Patent - Triz40 Mappingansätze

René Brückner

30. September 2019

Zusammenfassung

Im Laufe des Seminars *Widersprüche und Management-Methodiken* der Uni Leipzig im Sommersemester 2019 wurden die Studierenden immer wieder vor die Problematik der Zuordnung von Patenttexten zu Innovationsmethoden der 40 TRIZ Prinzipien gestellt. Als Beitrag des Forschungsseminars hat sich der Autor das Ziel gestellt, mittels Textprocessing und Information Retrieval Methoden, eine Auswahl möglicher der 40 TRIZ Methoden automatisiert zu Patenttexten zu ermitteln. Zu Beginn werden Grundlagen abgesteckt, welche für das weitere Verständnis vonnöten sind. Anschließend wird eine Zielstellung formuliert und auf die Ausgangssituation eingegangen. Auf Grundlage dieser beiden Kapitel werden drei Ansätze und deren entsprechenden Einschränkungen so wie Probleme vorgestellt. Aufgrund des Umfangs des Moduls ist eine Evaluierung der Ansätze nicht durchführbar. Zum Schluss wird ein Ausblick für eine mögliche weitere Bearbeitung des Themas gegeben.

Universität Leipzig
Sommersemester 2019

Fakultät für Mathematik und Informatik
Seminar Widersprüche und Management-Methodiken

Inhaltsverzeichnis

Kapitel 1

Einleitung

Mithilfe von Altshuller's Buch *40 principles: TRIZ keys to innovation.* [0] lassen sich Konflikte im Innovationsprozess überwinden. Seine 40 Methoden fanden Anwendung in vielen modernen Technologien und Erfindungen. Spuren dieses Einflusses lassen sich erkennen, indem man Patenttexte auf Parallelen zu seinen Prinzipien untersucht. Dies war Hauptbestantteil des Seminars *Widersprüche und Management-Methodiken* der Uni Leipzig im Sommersemester 2019. Die Recherche entsprechender Patente stellte dabei eine nicht triviale Aufgabe für die Teilnehmer so wie den Autor. Eine automatisierte Zuordnung erleichtert die Forschung und die Analyse bestehender Konfliktmanagementmethoden. Moderne Texmining und Information Retrieval Methoden haben eine Vielzahl von Algorithmen zum Textvergleich als Grundlage. Der Autor hat sich das Ziel gestellt, solche Algorithmen anzuwenden, um zu erproben, wie eine automatische Zuordnung der 40 TRIZ Prinzipien zu Patenttexten umgesetzt werden könnte. Dazu stellt der Autor zunächst kurz verwendete Grundlagen vor, um im folgenden 3 Ansätze zu erläutern.

Kapitel 2

Grundlagen

Zum Verständnis der Ansätze sind Grundlagen im Bereich des Textminings, Information Retrieval sowie mathematische Ähnlichkeitsmaße notwendig. Im Folgenden werden diese kurz benannt und definiert jedoch nicht im Detail erläutert. Für genaue Ausführungen verweist der Autor auf die Vorlesung *Textmining*, *Information Retrieval* sowie die Bücher *Corpus-based and knowledge-based measures of text semantic similarity* [5] und *Modern information retrieval*[2] .

2.1 Textmining / Information Retrieval Begriffserklärung

- **Tokenizing**
  Extrahieren einzelner Wörter (tokens) eines Satzes

- **Stopwort-Entfernung**
  Entfernen semantisch unwichtiger Wörter (z.B. *und, der, die, das, als,* etc.) via Listen

- **Stemming**
  Meist regelbasierte Reduzierung des Worts auf den Wortstamm (*Häuser Haus*, *gesprungen springen ...*)

- **POS Tagging**
  Part Of Speech - Tagging ist die Bestimmung der Wortart im Satz

2.2 Mathematische Grundlagen

Eine naive Form der Ähnlichkeitdefinition ist der sogenannte *Jaccard-Index* [4]. Aufbauend auf die Mengenlehre betrachtet man den Schnitt zweier Mengen.

