichsübersicht der Clusteringalgorithmen	39
Tabelle 4: Berechnungen des Fuzzy Davies-Bouldin Indizes	44

1 Einleitung

Das World Wide Web und seine Technologien entwickeln sich immer weiter. Facebook, Twitter und weitere soziale Netzwerke sind schon lange für jedermann ein Begriff und Aus Web 2.0 wird zusehends Web 3.0. Es gibt kaum mehr Teile dieser Erde, wo man nicht mit dem Web verbunden sein kann.

Immer mehr Menschen sehen im Begriff „Web“ etwas Alltägliches. Alte Kommunikationsmittel wie das Haustelefon und der Briefverkehr werden immer mehr durch neue ersetzt: Chats, E-Mails und Diskussionen in Internetforen bilden schon lange eine technologische Art der Kommunikation. Und jeder und alles hat eine eigene Homepage, um sich in dieser neuen Welt in Szene zu setzen.

Das alles ist Web 2.0. Immer mehr Informationen werden einfach so ins Netz gestellt, Web 3.0 versucht nun, auch für Maschinen daraus einen Nutzen zu schaffen.

1.1 Problembeschreibung

Es gibt immer mehr Webbenutzer. Jeder auf dieser Welt diskutiert zu den Themen, die er mag und zu denen er sich äussern möchte. Jede Gruppe von Menschen will für ihre Diskussionsrunde eine Homepage mit Forum und aktuellen Informationen.

1.1.1 Übersicht

Und hier beginnt das eigentliche Problem: Das Web stellt ein Überangebot an Informationen zur Verfügung. Wer mit einer Suchmaschine nach etwas Speziellem sucht, wird vor lauter Suchtreffern förmlich erschlagen. Es gibt mittlerweile so viele Seiten im Web, dass man sich anstrengen muss, um auf Nummer sicher gehen zu können und eine Seite als verlässliche Quelle zu akzeptieren.

Einen Ansatz zur Verbesserung dieses Problems stellt das Semantische Web (Englisch: Semantic Web) dar. Das Semantic Web stellt die Informationen nach (Hitzler, et al., 2008) so dar, dass diese durch Maschinen lesbar und verarbeitbar sind. So sollten diese Maschinen beispielsweise Zusammenhänge verschiedener Begriffe bei mehrmaligem gemeinsamem Auftreten als solche registrieren.

Gerade die schon angesprochenen Suchmaschinen haben heutzutage riesige Aufgaben zu bewältigen. Dank statistischen Verfahren können aber die Suchtreffer der Relevanz nach für den suchenden Nutzer sortiert und angeordnet werden. Für jeden Nutzer wird so ein Profil angelegt, das seine Präferenzen wertet und diese jederzeit so verändert, damit es dem aktuellen Verhalten weiter angepasst wird.

1.1.2 youReputation

Das in dieser Arbeit im Hauptteil untersuchte Problem dient der Bildung einer maschinenlesbaren Wissensrelation zwischen verschiedenen gegebenen Begriffen. Diese Berechnungen, welche durchgeführt werden, dienen dem Projekt youReputation. Dies ist ein interuniversitäres Projekt zwischen der NUS (National University of Singapore) und der Universität Freiburg mit dem Ziel, einen Prototypen zu erstellen, um Reputationen auf Basis von verschiedenen Klassen zu analysieren.

Das Projekt youReputation besteht aus verschiedenen Teilprojekten. Der bereits erwähnte Hauptteil dieser Arbeit wird ebenfalls ein solches Teilprojekt beschreiben. Es geht nicht nur darum, irgendeine Relation zu berechnen, sondern anhand verschiedener Algorithmen zu bestimmen, wie diese Wissensrelationen am bestmöglichen berechnet werden können und das „attraktivste“ Resultat ausgibt.

1.2 Ziele/Forschungsfragen

Ein Datensatz soll so verarbeitet werden, dass die Outputs des verarbeitenden Algorithmus weiter verwendbar werden für Maschinen. Dazu braucht es Zwischenschritte, welche als Ziele dieser Arbeit betitelt werden. Diese Schritte führen Stück für Stück zum Endziel, welches die Ausgabe der Verarbeiteten Daten in maschinenlesbarer Sprache ist. Damit können diese Daten für den Prototyp verwendet werden.

Um den ganzen Prozess zu verstehen, braucht es zuerst eine kurze Einführung in andere Themen, welche die Teilschritte der ganzen Arbeit darstellen. Dazu wird als Ziel gesetzt, auf die folgenden Fragen einzugehen, welche als Leitfragen dieser Arbeit dienen:

- Was bedeuten die Begriffe Social Web, Semantic Web und Social Semantic Web? Wie werden Social und Semantic Web zu Social Semantic Web verbunden?
- Was ist Fuzzy Logic und wozu dient diese? Was ist der Gedanke hinter Fuzzy Logic?
- Was ist Clustering und was Fuzzy Clustering, wo und wozu wird es benutzt?

Mit Antworten auf die letzten beiden Fragen, wird man sich auch im Hauptteil dieser Arbeit gut zu Recht finden, auch wenn dieser sehr mathematisch aufgebaut ist. Diese drei Leitfragen führen durch die ganze Arbeit, sie bilden die Theorie, auf der der Hauptteil aufbaut. Es wird also erst auf die Leitfragen eingegangen, dann wird die Datenverarbeitung erklärt.

1.3 Methode

Um sich dennoch ein Bild zu machen, wird die Methode der Datenverarbeitung in maschinenlesbare Sprache bereits kurz eingeführt. Es geht darum, einen Algorithmus auszuwählen, der die Daten so berechnet, dass diese brauchbar werden für die weiteren Stufen im Prototyp. Es gilt also folgende drei Punkte chronologisch nacheinander auszuführen:

- 1) Finde den „attraktivsten“ mathematischen Algorithmus zur Verarbeitung der Daten
- 2) Wende den Algorithmus auf den Datensatz an
- 3) Konvertiere den Output des Algorithmus in maschinenlesbare Sprache

Punkt 1) wird den grössten Teil an Zeit in Anspruch nehmen, da ein ausführlicher Vergleich aufgestellt werden muss, um den „attraktivsten“ Algorithmus zu erküren. Die Algorithmen werden in voller Länge und mit jedem Detail vorgestellt, Kriterien werden festgelegt, die Kriterien werden auf die Algorithmen angewandt und dann wird verglichen. Zuletzt kann ein Algorithmus zur Verarbeitung gewählt werden.

Um den Algorithmus, wie in Punkt 2) erwähnt, auf den Datensatz anzuwenden, müssen die Daten in die benötigte Form gebracht werden und der Algorithmus kann schon darauf angewendet werden.

Das Resultat aus Punkt 2) muss nun in Punkt 3) maschinenlesbare Sprache gebracht werden. Dafür muss direkt der Output des Programms, welches die Berechnungen durchführt, so geformt werden, dass die Ausgabedatei von gewünschter Form ist.

2 Social Semantic Web

Das Social Semantic Web ist nur ein anderer Name für das Web 3.0, ist aber so besser und schneller verständlich. Es gibt eine ganz einfache Gleichung für das Web 3.0 (Berners-Lee, et al., 2001):

$$\mathbf{Web\ 2.0 + Semantic\ Web = Web\ 3.0}$$

Diese Gleichung kann auch detaillierter dargestellt werden:

$$\mathbf{Social\ Web = Web\ 2.0 + Semantic\ Web = Web\ 3.0 = Social\ Semantic\ Web}$$

Das bedeutet, dass das Social Semantic Web nichts anderes ist, als eine Komposition aus Social und Semantic Web und deswegen auch diesen Namen besitzt.

2.1 Social Web

Über das Social Web braucht man in der heutigen Zeit kaum mehr Worte zu verlieren.

Es ist genau das, was die meisten Menschen vom Web zurzeit kennen.

Webplattformen mit sozial betriebenen Foren, sozialen Netzwerken und andere Seiteninhalte, welche von jedem User mitgestaltet werden können.

2.2 Semantic Web

Das Semantic Web dient dazu, die Probleme der Informationsintegration und des impliziten Wissens für Computer zu lösen (Hitzler, et al., 2008). Damit Computer „lernen“ können, braucht es Daten, die für die Computer lesbar und interpretierbar sind. Um im Web nach brauchbaren Daten suchen zu können, müssen diese Daten aber auch in einer bestimmten Form vorhanden sein. Diese Form unterscheidet sich

aber von der Form, die für uns von Nutzen ist: Aus geschriebenem Text kann ein Computer nur schwer lernen.

Es braucht einen Standard, in welcher Form die Webseiten angeboten werden. Wenn alle Seiten ihre Daten in maschinenlesbarer Sprache (lesbar für Computer) anbieten, dann kommt eine ganze Menge an Informationen zusammen, aus denen ein Computer lernen kann. Er kann dann auf Grund logischer Verbindungen selbst Schlüsse ziehen, er beginnt Zusammenhänge zu erkennen und weiss, was dem Nutzer anhand seines bisherigen Surfverhaltens am stärksten zusagt, beispielsweise bei einer Suche mit einer Suchmaschine.

Das Semantic Web und das Social Web bilden gemeinsam das Social Semantic Web (vgl. Formel oben). „Gemeinsam bilden“ kann sowohl heissen, dass aus dem Social Web mit Hilfe des Semantic Web das Social Semantic Web hervorgeht, als auch, dass umgekehrt, aus dem Semantic Web mit Hilfe des Social Webs das Social Semantic Web hervorgeht (siehe Abb. 1).

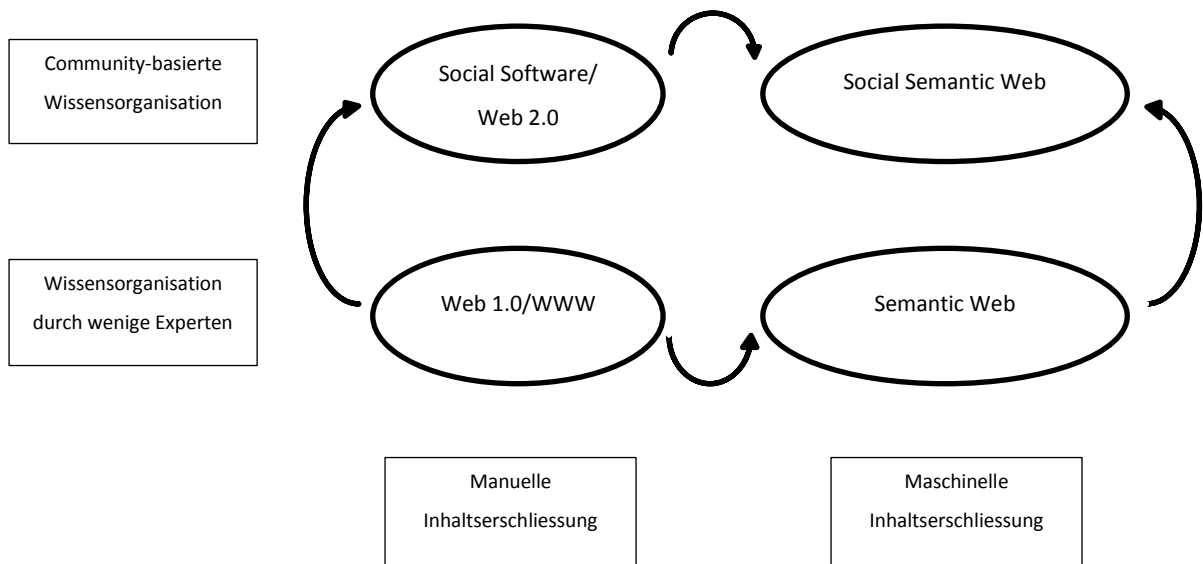


Abbildung 1: Entwicklungspfade zum Social Semantic Web (Blumauer, et al., 2009)

2.3 Von Web 2.0 zu Web 3.0

Es kam aber nicht von einem Tag auf den anderen dazu, dass aus Web 2.0 die „Version“ Web 3.0 wurde. Web 2.0 wurde dadurch geprägt, dass die Nutzer im Web

mitmachen konnten. Es gab nicht nur mehr statische Homepages irgendwelcher Unternehmen, viel mehr begann jeder sich aus seiner Anonymität zu lösen und sich im Web zu etablieren. Ob durch eine eigene Homepage, Foreneinträge, Auftreten in Chatrooms oder später auch mit einem Twitter-, Facebook- und anderen Social Network-Account, der Präsenz im Web wurden kaum mehr Grenzen gesetzt.

