Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

BACHELORTHESIS

# Wenn dies deine Idee ist, wird dich auch jene interessieren – Innovationsboost für Ideenplattformen durch automatisiertes Matching

vorgelegt von

Konstantinos Tsiamis

MIN-Fakultät

Fachbereich Informatik

Arbeitsbereich Language Technology(LT)

Studiengang: Software-System-Entwicklung

Matrikelnummer: 6662475

Erstgutachter: Prof. Dr. Chris Biemann

Zweitgutachter: Prof. Dr. Eva Bittner

# Abstract

**Problemstellung** Heute haben Ideenplattformen an großer Bedeutung gewonnen und haben großen Einfluss auf den Prozess von Innovationen. Nur das Problem ist, dass die Besitzer und Mitarbeiter solcher Plattformen vor großen Herausforderungen stehen. Durch knappe Ressourcen ist es fast unmöglich, die Ideen von Nutzer richtig zu bearbeiten und zu sortieren und dann auf Relevanz, Widersprüchlichkeit und Umfang zu prüfen. Hier kommt die Sprachtechnologie im Einsatz.

**Forschungsfrage** Hier in dieser Arbeit wird der Fokus auf sprachtechnologische Fragen gelegt. Hier wird durch Experimente untersucht, welche Methoden und Techniken existieren, um ein System für städtische Ideenplattform zu entwickeln, welchem den Benutzer zu seiner vorgeschlagene Idee ähnliche Ideen vorschlägt und daraufhin zu bewerten, wie semantische ähnlich diese Ideen für den Benutzer sind. Dabei werden alle mögliche Ähnlichkeitsberechnungen untersucht auf semantische Ähnlichkeit und evaluiert und inwiefern diese zu dem gegeben Kontext passt.

**Methoden** Bei der Untersuchung wurde ein mehrschichtiges Perzeptron verwendet und als Validierungstechnik die Kreuzvalidierung verwendet. Die Daten mit 9250 Satzpaare und 1850 Ideen wurden mit Hilfe der natürlichen Sprachverarbeitung vorbereitet. Das neuronale Netzwerk bekam als Feature BOW, TFIDF, LCS, LDA, Word2vecaverage, Word2Vectf, Jaccard und Doc2vec. Dabei wurde jedes Feature nach jedem Experiment hinzugefügt. Als Evaluation wurde das NDCG verwendet.

**Ergebnisse** Es wurde ein Ergebnis von 0.95 erzielt. Es wurde auch gezeigt, dass die Frameworks Deeplearninn4j, Mallet und Lucene sehr gut zu diesem Kontext passen und dass die Kombinationen von Features, die auf Vektoren basieren und Verfahren, die nur auf Strings basieren, gute Ergebnisse erzielen können.

# Inhaltsverzeichnis

# 1 Einführung

Ideenplattformen werden bei Unternehmen und öffentliche Institutionen immer beliebter. Am Innovationsprozess sind eine große Mengen von Leuten, wie zum Beispiel Kunden oder Bürger beteiligt. Dadurch entsteht eine hohe Belastung für die Betreiber solcher Ideenplattformen. Einige Probleme zum Beispiel sind, das Erstens sehr wenig Arbeitskräfte existieren. Des weiteren müssen, diese wenige Arbeiter eine sehr große Menge an Ideen moderieren und weiter bearbeiten. Dann kommt noch das Problem, dass die Ideen widersprüchlich, unvollständig oder nicht sinnvoll sind. Um genau die richtige Ideen zu bekommen, als auch die Arbeiter solcher Plattformen zu entlasten, kommt das Ideenmatching ins Spiel. Dabei geht es darum, dass ein Nutzer, zu seiner beschriebenen Idee ähnliche Idee von anderen Nutzer als Empfehlung vorgeschlagen bekommt. Anschließend kann der Nutzer die empfohlenen Ideen auf bestimmte Kriterien bewerten als auch kommentieren. Diese Bewertung der andere Ideen hilft den Arbeiter die Idee richtig weiter zu bearbeiten und zu analysieren.(Quellen:[1])LaTeX.

## 1.1 Ziel

Bei der Bachelorarbeit ist eine Webapplikation zu erstellen, die eine städtische Ideenplattform ist. Dabei soll diese Webapplikation konzipiert, erstellt und untersucht werden. Bei dieser Applikation soll der Benutzer eine Idee eingeben können, und dann soll ähnliche Ideen mit Hilfe der semantische Ähnlichkeit vorgestellt werden. Diese Ideen sind in einer Plattform gespeichert. Anschließend kann der Nutzer Aktionen durchführen, wie zum Beispiel die Ideen zu bewerten, wie ähnlich sie tatsächlich sind. Bei dieser Bachelorarbeit ist der Fokus auf die Sprachtechnologie. Hier ist die Forschungsfrage, was für Verfahren gibt es, um die Ähnlichkeit zwischen zwei Texten zu berechnen und wie gut passen sie in diesem Kontext. Bei der Evaluation wird untersucht, wie gut die Ähnlichkeit ist, sprich ob es sich um wirklich ähnliche Beschreibungen handelt. Hier werden Experimente durchgeführt mit unterschiedlichen Ähnlichkeitsberechnungen. Es werden verschiedene Berechnungen kombiniert, um das ideale Ergebnis zu bekommen.(Quellen:[1])

## 1.2 Aufbau der Bachelorarbeit

Zunächst wird im Abschnitt 2, die verwandte Arbeiten zu dieser These erläutert. Anschließend werden unterschiedliche Vektorrepräsentation für Texte vorgestellt und Verfahren für die Ähnlichkeitsberechnungen vorgestellt, die sowohl auf Vektoren als auch auf Strings basieren. Im Kapitel 3 wird gezeigt, wie die Datensätze bearbeitet wurden, wie der Versuch aufgebaut wird Zu guter Letzt wird noch erläutert welche Methode für die Evaluation verwendet wurde. Des weiteren wird erklärt, welche Techniken verwendet wurden und wie die Parameter gesetzt

wurden. In Kapitel 4 werden die Experimente ausgewertet und die Stärken und die Schwächen des Systems erläutert. Im Kapitel 5 wird die Webapplikation vorgestellt und wie das System funktioniert. Im Kapitel 6 folgt das Fazit und eine Zusammenfassung.

# 2 Hintergrund

## 2.1 verwandte Arbeiten

Die Semantische Textähnlichkeit(Semantic Textual Similarity kurz STS) hat in den letzten Jahren in der Forschung an großes Interesse gewonnen. Da dieses Thema bei vielen Aufgaben wenn es, um die natürliche Sprachverarbeitung geht, ein wesentliches Problem ist. Zu diesem Thema gibt es sehr viele Arbeiten, wo gezeigt wird, mit welchen Verfahren man die semantische Ähnlichkeit berechnet. Hier in diesem Abschnitt werden 10 Arbeiten vorgestellt, die für Bearbeitung der Bachelorarbeit wichtig waren.

**Computing Semantic Text Similarity Using Rich Features** Der Fokus in dieser Arbeit legten die Wissenschaftler auf die Berechnung von semantischer Ähnlichkeiten zwischen zwischen zwei Texte. Dafür verwendeten Sie als Modell ein Support-Vektorregressionsmodell, die als Eingabe Features bekomment hat wie WordNet-basierte Funktionen, Corpus-basierte Funktionen, Word2Vec-basierte Funktionen, Alignment-basierte Funktionen und literalbasierte Funktionen. Dieses Modell kann für englische Satzpaare vorhersagen, wie ähnlich diese Paare sind.(Quellen:[16])

**Description and Evaluation of Semantic similarity Measures Approaches** Wichtig für die Wissenschaftler für diese Arbeit war, welche Maßnahmen existieren für die semantische Ähnlichkeit und welche Methoden, zu welchen Anforderungen passen. Dabei wurde untersucht, ausgewertet und bewertet, wie gut diese Methoden sind.(Quellen:[24])

**UniMelb NLP-CORE: Integrating predictions from multiple domains and feature sets for estimating semantic textual similarity** In diesem Paper wurde ein System entwickelt, welche sagt, wie ähnlich zwei Texte sind. Dabei wurde ein Regresionsmodell erstellt und mit Features gefüttert. Bei den Features handelt es sich um Stringähnlichkeitsberechunungen und LDA. Dabei wurde unterschiedliche Features kombiniert und bei unterschiedliche Datensätze eingesetzt.(Quellen:[15])

**Cross-Language Plagiarism Detection Methods for Semantic Textual Similarity** Hier wurde ein System präsentiert, welche anhand von englischen oder spanischen Satzpaare bestimmen muss, wie semantisch ähnlich sind. Dabei wird eine Punktzahl von 0 bis 5 gegeben. Ihr System verwendete als Featues syntaxbasierte, wörterbuchbasierte, kontextbasierte und MT-basierte Methoden. Diese Features wurden miteinander kombiniert und an unterschiedlichen Techniken von maschinelles Lernen angewendet.(Quellen:[22])

**Semantic Textual Similarity Estimation of English Sentence Pairs Using Regression Model over Pairwise Features** Dieses Papier präsentiert ein Regresionsmodell, welche die semantische Ähnlichkeit von englischen Satzpaare bestimmt. Auch hier mit einer Punkte Skala von 0 bis 5. Als Features wurden Worteinbettungsvektoren, Part-of-speech-Tagging(POS) ,Namensentitäten(NE) und Bag of words(BOW) verwendet. Wie bei den anderen Papers wurde die Features miteinander kombiniert.(Quellen:[21])

**Application of Ensemble learning for computing semantic textual similarity** Hier wurden ein Modell entwickelt, die semantische Ähnlichkeit von englischen Satzpaare bestimmen soll. Bei diesem Experiment wurde ein rekursiven neuronale Netzwerk (RNN) verwendet. Das Netzwerk bekommt als Eingabe drei verschiedene Features. Einmal Worteinbettungsvektoren mit Gewichtung, Part-of-speech-Tagging(POS) und eine Methode mit Wortabdeckung(TakeLab).(Quellen:[29])

**Semantic Textual Similarity Based on Word Embeddings** Forscher entwickelten hier 3 unüberwachte Systeme, die mit Worteinbettungen gefüttert wurden. Für die Forscher war die Vorhersage der semantische Ähnlichkeit nicht wichtig, sondern was für eine Rolle spielt Verarbeitung von Datensätzen. Das Resultat war, dass die Techniken wie Lemmatisierung,Tokenisierung , Extraktion von Inhaltswörter und Entfernen von Stoppwörtern eine große Rolle auf das Ergebnis haben.(Quellen:[30])

**Word Similarity Based on Word Embedding and Knowledge Base** Die Wissenschaftler entwickelten ein Modell, dass die Ähnlichkeit zweier Artikel berechnen. Das Modell wurden mit Worteinbettungen und wissensbasierte Methoden gefüttert. Das Resultat zeigt, dass die Kombination dieser beiden Features gute Ergebnisse liefern. Wenn man sie einzeln verwendet, sind sie nicht so gut und müssen nachgebessert werden.[31]

**Leverage Kernel-based Traditional NLP features and Neural Networks to Build a Universal Model for Multilingual and Cross-lingual Semantic Textual Similarity** Die Forscher haben ein System entworfen, die mehrere Sprachen beherrscht als auch sprachübergreifend, um die semantische Ähnlichkeit für solche Satzpaare zu bestimmen. Das System übersetzt es die Fremdsprache ins Englische und macht die Vorhersage. Dabei wurde das System mit 3 Features gefüttert, einmal Worteinbettungen, "Dependency Featuresünd "BOW Features".(Quellen:[53])

**Predicting Semantic Textual Similarity with Paraphrase and Event Embeddings** Die Wissenschaftler entwickelten ein Regressionsmodell mit zwei Subsysteme entwickelt und dieses Modell wurde mit den Features Paraphrasen und Ereigniseinbettungen gefüttert.(Quellen:[32])

## 2.2 semantische Ähnlichkeit

### 2.2.1 Vektorrepräsentationen

In diesem Abschnitt wird erklärt, welche Vektorrepräsentationen für Sätze benutzt werden, damit diese Sätze für die Maschinen lesbar sind und welche Ähnlichkeitsberechnungen durchgeführt wurden, um die semantische Ähnlichkeit zu finden.

**BoW** Bei ersten Feature handelt es sich um das Bag of Words Model. Es ist das simpelste Modell für die Darstellung für Texte im im Bereich natürliche Sprachverarbeitung und in Informationsbeschaffung. Auch als Vektorraummodel bekannt. Erstmal wird ein Wörterbuch erzeugt, dass alle Wörter extrahiert von dem untersuchten Dokument. Das Modell sortiert dann, die Wörter da nach, wie häufig sie vorkommen. Dabei das Häufigste an erster Stelle etc. In Dokumente guckt er dann, wie häufig, kommt es im Satz vor und notiert das. (Quellen: [34][56][18][2]) Hier ein Beispiel:

| | I | love | dogs | hate | and | knitting | is | my | hobby | passion |
|---|---|---|---|---|---|---|---|---|---|---|
| Doc 1 | 1 | 1 | 1 | | | | | | | |
| Doc 2 | 1 | | 1 | 1 | 1 | 1 | | | | |
| Doc 3 | | | | | 1 | 1 | 1 | 2 | 1 | 1 |

Abbildung 2.1: Bag of words

**TFIDF** TFIDF steht für term frequency-inverse document frequency. Es ist ähnlich aufgebaut wie das BoW, jedoch kriegen die Wörter eine Gewichtung. (Quellen: [33][35][36][2][18][19]) Diese Gewichtung berechnet sich wie folgt: Das TfIDF verwendet zwei Rechnungen. Die Rech-

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

Abbildung 2.2: TFIDF-Formel

nung lautet die Anzahl des Wortes, welche in dem Dokument erscheint, dividiert durch die insgesamt Anzahl der Wörter in dem Dokument(TF), dann die zweite Rechnung ist, den Logarithmus zu ziehen von der Anzahl der Dokumente, die im Textkorpus vorliegen, dividiert durch die Anzahl der Dokumente, in dem das untersuchte Wort auftaucht(IDF). Hat man beides, multipliziert die miteinander und kriegt, so die Zahlen für die Vektoren.

| | I | love | dogs | hate | and | knitting | is | my | hobby | passion |
|---|---|---|---|---|---|---|---|---|---|---|
| Doc 1 | 0.18 | **0.48** | 0.18 | | | | | | | |
| Doc 2 | 0.18 | | 0.18 | **0.48** | 0.18 | 0.18 | | | | |
| Doc 3 | | | | | 0.18 | 0.18 | **0.48** | **0.95** | **0.48** | **0.48** |