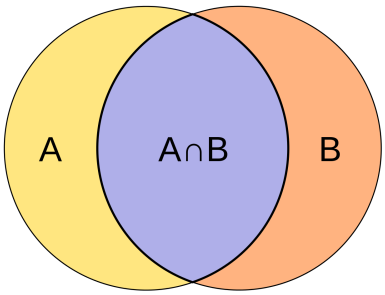

Abbildung 2.1: Veranschaulichung Schnittmenge

In Abbildung 1 wird der Schnitt in einem Venn diagramm veranschaulicht.

$$J_\delta(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

Abbildung 2.2: Jaccard-Index

Die Quantifizierung dieser Ähnlichkeitsform zwischen 0 und 1 ist daraus abgeleitet in Abbildung 2 als Formel repräsentiert.
Die Kosinusähnlichkeit[9] ist ein besseres Maß zur Ähnlichkeitsbestimmung und fundiert auf der linearen Algebra.

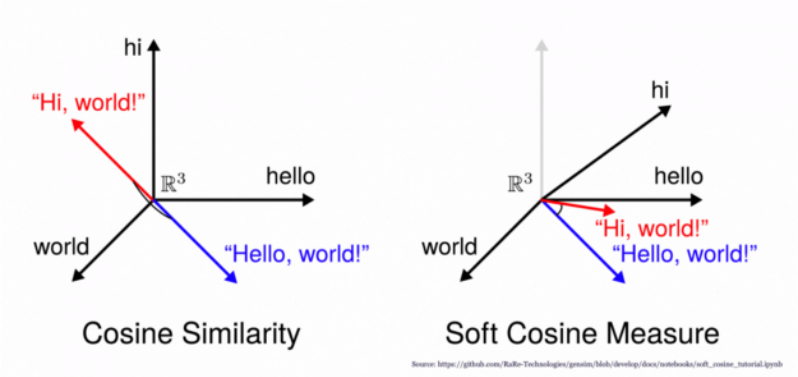

Abbildung 2.3: Veranschaulichung Kosinusähnlichkeit[8]

Dafür wird der Winkel zweier Vektoren, die das Dokument repräsentieren, ermittelt.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Abbildung 2.4: Kosinusähnlichkeit

Mathematisch ausgedrückt entspricht die Kosinusähnlichkeit der in Abbildung 2.4 dargestellten Formel.

Kapitel 3

Zielstellung

Grundlegend ist das Ziel, semantische Ähnlichkeiten zwischen einem Patenttext und allen 40 TRIZ Methoden zu quantifizieren. Im Fokus stand für den Autor eher die Erprobung verschiedener Verfahren als eine genaue Einordnung. Die Zuordnung zu möglichen Kandidaten ist dabei wichtiger als eine exakte Bestimmung aller angewandten Methoden. Zunächst wurde die Beschaffenheit der verschiedenen Texte betrachtet, um eine Abschätzung der Umsetzbarkeit zu ermöglichen. Patenttexte sind in einem sehr speziellen Vokabular verfasst. Beschreibende technische Bezeichnungen und Erläuterungen ihrer Interaktionen untereinander sowie mit einem potenziellen Nutzer stehen im Mittelpunkt. Aufgrund dieser Sonderheiten hat jeder Text markante Eigenschaften, welche einen Vergleich ermöglichen.

Die TRIZ40 Methoden hingegen haben eine allgemein gehaltene Beschreibung ohne Konkretisierung von entsprechenden Bauteilen. Verwendet wurde eine detaillierte Übersetzung von Altshuller's *40 principles: TRIZ keys to innovation*[0] ins englische sowie ins deutsche von *www.triz40.com*[3]. Aufgrund dieser allgemein gültigen Formulierungen besteht die Möglichkeit, Parallelen zwischen der entsprechenden Methode und einem Patenttext zu ziehen.