Wie schon angedeutet wurde, bringt eine solche Vielzahl an Informationen auch negative Aspekte mit sich: Beispielsweise kann eine Suche nach einer Seite mit dem gewünschten Inhalt zu einer Riesenaufgabe werden.

Und genau da beginnt die Wirkung des Semantic Web: Das Semantic Web basiert auf den erwähnten Standards, welche in diesem Kapitel erklärt werden. Wie bereits erwähnt, bildet das Web 2.0 zusammen mit dem Semantic Web das Web 3.0. Dies bedeutet nichts anderes, als dass sich für uns als Mensch von der Ansicht einer Webseite nichts verändert, für die Maschine, welche die Seite „betrachtet“, aber schon. Für menschliche Nutzer im Normalfall nicht sichtbare Relationen, welche eine Seite beschreiben, geben einer Maschine genügend Informationen, um die Seite einstufen zu können.

2.4 Ontologien und ihre Sprachen

2.4.1 Ontologie in der Theorie

Informationen für Ontologien, welche im vorangegangenen Unterkapitel erwähnt wurden, bestehen aus Tripeln. Diese Tripel haben eine feste Form: Sie bestehen aus Subjekt, Prädikat und Objekt (Hitzler, et al., 2008). Diese Tripel stellen die Hauptinformationen einer Ontologie dar. Eine Ontologie weiss also von jedem Subjekt, welche Beziehung es zu einem Objekt hat. Um dann eine ganze Ontologie zu definieren, braucht es auch noch Klassen, zu denen die Subjekte gehören, Hierarchien von Klassen und Unterklassen.

Diese maschinenlesbaren Wissensrepräsentationen der Seiten zur Verfügung zu stellen, ist das Ziel von Web 3.0. Solche Wissensrepräsentationen mit fest vorgegebener Struktur nennt man Ontologie (Ultes-Nitsche, 2010).

2.4.2 Ontologie als Beispiel

All das kann man sich am besten als Graph vorstellen. Ohne bereits die RDF und RDFS Terme zu beachten, kann man sich eine kleine Ontologie folgenderweise vorstellen (siehe Abbildung 2):

Max Muster lebt in Bern. Dies ist die Hauptaussage dieser Ontologie. Nun gibt es aber noch weitere Informationen, welche diese Ontologie aussagt. Nämlich dass Max Muster zu den stimmberechtigten Einwohnern gehört. Diese Untergruppe stellt zusammen mit den nicht stimmberechtigten Einwohnern die Gruppe aller Bürger dar.

Auf der anderen Seite informiert die Ontologie darüber, dass Bern eine Stadt ist. Die Untergruppe der Städte bildet zusammen mit den Dörfern den jeweiligen Kanton. Alle Kantone zusammen bilden dann das Land Schweiz.

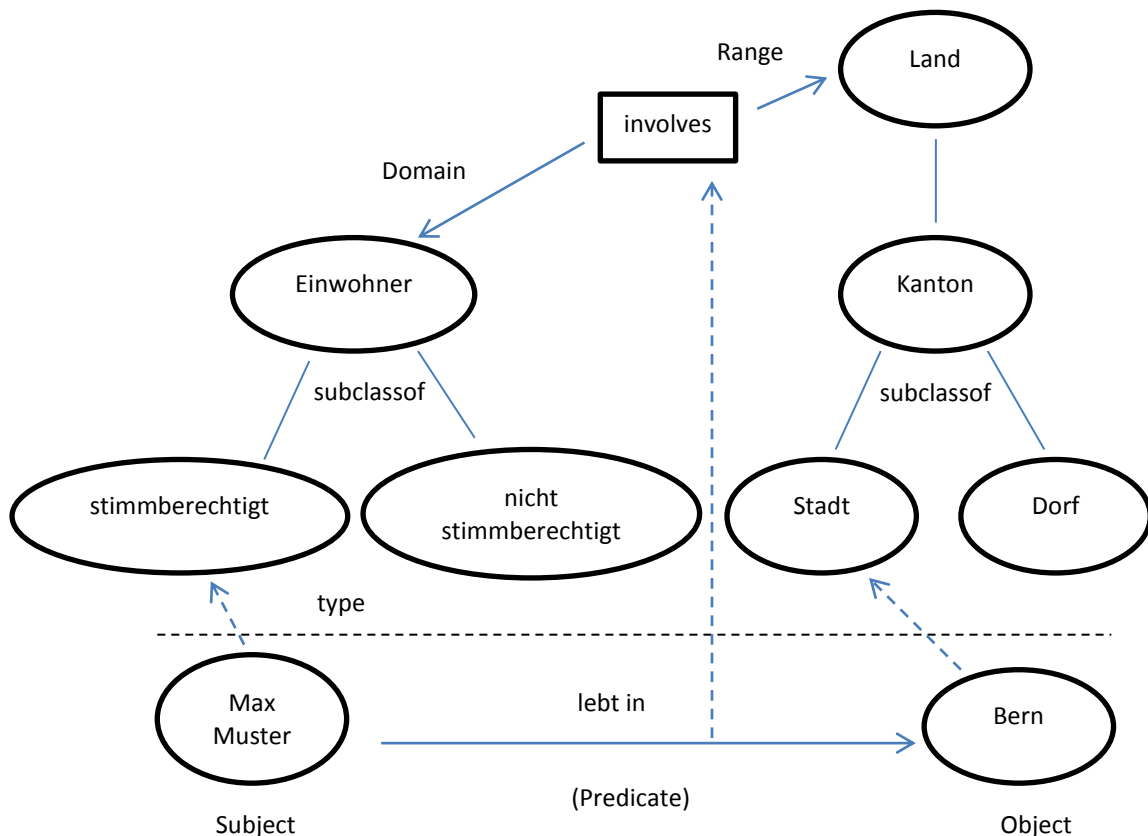


Abbildung 2: Beispiel einer kleinen Ontologie (Anlehnung an (Thatte, 2008))

Diese beiden Seiten können verbunden werden, indem die Einwohner dem Land Schweiz „zugeordnet“ werden und umgekehrt auch das Land Schweiz den Einwohnern „zugeordnet“ wird.

Alle solche Ontologien beschreibenden Sprachen, welche in diesem Kapitel erwähnt werden, basieren formell auf der Syntax von XML und wurden vom W3C ebenfalls standardisiert.

2.4.3 RDF

RDF steht für Resource Description Network. Wie der Name schon sagt, beschreibt RDF ein Netzwerk von „Ressourcen“ in Form von Informationen. RDF ist die simpelste Art der Beschreibung einer Ontologie, daher kann auch nur eine Ontologie einfachster Art beschrieben werden. RDF verfügt nur über Triple-Statements, also Subjekt, Prädikat und Objekt, ohne Klassen, Unterklassen oder anderen Definitionen (siehe Abb. 2 unter der gestrichelten Linie). Die einzige Spezialisierung von RDF ist es, Elemente einem Typ zuzuordnen, also sie einer Gruppe von gleichartigen Elementen hinzuzufügen (Hitzler, et al., 2008).

RDF kann einfache Relationen zweier Begriffe beschreiben. Diese Relationen können auch weiter vernetzt werden. Ein kleines und einfaches Beispiel stellt Abbildung 3 dar: Ein Begriff kann sowohl Objekt eines Tripels wie auch zugleich Subjekt eines anderen Tripels sein. Dies lässt die Bildung eines grösseren Netzwerks zu.



Abbildung 3: Doppeltes RDF Tripel

2.4.4 RDF-Schema

RDF-Schema (kurz: RDFS) ist eine Spezialisierung von RDF. Die Syntax von RDFS kann auch von einem Programm gelesen werden, das nur RDF unterstützt. Es werden dann einfach einige Informationen verloren gehen, welche RDFS spezifisch sind (Hitzler, et al., 2008).

RDF-Schema ist nichts anderes als eine Schema Sprache von RDF. Es gibt verschiedene Arten von Schema Sprachen: Eine Schema-Sprache kann dazu da sein, eine gewisse Ordnung zu fordern. Beispielsweise hält das XML-Schema fest, in welcher Struktur eine XML-Datei zu sein hat. Eine Schema-Sprache kann aber auch helfen, Daten zu interpretieren. Ein Datenbankschema einer relationalen Datenbank gibt beispielsweise Schlüsselinformationen an. Es gibt weitere Schemas, die zum Teil verschieden wirken, aber in einem Punkt doch eines gemeinsam haben: Das Schema gibt Informationen über die jeweiligen Daten preis, die im jeweiligen System vorhanden sind (Allemang, et al., 2008).

Das RDF Schema wird anhand der Abbildung 2 sehr gut veranschaulicht. Die eigentliche Information ist nur, dass Max Muster in Bern lebt, welche als RDF Tripel angesehen wird. Durch die ganzen Meta-Daten können aber all die Informationen gewonnen werden, welche am Anfang dieses Kapitels beschrieben werden.

RDFS kann, im Gegensatz zum „Typ zuordnen“ von RDF, das Ganze viel strukturierter und nicht nur so allgemein gestalten: Klassen können definiert werden und Instanzen können Klassen hinzugefügt werden. Unterklassen von Klassen können erstellt werden und so Klassenhierarchien festgelegt werden. Eigenschaften können Klassen und Instanzen zugeordnet werden, dasselbe gilt für Untereigenschaften, gemeinsam können auch diese eine Hierarchie von Eigenschaften bilden.

Eines der Probleme mit RDFS stellen die negativen Beziehungen dar: Aussagen mit negativen Beziehungen, also Beziehungen die nicht gelten, können mit RDFS nicht dargestellt werden. Es kann zwar jemand als Nichtraucher und ein anderer als Raucher definiert werden, dafür müssen diese aber zwei verschiedenen Klassen hinzugefügt werden. Es ist dementsprechend möglich, jemanden der Klasse der Raucher hinzu zu fügen, aber man kann niemanden explizit als nicht in dieser Klasse definieren (Hitzler, et al., 2008).

RDFS ist also fähig, eine Wissensrepräsentation/Ontologie darzustellen, hat aber noch Defizite, welche konkret auf den Programmierer zukommen, sobald die Ontologien noch etwas genauer dargestellt werden sollen.

2.4.5 OWL

Die Web Ontology Language OWL baut auf RDFS auf und ist eine ausdrucksstarke Repräsentationssprache. Eine OWL-Ontologie besteht, wie schon grösstenteils RDFS, aus Klassen, Eigenschaften (in OWL auch Rollen genannt) Instanzen und Operationen (Ultes-Nitsche, 2010). OWL erweitert RDFS und versucht damit die erwähnten Limitationen zu überwinden.

OWL ist in drei Teilsprachen unterteilt: OWL Full, OWL DL und OWL Lite. Tabelle 1 gibt einen Überblick über die Eigenschaften dieser drei OWL Teilsprachen:

OWL Full	OWL DL	OWL Lite
enthält OWL DL und OWL Lite	enthält OWL Lite und ist Teilsprache von OWL Full	ist Teilsprache von OWL DL und OWL Full
enthält als einzige OWL-Teilsprache ganz RDFS	entscheidbar	entscheidbar
sehr ausdrucksstark	wird von aktuellen Softwarewerkzeugen fast vollständig unterstützt	weniger ausdrucksstark
Semantik enthält einige Aspekte, die aus logischem Blickwinkel problematisch sind	Komplexität: NExpTime (worst-case)	Komplexität: ExpTime (worst-case)
unentscheidbar		
wird durch aktuelle Softwarewerkzeuge nur bedingt unterstützt		

Tabelle 1: Die drei Teilsprachen von OWL und ihre wichtigsten Eigenschaften (Hitzler, et al., 2008).

Neu dazu kommen in OWL vor allem die Restriktionen, die auch für die Komplexitätsbeziehungen von Bedeutung sind. Bisher, in RDF und RDFS, galt die sogenannte AAA Regel: Anyone can say Anything about Any topic (Irgendjemand kann Irgendetwas über Irgendein Thema sagen) (Allemang, et al., 2008). Es war also bisher

entgegen aller Logik möglich, völlig sinnlose Tripel zu erstellen, ohne dass irgendein System einen Fehler hätte entdecken können.