Abbildung 2.3: TFIDF-Vektor

**Word2vec** Die nächste Vektorrepräsentation ist das Word2vec. Es ist ein Modell, welche aus Wörter Vektoren darstellt. Das Modell ist ein zweischichtiges neuronales Netz. Als Eingabe wird ein Textkorpus gegeben, wo daraufhin ein Vektorraum erzeugt wird. In diesem Raum hat jedes Wort ein Vektor und je dichter sie beieinander liegen, umso semantisch ähnlicher sind sie. Wie das aussieht, kann man im folgen Bild sehen:

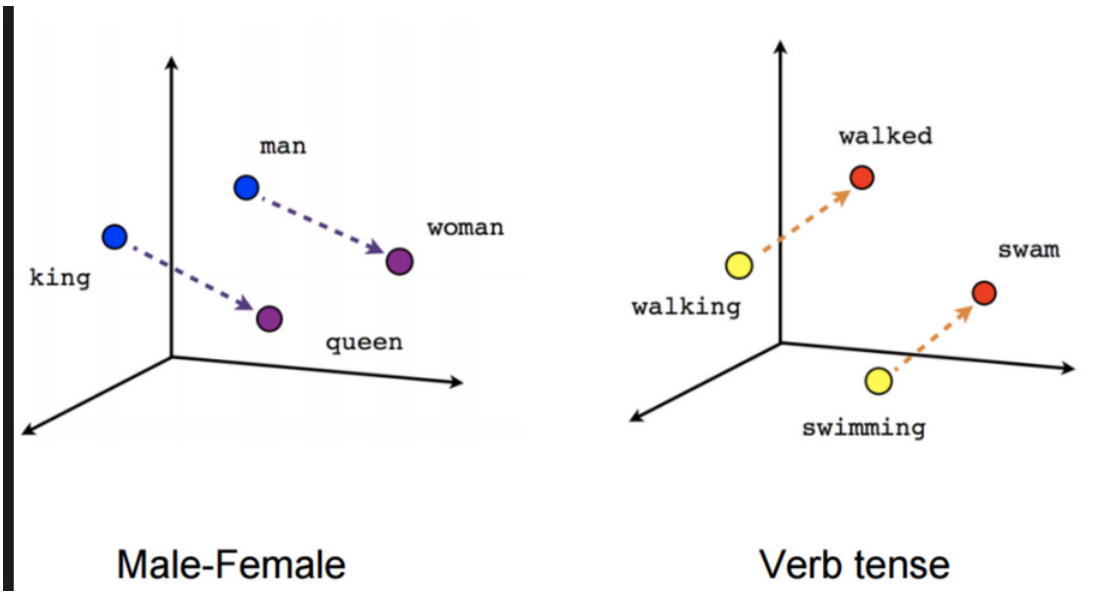



Abbildung 2.4: Word2Vec

Da wir ausschließlich mit Sätze befassen und wir Sätze als Vektoren darstellen möchten. Wurde zwei verschiedene Vorgehensweise im Word2Vec benutzt. Einmal wurde jedes einzelne Wort in dem Satz in Vektor umgewandelt und anschließend wurden alle Vektoren miteinander addiert. Zu guter Letzt wurde der Durchschnitt genommen und das repräsentiert den Satzvektor(Word2vecaverage). Der zweite Ansatz ist nachdem man die Wörter in dem Satz als Vektoren umgewandelt hat, ist die TFIDF-Formel von jedem Wort zu berechnen, dann wird das TF-IDF-Wert multipliziert, mit dem Vektor zum entsprechenden Wort. Danach addiert man alle Vektoren miteinander und nimmt dann den Durchschnitt für den Satzvektor(Word2Vectf).(Quellen:[46][2][10][11][58])

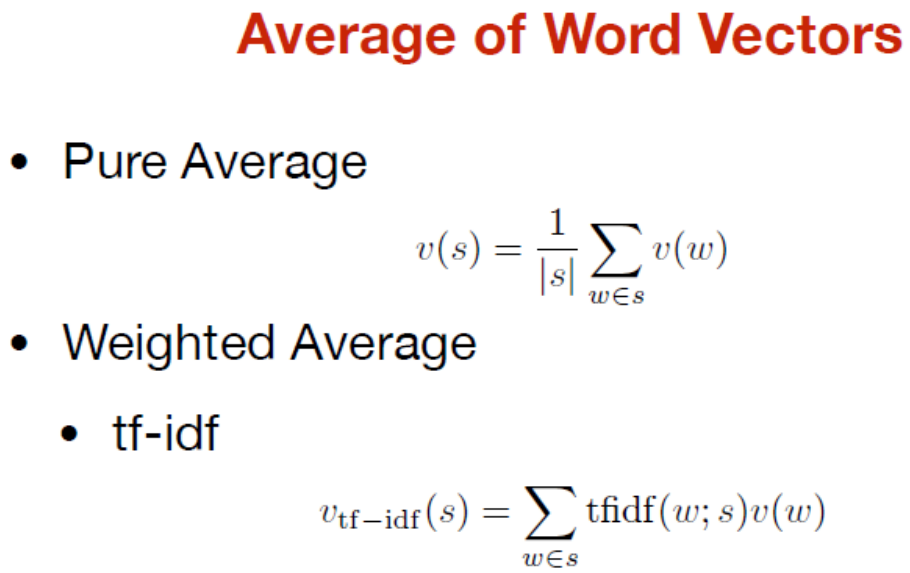



Abbildung 2.5: Word2Vecaverage and tfidf

**Doc2vec** Doc2vec ist ein ähnliches Modell wie das Word2Vec, jedoch wurde dies extra für Sätze entwickelt. Sprich Sätze werden sofort in Vektoren umgewandelt. Es wird zu jedem Wort im Dokument ein Vektor erstellt und jedem Dokument ein Vektor erstellt. Dann wird eine Stufe höher, aus diesen beiden Komponenten, ein neues Dokument erstellt. Danach werden Gewichtungen festgelegt, um den gewünschten Vektor von dem Satz zu bekommen.(Quellen: [2][12][42][43]) Wie das aussieht kann man im folgen Bild sehen:

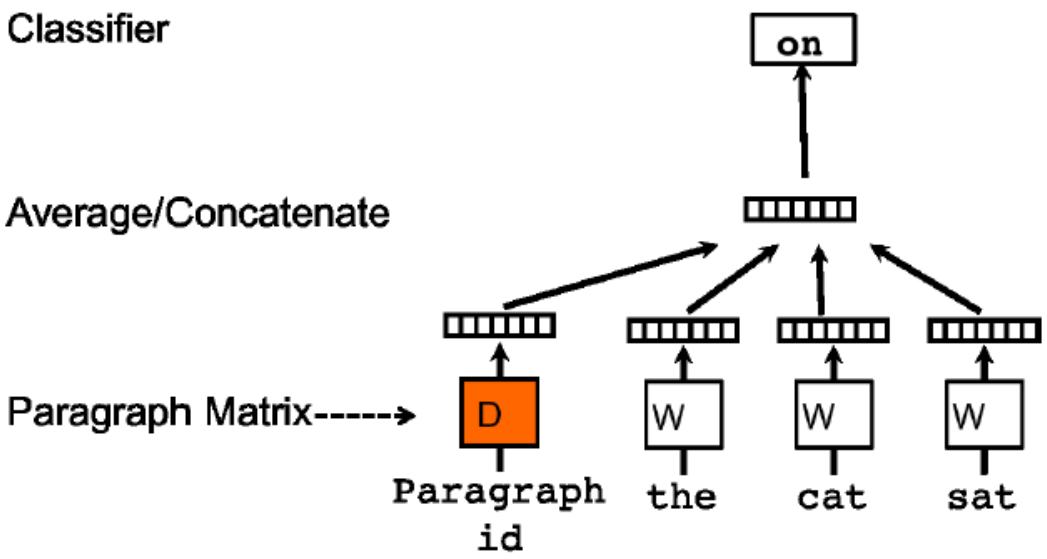
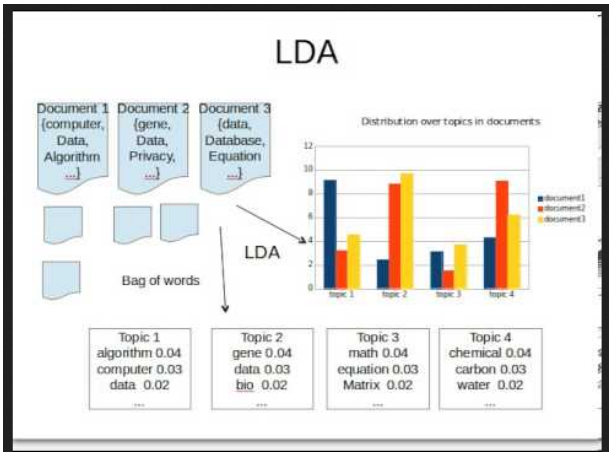



Abbildung 2.6: Doc2vec

**LDA** Die nächste Vektorrepräsentation ist das LDA und steht für Latent Dirichlet allocation. Es werden Vektoren erstellt, in dem man jedes einzelne Wort in einem Dokument, eine Wahrscheinlichkeit zu teilt, zu welchem Thema es gehört. Die Themen und die Anzahl werden vom Anwender festgesetzt. Dabei berechnet sich die Wahrscheinlichkeit aus der Verteilung von der Anzahl der Themen und aus der Dirichlet-Verteilung.(Quellen: [38][39][8][61][59][62])



Hier ist die Formel für die Dirichlet-Verteilug. Die Dirichletverteilung gehört zu der Gruppe

$$f(x_1,\ldots,x_K;\alpha_1,\ldots,\alpha_K) = \frac{1}{\mathrm{B}(\alpha)}\prod_{i=1}^{K} x_i^{\alpha_i-1}$$

Abbildung 2.7: Dirichlet-Verteilung

$$\mathrm{B}(\alpha) = \frac{\prod_{i=1}^{K}\Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{K}\alpha_i\right)}, \qquad \alpha = (\alpha_1,\ldots,\alpha_K).$$

Abbildung 2.8: Dirichlet-Verteilung

der multivariaten Wahrscheinlichkeitsverteilungen. Der Begriff multivariat bedeutet, dass es von mehreren Variablen abhängt. Dabei ist sie stetig und ist von den Parameter Alpha und Beta abhängig. Bei der Formel oben links handelt es sich, um eine Dichtefunktion und gibt zu den Anzahl der Themen die Wahrscheinlichkeit an.

### 2.2.2 Ähnlichkeitsberechnungen für Vektoren

**Kosinus** Die erste Ähnlichkeitsberechnung ist die Kosinusähnlichkeit. Dabei werden einfach der Kosinus Winkel zwischen zwei Vektoren berechnet. Es kommen nur Werte zwischen 1 und 0 raus. Wenn sie senkrecht zueinander steht, ist es eine 0, selbe Orientierung eine 1.(Quellen: [54]) Die Formel lautet:

$$\text{Kosinus-Ähnlichkeit} = \cos(\theta) = \frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{a}\|_2\|\mathbf{b}\|_2} = \frac{\sum_{i=1}^{n} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{n}(a_i)^2}\cdot\sqrt{\sum_{i=1}^{n}(b_i)^2}}$$

Abbildung 2.9: Kosinusähnlichkeit

**Euklidische Distanz** Es berechnet den Abstand zweier Punkte.

$$d(p,q) = \|q-p\|_2 = \sqrt{(q_1-p_1)^2+\cdots+(q_n-p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}$$

**Canberra Distanz** Es berechnet den Abstand zwischen Paaren von Punkten.

$$d(\mathbf{p},\mathbf{q}) = \sum_{i=1}^{n}\frac{|p_i-q_i|}{|p_i|+|q_i|}$$

**Chebyshev Distanz** Es berechnet die Distanz zwischen zwei Vektoren, wo die größte Differenz entlang der Koordinatendimension ist.

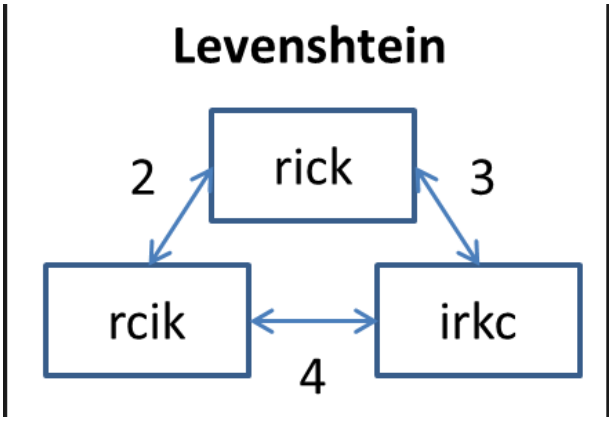
$$D_{\text{Chebyshev}}(p,q) := \max_i(|p_i-q_i|).$$

**Manhattan Distanz** Es berechnet die Distanz zwischen zwei Punkte, welche die Summe der Differenz ihrer Koordinaten bildet.

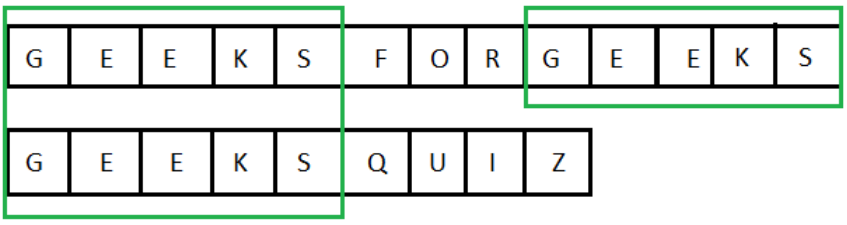
$$d_1(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_1 = \sum_{i=1}^{n} |p_i - q_i|,$$

### 2.2.3 Ähnlichkeitsberechnungen für Texte

**Levensthein** Bei der Levenshtein-Distanz berechnet es, die Differenz zwischen zwei Sequenzen, sprich wie oft muss ich das Wort ändern, um es in den anderen Text zu bekommen. Um das Wort zu ändern wird eingefügt, gelöscht und ersetzt.(Quellen: [49][50][51][60]) Hier ist kleines Beispiel, wie das aussehen kann:




**LCS** LCS steht für longest common substring. Dabei schaut es sich in beiden Text an, an welcher Stelle die längste Folge von Zeichens, die identisch sind. (Quellen: [47][48])




OUTPUT: 5
As longest Common String is "Geeks"

**Jaccard-Koeffizient** Hier hat man zwei Mengen, also unsere Texte und guckt wie viele Wörter, die gemeinsam haben und teilt es durch die gesamte Länge beider Texte.(Quellen: [44][45]) Die Formel lautet:

$$\mathrm{J}(A,B) = \frac{\left|A \cap B\right|}{\left|A \cap B\right|}$$

### 2.2.4 Frameworks

Die Experimente wurden alle in der Programmiersprache Java durchgeführt.