Ziel ist nun, ein Verfahren zu finden, welches möglichst viele treffende Kandidaten der 40 TRIZ Methoden einem Patent zuordnen kann. Aufgrund von fehlenden Daten zum Vergleich und mangelnder Bearbeitungszeit ist es dem Autor nicht möglich, eine genaue Evaluierung der Ergebnisse vorzunehmen. Vielmehr sollen die Ansätze einen Einstieg für eine weitere Bearbeitung dieses Themas bieten.

Kapitel 4

Ansätze

Für die Präsentation während des Seminars sowie zur Erprobung wurden beispielhaft drei Ansätze in Python implementiert. Zur Erleichterung wurde ein Modul erstellt, welches die 40 TRIZ Methoden in Englisch sowie Deutsch enthält. Diese wurden Patenttexten gegenübergestellt. Inspiration der Verwendung dieser Textvergleiche und zugleich Grundlage bildete der Artikel *Quick review on Text Clustering and Text Similarity Approaches*[7] sowie *Text Similarities : Estimate the degree of similarity between two texts*[1]. Die erstellten Skripts sind im Github-Repository[1] des Seminars zu finden. In den folgenden Kapiteln werden sie genauer erläutert.

4.1 TF-IDF

Bei der Termfrequenz - Inverse Dokumenten Frequenz (TF-IDF) wird das Auftreten einzelner Terme eines Dokuments in einer Matrix abgebildet. Genauer wird das Vorgehen in *Modern information retrieval: A brief overview*[2] erläutert. Es handelt sich um eine Methode des Textmining und des Information Retrieval welches ursprünglich für Suchmachinen verwendet wird.

Zur Umsetzung wurde die Python Implementierung von *sklearn* verwendet. Diese unterstützt das Entfernen von Stopwörtern, welche das Ergebnis verfälschen. Beispielsweise würden Texte, welche viele Stopwörter verwenden automatisch als ähnlicher eingeschätzt werden. Die Implementierung erfolgte wie in Folgendem Ausschnitt.

```
1 def tfidf(text, compareset, stopwords):
2     vect = TfidfVectorizer(min_df = 1, stop_words = stopwords)
3     tfidf = vect.fit_transform([text] + compareset)
4     return((tfidf * tfidf.T).A[0][1:])
```

Die Funktion gibt ein Array mit 40 Zahlenwerten zwischen 0 und 1 zurück, wobei Eins die größte und Null die kleinste Ähnlichkeit ausdrückt. In Zeile 2 wird das minimale Vorkommen der zu berücksichtigenden Wörter festgelegt und zu entfernende Stopwörter

[1] https://github.com/wumm-project/Leipzig-Seminar/tree/master/2019-07-04/Mappingansaetze/

übergeben. In Zeile 3 wird der Vergleich ausgeführt und in einer Matrix abgebildet, wovon schließlich in Zeile 4 die erste Reihe ausgegeben wird.

Mehrdeutigkeiten oder andere Formulierungen können das Ergebnis stark verfälschen, da diese nicht berücksichtigt werden.

4.2 Wordnet

Ein etwas komplexerer Ansatz ist die Realisierung eines Vergleichs beider Texte unter der Berücksichtigung von möglicher Mehrdeutigkeit und semantischer Ähnlichkeit der Wörter. Wordnet ist ein lexikalisch semantisches Netzwerk, welches 1998 von George Miller im Buch *WordNet: An electronic lexical database*[6]. Es ermöglicht Bedeutungsähnlichkeit zwischen Wörtern zu quantifizieren. Da die Zielstellung jedoch Textvergleiche und nicht Wortvergleiche vorraussetzt, bedient sich der Autor an einer, in *Corpus-based and knowledge-based measures of text semantic similarity* [5] vorgestellten, Formel für Textvergleiche.

$$\mathrm{Sim}(T_1, T_2) = \frac{1}{2}\left(\frac{\sum\limits_{w \in T_1} \mathrm{maxSim}(w, T_2) \cdot idf(w)}{\sum\limits_{w \in T_1} idf(w)} + \frac{\sum\limits_{w \in T_2} \mathrm{maxSim}(w, T_1) \cdot idf(w)}{\sum_{w \in T_2} idf(w)}\right)$$

Abbildung 4.1: Textvergleich via Wordnet[5]

Dies ermöglicht Texte über ihre eigentliche Formulierung hinaus semantisch gegenüberzustellen. Dieses Vorgehen wurde mithilfe der Python Bibliothek *nltk*, welche Wordnet beinhaltet wie in folgendem Quelltext-abschnitt umgesetzt.