In OWL kann man aber dagegen vorgehen, indem man Restriktionen einbaut. Ein Beispiel könnte folgendermassen aussehen: Lionel Messi ist ein Fussballspieler. Wenn man nun ein Kader einer Eishockeymannschaft definieren möchte, dann muss man die Restriktion einbauen, dass jeder Spieler des Teams Eishockeyspieler sein muss.

Das gleiche Prinzip gilt für die Komplexitätsbeziehungen einzelner Klassen und deren Instanzen: Beim Definieren einer Startaufstellung einer Fussballmannschaft können nicht einfach alle Spieler aufs Feld geschickt werden, es muss eine Restriktion geben, welche besagt, dass genau 11 Spieler aufgestellt werden müssen. Auch nicht genaue Kardinalitäten wie „mindestens“ oder „maximal“ können dargestellt werden: Eine Sportmannschaft muss mindestens einen Trainer haben, ein Kader einer Nationalmannschaft darf aus maximal 23 Spielern bestehen.

OWL lässt auch intelligentere Ontologien zu als RDF(S): Eigenschaften von OWL sind die Erkennung von Symmetrie, Transitivität und Inversität:

Die Symmetrie besagt, dass wenn Lionel Messi mit David Villa spielt, dann spielt auch David Villa mit Lionel Messi. Natürlich unter der Vorgabe, dass es eine Klasse mit Instanzen gibt, die zusammen gehören und in diesem Beispiel zusammen spielen.

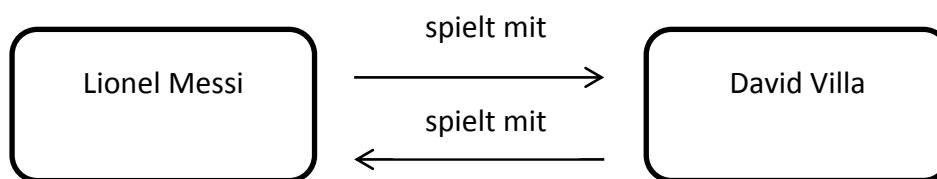


Abbildung 4: Symmetrie in OWL

Der Vorteil der Erkennung der Transitivität ist, dass nicht jede Hierarchiestufe definiert werden muss: Der Abteilungsleiter ist dem Teamleiter übergeordnet und der Teamleiter ist dem Teammitglied übergeordnet. Also ist der Abteilungsleiter auch dem Teammitglied übergeordnet.

Bei entsprechender Deklaration kann OWL auch Inversität erkennen: Der Spieler attackiert Gegenspieler, also wird der Gegenspieler vom Spieler attackiert. Auch hier ist der Vorteil der, dass nicht alles in Tripel beschrieben werden muss, ein Weg reicht, der andere wird von OWL erkannt.

2.4.6 Zusammenfassung der Ontologiesprachen

Einen guten Überblick darüber, wie XML, RDF, RDFS und OWL verbunden sind, bietet Abbildung 5:

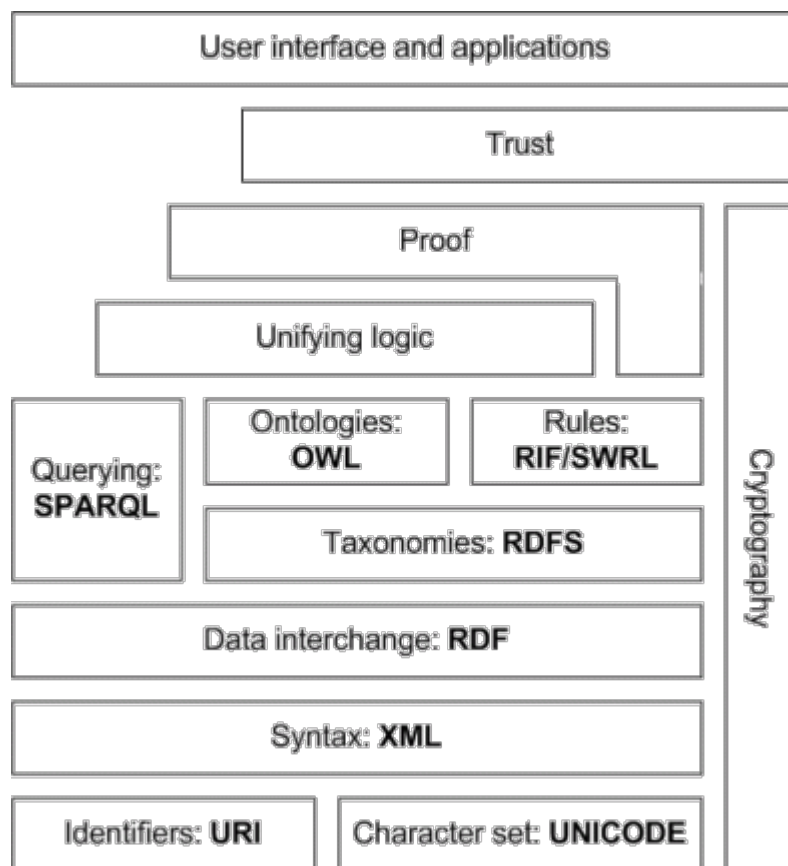


Abbildung 5: Lage von RDF(S) in der Architektur des Semantic Web (Wikipedia)

OWL ist eine Schicht oberhalb RDFS und dies direkt oberhalb RDF, da, wie schon erwähnt, die OWL Syntax für eine RDF interpretierende Software eine normale RDF Syntax ist, mit OWL Inhalten, die dann einfach verschwinden.

Für die angegangenen Sprachen kann also festgehalten werden, dass RDF mit RDFS ausreicht, um kleine „unkomplizierte“ Ontologien darzustellen. Sobald komplexere und grössere Ontologien dargestellt werden sollen, kommen RDF und RDFS an ihre

Grenzen und es braucht die ausdrucksstärkere Ontologiesprache OWL, um alle Wissensrelationen exakt darstellen zu können.

3 Unscharfe Logik (Fuzzy Logic)

Der Mensch neigt in seiner Wortwahl oft dazu, Begriffe sehr objektiv auszusuchen. Wenn ein Mensch von einem „grossen“ Mann spricht, wissen seine Gesprächspartner oft nicht so genau, was dieser darunter versteht, können sich aber aufgrund von Erfahrungen und Gebräuchlichkeit selbiger Terme dennoch recht gut ein Bild davon machen. Es treten immer wieder solche Wörter auf, die mehrere Interpretationsmöglichkeiten zulassen oder jedenfalls einen breiten Spielraum dafür bieten.

3.1 Geschichte

Früher und auch heute noch heisst es oft einfach „wahr“ oder „falsch“, „Ja“ oder „Nein“, oder wie in der Informatik 1 oder 0. Die ersten mathematischen Aspekte, die nicht auf sturer wahr-falsch Denkweise beruhten, also auf „unsicherer“ oder ungenauer Mathematik, kamen im 16. Jahrhundert auf, als Glücksspieler begannen, ihre Chancen in den verschiedenen Spielen auszurechnen (Ross, 2010). So wurde aus jenen, die sich der Wahrscheinlichkeitsrechnung bedienten, statt purer Glücksspieler kühne Strategen, welche immer den Weg wählten, der für einen Gewinn am wahrscheinlichsten war.

Über den Begriff der Vagheit (engl. vagueness) (Black, 1937) führte die Geschichte der unsicheren Mathematik durch Zadeh zum Begriff „unscharf“, welcher in seiner Veröffentlichung „Fuzzy Sets“ eingeführt wurde (Zadeh, 1965). „Fuzzy Sets“ stellte damals schon neue Möglichkeiten an Berechenbarkeit vor, an denen noch heute geforscht wird.

3.2 Formale Erklärung

Die Fuzzy Sets werden im gleichnamigen Werk beschrieben als Zugehörigkeitsgrade zu verschiedenen Klassen (Zadeh, 1965). In einem Satz mit unscharfen Begriffen gibt es zu jeder Menge, zu der ein solcher Begriff gehört, eine Zugehörigkeitsfunktion für die

Fuzzy Logic (unscharfe Logik). „Vor wenigen Tagen sah ich einen grossen Mann hier vorbei laufen!“ In diesem Satz sind die umgangssprachlichen unscharfen Mengen die Grösse und die Anzahl der Tage, welche in der Literatur der unscharfen Logik auch als linguistische Variablen bezeichnet werden. Die dazugehörigen Werte, wie in diesem Satz „wenige“ und „gross“, werden analog dazu als linguistische Terme bezeichnet (Schmidt, et al., 2010).

Eine zugehörige Funktion für die Grösse (und mit anderen Zahlen auch für die Anzahl Tage) könnte beispielsweise so aussehen:

Term	Funktionsbereich	Funktionswert
x = klein	$x < 150\text{cm}$	$f(x) = 1$
	$150\text{cm} \leq x < 180\text{cm}$	$f(x) = 1 - (x - 150) / 30$
	$180\text{cm} \leq x$	$f(x) = 0$
x = mittel	$x < 160 \text{ cm}$	$f(x) = 0$
	$160\text{cm} \leq x < 170\text{cm}$	$f(x) = (x - 160) / 10$
	$170\text{cm} \leq x < 190\text{cm}$	$f(x) = 1 - (x - 170) / 20$
	$190 \text{ cm} \leq x$	$f(x) = 0$
x = gross	$x < 170 \text{ cm}$	$f(x) = 0$
	$170\text{cm} \leq x < 190\text{cm}$	$f(x) = (x - 170) / 20$
	$190 \text{ cm} \leq x$	$f(x) = 1$

Tabelle 2: Beispiel einer Zugehörigkeitsfunktion aus dem Beispiel der Körpergrösse

Diese Zugehörigkeitsfunktion kann als grafisch dargestellte Funktion am besten visualisiert werden. Am Beispiel dieser Körpergrösse kann man die Tabelle 2 auch als drei Teilfunktionen der jeweiligen Terme darstellen. Auf der Abbildung wird gezeigt, welcher Term in der Beispielfunktion je nach Grösse welchen Wert besitzt und die Grafik kann abgelesen werden: Ein Mann, der 180cm misst, gehört in diesem Beispiel nicht mehr zur Klasse „klein“, mit 0.5 Zugehörigkeitsanteil zur Klasse „mittel“ und mit 0.5 Zugehörigkeitsanteil zur Klasse „gross“.

Bei der Erstellung solcher Funktionen besteht eine grosse Freiheit, denn eine Variable muss nicht nur aus linearen Termen bestehen und sie kann auch aus mehr als drei verschiedenen Termen bestehen. Ein Term, der nicht in der Randgruppe ist (in diesem Beispiel nur der „mittel“ Term), muss auch nicht nur aus einer einzelnen „Spitze“ bestehen, sondern kann beispielsweise auch trapezförmig sein, will heissen, dass es im

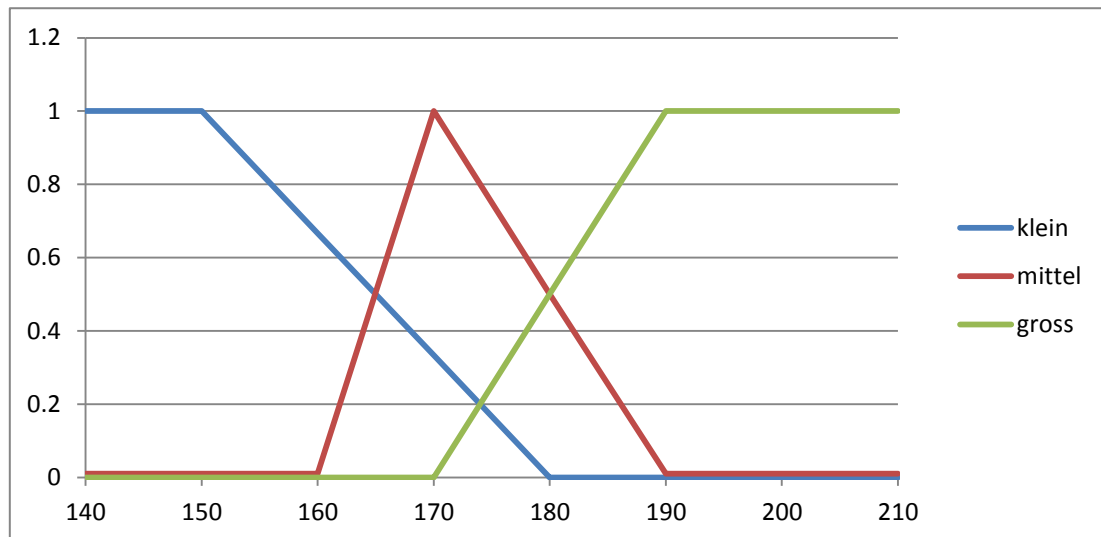


Abbildung 6: Grafische Darstellung der Beispielfunktion zur Körpergrösse

Beispiel auf Abbildung 6 auch möglich gewesen wäre, dem Term „mittel“ nicht nur bei 170cm volle Zugehörigkeit zuzuordnen, sondern beispielsweise von 170cm bis 180cm gilt alles als mittelgross.