**Deeplearning4j** Deeplearning4j ist ein Framework, welches maschinelles Lernen, künstliche Intelligenz und natürliche Sprachverarbeitung anbietet. Mit Hilfe dieses Frameworks können Word2vec, Doc2vec, TFIDF und Bow Modelle erstellt werden.(Quellen: [2])

**Mallet** Mallet ist ein Framework, welches viele Tools für natürliche Sprachverarbeitung anbietet. Es bietet sehr viele unterschiedliche Implementation zu LDA an.(Quellen: [8])

**Lucene** Bei Apache LuceneTM handelt es sich um eine Suchmaschinenbibliothek. So bietet sich Funktionen an, die dabei helfen Dokumente zu indexieren und sofort abzurufen. (Quellen: [9])

**Weka** Weka (Waikato Environment for Knowledge Analysis) ist ein weiteres Framework, welches viele Tools für maschinelles Lernen anbietet. Hier wurde nur der mehrschichtige Perzeptron(kurz MLP) gebraucht.(Quellen: [3])

**Apache Common Text** Dieses Framework bietet für Strings viele Ähnlichkeitsberechnungen, um die semantische Ähnlichkeit zu finden. (Quellen: [57])

### 2.2.5 Evaluation

Für die Evaluation der Experimente wurde das NDCG verwendet, welches für Normalized Discounted Cumulative Gain steht.(Quellen: [55]) Die Formel lautet:

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p}.$$

Der erste Term DCG steht für Discounted Cumulative Gain (DCG). Dieser Wert sagt aus, wie gut das Ranking ist. Dabei wird der Wert durch die Relevanz des Dokuments, als auch die Postion des Dokuments in der Liste bestimmt. Der zweite Term ist das IDCG für Ideal Discounted Cumulative Gain. Das IDCG gibt den maximalen DCG Wert, dabei werden Dokumente nach ihrer Relevanz sortiert und dann die Formel für das DCG angewendet, die wie folgt lautet:

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

Damit, die Leistung unseres Systems evaluiert werden kann, müssen die Werte normalisiert werden, da reicht das DCG alleine nicht aus. Dieses klappt mit dem IDCG. Man kann alle NDCG-Werte, die man mit allen Abfragen ermittelt hat, den Durchschnitt nehmen, dass als Maß für die Leistung des Ranking-Algorithmus einer Suchmaschine nehmen. Der NDCG kann Werte von 0 bis 1 annehmen.

# 3 Vorgehensweise

## 3.1 Datensatz

Erstmal zum Datensatz. Zu Beginn wurden mir 3 verschiedene Tabellen gegeben, bei denen es sich, um unstrukturierte Textdateien handelt. Aus drei verschiedenen Tabellen wurden die Ideenbeschreibung, extrahiert und alle in einem Textfile abgespeichert. Dieses File besitzt alle Ideen ohne Duplikate von allen 3 Tabellen. Zu dem wurde ein zweites File erstellt, welches Themen dieser Ideenbeschreibung abspeichert. Das zweite File ist später relevant für das LDA, da es Startparameter, wie die Anzahl der Themen braucht. Bei den Ideen handelt es sich um ausschließlich deutsche Texte. Anschließend wurden die erstellten Dokumente indexiert, so dass jede einzelne Idee sofort aufrufbar ist, mit Hilfe der Textsuchmaschine Lucene. So muss man nicht durch etliche Schleifen durchlaufen, um die benötigte Ideenbeschreibung zu finden. Nach diesem Prozess wurde ein Iterator angewendet von Deeplearning4j der Sentence Iterator. Der Iterator wird eingesetzt bei Word2vec, TFIDF, Bow und Doc2vec verwendet. Es sorgt dafür Texte in Form von Vektoren zu bringen, die für neuronale Netze lesbar sind. Danach wird Ausgabe vom Iterator zum Tokenizer von Deeplearning4j weitergegeben. Hier wird das Verfahren Tokenisierung durchgeführt, was bedeutet, dass Texte in einzelne Wörter aufgeteilt wird. Zusätzlich werden die Wörter in Kleinbuchstaben geschrieben. Zu guter Letzt werden die Stoppwörter entfernt. Stoppwörter sind die Wörter, die keine Relevanz für den Inhalt haben und somit ignoriert werden. Jetzt wurden die Daten soweit aufbereitet, dass man die Vektoren erstellen kann und Ähnlichkeitsberechnungen anwendet mit Vektoren oder mit Texte string basierte Verfahren anwendet.

## 3.2 Versuchsaufbau

Kommen wir zum Versuchsaufbau. Der erste war der Annotationsprozess. Dabei wurde zu jeder Idee im File, willkürlich fünf andere Ideen ausgewählt und bewertet, sprich man schaut sich an ,wie ähnlich diese zwei Ideen sind und gibt Punkte. Meine Punkteskala geht von 0 bis 3 punkte. Dabei steht 0 für nicht semantisch ähnlich, für total semantisch ähnlich eine 3, 1 ein bisschen semantisch ähnlich und 2 bedeutet ähnlich. Das wurde in File gemacht und als "ground truth"bezeichnet. "Ground thruth"braucht man beim überwachten maschinellen Lernen, um zu gucken, wie genau Vorhersage von der Maschine ist, für die semantische Ähnlichkeit. Die Werte von der Maschine werden mit dem "Ground thruth"verglichen. Hier ist kleine Illustration, wie eine Annotation bei deiner Idee aussieht.

Eine kleine Illustration:

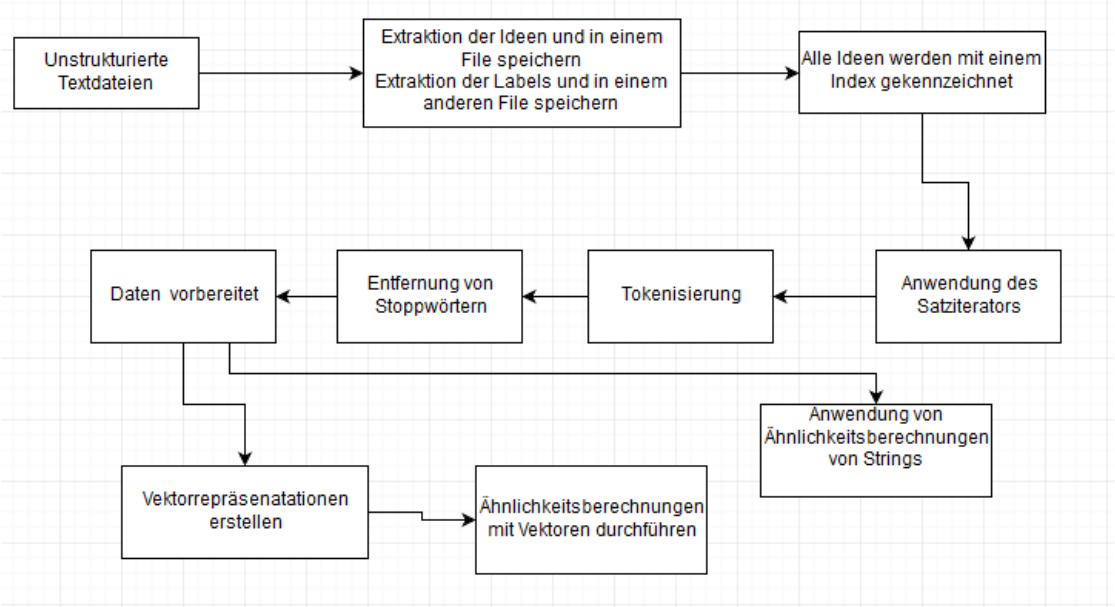



Abbildung 3.1: Verarbeitung vom Text

Idee 1 —> Lösung 1 –> ground truth: 3

Idee 1 —> Lösung 2 –> ground truth: 2

Idee 1 —> Lösung 3 –> ground truth: 0

Idee 1 —> Lösung 4 –> ground truth: 1

Idee 1 —> Lösung 5 –> ground truth: 1

Anschließend wird diese richtig geordnet in 3,2,1,1,0 und dann das IDCG angewendet und dieser Wert wird notiert, da es wichtig für die Auswertung der Experimente ist. Diese Annotation wurde mit allen 1850 Ideen gemacht, sprich wir haben insgesamt 9250 deutsche Satzpaare, die manuell per Hand annotiert wurden. Jetzt kommen unsere Features und neuronales Netzwerk im Spiel. Bei einem Feature handelt es sich um eine Ähnlichkeitsberechnung von der Idee und die ausgewählte Idee und der errechnete Wert dient als Eingabe zum neuronalen Netzwerk. Als Ausgabe für das Netzwerk ist unsere Punkteskala von 0 bis 3.

Eine kliene Illustration wie das aussieht

f1, f2, f3, f4, f4, ....etc... Label (ground truth)

f11, f21, f31, f41 ....etc... 3

f12, f22, f32, f42 ....etc... 2

f13, f23, f33, f43 ....etc... 0

f14, f24, f34, f44 ....etc... 1

f15, f25, f35, f45 ....etc... 1

Anschließend wird eine Netzwerk trainiert, es wird eine Vorhersage getroffen. Dann bekommt man den Wert:

BOW, topic, doc2vec, longest common sub-sequence , Euclidean distance, ....usw... Label (ground truth) | Vorhersage

f11, f21, f31, f41 ....etc... 3 –> Vorhersage: 3

f12, f22, f32, f42 ....etc... 2 –> Vorhersage: 1

f13, f23, f33, f43 ....etc... 0 –> Vorhersage: 0

f14, f24, f34, f44 ....etc... 1 –> Vorhersage: 0

f15, f25, f35, f45 ....etc... 1 –> Vorhersage: 0

Dann werden die Ausgabe vom Netzwerk sortiert und das DCG bei 3,1,0,0,0 angewendet. Jetzt man die NDCG-Formel anwenden und dann vergleichen. Ab diesem Punkt kann experimentieren, man kann nach jedem Durchlauf neue Feature hinzufügen. So laufen die Experimente hier ab. Bei dem Experiment wurde ein mehrschichtiges Perzeptron verwendet und als Technik die Fehlerrückführung(backpropagation). Dabei werden die Werte solange geändert bis es einem gewissen Muster entspricht und dann kann die Vorhersage durchgeführt werden. Hier eine kleine Übersicht über die Architektur:

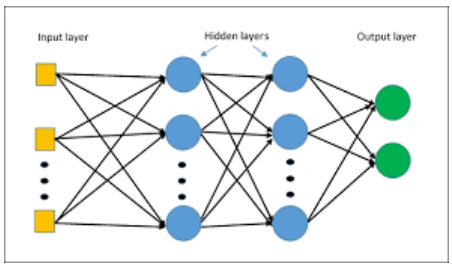



Abbildung 3.2: neuronales Netz

Für die Validierung wurde als Technik, die Kreuzvalidierung als Bewertung des Vorhersage-

modells genommen. Der Zweck dieser Technik ist, wie gut unser Modell mit Daten umgeht, die unbekannt für das System sind und wie es mit unbekannte Dokumente umgeht. Hier in unserem Experimente haben wir 10 folds benutzt.(Quellen: [40][41][52][63]) Ein Bild wie diese Technik aussieht:

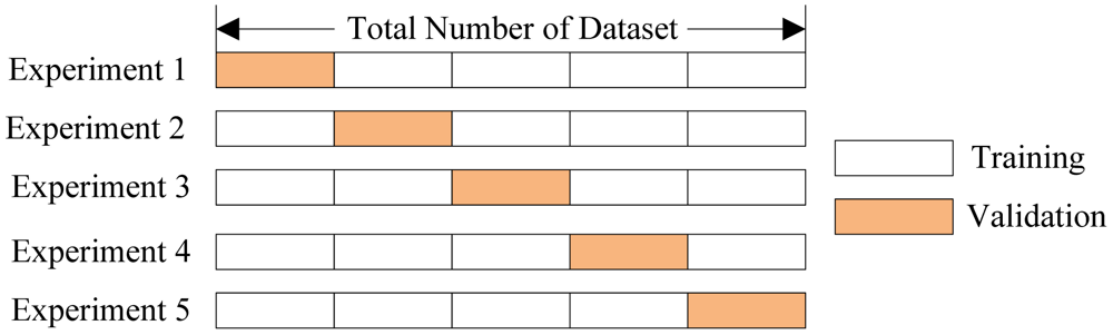



Abbildung 3.3: Kreuzvalidierung

## 3.3 Techniken

Hier werden vorgestellt, welche Parameter für Vektorrepräsentationen eingestellt wurde, die dann bei allen Experimente durchgeführt wurden sind.