Problematisch bei diesem Ansatz ist, dass der Vergleich der Synonyme abhängig von dem von Wordnet bereitgestellten semantischen Netz ist. Eine technische Beschreibung von Patenten hat ein sehr spezielles Vokabular, dessen Synonyme nicht im Netz enthalten sein könnten.

```
1  #sentence similarity using Wordnet
2  def sentence_similarity(sentence1, sentence2):
3      # Tokenize and tag
4      sentence1 = pos_tag(word_tokenize(sentence1))
5      sentence2 = pos_tag(word_tokenize(sentence2))
6      # Get the synsets for the tagged words
7      synsets1 = [tagged_to_synset(*tagged_word) for tagged_word in
       ↪  sentence1]
8      synsets2 = [tagged_to_synset(*tagged_word) for tagged_word in
       ↪  sentence2]
9      # Filter out the Nones
10     synsets1 = [ss for ss in synsets1 if ss]
11     synsets2 = [ss for ss in synsets2 if ss]
12     score, count = 0.0, 0
13     # For each word in the first sentence
14     for synset in synsets1:
15         # Get the similarity value of the most similar word in the other
           ↪  sentence
16         scores = []
17         for ss in synsets2:
18             simscore = synset.path_similarity(ss)
19             if simscore == None:
20                 scores.append(0)
21             else:
22                 scores.append(simscore)
23         best_score = max(scores)
24         # Check that the similarity could have been computed
25         if best_score is not None:
26             score += best_score
27             count += 1
28     # Average the values
29     score /= count
30     return score
31
32 def symmetric_sentence_similarity(sentence1, sentence2):
33     return (sentence_similarity(sentence1, sentence2) +
       ↪  sentence_similarity(sentence2, sentence1)) / 2
```

In Zeile 4 bis 12 werden die Wörter der Dokumente extrahiert, mit ihrem POS-Tag versehen und entsprechende Synonymmengen erstellt. In der Schleife ab Zeile 14 wird die Berechnung des einen Summanden der zuvor benannte Formel umgesetzt. Schließlich nutzt die Funktion in Zeile 32 die Summanden um die Berechnung durchzuführen.

4.3 Word2Vec

Als letzter Ansatz wurde ein machinelearning basierter Ansatz verfolgt. Bei Word2Vec handelt es sich um ein sogenanntes Wordembedding. Wordembeddings sind ... Dazu wird zunächst ein Beispielmodell trainiert. Als Trainingsdaten wurden in diesem Fall ausschließlich die Texte der 40 TRIZ Methoden verwendet. Empfehlenswert wäre ein Datensatz angemessener Größe, um die komplexen Zusammenhänge der technischen Beschreibungen und Methoden zu erfassen.

Die Funktionalitäten der Modellerstellung sowie der eigentliche Vergleich, mittels Kosinusähnlichkeit, wurden mit der Python Bibliothek *gensim* realisiert.

```
1  def trainModel():
2      sentences = []
3      for methode in methoden:
4          sentences.append(methode.split(' '))
5      model = gensim.models.Word2Vec(sentences, min_count=1)
6      model.wv.save('./models/model.bin')
7
8  def w2vCompare(s1, s2, wordmodel):
9      s1wordsset = set(s1.split())
10     s2wordsset = set(s2.split())
11     for word in s1wordsset.copy():
12         if (word not in wordmodel.vocab):
13             try:
14                 s1wordsset.remove(word)
15             except KeyError:
16                 pass
17     for word in s2wordsset.copy():
18         if (word not in wordmodel.vocab):
19             try:
20                 s2wordsset.remove(word)
21             except KeyError:
22                 pass
23     return wordmodel.n_similarity(list(s1wordsset), list(s2wordsset))
```

Die Funktion *trainModel()* benuzt die Stringoperation Splitt um die Texte zu Token umzuwandeln. Diese werden genutzt, um ein Modell zu trainieren und selbiges zu speichern. Die Funktion *w2vCompare* bekommt als Parameter zwei zu vergleichende Texte und ein Modell, um eine Schnittmenge aus den Vokabulars als Verktorraum aufzuspannen und mittels Kosinusähnlichkeit zu vergleichen.