Fuzzy Sets können also sehr unterschiedlich sein. Eine zugehörige Funktion kann jegliche Formen annehmen, sofern dies in jedem Fall sinnvoll erscheint und ein Fuzzy Set mit der jeweiligen Funktion der unscharfen Logik am besten dargestellt werden kann.

3.3 Vergleich unscharfer Logik und scharfer Logik

Ein Vergleich von unscharfer Logik mit scharfer Logik ist vor allem ein Vergleich von Mensch und Maschine, genauer Computer. Ein Computer kann nur Befehle ausführen, die ihm exakt aufgetragen werden. Variablen in Programmen werden scharf deklariert, Berechnungen laufen exakt ab.

Aber für Berechnungen mit unscharfen Eingaben, ist der Computer nicht gewappnet. Deshalb muss er für solche Berechnungen die Funktionen der linguistischen Variablen kennen. Erst dann kann ein Computer wieder seine üblichen exakten Berechnungen durchführen, jetzt mit den fuzzyfizierten Daten, und für den Nutzer eine natürliche Ausgabe erzeugen, welche dieser weiterverwenden kann.

Die scharfe Logik ist im Gegensatz dazu zu verstehen wie die normale Mengenlehre, der Grundlage jeglicher Mathematik. Entweder gehört ein Objekt in eine Menge, oder es gehört nicht dazu. Darin besteht der Unterschied zur unscharfen Logik. Die scharfe Logik hat klare und feste Grenzen zwischen dabei und nicht dabei: Es gibt volle Zugehörigkeit und keine Zugehörigkeit, welche gleichzeitig die beiden Extrema bilden in der unscharfen Logik.

3.4 Anwendungsfelder

Die unscharfe Logik ist laut (Ross, 2010) in folgenden beiden Situationen sehr nützlich:

- Wenn in komplexen Systeme bestimmte Verhaltensweisen auftreten, die nicht wohlverstanden werden.
- Wenn eine annähernde aber schnelle Lösung gewählt wird.

Zur Behandlung von konkreten Fällen werden die Berechnungen eines Computers gebraucht. Ein Computer arbeitet mit scharfer Logik: 1 oder 0, Strom oder kein Strom, übertrieben dargestellt. Um einem Computer und seinen Programmen eine natürlichere „Denkweise“ zu vermitteln, bedarf es der unscharfen Logik. Mit Hilfe dieser unscharfen Logik können vage Aussagen in mathematische Gleichungen umgeformt werden (Portmann, et al., 2010). Die Programme können so, trotz ihrer Scharfe-Logik-Berechnungen, Resultate ausgeben, die für den Endnutzer viel natürlicher und weniger als eine scharfe, exakte Ausgabe wirken.

Dies kann in verschiedenen Disziplinen mit menschlicher Komponente, wie in der Medizin oder in der Biologie, zum Beispiel in der Bioinformatik, eine enorme Menge an neuen Möglichkeiten generieren. Diese menschlichen Komponenten können auch aus sozialen, wirtschaftlichen oder politischen Systemen stammen (Ross, 2010).

Es wird dabei angewendet, was bis hierhin erklärt wurde: Die Funktionen der Fuzzy Sets werden vom Benutzer definiert, damit der Computer von diesem zusätzlichen Wissen Gebrauch machen kann und so wie bereits erwähnt die natürlichen Ausgaben berechnen und erzeugen kann.

4 Clustering

Als Clustering bezeichnet man einen Vorgang in der Informatik, bei welchem man eine bestimmte Datenmenge der Ähnlichkeit von Eigenschaften in verschiedene Gruppen unterteilt. Ähnlichkeit ist in diesem Fall eine mathematische Ähnlichkeit in einem wohldefinierten Sinn (Balasko, et al., 2005). Solche Gruppen nennt man Cluster. In Datenmengen, welche beispielsweise Punkte in einem Koordinatensystem darstellen, ist die bestimmende Eigenschaft die Nähe zu den anderen Punkten. Punkte, die nahe beieinander liegen, bilden ein Cluster (Ross, 2010). Ein gutes Cluster besitzt die beiden Eigenschaften, eine hohe Ähnlichkeit der Punkte im selben Cluster und eine niedrige Ähnlichkeit der Punkte im Cluster mit Punkten aus anderen Clustern zu haben (Miyamoto, et al., 2008).

In der Praxis können solche Cluster für verschiedene Zwecke gebraucht werden und verschiedene Formen annehmen. Ein Cluster ist zum Beispiel ein Schulkreis. Obwohl es da nicht nur auf die Distanz ankommt, kann dennoch die Form eines Clusters erkannt werden. Politische Grenzen sind manchmal für die Schulkreise nicht sehr sinnvoll, und so gibt es neben politischen Gemeinden, Bezirken oder Stadtkreisen auch davon grösstenteils unabhängige Schulkreise.

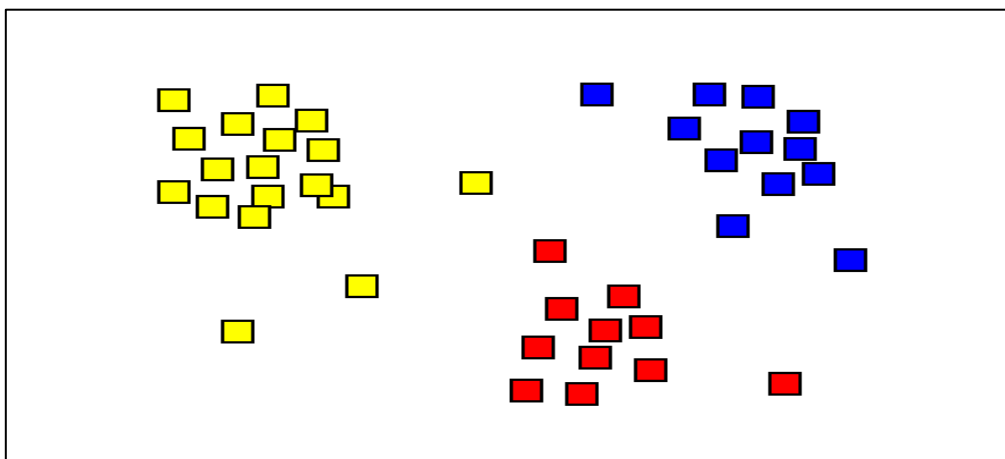


Abbildung 7: Einfaches Beispiel dreier Cluster nach einem Clustering (Wikipedia).

Clustering-Algorithmen werden überall eingesetzt, wo immer man Daten gruppieren möchte. Gerade in der Statistik und in neuronalen Netzwerken wurde schon früh versucht, Daten zu klassifizieren (Miyamoto, et al., 2008). Clustering-Analyse wird auch vermehrt gebraucht, um Informationen in der Genforschung zu gewinnen (Fu, et al., 2007).

Um diese Cluster zu bilden, wendet man Clustering-Algorithmen auf die Datenmengen an. Es gibt viele verschiedene Algorithmen, die mit der Zeit erfunden und erforscht wurden. Eine wichtige Eigenschaft dieser Algorithmen ist dabei, ob sie scharf (crisp oder hard Clustering) oder unscharf (fuzzy Clustering) sind.

4.1 Hard Clustering

Ein Clustering, bei dem die Datenpunkte entweder zum einen oder zum anderen Cluster gehören, nennt man ein hard Clustering. Es gibt dabei, wie auf Abbildung 7 zu sehen, keine Möglichkeiten für einen Datenpunkt, zu mehr als zu einem Cluster zu gehören. Die wenigen Punkte zwischen gelb und rot wurden zwar zu gelb gruppiert, aber hätten gerade so gut rot sein können, sie mussten aber entweder gelb oder rot gruppiert werden, und hätten nicht halb rot und halb gelb oder folgerichtig orange gefärbt werden können.

Das hard Clustering dient also dazu, feste Gruppierungen zu erschaffen. Ein hard-Clustering-Algorithmus hat dementsprechend die Eigenschaft, jeden Punkt exakt einem Cluster hinzuzufügen.

4.2 Fuzzy Clustering

Da Datenpunkte manchmal sowohl zu einer wie auch zu einer anderen Gruppe gehören, gibt es auch unscharfe Clustering-Algorithmen (fuzzy Clustering). Mit Hilfe der unscharfen Logik können fuzzy Cluster berechnet werden.

Fuzzy Cluster unterscheiden sich von harten Clustern darin, dass Datenpunkte zu mehreren unscharfen Clustern gehören dürfen. Dies führt dazu, dass all die erwähnten Möglichkeiten und Vorteile der unscharfen Logik auch auf die Cluster angewendet werden können.

Fuzzy Clustering ist die Methode, womit die in Kapitel 1.1.2 erwähnten Wissensrelationen bestimmt werden können. Mit fuzzy Clustering werden die gecrawlten Daten zu unscharfen Clustern berechnet und danach als Wissensrelationen ausgegeben.

5 Vergleich dreier Clustering-Algorithmen

Um den „attraktivsten“ Clustering Algorithmus zu erküren, braucht es einen Vergleich verschiedener Algorithmen. Es gibt einige bekannte Algorithmen, die ein solches Clustering durchführen. Gesucht sind unscharfe Clustering-Algorithmen, also nicht jegliche Clustering Algorithmen, wenn es auch zu einigen eine unscharfe Version gibt. Verglichen werden in der Folge drei verschiedene Algorithmen. Bevor diese aber einander gegenüber gestellt werden, wird zuerst jeder einzeln vorgestellt, kurz beschrieben und die Kriterien für einen Vergleich festgelegt.

5.1 Beschreiben der drei Algorithmen

Der älteste und einfachste der hier vorgestellten Algorithmen ist der Fuzzy C-Means, dieser stammt aus den 80er Jahren. Der Zweite ist der Gustafson-Kessel-Algorithmus, eine Erweiterung des FCM. Der neuste hier vorgestellte Clustering Algorithmus ist der FLAME (Fuzzy clustering by Local Approximation of MEmbership) Algorithmus.

Um solche Algorithmen auf Daten anwenden zu können, benötigt man erst einmal diese Daten in normalisierter Form. Um die Distanzen einzelner Einheiten von anderen zu berechnen, gibt es verschiedene Metriken (Portmann, et al., 2010). Erst wenn man alle Daten in der Form hat, dass jeder seine Nachbarn und die Distanz zu ihnen kennt, ist die Datenmenge bereit zum Clustering. Um das Ganze etwas besser vorstellbar zu machen, werden die Daten als Punkte in einem n-dimensionalen Raum bezeichnet.

5.1.1 Erklärung der Auswahl

Der Fuzzy C-Means-Algorithmus ist einer der ersten unscharfen Clustering Algorithmen und sehr einfach zu verstehen. Er bietet eine gute Grundlage für Vergleiche und an ihm muss sich erst einmal jeder Algorithmus messen. Aus diesen Gründen wurde der FCM in die Auswahl der drei zu vergleichenden Algorithmen aufgenommen.

Der FLAME Algorithmus ist konträr zu FCM, da er eher neu und nicht nur mit geometrischen Daten ausführbar ist. Er arbeitet mit Ähnlichkeiten zu seinen Nachbarn.

Als dritter Algorithmus wurde Gustafson-Kessel gewählt. Gustafson-Kessel soll eine Zwischenform darstellen, zwar nicht ganz neu, aber dennoch ist er eine Erweiterung des Fuzzy C-Means Algorithmus.

Diese drei Algorithmen wurden aus einer Auswahl von zahlreichen unscharfen Clustering-Algorithmen ausgewählt und stellen einen guten Mix dar, der von jung bis alt und von grundlegend bis innovativ reicht. Es gäbe hier weitere Algorithmen, die verglichen werden könnten, (Miyamoto, et al., 2008), (Balasko, et al., 2005) und (Fu, et al., 2007) stellen eine ganze Liste davon dar.