Word2vec

| Parameter | Wert |
|---|---|
| layerSize | 20 |
| windowSize | 5 |
| iterations | 20 |
| minWordFrequency | 1 |

Doc2vec

| Parameter | Wert |
|---|---|
| layerSize | 100 |
| learningRate | 0.025 |
| epochs | 1 |
| iterations | 100 |
| minWordFrequency | 1 |
| windowSize | 5 |

LDA

| Parameter | Wert |
|---|---|
| Anzahl an Themen | 30 |
| alpha | 1 |
| beta | 1 |
| iterations | 2000 |

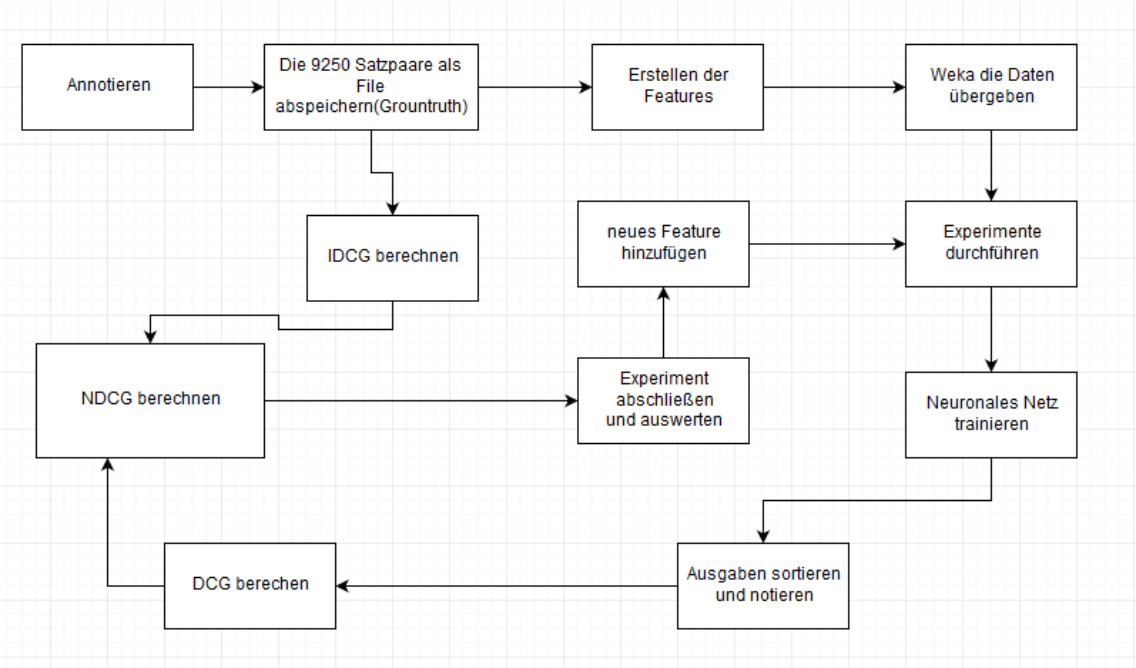
alpha ist das Dirichlet für die Verteilung über die Themen
beta ist pro Thema die Multinomialverteilung über Wörter

Mlp

| Parameter | Wert |
|---|---|
| epochs | 500 |
| momentum | 0.2 |
| learningrate | 0.3 |
| iterations | 2000 |

minWordFrequency: Ist die minimale Anzahl Wörter, die vorkommen muss, um das Wort zu merken.
layerSize: Bestimmt wie viele Features ein Vektor hat.
learningRate: Wann die Werte aktualisiert, sprich die Wortvektoren im Raum neu zu positionieren.
epochs: Wie oft man das Netz durchläuft.

Hier ist nochmal ein Bild, was den Aufbau des Experiments zeigt:


Annotieren
Die 9250 Satzpaare als File abspeichern(Grountruth)
Erstellen der Features
Weka die Daten übergeben
IDCG berechnen
neues Feature hinzufügen
Experimente durchführen
NDCG berechnen
Experiment abschließen und auswerten
Neuronales Netz trainieren
DCG berechen
Ausgaben sortieren und notieren

# 4 Auswertung

In diesem Kapitel prüfen wir die Qualität der Ähnlichkeitsberechnungen unseres Systems. Bei der Auswertung wird geachtet, wie sich die Ranking-Qualität(also NDCG) ändert, nach jedem Experiment. Wie im Kapitel 3 angedeutet, wird nach jedem Experiment ein Feature hinzugefügt und ausgewertet. Es wurden folgende Features hinzugefügt, dabei handelt es sich, um eine absteigende Reihenfolge. Das erste Feature ist der Beginn des Experiments, mit dem angefangen wurde, daraufhin wurden weitere Features hinzugefügt.
Hier eine kleine Anschauung: (1)Bag of words
(2)TFIDF
(3)Latent Dirichlet Allocation
(4)Longest Common Substring
(5)Word2Vec average
(6)Word2vectfidf
(7)Jaccard-Koeffizient
(8)Doc2Vec

Wie bereits im Kapitel 3 erläutert, wird das NDCG berechnet per Hand. Als Visualisierung wurde die WEKA Software benutzt, als auch die Durchführung der Experimente. Mit dem Tool evaluieren wir die Ergebnisse die WEKA liefert, dabei werden die 50 Sätze mit den jeweils fünf zugeordnete Sätze raus gesucht. Daraufhin wird das NDCG berechnet pro Satz und dann der Durchschnitt genommen. Je höher das NDCG, um so besser. So ist es mit WEKA möglich, die Qualität des Ranking zu untersuchen und das bestmögliche Ergebnis heraus zu bekommen. Hier sind einige Ausschnitte von WEKA:

Wie man im ersten Bild sieht, kann man erstmal die Features, die an Weka gegeben wurden, nochmal bearbeiten oder wenn man möchte, editieren und welche entfernen. Man kann sie auch visualisieren, um zu sehen, welche Werte die Features annehmen können. Im zweiten Bild wurde ein Ausschnitt aus dem Menü Klassifizierung genommen. Hier legen wir fest, mit welchem Modell wir eine Vorhersage treffen. Wie im Kapitel 3 erwähnt handelt sich, um den mehrlagiges Perzeptron(eng. MultilayerPerceptron) mit den beschriebenen Parameter. Hier kann man noch gut sehen, dass das Kreuzvalidierungsverfahren angewendet wurde mit 10 Folds, sprich pro Fold wurde ein Datensatz mit 925 Satzpaare getestet, sodass jedes Satzpaare geprüft wurde. Das dritte Bild zeigt einen Ausschnitt, wie die Ergebnisse aussehen. Da es sich um eine Liste handelt mit fester Reihenfolge, welche vorher als Option gewählt wurde, weiß man sofort zu welchem Satzpaar, die erste Instanz gehört. Man sieht den realen Wert, der vor mir gesetzt wurde, daneben der Wert der vorhergesagt wird von der Maschine und anschließend die Abweichung. Die Werte, die vorhergesagt werden, haben ein Intervall von 0 bis 3 annehmen.

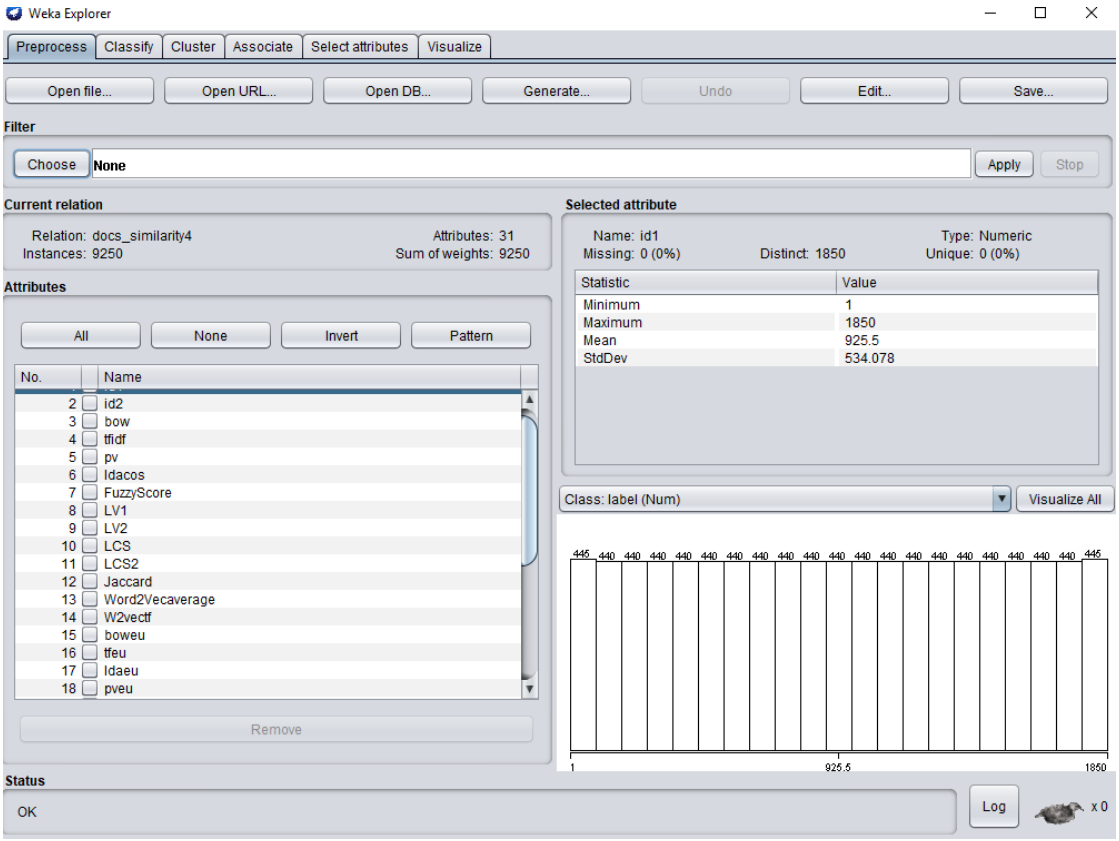



Abbildung 4.1: Weka-Ausschnitt

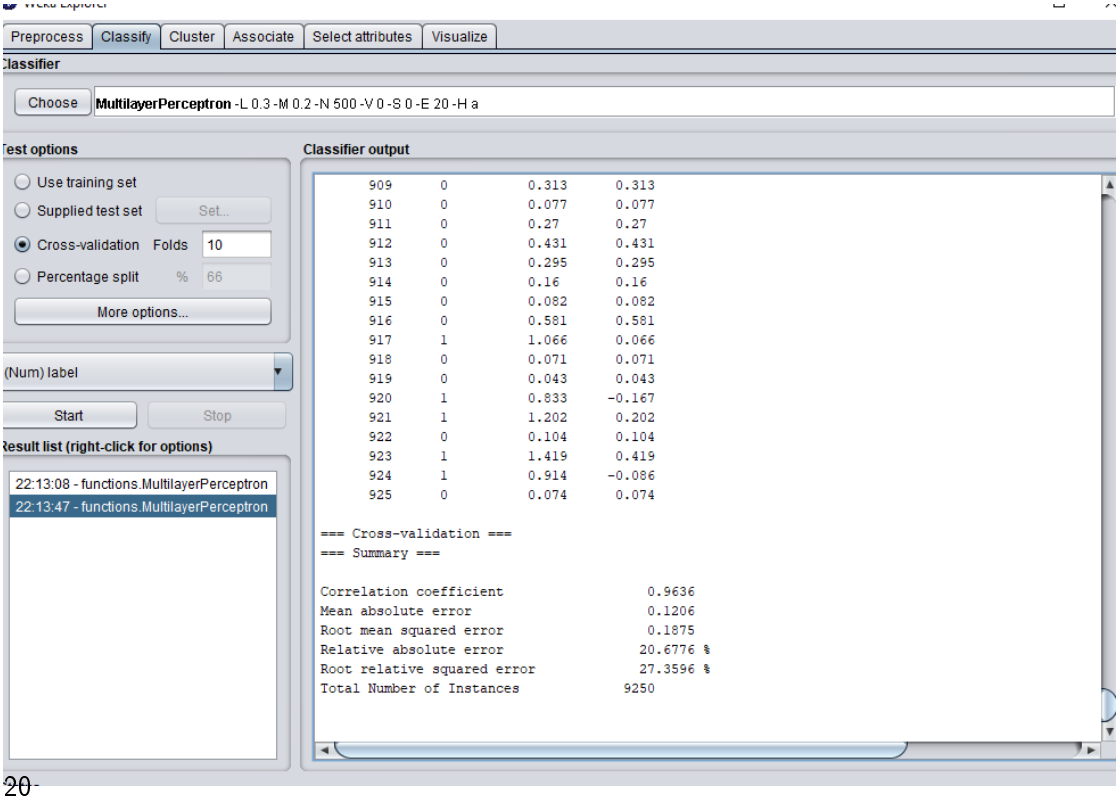





Abbildung 4.2: Weka-Ausschnitt2

| inst# | actual | predicted | error |
|---|---|---|---|
| 1 | 2 | 1.832 | -0.168 |
| 2 | 0 | 0.046 | 0.046 |
| 3 | 0 | 0.037 | 0.037 |
| 4 | 1 | 1.162 | 0.162 |
| 5 | 0 | 0.038 | 0.038 |
| 6 | 1 | 1.03 | 0.03 |
| 7 | 1 | 1.417 | 0.417 |
| 8 | 1 | 0.923 | -0.077 |
| 9 | 0 | 0.037 | 0.037 |
| 10 | 2 | 1.995 | -0.005 |
| 11 | 1 | 1.292 | 0.292 |
| 12 | 0 | 0.035 | 0.035 |
| 13 | 1 | 1.211 | 0.211 |
| 14 | 0 | 0.575 | 0.575 |
| 15 | 0 | 0.1 | 0.1 |
| 16 | 0 | 0.32 | 0.32 |
| -- | - | - -- | - -- |

Abbildung 4.3: Weka-Ausschnitt3

Bei dem ersten Experiment wurde als Feature das Bag of words Modell genommen und als Ähnlichkeitsberechnung die Kosinusähnlichkeit genommen. Als nächstes Experiment wurde BOW und TFIDF als Kombination getestet und beide jeweils arbeiten mit Kosinusähnlichkeit. Als nächstes Feature wurde das LDA hinzugenommen mit der selben Ähnlichkeitsberechnung. Anschließend wurden eine weitere Ähnlichkeitsberechnung genommen nämlich Longest Common Substring, eine string basierte Methode. Daraufhin kommt das Word2Vecaverage mit Kosinusähnlichkeit und dann wurde Word2Vectf hinzugefügt. Danach Jaccard-Koeffizient und zu guter Letzt Doc2vec. Man muss erwähnen, dass ich nicht alle Features nehmen konnte und unterschiedliche Kombinationen durchführen konnte. So habe ich insgesamt 28 Features erstellt, wie zum Beispiel die euklidische Distanz oder andere string basierte Methoden, die im Kapitel 2 erwähnt wurde. Jedoch habe sie als Kombination ausgelassen und auch jedes einzelne Feature zu untersuchen, wurde ausgelassen. Zudem wurde jedes einzelne, Feature die selbe Prozedur durchgeführt, wie im Kapitel 3 erzählt wurde. Hier noch einmal, welche Werte die Features angenommen haben. Beim Diagramm kann man sehr gut sehen, dass