5.1.2 Fuzzy C-Means clustering algorithm (FCM)

Fuzzy C-Means ist eine unscharfe Version des ursprünglichen C-Means (dieser wird manchmal auch K-Means genannt, hier wird C-Means und vor allem die Abkürzung FCM für Fuzzy C-Means verwendet). Um FCM zu verstehen, muss erst der normale C-Means Algorithmus verstanden werden.

1. In C-Means wird die Anzahl Clusterzentren manuell gewählt (Faustregel: $c \approx \sqrt{n/2}$, c =Anzahl Zentren, n =Anzahl Daten), dann werden diese zufällig in der Datenmenge (im m -dimensionalen Koordinatensystem) verteilt.
2. Nun wird mit derselben Metrik wie vorher die jeweilige Distanz jedes Punktes zu den Zentren berechnet. Jeder Punkt wird nun dem nächstgelegenen Zentrum zugeordnet. Jede dieser Punktmengen stellt nun ein Cluster dar.
3. Die Zentrumsunkte der einzelnen Cluster werden nun versetzt, indem man das Zentrum des gerade entstandenen Clusters neu berechnet.
4. Schritte 2 und 3 werden so lange wiederholt, bis sich die Cluster nicht mehr verändern oder eine vorgegebene Anzahl Iterationen durchgeführt wurde.

Um nun diesen Algorithmus für unscharfe Datensätze anzuwenden, wird er folgenderweise verändert:

1. In FCM wird ebenfalls die Anzahl Clusterzentren manuell bestimmt.
2. Jedem Punkt werden die Angehörigkeitskoeffizienten zu jedem Cluster zufällig hinzugefügt. Es gilt jedoch folgende Regeln zu beachten:
 - a Die Summe aller Zugehörigkeitskoeffizienten eines Punktes ist genau 1 (=100%).
 - b Die Summe aller Zugehörigkeitskoeffizienten eines Clusters ist grösser als 0 (kein Cluster ist leer).
3. Folgende beide Punkte werden wiederholt iteriert, bis der Algorithmus konvergiert:
 - a Die Clusterzentren werden berechnet.
 - b Nun werden für jeden Punkt die Zugehörigkeitskoeffizienten zu jedem Cluster berechnet.

5.1.3 Pseudocode FCM

1. *Bestimmung der Anzahl Cluster: $c \approx \sqrt{n/2}$*
2. *Jedem Punkt werden die Angehörigkeitskoeffizienten zu jedem Cluster zufällig hinzugefügt. Es gilt jedoch folgende Regeln zu beachten:*

- a. *Die Summe aller Zugehörigkeitskoeffizienten eines Punktes ist genau 1 (=100%)*

$$\forall i \left(\sum_{l=1}^c u_l(x_i) = 1 \right)$$

wobei x ein Datenpunkt und $u_l(x)$ der Zugehörigkeitskoeffizient zu Cluster l .

- b. *Die Summe aller Zugehörigkeitskoeffizienten eines Clusters ist grösser als 0 (kein Cluster ist leer).*

$$\forall v \left(\sum_{i=1}^n u_v(x_i) > 0 \right)$$

wobei v ein Clusterzentrum ist.

3. *Folgende beide Punkte werden wiederholt iteriert, bis der Algorithmus konvergiert, also folgende Abbruchbedingung erfüllt wird:*

$$\|U^r - U^{r-1}\| < \varepsilon$$

wobei ε ein frei wählbarer Koeffizient ist und U^r die Zugehörigkeitsmatrix nach r Durchgängen ist.

a. Die Clusterzentren werden berechnet:

$$v_l = \frac{\sum_i u_l(x_i)^m x_i}{\sum_i u_l(x_i)^m} \quad \forall l$$

wobei $m(\geq 1)$ der „fuzzifizier-Faktor“ ist.

b. Nun werden für jeden Punkt die Zugehörigkeitskoeffizienten zu jedem Cluster berechnet.

$$u_l(x_i) = \frac{1}{\sum_j \left(\frac{d(v_l, x_i)}{d(v_j, x_i)} \right)^{2/(m-1)}} \quad \forall l$$

wobei $d(v_j, x_i)$ die Distanz in der gewünschten meist euklidischen Metrik ist. Ist $m=2$, so kommt das der linearen Normalisierung gleich.

5.1.4 Gustafson-Kessel (GK)

Der Gustafson-Kessel Algorithmus ist dem FCM sehr ähnlich. Die Initialisierung des Ganzen geschieht gleich wie beim FCM:

1. In FCM wird ebenfalls die Anzahl Clusterzentren manuell bestimmt ($c \approx \sqrt{n/2}$).
2. Jedem Punkt werden die Angehörigkeitskoeffizienten zu jedem Cluster zufällig hinzugefügt. Es gilt jedoch folgende Regeln zu beachten:
 - a Die Summe aller Zugehörigkeitskoeffizienten eines Punktes ist genau 1 (=100%).
 - b Die Summe aller Zugehörigkeitskoeffizienten eines Clusters ist grösser als 0 (kein Cluster ist leer).
 - c Die Clusterzentren werden erstmals berechnet.
3. Folgende beide Punkte werden wiederholt iteriert, bis der Algorithmus konvergiert
 - a Nun wird anders als beim FCM eine Norm-bestimmende-Matrix für jedes Cluster berechnet, damit Cluster von verschiedenen

geometrischen Formen möglich sind, die den Punkten besser entsprechen.

- b Mit Hilfe dieser geometrischen Formen, welche durch die Normbestimmenden-Matrizen bestimmt werden, werden die neuen Clusterzentren berechnet. Dazu wird nicht die übliche euklidische Distanz verwendet, sondern die Mahalanobis-Distanz.

5.1.5 Pseudocode GK

1. In GK wird die Anzahl Clusterzentren manuell bestimmt ($c \approx \sqrt{n/2}$).
2. Jedem Punkt werden die Angehörigkeitskoeffizienten zu jedem Cluster zufällig hinzugefügt. Es gilt jedoch folgende Regeln zu beachten:

- a. Die Summe aller Zugehörigkeitskoeffizienten eines Punktes ist genau 1 (=100%).

$$\forall i \left(\sum_{l=1}^c u_l(x_i) = 1 \right)$$

- b. Die Summe aller Zugehörigkeitskoeffizienten eines Clusters ist grösser als 0 (kein Cluster ist leer).

$$\forall v \left(\sum_{i=1}^n u_v(x_i) > 0 \right)$$

3. Folgende beide Punkte werden wiederholt iteriert, bis der Algorithmus konvergiert, also folgende Abbruchbedingung erfüllt wird:

$$\|U^r - U^{r-1}\| < \varepsilon$$

- a. Die Clusterzentren werden berechnet:

$$v_l = \frac{\sum_i u_l(x_i)^m x_i}{\sum_i u_l(x_i)^m} \quad \forall j$$

- b. Nun wird eine Norm-bestimmende Matrix (Kovarianzmatrix) berechnet.

$$\Sigma_l = \frac{\Sigma_l^*}{\sqrt{\det(\Sigma_l^*)}}, \quad \forall i \in \text{Clusterzentrum}$$

wobei

$$\sum_l^* = \frac{\sum_{j=1}^n u_l(x_j)(x_j - c_l)(x_j - c_l)^T}{\sum_{j=1}^n u_l(x_j)}$$

- c. Nun werden für jeden Punkt die Zugehörigkeitskoeffizienten zu jedem Cluster berechnet.

$$u_l(x_i) = \frac{1}{\sum_j \left(\frac{d(v_l, x_i)}{d(v_j, x_i)} \right)^{2/(m-1)}} \quad \forall l$$

wobei hier die Mahalanobis-Distanz angewendet wird für $d(v_l, x_i)$.

$$d^2(x_j, v_l) = (x_j - v_l)^T \sum_l^{-1} (x_j - v_l)$$

5.1.6 Fuzzy clustering by Local Approximation of MEMbership (FLAME)

Der FLAME Algorithmus ist ein eher neuer Clustering-Algorithmus, entwickelt von (Fu, et al., 2007). Der Algorithmus basiert auf einer Annäherung der jeweiligen Nachbarn und funktioniert folgendermassen:

1. Als Initialisierung müssen die Cluster Supporting Objects (CSOs) gefunden werden.
 - a Jeder Punkt wird mit seinen k-nächsten Nachbarn (KNN) verbunden, wobei k eine wählbare Konstante ist.
 - b Mit der Summe der Distanzen dieser Verbindungen, wird die Dichte jedes Punkts berechnet.
 - c Jeder Punkt, dessen Dichte grösser ist als diejenige aller seiner Nachbarn, wird als CSO bezeichnet. Outliers (Aussenseiter) werden ebenfalls bestimmt, sie besitzen eine kleinere Dichte als ein vorgegebener Grenzwert.
 - d Es bestehen nun c Cluster und 1 Outlier Gruppe.
 - e Den Outliers wird volle Zugehörigkeit zur Outlier Gruppe angegeben.
 - f Den CSOs wird volle Zugehörigkeit zu ihrem eigenen Cluster angegeben.
 - g Jedem übrig gebliebenen Punkt wird nun zu jedem Cluster und der Outlier Gruppe derselbe Zugehörigkeitskoeffizient hinzugefügt.

2. Nun beginnt der iterative Teil des Algorithmus, welcher die Zugehörigkeitskoeffizienten der Punkte aus Schritt 1. g berechnet. Dies geschieht anhand einer linearen Kombination der Zugehörigkeitskoeffizienten der nächsten Nachbarn jedes Punktes, gewichtet mit der Distanz zum jeweiligen Nachbarn (je näher, umso stärkere Gewichtung).
3. Schritt 2 wird solange iteriert, bis das Ganze konvergiert.

5.1.7 Pseudocode FLAME

1. Als Initialisierung müssen die Cluster Supporting Objects (CSOs) gefunden werden.
 - a. Jeder Punkt wird mit seinen k -nächsten Nachbarn (KNN) verbunden, wobei k eine wählbare Konstante ist.

$$\forall i, j \begin{cases} a_{i,j} = 1, \text{ wenn } x_j \text{ zu den } k \text{ nächsten Nachbarn von } x_i \text{ gehört} \\ a_{i,j} = 0, \text{ sonst} \end{cases}$$
 wobei $a_{i,j}$ der Nachbarschaftsstatus von x_j zu x_i ist, 1 wenn Nachbar, sonst 0.
 - b. Mit der Summe der Distanzen dieser Verbindungen, wird die Dichte jedes Punkts berechnet.

$$\rho_i = \frac{1}{d_i} = \frac{1}{\sqrt{\sum_{j=1}^n a_{i,j} (x_i - x_j)^2}} \quad \forall x$$

wobei ρ_i die Dichte des Datenpunkts x_i ist und das Inverse der Distanz d_i vom Datenpunkt x_i zu seinen k -nächsten Nachbarn ist, also sind Punkte mit kurzen Verbindungen als dichter zu betrachten als solche mit längeren Verbindungen. (Die Berechnung kann auch mit anderer Metrik durchgeführt werden.)