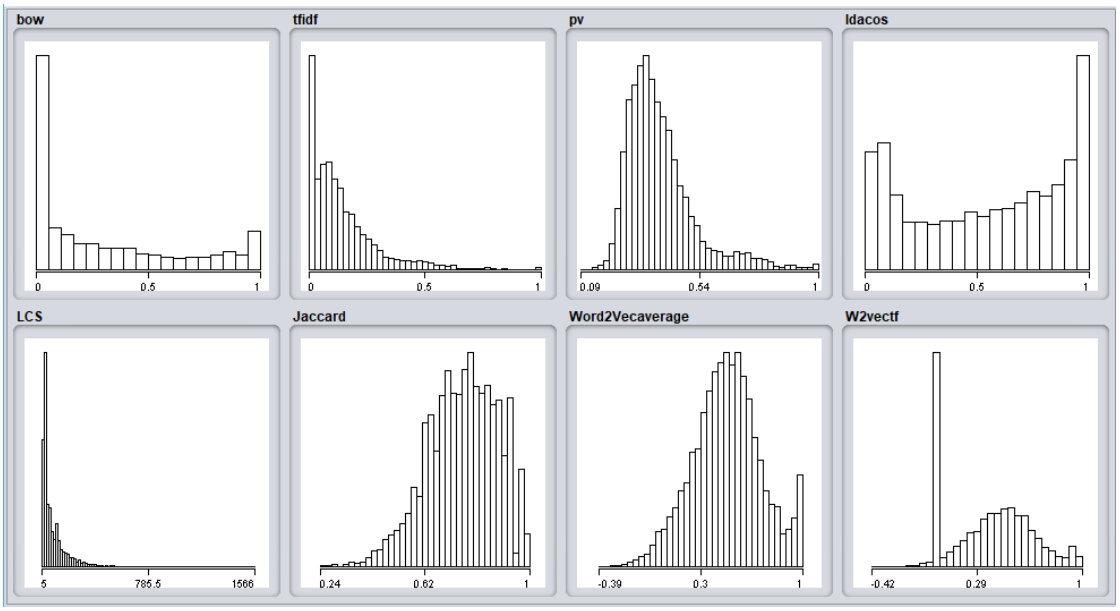



Abbildung 4.4: Weka-Diagramm

jedes Feature einen anderen Verlauf bei ihren Werten hat. So siedeln sich die meisten Werte beim Bag of words im dem Zahlenbereich gleichmäßig von 0 bis 1. Bei TFDIF befinden sich die meisten Werte im Zahlenbereich von 0 bis 0.4. Beim Doc2vec ist eine große Ansammlung im Zahlenbereich von 0.3 bis 0.52 zu sehen. Das LDA hat ist gleichmäßig verteilt, jedoch befinden sich die meisten Werte bei 0.9 und 1. Das Longest Common Substring hat die Werte eher am Anfang der Grafik. Bei Jaccard siedeln sich die Wert eher bei 0.4 bis 1. Das Word2Vecaverage hat eine ähnlich Siedlung an Werte, wobei hier die meisten Werte eher bei 0.3 und 0.6 liegen. Im Gegensatz zum Vorgänger besitzen die meisten Feautes in Wert von

0 bis 0.2. Dann folgt eine gleichmäßige Verteilung von 0.3 bis 1. Wie man sehen kann, hat jedes Feature eine eigene Vorstellung von semantischer Ähnlichkeit. Hier sind die Ergebnisse der Experimente:

| Feature | NDCG |
|---|---|
| BOW | 0.62 |
| TFIDF | 0.7772 |
| LDA | 0.8234 |
| LCS | 0.847 |
| W2average | 0.853 |
| W2tf | 0.8956 |
| Jaccard | 0.8968 |
| Doc2Vec | 0.95 |

Das Ergebnis zeigt, das nach jeder Iteration, sich die Ranking-Qualität, sich verbessert hat. Man braucht nur 8 Iteration, um ein sehr gutes Ergebnis in unserem Kontext zu erschließen. Dabei haben die Vektorrepräsentationen, die größte Auswirkung auf die Ranking-Qualität so kann man sehen dass TFIDF den höchsten Anstieg gemacht hat von 0.25. Anschließend das Doc2Vec mit 0.6 und dann Word2Vec mit TFIDF von 0.4. Dicht gefolgt von LDA. Deutlich zu sehen ist das Word2vecaverage und Jaccard keine nennenswerte Auswirkung auf die Qualität hat, sprich man hätte sie, weglassen können. Was auffällt vom Ergebnis ist, das String basierte Methoden als reine Ähnlichkeitsberechnung für sich nicht viel taugen, aber als Kombination mit Features die Vektoren Repräsentieren äußerst hilfreich sind. Es fällt auf das Word2vecaverage keine gute Repräsentation für Sätze darstellt, da es sich nicht mit der Semantik im ganzen Satz beschäftigt, sondern nur mit jedem einzelnen Wort, dies ist nur hilfreich wenn man nur ein Wort angibt.

Aber in unsrem Fall haben wir mit kurze Texte, was die Sache mit der Ranking-Qualität erklärt. Auch BOW ist nicht gut allein zu verwenden, da es in diesem Modell nicht kümmert, welche Postion die Wörter im Text annimmt, somit die semantische Bedeutung mit ins Spiel bringt. TFIDF und Bow sind sich ähnlich, jedoch wird auf die semantische Bedeutung geachtet, da jeder Term im Text eine Gewichtung bekommt, was den Anstieg erklärt. Bei der Jaccard-Koeffizient ist es so, dass die semantische Ähnlichkeit nicht berücksichtigt, da die Texte als eine Menge gesehen wird und geguckt wird, wie viele gemeinsame Elemente die haben. Da sieht man, dass es kaum Einfluss auf die Verbesserung der Ranking-Qualität hat. Bei Sätze mit den gleichen Term ist es hilfreich. Lda ist ein Feature, was mal schlecht oder mal gut sein kann, denn der Nachteil ist man muss die passende Hyperparameter wählen und passend Themen deklarieren, um ein gutes Ergebnis zu erzielen. Als Ergänzung nützlich, jedoch sich nur auf dieses Feature zu verlasen nicht empfehlenswert. Jedoch achtet es sehr auf die Semantik, da einzelne Wörter zu mehrere Themen zugeordnet werden kann, was zu einer gute semantische Ähnlichkeit führen kann. Doc2Vec ist das beste Model für Ähnlichkeitsberechnung, da es sich um die Semantik für ganze Texte beachtet und nicht nur einzelne Wörter untersucht.

Bei meinem System gibt es natürlich Stärken und Schwächen. Die größte Schwäche ist, dass das System bei Wörter, die nicht im Textkorpus vorhanden sind, nicht die richtige Idee

vorschlagen kann, da dies für die Modelle unbekannt sind. Weil die Modelle können nur richtige Vektorrepräsentationen machen, wenn sie im Textkorpus drin sind. Bei kurze Texte und einzelne Wörter ist das System sehr gut, wenn die Vokabeln, dem System bekannt sind. Bei sehr langen Texte oder generell bei Texte spielt die Qualität eine Rolle, da hier die Schlüsselwörter(Wörter im Korpus vorhanden) eine Rolle spielen, wenn einige vorhanden kriegt man schon ähnliche Beschreibung. Die Frage, die man sich hier natürlich stellt ist, was ist die Lösung für dieses Problem. Einmal müsste man jedes entdeckte neue Wort sofort im Textkorpus speichern, dann die Modelle trainieren mit den neuen Vokabeln. Das führt dazu, dass enorme Rechenleistung beansprucht. Eine weitere Möglichkeit wäre, wenn man nach neue Features und Modelle sucht, die sich mit solchen Dinge befassen. Es gibt momentan ein Modell in Bereich NLP, welches sich mit diesem Problem beschäftigt. Die rekurrente neuronale Netze, die Texte überfliegen können und dann eine Vorhersage überprüfen können. Als Feature kann man andere Worteinbettungsmodelle nehmen, welche hier nicht vorgestellt wurden oder ein Convolutional Neural Network (kurz CNN), welche Sätze als Vektoren darstellen oder spezielle Verfahren, die die neuronale Netze verwenden, wie zum Beispiel Long short-term memory (LSTM). Auch bei der Aufbereitung der Dateien kann man noch effizientere Ergebnisse erzielen, in dem man zum Beispiel Part-of-speech-Tagging(POS-Tagging) verwendet. Hier werden Wörter und Satzzeichen deklariert zu bestimmte Wortarten. Die Quellen aus diesem Kapitel entstammen aus dem Tool WEKA und [3].

# 5 Webapplikation

## 5.1 Konzeption

Ziel ist es eine Webapplikation zu entwickeln und implementieren, die eine städtische Innovationsplattform darstellt. Dabei soll der Benutzer eine Idee eingeben und dann auf Suche klicken und er bekommt dann ähnliche Ideen vorgeschlagen und kann anschließend diese vorgeschlagene Ideen bewerten, wie ähnlich sie sind, sprich ob sie zum gewünschten Ergebnis passen. Anhand des Feedbacks kann man die Webapplikation dann so optimieren, dass die Resultate für vorgeschlagene Ideen immer besser wird.

Um so eine Applikation zu erstellen, ist eine Datenbank, ein Backend und ein Frontend zu gebrauchen und daraus ein Rest Web Service zu erschaffen. Bei einer Datenbank handelt es sich, um ein System, welche Daten speichern, verwalten, bearbeiten etc. kann. Diese Datenbank ist notwendig, damit die Bewertungen der Ideen gespeichert werden können als auch jederzeit abrufbar sind, um zu sehen, wie gut die Ergebnisse sind.

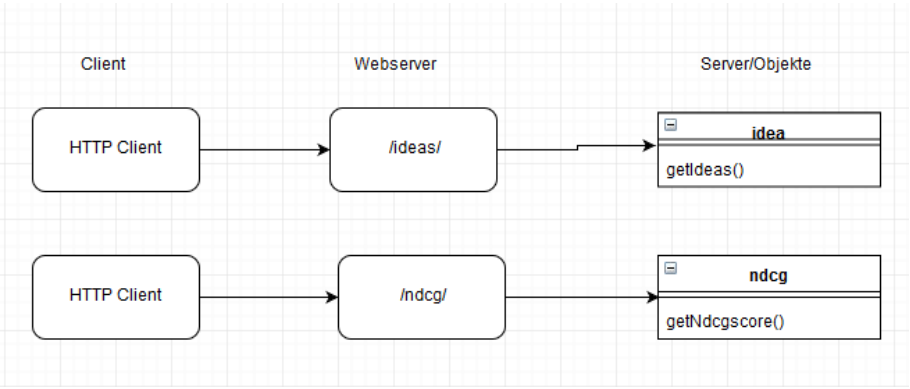
Als Backend wird der Teil im System bezeichnet, der für die Prozesse im Hintergrund verantwortlich ist. Dort werden die Abfragen zur Datenbank durchgeführt, Berechnungen durchgeführt oder bestimmte Skripte durchgeführt. Das Frontend ist grafische Benutzeroberfläche, bekommt nur die Resultate angezeigt, die im Hintergrund berechnet wurden. Im unserem Kontext ist die Oberfläche die Suchleiste, wo man die Idee eingibt und dann die Ergebnisse angezeigt bekommt.

Unser System baut auf die Architektur von Rest auf und steht für Representational State Transfer. Diese Architektur findet sehr häufig Anwendung in Suchmaschine, was in unserem Kontext sehr passt, da unsere Plattform eine Suchmaschine ist. Rest baut vier HTTP Methoden auf:

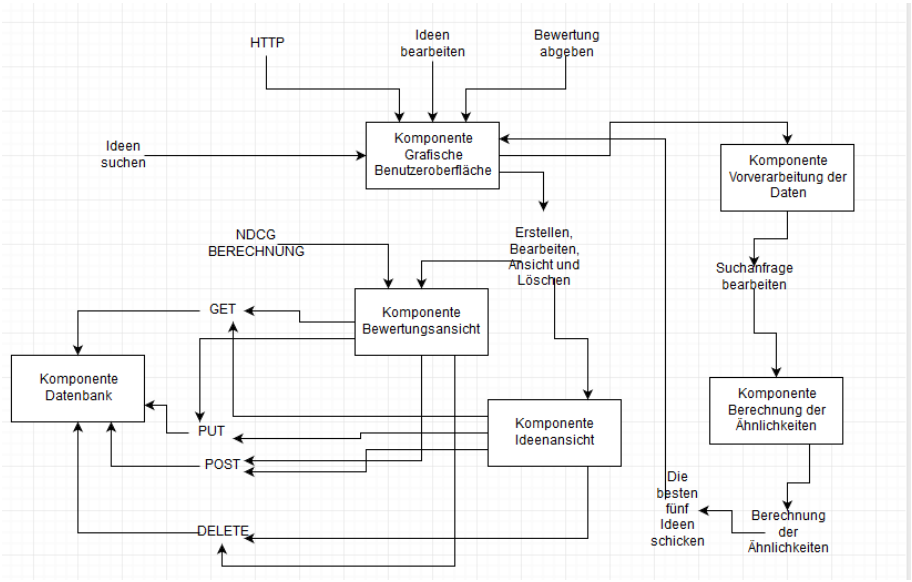
Einmal Get: Diese Methoden stellt Anfragen über die Ressourcen her, sprich unsrem Fall, fragen wir nach den vorgeschlagenen Ideen.
Die nächste Anweisung ist POST: Diese Methode sorgt dafür, dass eine Ressource hinzugefügt werden, sprich in unsrem Fall, wäre das zum Beispiel eine Idee zu werten oder neue Ideen hinzufügen.
Die nächste Anweisung ist PUT: Damit können neue Ressourcen erstellt werden oder bereits vorhandene Ressourcen verändert werden, also bei uns, wäre das die Bewertung zu verändern oder vorhandene Ideen zu verändern.
Die letzte Anweisung ist Delete: Damit können wir eine Ressource löschen. In unserem Kontext

bezogen, ist es zum Beispiel das Löschen einer Bewertung.

Hier ist nochmal ein Bild, wie ein Rest aussehen kann, in unserem Kontext:




Das jetzige Bild zeigt das Konzept der Architektur unseres System:




In diesem Abbild zeigen wir meine Anwendungsarchitektur, wo einzelne Komponente abgebildet sind und in, welcher Beziehung sie zu den anderen Komponenten stehen. Die Idee ist, dass die Webapplikation über HTTP aufrufbar ist und über eine Oberfläche die Anfragen über die Komponenten an die Datenbank zu machen, um die gewünschte Dateien zu bekommen. Außerdem soll es möglich sein, ein Ranking zu Ideen von der Benutzeroberfläche aus, zu speichern. Dann kann man diese Auswertung jederzeit von der Oberfläche es aus in Form einer Liste abrufen, um zu sehen, was für eine Ranking-Qualität in Form von NDCG bei bisherigen Suchanfragen ergeben hat. Dies erfolgt über die Komponente Bewertungsansicht. Das Bearbeiten von Ideen wird durch die Komponente Ideenansicht erledigt. Die Suchanfrage, um die besten Ideen zu der angegeben Beschreibung zu erhalten, wird über die Komponenten Vorverarbeitung der Daten und Berechnung der Ähnlichkeiten. Die Anfrage wird präpariert, so dass Ähnlichkeitsberechnungen durchgeführt werden können mit Hilfe von Algorithmen.