- c. Jeder Punkt, dessen Dichte grösser ist als diejenige aller seiner Nachbarn, wird als CSO bezeichnet. Outliers (Aussenseiter) werden ebenfalls bestimmt, sie besitzen eine kleinere Dichte als ein vorgegebener Grenzwert.
- d. Es bestehen nun c Cluster und 1 Outlier Gruppe.

e. Den Outliers wird volle Zugehörigkeit zur Outlier Gruppe angegeben.

$$\forall i \in \text{Outlierpunkte}: u_{c+1}(x_i) = 1$$

$$\forall i, l \leq c: u_l(x_i) = 0$$

f. Den CSOs wird volle Zugehörigkeit zu ihrem eigenen Cluster angegeben.

$$\forall i \in \text{CSO}: u_{l_i}(x_i) = 1$$

$$\forall i \neq j: u_{l_j}(x_i) = 0, j \neq i$$

wobei l_i das Cluster mit i als CSO ist.

g. Jedem übrig gebliebenen Punkt wird nun zu jedem Cluster und der Outlier Gruppe derselbe Zugehörigkeitskoeffizient hinzugefügt.

$$\forall x \notin (\text{Outlierpunkte} \vee \text{CSO}), \forall l: u_l(x) = \frac{1}{c+1}$$

h. Es gelten auch in Flame folgende Gesetze, welche mit den vorangehenden Formeln eingehalten werden:

Die Summe aller Zugehörigkeitskoeffizienten eines Punktes ist genau 1 (=100%)

$$\forall i \left(\sum_{l=1}^c u_l(x_i) = 1 \right)$$

und die Summe aller Zugehörigkeitskoeffizienten eines Clusters ist grösser als 0 (kein Cluster ist leer).

$$\forall l \left(\sum_{i=1}^n u_l(x_i) > 0 \right)$$

i. Um die Iteration durchführen zu können, müssen erst noch die Gewichtungen jedes k nächsten Nachbars bestimmt werden, mit der dieser auf den jeweiligen Punkt x einwirkt:

Die Summe aller Gewichtungen der KNN eines Punktes beträgt 1.

$$\sum_{x_j \in \text{KNN}(x_i)} w_{ij} = 1$$

wobei w_{ij} die Gewichtung von j auf i ist.

Die einzelnen Gewichtungen werden durch die Ähnlichkeitsfaktoren s_{ij} berechnet. Damit die obere Gleichung stimmt, müssen diese normalisiert werden.

$$w_{ij} = \frac{s_{ij}}{\sum_{x_m \in KNN(x_i)} s_{im}}$$

wobei s_{ij} der nicht normalisierte Ähnlichkeitsfaktor ist (der egal in welcher Metrik erstellt werden kann, wird bei geometrischen Datenpunkten meist mit dem inversen der Strecken berechnet).

2. Nun beginnt der iterative Teil des Algorithmus, welcher die Zugehörigkeitskoeffizienten der Punkte Schritt für Schritt berechnet. Dies geschieht anhand einer linearen Kombination der Zugehörigkeitskoeffizienten der nächsten Nachbarn jedes Punktes, gewichtet mit den normalisierten Ähnlichkeitsfaktoren.

$$\forall i, l: u_l(x_i) = \sum_{x_j \in KNN(x_i)} w_{ij} \cdot u_l(x_j)$$

3. Schritt 2 wird solange iteriert, bis das Ganze konvergiert, also $E(\{u\})$ kleiner wird als eine vorgegebene Zahl.

$$E(\{u\}) = \sum_{x_i \in X} \left\| u(x_i) - \sum_{x_j \in KNN(x_i)} w_{ij} u(x_j) \right\|^2$$

wobei $u(x_i)$ der Vektor der Zugehörigkeiten ist, also für jedes CSO: $u_l(x_i)$ für $l=1, \dots, c+1$.

5.1.8 Abschluss eines Algorithmus

Nachdem jeder der drei Algorithmen konvergiert, kann die jeweilige Ontologie verarbeitet werden. Es gibt verschiedene Methoden, um die Daten der Ontologie weiter zu verarbeiten und somit die Clusterzugehörigkeit endgültig zu klären:

1. Entweder gehört jeder Punkt zu dem Cluster, zu dem er den grössten Zugehörigkeitskoeffizienten hat.
2. Oder, was bei unscharfer Logik das Ziel ist, jeder Punkt kann zu mehreren Clustern gehören, nämlich zu allen Clustern, zu denen er einen grösseren Zugehörigkeitskoeffizienten hat, als eine vorbestimmte Untergrenze besagt.

3. Oder die Daten werden als Ganzes weiterverarbeitet, das heisst, jede Zugehörigkeit irgendeines Datenpunkts zu einem Cluster wird mitsamt dem Zugehörigkeitskoeffizienten als Ontologie gespeichert.

5.2 Kriterien für einen Vergleich

Damit man diese Algorithmen in einem sinnvollen Verhältnis einander gegenüber stellen kann, braucht es passende Kriterien für verschiedene Vergleiche. Die Algorithmen werden in einem Raster mit diesen Kriterien konfrontiert und dadurch sieht man, welcher Algorithmus welche Vor- und Nachteile hat gegenüber den anderen.

Da die Algorithmen etwas mathematisch Vergleichbares sind, müssen die Kriterien messbar und vor allem beweisbar sein. Benutzerabhängige Kriterien kommen in diesem Fall also nicht zur Geltung. Für jeden Algorithmus gilt es also heraus zu finden, ob er das jeweilige Kriterium erfüllen oder nicht erfüllen kann.

5.2.1 Erklärung der Auswahl

Die Kriterien für die Gegenüberstellung der Algorithmen sind Komplexität, Konstanz/Zufallseinfluss und Anpassungsfähigkeit. Diese Kriterien wurden aufgrund der Analyse der Algorithmen und deren Ausgaben gewählt. Voraussetzung für die Kriterien war es daher, auf die Algorithmen anwendbar zu sein und besonders für den speziellen, in dieser Arbeit angewendeten, Fall der unscharfen Clustering-Algorithmen sinnvoll zu sein. Um solche Kriterien zu finden, galt es, in allen drei Algorithmen nach möglichen vergleichbaren Eigenschaften zu suchen und diese für einen Vergleich zu formulieren.

5.2.2 Komplexität

Es stimmt zwar, dass ein Algorithmus schnell sein muss, aber anstatt schneller als die anderen Algorithmen zu sein, muss vor allem jeder Algorithmus für sich selbst das Kriterium erfüllen, in nutzbarer Zeit durchführbar zu sein.

Da die zu vergleichenden drei Algorithmen von verschiedenen Personen implementiert wurden, macht es in diesem Vergleich keinen Sinn, sich auf Zeittests zu stützen, zu verschieden können die MatLab-Kenntnisse und die Implementier-Details sein.

Vielmehr muss man die Komplexität beachten. Die Komplexität eines Problems darf maximal polynomial sein. Denn wenn ein Problem nicht in Polynomialzeit lösbar ist, gilt es nach (The Intrinsic Computational Difficulty of Functions, 1965) als nicht in nutzbarer Zeit lösbares Problem. Dieses Problem wenden wir nun auf unsere Algorithmen an, welche also dementsprechend maximal von polynomialer Komplexität sein dürfen.

5.2.3 Konstanz/Zufallseinfluss

Ein Algorithmus sollte eine gewisse Konstanz haben. Veränderungen bei Anpassungen von aussen sollten minimalisiert werden, so dass nicht jede kleine Veränderung eine komplett neue Ontologie erstellt. Ausserdem sollte der Zufall eine möglichst kleine Rolle spielen.

Diese beiden Kriterien werden zusammen genommen, weil sich der Zufall in einem Algorithmus auf die Konstanz auswirkt. Denn wenn der Zufall starken Einfluss auf das Resultat hat, so berechnet ein Algorithmus bei jeder Durchführung ein komplett anderes Resultat. Dies sollte nicht der Fall sein, da dies zeigen würde, dass in diesem Fall die Cluster völlig willkürlich generiert würden.

5.2.4 Anpassungsfähigkeit

Ist aber das Resultat nicht von der gewünschten Form oder sonst unbefriedigend, so sollte dieses mit Hilfe von aussen angepasst werden können, indem man eine Konstante verändert, so dass das Resultat verändert wird.

Oft erhält man nach Durchführung eines Programms/Algorithmus ein Resultat, das zwar richtig ist, aber nicht ganz den Erwartungen entspricht. Wenn der Benutzer dann irgendwie eine Veränderung herbeiführen möchte, so sollte ihm dies möglich sein. Denn ist eine Programmausgabe nicht von gewünschter Form, kann der Benutzer diese

möglicherweise nicht weiterverwenden und man muss einen komplett anderen Weg suchen, eine gewünschte Ausgabe zu erhalten.

5.2.5 Erklärung

Zu den Kriterien Konstanz/Zufallseinfluss und Anpassungsfähigkeit muss angemerkt werden, dass bei der Konstanz kleine Veränderungen das Resultat nicht stark verändern sollten, aber falls man eine Änderung möchte oder sogar braucht, dann sollte man diese mittels Anpassung von aussen dennoch herbei führen können.

5.3 Anwendung der Kriterien auf die Algorithmen

Um einen Einblick zu erhalten, werden die Kriterien erst auf jeden Algorithmus angewendet. Erst später werden diese dann einander gegenübergestellt. So erhält man einen ersten Überblick, auf welche Weise sich jeder Algorithmus zu den Kriterien verhält und wo die grössten Vor- und Nachteile liegen.

5.3.1 FCM

- Die höchste Komplexität ist quadratisch, also wird das Kriterium der Komplexität durch den Algorithmus erfüllt.
- Weil die Zugehörigkeiten anfangs zufällig gesetzt werden, können die Cluster bei jeder Durchführung des Algorithmus immer etwas anders aussehen. Ausserdem wird durch den Fakt, dass man beim Start des FCM die Anzahl der Cluster mitgeben muss, logischerweise bei jeder Änderung eine andere Clusterverteilung erstellt. Das Kriterium der Konstanz und des Zufalls ist nicht gut erfüllt.
- Dies kann ein Nach- wie auch ein Vorteil sein, denn durch diese Konstante, die Anzahl der Cluster, kann vor dem Start des Algorithmus schon eine gewünschte Ausgabe angesteuert werden. Das Kriterium der Anpassungsfähigkeit ist erfüllt.

5.3.2 GK

- Auch beim GK ist die höchste Komplexität quadratisch, also wird das Kriterium der Komplexität vom Algorithmus auch erfüllt.
- Das Problem mit der mitgegebenen Clusterzahl bleibt hier dasselbe wie beim FCM. Jedoch werden die Cluster weniger stark verändert, da geometrische

Formen erkannt werden. Das Kriterium der Konstanz und des Zufalls ist nicht gut erfüllt.

- Auch hier gilt dasselbe wie beim FCM, die Wahl der Anzahl Cluster lässt eine gewünschte Anzahl Cluster zu, was aber nicht zwingend ein Vorteil sein muss. Das Kriterium der Anpassungsfähigkeit ist erfüllt.

5.3.3 FLAME

- Auch der Flame hat die höchste Komplexität quadratisch, also wird das Kriterium der Komplexität vom Algorithmus ebenfalls erfüllt.
- Der signifikante Unterschied des FLAME zum FCM und GK liegt in der Veränderung der KNN zum Start des Algorithmus. FLAME erstellt nicht automatisch eine andere Anzahl Cluster. Ausserdem gibt es keine Zufallseinflüsse, jede Durchführung ergibt das exakt gleiche Resultat. Das Kriterium der Konstanz und des Zufalls ist erfüllt.
- Erst ab einer grösseren Veränderung der KNN gibt es wieder eine Veränderung der CSOs, der Clusterzentren. Aber dennoch kann ein ungewünschtes Resultat verändert werden. Das Kriterium der Anpassungsfähigkeit ist erfüllt.

5.4 Vergleich

Die Priorität der Kriterien ist wie folgt zu bewerten: Die Komplexität darf maximal polynomial sein, sonst scheidet der Algorithmus automatisch aus. Ein Algorithmus, der nicht in, wie bereits erwähnt, nutzbarer Zeit durchführbar ist, kann nicht ausgewählt werden.

Nach dem Hauptkriterium der Komplexität bleiben die beiden anderen Kriterien, die sich von ihrer Wichtigkeit nicht unterscheiden. Beide sind für ein erfolgreiches unscharfes Clustering eines Datensatzes von Bedeutung, keines kann aber das andere wirklich überwiegen.