## 5.2 Implementierung der städtischen Innovationsplattform

**Voraussetzungen und Server** Die Webapplikation wurde unter Windows 10 durchgeführt. Bei der Entwicklungsumgebung handelt es sich um Eclipse, und das Projekt wurde als Maven Projekt deklariert. Maven ist ein Paketmanager für Java, dies wurde als Plugin in Eclipse installiert. Dadurch ist möglich alle Frameworks, die wir im Bereich NLP, neuronale Netzwerke und Rest- Architektur direkt vom Internet herunterladen. Das Frontend basiert auf die Programmiersprache Javascript und hat als Paketmanager npm. Dies sorgt dafür, dass alle notwendige Frameworks für Javascript aus dem Internet heruntergeladen wird. Für das Frontend wurde keine Entwicklungsumgebung benutzt hier wurde mit einem einfachen Editor gearbeitet, der dann compiliert wird, um das Frontend zu starten. Zu Beginn wurde die MySQL Datenbank konfiguriert, so dass das Backend auf die MySQL Datenbank zugreifen kann mit der essentiellen API MySQL-Connector.

**Implementierung auf Datenbankebene** Wie zuvor erwähnt, wurde als Datenbank MySQL genommen. Eine Open-Source Software für Datenbanken.Die Datenbank hat zwei relevante Tabellen einmal die Tabelle ïnput ". Diese enthält eine Id und eine Ideenbeschreibung zum Beispiel sieht, dass so aus:

| id | text |
|---|---|
| 5 | St. Pauli ohnehin schon stark belastet |

Diese Tabelle enthält alle Ideen, die dieser Plattform zur Verfügung steht. Hier werden die Beschreibungen verändert, gelöscht oder neue Ideen hinzugefügt. Die zweite Tabelle, die eine Rolle spielt ist die Tabelle "ndcg". Hier wird die beschriebene Idee als auch die vorgeschlagene 5 Ideen des Systems gespeichert als Id. Zudem werden auch die Bewertung der Ideen gespeichert und das daraus resultierende NDCG gespeichert. Man kann auch hier ein Kommentar zu dem Vorschlag geben.

| field | value |
|---|---|
| id | 4 |
| idea | sport |
| s | 212 |
| s2 | 840 |
| s3 | 555 |
| s4 | 700 |
| s5 | 600 |
| id1 | 1 |
| id2 | 1 |
| id3 | 1 |
| id4 | 1 |
| id5 | 1 |
| ndcg | 1 |

Diese Tabelle ist essentiell für die Bewertung von Ideen. Hier kann man nämlich prüfen, wie gut die vorgeschlagenen Ideen sind. Anhand der Scores kann man sich dann überlegen, ob das System gut ist oder eventuell bearbeiten werden muss. So viel zur Datenbank.

**Backend** Wie ich vorher erwähnt habe, wurde mit Java gearbeitet und als Backend wurde das Framework Spring verwendet. Es bietet eine sehr große Auswahl an Funktionalitäten für die Umsetzung eines Rest Webful Service. Zudem bietet dieses Framework sein eigener Server an. Es basiert auf den Tomcat-Server von Apache und kann für Webanwendungen verwendet werden. Das Besondere an diesem Framework, dass drei besondere Funktionalitäten anbietet, die für die Webapplikation nützlich sind. Einmal ist es die "Dependency Injection": Dies ermöglicht, dass Objekte nicht selber gesucht wird, da dies das Framework von alleine, es ordnet selber die Objekt zu den Ressourcen zu. Das nächste ist die Äspektorientierte Programmierung": Hier kann Transaktionen oder Sicherheiten vom wichtigen Programmcode fern halten. Eine dritte Funktionalität ist, dass Ressourcen automatisch aufgeräumt werden.

Entscheidend für die Sprachverarbeitung und Ähnlichkeitsberechnung, wurden diese Frameworks verwendet: Mallet, Deeplearning4j, Lucene und Apache common Text. Diese werden in dieser Reihenfolge erläutert:

Mallet bietet eine sehr große Sammlung von natürlichen Sprachvearbeitungsalgorithmen und Werkzeuge. Es kann für sehr viele Anwendungen verwendet werden, wie zum Beispiel Dokumentklassifizierung, Dokumentklustering, Informationsextraktion und Topic Modelling. Das letztgenannte ist für unsere Experimente sehr wichtig. Mallet bietet zwar keine Worteinbettungsvektoren an, jedoch bietet es sehr viele Implementation für LDA an. Es existieren verschiedene LDA: "Parallel LDA, DMR LDA, Hierarchical LDA, Labeled LDA, Polylingual Topic Model, Hierarchical Pachinko Allocation Model (PAM), Weighted Topic Model,LDA with integrated phrase discovery". Hier wurde das Parallel LDA gewählt. Der Vorteil bei Mallet ist , es enthält eigene Methoden für die Textverarbeitung, um sie für das LDA zu optimieren. Das Mallet wurde hauptsächlich für die Implementation von LDA gebraucht. Hier sind zwei Codeausschnitte. Das erste Zeigt, dass die Bearbeitung der Texte für das Lda und die zweite zeigt das Erstellen des LDA:

```
ArrayList<Pipe> pipeList = new ArrayList<Pipe>();
List<Instance> test = new ArrayList<>();
pipeList.add(new Input2CharSequence("UTF-8"));
Pattern tokenPattern = Pattern.compile("[\\p{L}\\p{N}_]+");
pipeList.add(new CharSequence2TokenSequence(tokenPattern));
pipeList.add(new TokenSequenceLowercase());
pipeList.add(new TokenSequenceRemoveStopwords(new File(stopListFilePath), "utf-8", false, false, false));
pipeList.add(new TokenSequence2FeatureSequence());
pipeList.add(new Target2Label());
SerialPipes pipeline = new SerialPipes(pipeList);
// Construct a new instance list, passing it the pipe
//  we want to use to process instances.
InstanceList instances = new InstanceList(pipeline);
```

```
// Create a model with 30 topics, alpha_t = 0.01, beta_w = 0.01
//  Note that the first parameter is passed as the sum over topics, while
//  the second is the parameter for a single dimension of the Dirichlet prior.
int numTopics = 30;
ParallelTopicModel model = new ParallelTopicModel(numTopics, 0.01, 0.01);

model.addInstances(instances);

// Use two parallel samplers, which each look at one half the corpus and combine
//  statistics after every iteration.
model.setNumThreads(4);

// Run the model for 2000 iterations and stop
model.setNumIterations(2000);
model.estimate();
```

Deeplearning4j bietet eine Sammlung an verschiedene Implementation von neuronale Netzwerke an vom Standard wie das mehrschichtiges Perzeptron bis hin zu rekurrente neuronales Netze, als auch Netzwerke wie das Convolutional Neural Network. Es bietet nahezu alle verschiedene Architekturen von neuronalen Netzwerke an, sowie die Datenverarbeitung für die Netze. Entscheidend für die Webapplikation war die Implementation vom mehrschichtiges Perzeptron, die das Framework anbietet, auch das komplette Paket für die natürliche Sprachverarbeitung. Hier bietet es die Implementation Word2Vec, Doc2Vec, Bow, TFIDF als auch die Tokenisierung, Lemmatisierung,und POS-TAGGING. an. Ein paar Codeausschnitte:

```
final int numInputs = 7;
int outputNum = 4;
int epochs = 5000;
long seed = 6;


MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
        .seed(seed)
        .activation(Activation.TANH)
        .weightInit(WeightInit.XAVIER)
        .updater(new Sgd(0.1))
        .l2(1e-4)
        .list()
        .layer(0, new DenseLayer.Builder().nIn(numInputs).nOut(6).build())
        .layer(1, new DenseLayer.Builder().nIn(6).nOut(6).build())
        .layer(2, new OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
                .activation(Activation.SOFTMAX).nIn(6).nOut(outputNum).build())
        .backprop(true).pretrain(false)
        .build();

//run the model
MultiLayerNetwork model = new MultiLayerNetwork(conf);
model.init();
model.setListeners(new ScoreIterationListener(100));

for( int i=0; i<epochs; i++ ) {
    model.fit(trainingData);
}
```

```
LabelsSource source = new LabelsSource("DOC_");

ParagraphVectors vec = new ParagraphVectors.Builder()
        .minWordFrequency(1)
        .stopWords(getStopwords())
        .iterations(100)
        .epochs(1)
        .layerSize(100)
        .learningRate(0.025)
        .labelsSource(source)
        .windowSize(5)
        .iterate(iter)
        .trainWordVectors(true)
        .trainSequencesRepresentation(true)
        .trainElementsRepresentation(true)
        .vocabCache(cache)
        .tokenizerFactory(t)
        .sampling(0)
        .build();

vec.fit();


WordVectorSerializer.writeParagraphVectors(vec, "pv.txt");
```

```
LabelAwareSentenceIterator iter = new LabelAwareFileSentenceIterator(rootDir);
AbstractCache<VocabWord> cache = new AbstractCache<>();
   TokenizerFactory t = new DefaultTokenizerFactory();
   t.setTokenPreProcessor(new CommonPreprocessor());
   TfidfVectorizer vec = new TfidfVectorizer.Builder().
         setIterator(iter).
         setMinWordFrequency(1)
         .setStopWords(getStopwords())
         .setTokenizerFactory(t)
         .setVocab(cache)
         .build();
   vec.fit();
```

Das Framework Lucene wurde schon angedeutet, dass es bei Backend als Textsuchmaschine gilt. Im Backend verwende ich, dass um eine gewissen Vorselektion durchzuführen, sprich es werden Dokumente erst gefiltert und die besten Dokumente auszusuchen. Mit den besten Dokumente wird dann weiter gearbeitet und Ähnlichkeitsberechnungen durchgeführt. Zusätzlich werden unbekannte Wörter oder Beschreibungen auch sofort gespeichert im Index. Bevor es zu dieser Selektion kommt, müssen die Daten erstmal indexiert werden und hier kommen die Stärken von Lucene zum Vorschein. Obwohl es sich um ein Open-Source Framework handelt, hat sehr gute Features die Performance des Systems verbessert. So verbraucht es sehr wenig RAM, also nur 1 MB HEAP. Des weiteren kann es Dokumente so schnell indexieren, wie im Batch und es braucht nur 20 Prozent des Volumens von ganzen Dokumente als Index. Weitere Vorteile sind, dass es verschiedene Suchalgorithmen existieren, es können einige Algorithmen implementiert werden oder durch Plugins weitere Algorithmen angewendet werden. Des weiteren kann man durch den Index Dokumente nach bestimmte Felder durchsuchen, wie zum Beispiel in nach Texte oder nur nach Ids. Man kann die Ranking manipulieren und eigene Gewichtungen festlegen. Es gibt noch viele weitere Optionen. Als Suchalgorithmus wurde das Okapi BM25 verwendet.

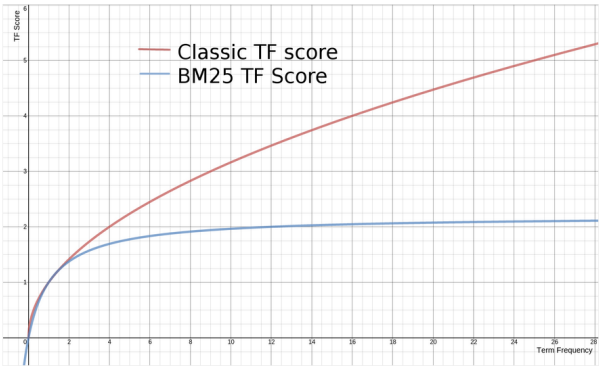
Das BM25 ist ein Suchalgorithmus und ist ein ähnliches Modell wie das BOW, sprich es achtet darauf wie häufig Terme in jedem Dokument auftauchen. Die Bewertungsfunktion besteht aus sehr viele Komponenten und Parametern. Gibt man eine Suchanfrage an, die einzelne Wörter, die in der Suchanfrage sind von q1,...qn, beträgt der BM25-Wert bei einem Dokument D:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)},$$

Bei f(qi,D) ist qi die Häufigkeit des Wortes im Dokument D, Der Betrag von D gibt die insgesamt Anzahl an Wörter, die im Dokument drin sind, avgdl ist die durchschnittliche Länge des Dokuments von der ganzen Dokumentensammlung berechnet wird. k1 und b sind frei wählbare Parameter. Untersuchungen haben gezeigt, um optimale Ergebnisse zu erzielen, soll k zwischen k1 zwischen 1.2 und 2 liegen und b soll, als festen Wert 0.75 haben. Danach kommt das IDF im Spiel, hier kriegen die Wörter eine Gewichtung:

$$\text{IDF}(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5};$$

N ist die insgesamte gesamte Anzahl von Dokumenten und n(qi) ist die Anzahl der Dokumente, die das Wort enthalten. Zu dieser Formel wurde eine Untersuchung gemacht und mit dem TFIDF verglichen:




Es zeigt, dass es höhere Werte, sprich mehr Relevanz und es erreicht nicht so schnell das Maximum. Her sind einige Ausschnitte von Lucene in 4 Schritten: 1.Indexieren

```
//Document document2 = createDocument(id, document.toLowerCase(), bow,
Document document2 = createDocument2(id, document.toLowerCase());
documents.add(document2);
```

```
//Let's clean everything first
writer.deleteAll();

writer.addDocuments(documents);
writer.commit();
writer.close();
```

2.Query und Suche

```
IndexSearcher searcher = createSearcher();
Directory dir = FSDirectory.open(Paths.get(INDEX_DIR));
IndexReader reader = DirectoryReader.open(dir);
TopScoreDocCollector collector4 = TopScoreDocCollector.create(11);
 Query q4 = new QueryParser("text", analyzer).parse(text.toLowerCase());
 searcher.search(q4, collector4);
 ScoreDoc[] hits4 = collector4.topDocs().scoreDocs;
```

3.Zeigen

```
for(int k=1;k<hits4.length;++k) {
    int docId = hits4[k].doc;
    Document d = searcher.doc(docId);
    System.out.println(d.getField("id"));
    String tempid = d.get("id");
       String temptext = d.get("text");
```

Das letzte Framework Apache Commons Text ist eine Bibliothek, welche Berechnung für Vektoren und Strings anbietet und womit man Ähnlichkeitsberechnungen durchführt. Dieses Tool wurde angewendet, um die Features zu erstellen für das neuronale Netzwerk.

Da im Backend das NDCG berechnet wird, wurde ein Code eingesetzt, um die Bewertung vom Nutzer auszuwerten. So wie im Kapitel 2 erläutert wurde, führt dieser Code das aus.

```
public static double calculateNDCG(List<Integer> realData, List<Integer> predictionData) {
    double dcg = 0;
    double idcg = calculateIDCG(realData);

    if (idcg == 0) {
        return 0;
    }

    for (int i = 0; i < predictionData.size(); i++) {

        // compute NDCG part
        int rank = i + 1;

        dcg += (Math.pow(2, predictionData.get(i)) - 1.0) * (Math.log(2) / Math.log(rank + 1));
    }

    return dcg / idcg;
}
```

**Frontend** Kommen wir nun zum Frontend. Wie bereits erwähnt, wurde als Programmiersprache Javascript verwendet. Dabei wurde als Framework Angular genutzt. Dieses Framework ermöglicht es Rest anfragen zum Backend zu machen und Ressourcen wieder zu empfangen. Des weiteren ist es für moderne Browser gemacht, sodass es nicht zu Problemen kommt. Das gute an diesem Framework ist, eine Bibliothek dafür gibt, die Sammlung vieler verschiedener User Interfaces anbietet. Man kann Tabellen erstellen und diese so modifizieren, dass dies nur eine bestimmte Anzahl an Dokumenten anzeigt, dass man Dokumente selektieren kann, um sie dann zu bearbeiten. Es bietet noch viele andere Features an. Man kann auch sofort CSV Dateien exportieren. So bei meinem Frontend ist es mögliche alle Ideen in einer Plattform zu zeigen, als auch neue zu erstellen, dabei werden die neue Ideen sofort von Lucene indexiert. Man kann auch sie bearbeiten oder sie von der Datenbank löschen. Hier sind einige Ausschnitte vom Interface:

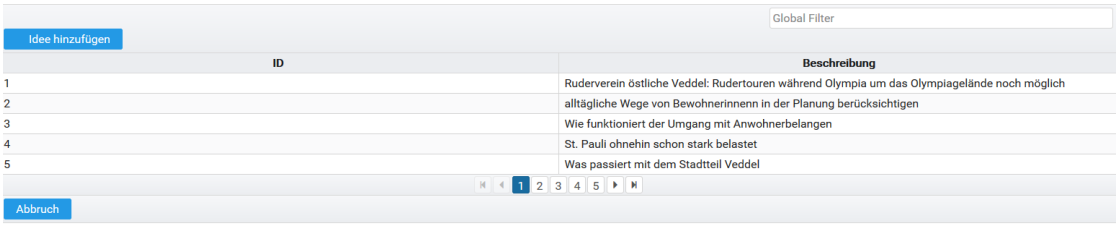



Abbildung 5.1: Ausschnitt 1

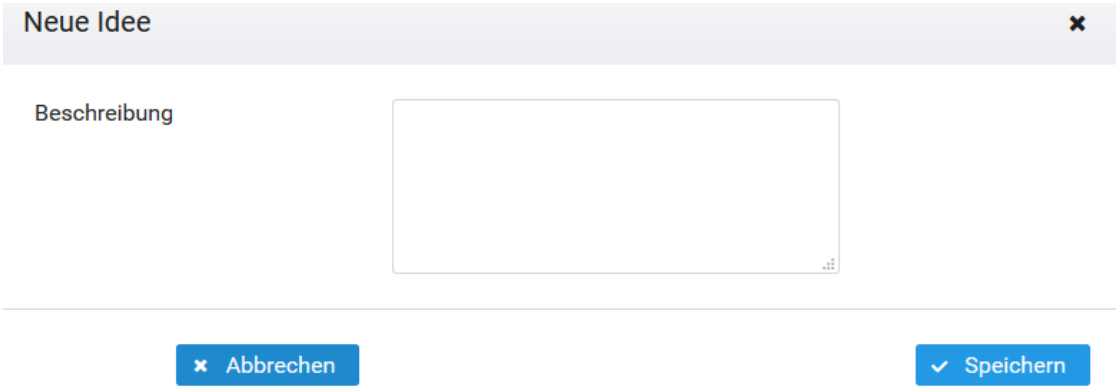



Abbildung 5.2: Ausschnitt 2

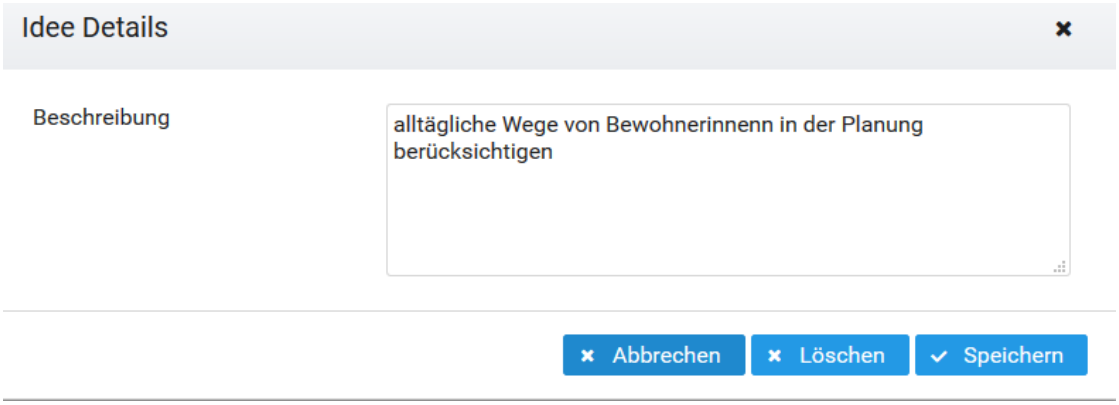



Abbildung 5.3: Ausschnitt 3

Zudem man kann seine Suchanfrage starten und das Backend liefert dann 5 ähnliche Ideen. Des weiteren kann man diese Ideen bewerten. Dann wird diese Auswertung zum Backend geschickt, um das NDCG zu berechnen. Später kann man die bei einer Übersicht, alle Bewertung von Suchanfragen sehen und gegebenenfalls löschen. Wie man weiter unten sehen kann, gibt man sein Text an und klickt auf "checkButton. Dann bekommt man die vorgeschlagene

Ideen. Dann kann man die Idee eine Punkteskala von 0 bis 3 Punkte geben, dann klickt man auf bewerten und es wird im Backend das NDCG berechnet. Wenn man auf Tabelle klickt, kriegt man eine Übersicht von allen Ideen, die bewertet wurden zur Idee, die der Benutzer eingegeben hat. Hier sind einige Ausschnitt von meinem Interface:

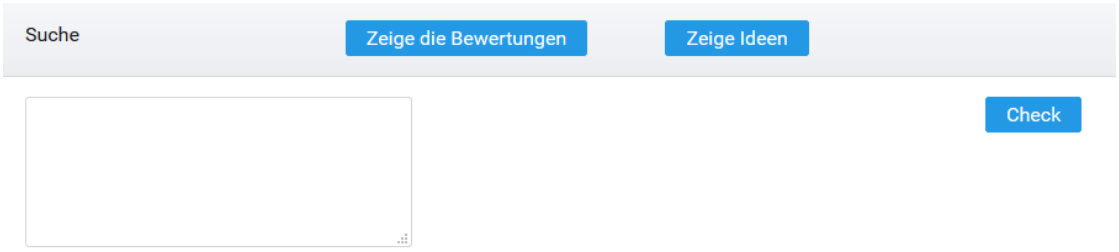



Abbildung 5.4: Ausschnitt 4

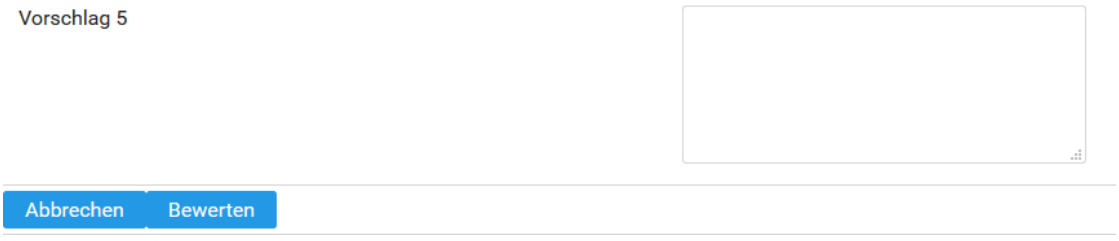



Abbildung 5.5: Ausschnitt 5

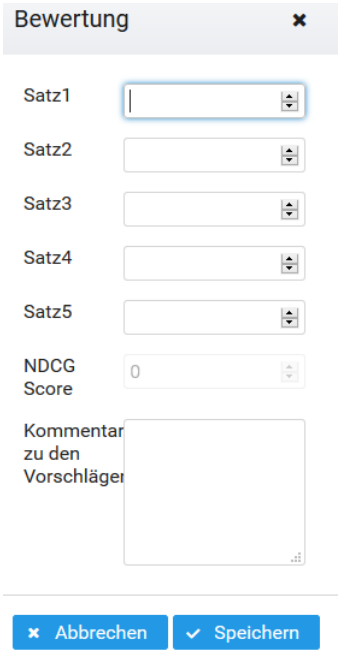



Abbildung 5.6: Ausschnitt 6

Abbildung 5.7: Ausschnitt 7

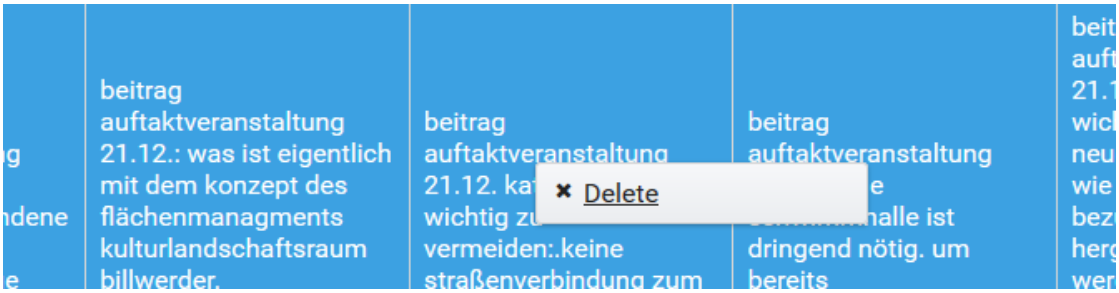
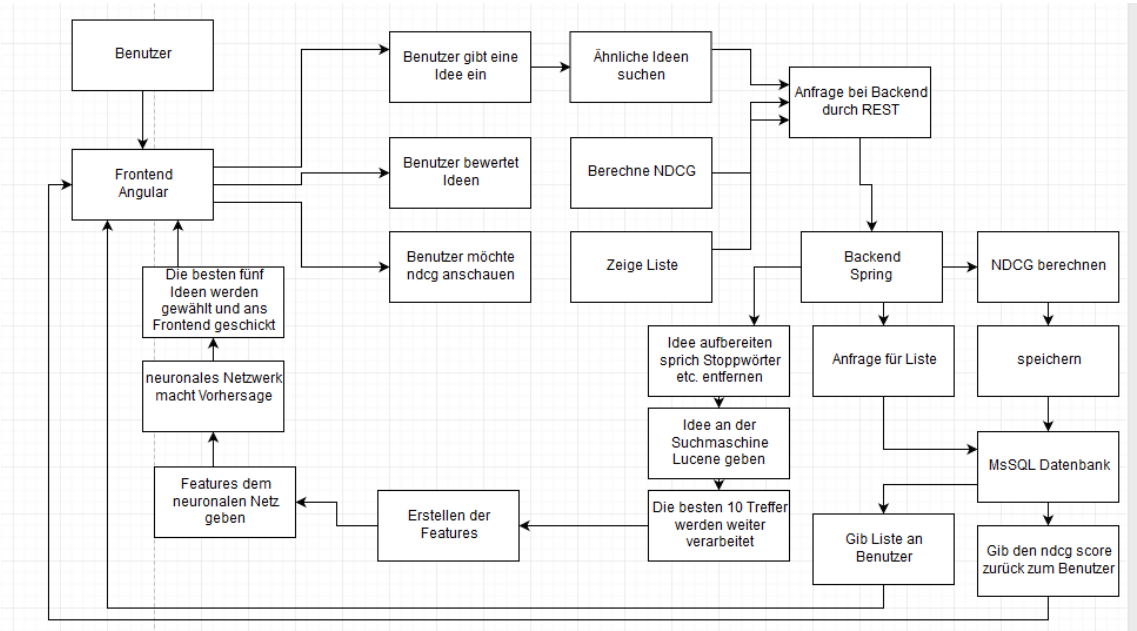



Abbildung 5.8: Ausschnitt 8

**Übersicht** Da ich ein einen sehr langsamen Rechner besitze und damit auch die Suche nicht 4 Stunden dauert bis das System alle 2000 Dokumente durchgegangen ist, Features erstellt hat und die passende Ideen zu finden. Habe ich Lucene als Suchmaschine genommen, der für mich im Voraus alle notwendige Dokumente liefert. Dabei wurden vor allem die Top 10 ähnlich Dokumente verwendet. Als Algorithmus verwendet Lucene das BM25 Similarity. Das ist eine Ranking-Funktion die nach einer Suchanfrage die relevante Dokumente liefert. Daraufhin wurden zu dem 10 Dokumente die Features erstellt. Als Feature wurde hier aufgrund der Rechnerleistung nur Features gewählt die schnell Resultate liefern. BOW, DOC2VEC, Levensthein 1, Levensthein 2, Longest common substring, Longest common substring 2 und Jaccard-Koeffizient verwendet. Bei den string basierte Methoden wurden Verfahren benutzt die detaillierter analysieren als normale Variante. Anschließend werden die Features an meinem neuronalen Netzwerk gegeben und eine Vorhersage getroffen. Danach werden die besten fünf Ergebnisse an dem Benutzer gesendet. Hier eine Übersicht:




## 5.3 Ausblick

Man konnte noch so viele Sachen implementieren und experimentieren. Jedoch würde der Zeitraum für eine Bachelorarbeit nicht ausreichen. Erstmal hätte ich noch weitere Suchalgorithmen von Lucene benutzt. Denn es bietet noch "Language modelsünd "Divergence of randomnessän. Da Performance bei Plattformen sehr wichtig ist, hätte ich noch Experimente durchgeführt für die Vorwahl, um nur mit den besten Dokumente Ähnlichkeitsberechnungen durchzuführen. Dann hätte man evaluiert, wie gut diese sich für die Vorwahl eignen. Natürlich alle 28 Features experimentieren, sprich alle einzeln probieren und dann anschließend verschiedene Kombinationen auszuprobieren. Dann die Evaluation machen, und gucken, wel-

che von der Performance her sehr schnell sind und gute Ergebnisse liefern. Natürliche auch verschiedene Techniken für überwachtes Lernen zu implementieren, nicht nur andere neuronale Architekturen, sondern auch Modelle die Klassifikation beherrschen. Wie zuvor erwähnt, eine gründlichere Aufbereitung der Texte mit POS-Tagging. Natürlich noch mehr Features, aber auch aus anderen Bereichen der natürlichen Sprachverarbeitung. Hier wäre interessant mit Hilfe neuronaler Netze aus Texte Vektoren zu erstellen und mit den Ähnlichkeitsberechnungen durchzuführen. Ein weiterer interessanter Punkt ist, Strukturen in den Ideen zu finden und dann mit einem Algorithmus, die Texte zu clustern. Auch dafür verschiedene Techniken vom unüberwachten Lernen zu implementieren. Natürliche, damit es verschiedene Benutzer mit unterschiedliche Rollen, die Plattform verwenden können, Rechte verteilen und Logins entwerfen. Die Quellen in diesem Kapitel entstammen aus meinem Programm und: [2][3][4][5][6][7][8][9].