5.4.1 Vergleichsübersicht

Algorithmus/Kriterium	Komplexität	Konstanz/Zufallseinfluss	Anpassungsfähigkeit
FCM	Polynomial	Kleine Änderungen nehmen starken Einfluss und wenig Zufall ist vorhanden	Anzahl Clusterzentren kann beliebig gewählt werden
GK	Polynomial	Kleine Änderungen nehmen starken Einfluss und wenig Zufall ist vorhanden	Anzahl Clusterzentren kann beliebig gewählt werden
FLAME	Polynomial	Kleine Änderungen nehmen kaum Einfluss und kein Zufall vorhanden	Anzahl Clusterzentren kann verändert aber nicht exakt gewählt werden

Tabelle 3: Vergleichsübersicht der Clusteringalgorithmen

5.4.2 Visueller Vergleich

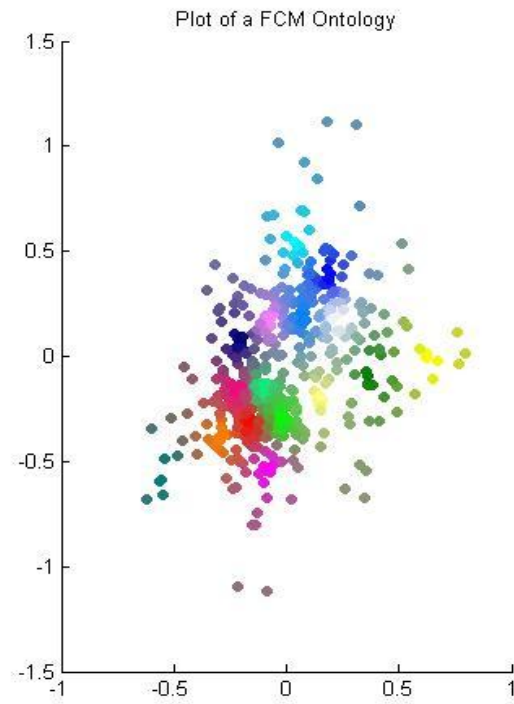


Abbildung 8: Plot einer FCM Ontologie

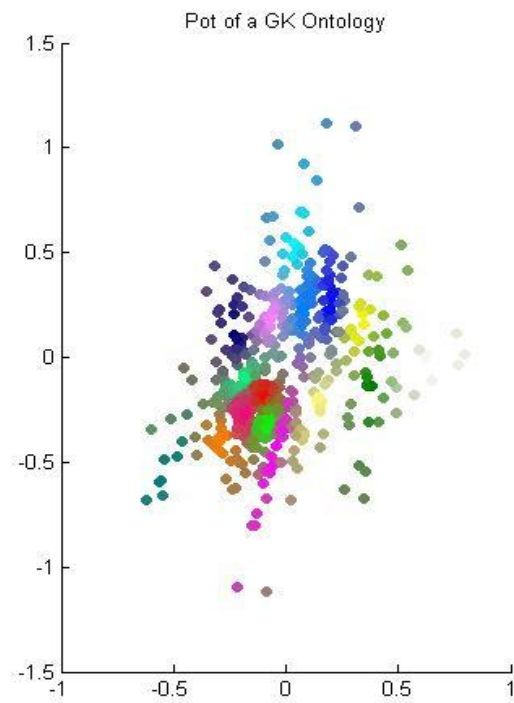


Abbildung 9: Plot einer GK Ontologie

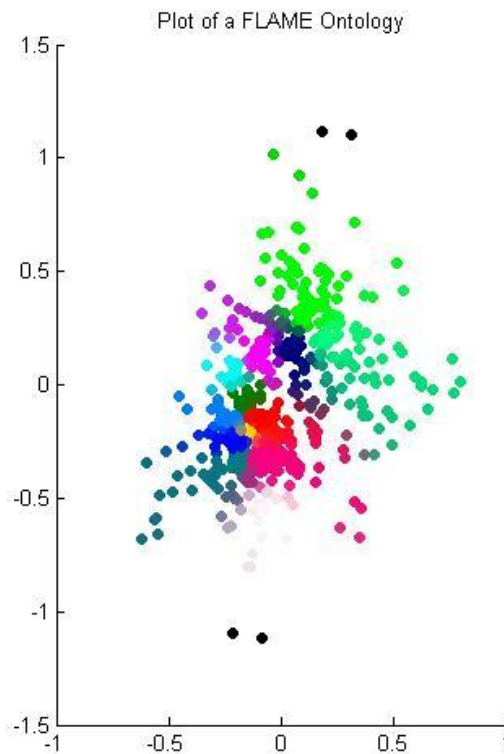


Abbildung 8: Plot einer FLAME Ontologie

5.4.3 Zusammenfassung

Da der FCM der erste unscharfe Clustering-Algorithmus überhaupt ist, ist er nicht sehr fortschrittlich und auf einfachen Prinzipien aufgebaut. So wird er immer noch regelmässig gebraucht, da er den einfachsten Standards genügt und benutzerfreundlich anzuwenden ist.

GK ist eine interessante Erweiterung des FCM, welcher in einigen Fällen sehr sinnvoll sein kann.

FLAME hat einen anderen Ansatz als die anderen beiden Algorithmen und sein grösster Vorteil liegt in der Bestimmung der Anzahl Cluster, die der Algorithmus durchführt und die nicht mitgegeben wird beim Start.

5.4.4 Schlussfolgerung

FLAME ist der „attraktivste“ Algorithmus, da er alle drei Kriterien mehrheitlich erfüllt. Zusätzlich ist die in der Zusammenfassung erwähnte und vom Algorithmus erstellte

Anzahl der Cluster viel sinnvoller, weil so die Clusterzentren auch wirklich zentrale Datenpunkte sind, FCM und GK formen Clusterzentren, weil sie noch zu wenige besitzen und so werden diese Clusterzentren gezwungenermassen erstellt.

Weiter sind die Clusterzentren in FCM und GK keine Punkte aus der ursprünglichen Datenmenge, da sie einfach das geometrische Zentrum der jeweiligen Punkte darstellen. So muss meist der nächste Datenpunkt als Clusterzentrum erhalten. Die Clusterzentren sind dann aber nicht mehr die exakten berechneten Zentren.

5.5 Notiz

Sowohl FCM wie auch GK basieren auf geometrischen Entfernungen, während für FLAME diese Entfernungen umgerechnet werden müssen in Ähnlichkeitsfaktoren. Diese Ähnlichkeitsfaktoren sind genau das, was mit den gecrawlten Daten von delicious hergestellt werden kann. Da die gecrawlten Daten von delicious nicht als geometrische Datenpunkte dargestellt werden können, kommt also nur FLAME in Frage um in diesem Fall eine Ontologie zu erstellen. Natürlich kann man mit der Jaccard-Metrik die Distanz zweier Punkte berechnen, die man nicht geometrisch hat, da FCM und GK aber, wie in der Schlussfolgerung erwähnt, keine im Datensatz vorhandenen Punkte als Clusterzentren haben, reicht die Jaccard-Metrik jedoch nicht aus.

6 Bewertung eines Clusters

Um eine Ontologie vergleichend mit einer anderen bewerten zu können, braucht es einen Index, der für jede auf demselben Datensatz gebildete Ontologie bestimmt werden kann. Es gibt dazu interne, externe und relative Bewertungen (Portmann, 2012), wobei für diese Ontologien hier nur die internen Bewertungen betrachtet werden, da die anderen beiden aus verschiedenen Gründen nicht hierhin passen.

6.1 Interne Bewertungen

Die beiden internen Bewertungen, welche von (Portmann, 2012) erwähnt werden, sind der Davies-Bouldin Index und der Dunn-Index. Da dies beides Indizes für hard-clustering Algorithmen sind, muss einer davon an ein unscharf gebildetes Cluster

angepasst werden. Wenn man die beiden Algorithmen analysiert, fällt als erstes auf, dass der Dunn-Index den Durchmesser des jeweiligen Clusters benutzt, was nur sehr schwer und umständlich für ein unscharfes Cluster angepasst werden könnte.

Der Davies-Bouldin Index hingegen basiert nur auf Distanzen:

$$\text{Davies - Bouldin Index: } DBI = \frac{1}{c} \sum_{i=1}^c \max_{i \neq j} \left(\frac{\sigma_i + \sigma_j}{d(v_i, v_j)} \right) \quad \forall j$$

wobei c die Anzahl Cluster ist, v_i das Clusterzentrum des Clusters i , σ_i die durchschnittliche Distanz der Datenpunkte x in v_i zum Zentrum des Clusters v_i und $d(v_i, v_j)$ ist die Distanz zwischen den beiden Clusterzentren v_i und v_j . Je kleiner der Index wird, umso besser wurden die Cluster erstellt und führten somit zu einer besseren Ontologie. So können die anpassbaren Werte bei der Ausführung des Algorithmus beispielsweise so angepasst werden, dass der FDBI kleiner wird.

Dieser Index ist, wie andere interne Bewertungen auch, nur für den Vergleich verschiedener Ontologien basierend auf demselben Datensatz sinnvoll, da ein Datensatz bestehend aus weit voneinander entfernten Datenpunktgruppen im Normalfall immer einen tieferen Index haben wird als ein dichter Datensatz, der alle Datenpunkte nahe beieinander hat.

6.1.1 Interne Bewertungen für Fuzzy Clustering

Um diesen Index anzupassen, stellt man sich die Daten des hard Clustering in einer fuzzy Version bereit: Jeder Punkt hat einfach volle Zugehörigkeit zu seinem Cluster $u_i(x_a)=1$ für das Cluster i in dem sich der Punkt befindet und 0 für die anderen. So wird also die Formel folgendermassen geschrieben, ohne Ergebnisveränderung:

$$\text{Fuzzy Davies - Bouldin Index: } FDBI = \frac{1}{c} \sum_{i=1}^c \max_{i \neq j} \left(\frac{\sigma_i + \sigma_j}{d(v_i, v_j)} \right) \quad \forall j$$

$$\text{mit } \sigma_i = \frac{1}{n} \sum_{a=1}^n d(x_a, v_i) * u_i(x_a) \quad \forall i$$

was bei hard Clustering keine Veränderung des Ergebnisses herbeiführt, die Formel aber für fuzzy Clustering anwendbar macht.

6.1.2 Vergleich der Fuzzy Davies-Bouldin Indizes der Ontologien

Um jetzt die Indizes auf die Ontologien der verschiedenen Algorithmen anzuwenden, wurde mit einem zufällig erstellten Datensatz M (zweidimensional, 500 Datenpunkte) mit jedem verglichenen Algorithmus eine Ontologie erstellt. Für diese kann der Fuzzy Davies-Bouldin Index berechnet werden. Die folgende Tabelle zeigt die Resultate aus jeweils 5 Berechnungen des FDBI:

FCM/16 Zentren	GK/16 Zentren	FLAME/KNN=10/20 Zentren
0.3213	1.2314	0.1794
0.8832	0.7672	0.1794
0.5475	0.9039	0.1794
0.9304	1.5314	0.1794
1.2964	1.0266	0.1794
0.7958	1.0928	0.1794

Tabelle 4: Berechnungen des Fuzzy Davies-Bouldin Indizes

Wie im Vergleich bereits erwähnt wird, wird die Ontologie mit FLAME nicht verändert, so bleibt auch der FDBI konstant derselbe.

6.2 Informationen zum DBI

Wie man aus der Formel des DBI entnehmen kann, basiert die ganze Berechnung dieses Indexes auf direkten Distanzen basierend. Dass GK durchschnittlich einen höheren Index als FCM hat, ist daher nachvollziehbar. Aber der FDBI zeigt sehr gut, dass FCM und GK durch die zufällige Verteilung bei der Initialisierung, sehr inkonstant sind.

Dass aber FLAME durchschnittlich so viel besser ist, wirkt hingegen auf den ersten Blick äusserst überraschend. Sieht man sich aber die Formel genauer an, so ist es damit zu erklären, dass FCM die Zentren im Normalfall in einem dichten Datensatz (siehe z.B. die Abbildungen 8-10) regelmässig verteilt, FCM aber nur an den dichtesten Stellen ein

Zentrum stellt. So kann es öfter vorkommen, dass die Zentren weiter voneinander entfernt sind, was automatisch zu einem tieferen FDBI führt.

Der Hauptpunkt für diesen besseren FDBI bei FLAME dürften die Outlier sein. Auch wenn es nicht viele sind, schrauben genau diese die Durchschnittsdistanz σ_i in FCM und GK in die Höhe, was sie in FLAME nicht tun können, da sie ein eigenes Cluster bilden.

6.3 Erklärung

Diese interne Bewertung wurde nicht als Kriterium in den Vergleich aufgenommen, da er, wie bereits erwähnt, nur von Distanzen abhängig ist. Die verglichenen Algorithmen erstellen ihre Cluster und somit Ontologien aber nicht nur aufgrund von Distanzen. Ausserdem hat trotzdem der „attraktivste“ Algorithmus den niedrigsten FDBI, was die Wahl des FLAME Algorithmus nachträglich stärkt.

6.4 Berechnung des FDBI für die erstellte Ontologie

Wie nach dem Vergleich der Algorithmen erwähnt, kann die gewünschte Ontologie mit den gecrawlten Daten aus delicious nur mit FLAME berechnet werden. Da aber ein Vergleich somit nicht möglich sein wird, steht der FDBI der erstellten Ontologie zu den Daten aus delicious ohne Gegenwert leer auf dem Papier: 0.5048.

7 Schlussteil

Im Schlussteil wird diese Arbeit zusammengefasst, damit danach ein Fazit gezogen werden kann. Zuletzt wird ein kurzer Ausblick auf mögliche zukünftige Forschungen mit ähnlichen Themen gewährt.

7.1 Zusammenfassung

Das Social Semantic Web besteht sowohl aus Social Web wie auch aus Semantic Web. Das Ziel des Social Semantic Web ist es, diese beiden Webs zu vereinen und eine Wissensbasis zu schaffen, die von einer Menschenmenge erstellt wurde und nun von Computern interpretiert werden kann.

Mit Hilfe der unscharfen Logik können Aussagen natürlicher gemacht werden. Damit Computer Aussagen auch natürlicher anwenden und ausgeben können, braucht es die

mathematische Formulierung der linguistischen Variablen mit unscharfer Logik. Der Computer rechnet dann mit diesen unscharfen Eingaben und gibt die Resultate dementsprechend unscharf aus. Diese natürlichen Ausgaben sind für den Endnutzer im Zusammenhang mit umgangssprachlichen Aussagen und Formulierungen natürlicher und angepasster, als wenn diese Berechnungen mit scharfer Logik getätigt worden wären.

Um Daten zu gruppieren, können Cluster genutzt werden. Um Cluster zu berechnen, gibt es verschiedene Algorithmen. Einige davon sind unscharfe Clustering-Algorithmen und können so die angesprochenen natürlicheren Ausgaben erzeugen, da sie mit unscharfer Logik berechnet werden.

Im erwähnten Prototypen stellt die Berechnung der Cluster und somit der Ontologie den mathematischen Teil dar: Die gecrawlten Begriffe werden mathematisch zu unscharfen Clustern berechnet und weiter verarbeitet, so dass die Relationen zwischen den verschiedenen Begriffen dargestellt werden können. Es wird daraus visuell ersichtlich, welcher Begriff mit welchen Begriffen stärker oder weniger stark verwandt ist. Dies dient einem Computer, damit er erkennt, welche Begriffe beispielsweise bei einer Suche ebenfalls gesucht werden können, da er Synonyme und verwandte Begriffe einordnen kann.

Als Fazit mit Bezug des Hauptteils auf die theoretischen Aspekte dieser Arbeit kann gesagt werden, dass die verschiedenen Clustering-Algorithmen für verschiedene Datensätze geeignet sein können. FLAME hat, gerade im behandelten Fall mit Verwandtschaften von Begriffen, den Vorteil, dass er mit Ähnlichkeitsfaktoren arbeitet und so der Problemstellung schon von Natur aus näher ist, als andere Clustering-Algorithmen.

7.2 Ausblick

In Zukunft werden Computer durch die Hilfe von riesigen Menschenmengen und den richtigen technischen Ansätzen immer intelligenter werden. Das Web enthält bereits heute unzählige Information, die durch Social Web Applikationen geschaffen wurden. Werden diese Informationen in Zukunft alle durch das Semantic Web strukturiert und

für Computer somit nutzbar sein, so wird das Social Semantic Web sehr mächtig und in verschiedenen Teilen der Forschung stark unterstützend werden.

Ob zur Erstellung von Ontologien das Clustering einen Hauptansatz darstellen wird, bleibt zu beobachten. Aber auf jeden Fall können sich die Ontologien so an den wichtigsten Begriffen orientieren.

Es wird aber auf alle Fälle immer wieder neue Technologien, Algorithmen und andere Ideen rund um das Web geben, um dem Computer weiterhin immer mehr Wissen beizubringen. So kann auch der Endnutzer davon profitieren, da dadurch diesem ein intelligenter Computer im Web das Leben ständig weiter erleichtern wird, in welchem Zusammenhang dies auch immer wünschenswert sein wird.

Literaturverzeichnis

Allemang, Dean und Hendler, Jim. 2008. *Semantic Web for the Working Ontologist*. Burlington : Morgan Kaufmann, 2008.

Balasko, Balazs, Abonyi, Janos und Feil, Balazs. 2005. *Fuzzy Clustering and Data Analysis Toolbox*. Veszprém : www.fmt.vein.hu, 2005. Dokumentation.

Berners-Lee, Tim. 2000. W3C: Semantic Web: Architecture. W3C. [Online] 2000. [Zitat vom: 11. November 2011.] <http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html>.

Berners-Lee, Tim, Hendler, James und Lassila, Ora. 2001. The Semantic Web. *Scientific American Magazine*. Mai 2001.

Black, Max. 1937. *Vagueness: An exercise in logical analysis*. 1937.

Blumauer, Andreas und Pellegrini, Tassilo. 2009. *Social Semantic Web: Web 2.0 - Was nun?* Berlin Heidelberg : Springer-Verlag, 2009.

Bradley, Aaron R. und Manna, Zohar. 2007. *The Calculus of Computation*. Stanford : Springer-Verlag, 2007.

de Oliveira, Jose Valente und Pedrycz, Witold. 2007. *Advances in Fuzzy Clustering and its Applications*. Chichester : John Wiley & Sons, 2007.

Fu, Limin und Medico, Enzo. 2007. *FLAME, a novel fuzzy clustering method for the analysis of DNA microarray data*. Candiolo, Italy : BioMed Central, 2007.

Hitzler, Pascal, et al. 2008. *Semantic Web*. Berlin Heidelberg : Springer-Verlag, 2008.

Miyamoto, Sadaaki, Ichihashi, Hidetomo und Honda, Katsuhiko. 2008. *Algorithms for Fuzzy Clustering: Methods in c-Means Clustering with Applications*. Berlin Heidelberg : Springer-Verlag, 2008.

Portmann, Edy und Meier, Andreas. 2010. *A Fuzzy Grassroots Ontology for improving Weblog Extraction*. 2010. S. 276-284.

Ross, Timothy J. 2010. *Fuzzy Logic*. Grossbritannien : John Wiley & Sons, 2010.

Schmidt, Jörg, Klüver, Christina und Klüver, Jürgen. 2010. *Programmierung naturanaloger Verfahren*. Wiesbaden : Vieweg + Teubner Verlag, 2010.

Thatte, Aditya. 2008. Wordpress. [Online] 18. November 2008. [Zitat vom: 2. November 2011.] <http://thesemanticway.wordpress.com/2008/11/16/rdf-schema-rdfs-example/>.

The Intrinsic Computational Difficulty of Functions. **Cobham, Alan. 1965.** Amsterdam : North Holland, 1965. Proceedings of the 1964 Congress for Logic, Mathematics, and Philosophy of Science. S. 24-30.

Ultes-Nitsche, Ulrich. 2010. Web 3.0 - wohin geht es mit dem World Wide Web. [Buchverf.] Urs Hengartner und Andreas Meier. *Web 3.0 & Semantic Web*. Heidelberg : dpunkt.verlag, 2010, S. 6-12.

Wikipedia. Cluster analysis - Wikipedia. *Wikipedia*. [Online] [Zitat vom: 11. 23 2011.] http://en.wikipedia.org/wiki/Cluster_analysis.

Wikipedia. Semantic Web Stack - Wikipedia. *Wikipedia*. [Online] [Zitat vom: 29. November 2011.] http://en.wikipedia.org/wiki/Semantic_Web_Stack.

Zadeh, Lotfi A. 1965. *Fuzzy Sets*. Berkeley : Departement of Electical Engineering and Electronics Research Laboratory, University of California, Berkeley, California, 1965.

Anhang: FLAME in MatLab für youreputation

flame_calc()

```
function [CluMat,turn,TagList,CSOs] =
flame_calc(XDat,Tags,KNN,Turns)
%FLAME_CALC claculates the memberships of the clusters
%
%   written by : Sandro Kolly
%       date : 2011/08/18

[~,N] = size(XDat);

[SimMat,CSOs,CluMat] = flame_init(XDat,KNN,N);

CluNum=numel(CSOs);
EyeMat=eye(CluNum+1);
turn=0;
sumep=zeros(N,1);
for epcso=1:CluNum+1
    for epsim=1:N
        sumep(epsim)=SimMat(:,epcso)'*CluMat(:,epcso);
    end
end

%calculate the Fuzzy Memberships by local Approximation
while turn<Turns && sum(sumep)>0.000001
    turn=turn+1;
    for line=1:N
        if ~ismember(CluMat(line),EyeMat)
            for row=1:CluNum+1
                CluMat(line,row)=SimMat(:,line)'*CluMat(:,row);
            end
        end
    end
    %calculate the E(p) (LAE/NAE)
    for epcso=1:CluNum+1
        for epsim=1:N
            sumep(epsim)=(SimMat(:,epcso)'*CluMat(:,epcso))^2;
        end
    end
end

TagList=cell(numel(CSOs),1);
for i=1:numel(CSOs)
    TagList(i)=Tags(CSOs(i));
end
TagList(numel(CSOs)+1)=cellstr('OutlierGroup');

end
```

flame_init()

```
function [SimMat,CSOs,CluMat] = flame_init(XDat,KNN,N)
%FLAME_INIT claculates all initialisation-work todo
%
%   written by : Sandro Kolly
%       date : 2011/08/18

M=zeros(N);
for l=1:N
    for r=1:N
        if l<r
            M(l,r)=XDat(l,r)/(XDat(l,l)+XDat(r,r));
            M(r,l)=XDat(r,l)/(XDat(l,l)+XDat(r,r));
        end
    end
end

for k=1:N
    M(k,k)=0;
end

% sort matrix descending
SMat=zeros(N);
for im=1:N
    S=sort(M(:,im),'descend');
    SMat(:,im)=S;
end

% take the sum of the KNN
SumList=sum(SMat(1:KNN,:));
AvSumList=SumList./KNN;
% take the longest of the KNN
DList=SMat(KNN,:);

% create adjacency matrix; if 1: adjacent
AdMat=zeros(N);
for ik=1:N
    for in=1:N
        if M(ik,in)>=DList(in) && ik ~= in;
            AdMat(ik,in)=1;
        end
    end
end

%calculate the CSOs
CSO=ones(1,N);
AdSumMat=zeros(N);
for ad=1:N
    AdSumMat(:,ad)=AvSumList'.*AdMat(:,ad);
end
```

```

CSOs=[];
for cs=1:N
    for cso=1:N
        if AvSumList(cs)<AdSumMat(cso,cs) &&
AdSumMat(cso,cs)~=0;
            CSO(cs)=0;
            break;
        end
    end
    if CSO(cs)==1
        CSOs=[CSOs,cs];
    end
end

%calculate the Outliers
%find CSO with smallest density
smallest=CSOs(1);
for el=CSOs
    if AvSumList(el)<AvSumList(smallest)
        smallest=el;
    end
end

%Elements with less than 25% density of the smallest CSO are
Outliers
Outliers=[];
for ol=1:N
    if 10*AvSumList(ol)<AvSumList(smallest)
        Outliers=[Outliers,ol];
    end
end

%fill ClusterMatrix with Memberships:
%every Member equal to every CSO
%and every CSO full Membership to itself.
NumOfCSO=numel(CSOs);
NumOfOut=numel(Outliers);
CluMat=ones(N,NumOfCSO+1)/(NumOfCSO+1);
CSOVecs=eye(NumOfCSO+1);
for cmlc=1:NumOfCSO
    CluMat(CSOs(cmlc),:)=CSOVecs(cmlc,:);
end
if NumOfOut>0
    for cmlo=Outliers
        CluMat(cmlo,:)=CSOVecs(NumOfCSO+1,:);
    end
end
% create AdjacencyMatrix with distances
InvMat=M.*AdMat;

```

```

%Similarity Matrix to get the weights from KNNs:
%InvMat=zeros(N);
%for sml=1:N
%    for smr=1:N
%        if AdDistMat(sml,smr)>0
%            InvMat(sml,smr)=1/AdDistMat(sml,smr);
%        end
%    end
%end

SimList=sum(InvMat);

SimMat=zeros(N);
for sm=1:N
    SimMat(sm,:)=InvMat(sm,:)./SimList;
end

return %% end of function flame_init()

```