# 6 Fazit

In dieser Bachelorarbeit wurde mit Hilfe der Datensätze für die Innovationsplattform, viele Experimente konzipiert, durchgeführt und evaluiert. Dadurch konnte eine kleine Webapplikation erstellt und implementiert werden, die es ermöglicht zu einer vorgeschlagene Idee, andere Ideen vorzuschlagen aufgrund deren berechneten Ähnlichkeit. Man hat festgestellt, dass die Wahl der Features für ein Modell Auswirkungen hat auf die semantische Ähnlichkeit und auf die Ranking-Qualität. Man konnte sehen, dass das Hinzufügen von Features nach jeder Iteration zu einer Verbesserung des NDCG geführt hat. Wir haben ein Ergebnis von 0.95 erreicht, was sehr gut ist. Man muss jedoch bedenken, dass die Kombination von Vektorrepräsentationen mit string basierte Methoden sorgfältig gewählt werden muss, damit ein gutes Ergebnis erreicht wird. Einzeln wäre das Ergebnis vermutlich nicht annähernd so gut wie die Kombination aus beidem. Unser System zeigt, dass es gut bei Texten mit bekannte Wörter im Korpus gut ist, jedoch wenn es um unbekannte Wörter geht, tut sich das System sehr schwer. Des weiteren muss man beachten, dass Vektorrepräsentationen und neuronale Netzwerke sehr viel Rechenleistung kostet und wenn die Datenbank eine hohen Menge an Datensätze verfügt, wird die Suche nach der korrekten Ideen sehr lange dauern. Hier muss man sich genauer untersuchen, welche Repräsentation von Vektoren und string basierte Verfahren am besten sind und gleichzeitig auch nicht so viel Rechenleistung kostet. Auch die Architektur von neuronalen Netzen können bearbeitet werden, um schnellere Ergebnisse zu suchen. Sehr gut bei meinem System finde eine Suchmaschine mit eingebaut zu haben, um an Rechenleistung zu sparen. Hier sollte, wenn man ein System entwickelt für Innovationsplattformen, ein Schwerpunkt setzen und nach bessere Methoden suchen, die korrekte Daten zu filtern und Features zu erstellen. Worauf ein Schwerpunkt setzen könnte, ist noch die Vorverarbeitung der Texte in dem man noch POS-Tagging anwendet. Für diesen Kontext sind TFIDF,DOC2VEC,LCS, LDA und Word2VeCtf sehr zu empfehlen, für die Berechnung von Ähnlichkeiten. Wichtig ist hierbei aber, dass alle Vorkehrungen der Vorverarbeitung der Texte durchgeführt werden, um die optimale Ranking-Qualität zu bekommen. Auf die Fragen bezogen eignet sich das Framework Deeplearning4j, Lucene und Mallet hervorragend für die Berechnung von Ähnlichkeiten, da deren Algorithmen für die Vektorrepräsentation und Kosinusähnlichkeit sehr gut zur Erstellung einer Ideenplattform passt. Für die Webapplikation sind Primeng und Spring sehr geeignet, da es gute Tools bietet für die Anfragen und Darstellung für Metadaten. Natürlich sind string basierte Verfahren als Ergänzung sehr hilfreich, jedoch sollte bei einer Innovationsplattform der Schwerpunkt auf die Vektorrepräsentationen gelegt werden und deren Distanzmetrik.

# 7 Referenzen

Die Informationen und Bilder entstammen aus folgenden Quellen und von meiner technischen Betreuerin Sarah Kohail:


[1] https://www.inf.uni-hamburg.de/inst/ab/wists/teaching/theses.html

[2] https://deeplearning4j.org/

[3] https://www.cs.waikato.ac.nz/ml/weka/

[4] https://spring.io/

[5] https://www.primefaces.org/primeng/

[6] https://angular.io/

[7] https://www.mysql.com/de/

[8] http://mallet.cs.umass.edu/

[9] http://lucene.apache.org/

[10] https://www.tensorflow.org/tutorials/representation/word2vec

[11] https://towardsdatascience.com/word-embedding-with-word2vec-and-fasttext-a209c1d3e12c

[12] Quoc Le QVL@GOOGLE.COM, Tomas Mikolov TMIKOLOV@GOOGLE.COM Google Inc, 1600 Amphitheatre Parkway, Mountain View, CA 94043: Distributed Representations of Sentences and Documents

[13] Christian Häusler Autumn term 2012/13 Instructor: Prof. Dr. Anke Dreiling Supervisor: Prof. Dr. Thomas Hanne: Methods for Determiningthe Similarity of Documents

[14] Minmin Chen Criteo Research Palo Alto, CA 94301, USA m.chen@criteo.com: EFFICIENT VECTOR REPRESENTATION FOR DOCUMENTS THROUGH CORRUPTION

[15] University of Melbourne: UniMelb NLP-CORE: Integrating predictions from multiple

domains and feature sets for estimating semantic textual similarity

[16] Yang Liu1, Chengjie Sun1, Lei Lin1, Yuming Zhao2 and Xiaolong Wang Harbin Institute of Technology, Harbin Heilongjiang 150001, China 2 Northeast Forestry University, Harbin Heilongjiang 150040, China: Computing Semantic Text Similarity Using Rich Features

[17] Dr. K. Anuradha1, Himaja Indukuri, Tilak Putta Department of Computer Science and Engineering. Gokaraju Rangaraju Institute of Engineering and Technology, Hyderabad, India: FEATURE ANALYSIS FOR SEMANTIC TEXTUAL SIMILARITY

[18] http://datameetsmedia.com/bag-of-words-tf-idf-explained/

[19] http://dovgalecs.com/blog/matlab-simple-tf-idf/

[20] https://medium.freecodecamp.org/an-introduction-to-part-of-speech-tagging-and-the-hidden-markov-model-953d45338f24

[21] Hussein T. Al-Natsheh, Lucie Martinet, Fabrice Muhlenbach, Djamel A. Zighed, Universite de Lyon, Lyon 2, ERIC EA 3083, 5 Avenue Pierre Mendes France, F69676 Bron Cedex - France CNRS, ISH FRE 3768, 14 Avenue Berthelot, F-69007 Lyon, France CESI EXIA/LINEACT, 19 Avenue Guy de Collongue, F-69130 Ecully, France UJM-Saint-Etienne, CNRS, Laboratoire Hubert Curien UMR 5516, F-42023 Saint Etienne, France, UdL at SemEval-2017 Task 1: Semantic Textual Similarity Estimation of English Sentence Pairs Using Regression Model over Pairwise Features

[22] Jeremy Ferrero Compilatio 276 rue du Mont Blanc 74540 Saint-Felix, France LIG-GETALP Univ. Grenoble Alpes, France, Laurent Besacier LIG-GETALP Univ. Grenoble Alpes, France, Didier Schwab LIG-GETALP Univ. Grenoble Alpes, France, Frederic Agnes Compilatio 276 rue du Mont Blanc 74540 Saint-Felix, France: Cross-Language Plagiarism Detection Methods for Semantic Textual Similarity CompiLIG at SemEval-2017 Task 1: Cross-Language Plagiarism Detection Methods for Semantic Textual Similarity

[23] Sarah Kohail and Chris Biemann, Matching, Re-ranking and Scoring: Learning Textual Similarity by Incorporating Dependency Graph Alignment and Coverage Features, Language Technology Group ,Computer Science Department, Universität Hamburg

[24] Thabet Slimani, Computer Science Department Taif University and LARODECLab: Description and Evaluation of Semantic similarity Measures Approaches

[25] Euripides G.M. Petrakis, Giannis Varelas, Angelos Hliaoutakis, Paraskevi Raftopoulou, Department of Electronic and Computer Engineering Technical University of Crete (TUC) Chania, Crete, GR-73100, Greece: Design and Evaluation of Semantic Similarity Measures for Concepts Stemming from the Same or Different Ontologies

[26] Danushka Bollegala, Mitsuru Ishizuka,Yutaka Matsuo, The University of Tokyo Hongo 7-3-1, Tokyo 113-8656, Japan, National Institute of Advanced, Industrial Science and Technology: Measuring Semantic Similarity between Words Using WebSearch Engines

[27] Blinov P. D., Kotelnikov E. V., Vyatka State Humanities University, Kirov, Russian Federation: Semantic Similarity for Aspect Based Sentiment Analysis

[28] Mohammad Taher Pilehvar and Roberto Navigli, Department of Computer Science Sapienza University of Rome: An Open-source Framework for Multi-level Semantic Similarity Measurement

[29] Martyna Spiewak, Piotr Sobecki and Daniel Karas, National Information Processing Institute, al. Niepodlegosci 188b, 00-608 Warsaw, Poland:OPI-JSA at SemEval-2017 Task 1: Application of Ensemble learning for computing semantic textual similarity

[30] Fanqing Meng, Wenpeng Lu*, Yuteng Zhang, Jinyong Cheng, Yuehan Du, Shuwang Han, Institute of Intelligent Information Processing, School of Information, QiLu University of Technology, Jinan, Shandong, China: QLUT at SemEval-2017 Task 1: Semantic Textual Similarity Based on Word Embeddings

[31] Fanqing Meng1, Wenpeng Lu*1, Yuteng Zhang1,Ping Jian,Shumin Shi,Heyan Huang, School of Information, QiLu University of Technology, Jinan, Shandong, China, School of Computer, Beijing Institute of Technology, Beijing, China: QLUT at SemEval-2017 Task 2: Word Similarity Based on Word Embedding and Knowledge Base

[32] I-Ta Lee, Mahak Goindani, Chang Li, Di Jin, Kristen Marie Johnson, Xiao Zhang, Maria Leonor Pacheco, Dan Goldwasser, Department of Computer Science, Purdue University West Lafayette, IN 47907: PurdueNLP at SemEval-2017 Task 1: Predicting Semantic Textual Similarity with Paraphrase and Event Embeddings

[33] https://de.wikipedia.org/wiki/Tf-idf-Maß

[34] https://machinelearningmastery.com/gentle-introduction-bag-words-model/

[35] http://www.tfidf.com/

[36] https://www.elephate.com/blog/what-is-tf-idf/

[37] https://medium.com/@lettier/how-does-lda-work-ill-explain-using-emoji-108abf40fa7d

[38] https://de.wikipedia.org/wiki/LatentDirichletAllocation

[39] https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24

[40] https://en.wikipedia.org/wiki/Multilayerperceptron

[41] http://deeplearning.net/tutorial/mlp.html

[42] https://medium.com/scaleabout/a-gentle-introduction-to-doc2vec-db3e8c0cce5e

[43] https://rare-technologies.com/doc2vec-tutorial/

[44] https://en.wikipedia.org/wiki/Jaccardindex

[45] http://www.statisticshowto.com/jaccard-index/

[46] https://en.wikipedia.org/wiki/Word2vec

[47] https://en.wikipedia.org/wiki/Longestcommonsubstringproblem

[48] https://www.geeksforgeeks.org/longest-common-substring-dp-29/

[49] https://en.wikipedia.org/wiki/Levenshteindistance

[50] http://www.levenshtein.de/

[51] https://en.wikibooks.org/wiki/AlgorithmImplementation/Strings/Levenshteindistance

[52] https://www.kaggle.com/dansbecker/cross-validation

[53] ECNU at SemEval-2017 Task 1: Leverage Kernel-based Traditional NLP features and Neural Networks to Build a Universal Model for Multilingual and Cross-lingual Semantic Textual Similarity, Junfeng Tian, Zhiheng Zhou1, Man Lan1„ YuanbinWu, Department of Computer Science and Technology, East China Normal University, Shanghai, P.R.China,Shanghai Key Laboratory of Multidimensional Information Processing

[54] https://de.wikipedia.org/wiki/Kosinus-Ähnlichkeit

[55] https://en.wikipedia.org/wiki/Discountedcumulativegain

[56] https://en.wikipedia.org/wiki/Bag-of-words-model

[57] https://commons.apache.org/proper/commons-text/

[58] https://openreview.net/pdf?id=SyK00v5xx
[59] https://3.bp.blogspot.com/-5ZSDw3tP1ho/V1WU4aNqPbI/AAAAAAAAAP0/5phmpChHYIoTyVztSxZIapC1
[60] https://siongui.github.io/2017/04/04/go-levenshtein-distance-recursive-implementation/
[61] https://www.kdnuggets.com/2016/07/text-mining-101-topic-modeling.html
[62] https://de.wikipedia.org/wiki/Dirichlet-Verteilung
[63] https://www.safaribooksonline.com/library/view/getting-started-with/9781786468574/ch04s04.html
[64] Buch: Mastering Java Machine Learning von Dr. Carlotta Domeniconi
[65] Buch: Deep Learning: Practical Neural Networks with Java von Yusuke Sugomori, Bostjan Kaluza, Fabio M. Soares und Alan M. F. Souza
[65] Online Tool für die Erstellung der Diagramme: https://www.draw.io/

# Abbildungsverzeichnis

## Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Bachelorstudiengang Software-System-Entwicklung selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den 17.09.2018 Konstantinos Tsiamis

## Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Hamburg, den 17.09.2018 Konstantinos Tsiamis