Der Vorteil dieses Ansatzes ist, dass er die Schwächen der beiden letzten Vorgehen berücksichtigt. Jedoch geschieht dies aufgrund der Abhängigkeit eines zu trainierenden Modells. Wie bereits zuvor erwähnt sollte die Modellerstellung wesentlich mehr Texte beinhalten.

Kapitel 5

Ausblick

Die vorgestellten Ansätze sind nicht in der Lage ein komplexes Mapping gut zu realisieren. Jedoch bieten sie ein Einblick in die Möglichkeiten einer Umsetzung. Im Grunde ist die vorgestellte Problemstellung der Klassifizierung sehr gut geeignet für ein neuronales Netz. Neuronale Netze sind state-of-the-art Methoden zum Lösen von Klassifizierungsproblemen. Dazu könnte auf dem in 4.3 vorgestellten Ansatz aufgebaut werden. Für das Seminar war dies jedoch nicht umsetzbar, da die Qualität der Klassifizierung mittels eines neuronalen Netzes immer von der Menge der Trainingsdaten abhängt. Um einen großen Trainingsdatensatz zu erstellen müsste eine händische Klassifizierung vorgenommen werden. Alternativ könnte man einen der anderen Ansätze verwenden, um eine Vorauswahl zu treffen, welche dann evaluiert in den Trainingsdatensatz mit einbezogen wird.

Anwendungsmöglichkeiten für diese Verfahren wäre die Patentrecherche sowie weitere Forschung bezüglich angewandter Innovationsmethoden. Auch eine Untersuchung, ob ein Patentkandidat notwendige Innovation aufweist, wäre vorstellbar. Zuletzt könnte ein bestehendes neuronales Netz um Konfliktarten erweitert werden, um entsprechende Innovationsmethoden zur Konfliktlösung vorzuschlagen.

Insgesamt ist hervorzuheben, dass dieser Bereich der Innovationsforschung im Bezug auf Methoden der modernen Informatik, speziell bezüglich Big Data und Machine Learning, noch sehr viel Potenzial mit sich bringt.

Quellenverzeichnis

[1] Sieg Adrien. *Text Similarities : Estimate the degree of similarity between two texts*. 2018. URL: https://medium.com/@adriensieg/text-similarities-da019229c894 (besucht am 30. 09. 2019) (siehe S. 8).

[0] Genrich Altshuller. *40 principles: TRIZ keys to innovation*. Bd. 1. Technical Innovation Center, Inc., 2002 (siehe S. 3, 7).

[2] Ricardo Baeza-Yates, Berthier Ribeiro-Neto u. a. *Modern information retrieval*. Bd. 463. ACM press New York, 1999 (siehe S. 4, 8).

[3] Olivier Goguel. *triz40.com*. 2014. URL: http://www.triz40.com (besucht am 30. 09. 2019) (siehe S. 7).

[4] Paul Jaccard. *Lois de distribution florale dans la zone alpine*. 1902 (siehe S. 5).

[5] Rada Mihalcea, Courtney Corley, Carlo Strapparava u. a. „Corpus-based and knowledge-based measures of text semantic similarity“. In: *Aaai*. Bd. 6. 2006. 2006, S. 775–780 (siehe S. 4, 9).

[6] George A Miller. *WordNet: An electronic lexical database*. MIT press, 1998 (siehe S. 9).

[7] Maali Mnasri. *Quick review on Text Clustering and Text Similarity Approaches*. 2016. URL: http://www.lumenai.fr/blog/quick-review-on-text-clustering-and-text-similarity-approaches (besucht am 30. 09. 2019) (siehe S. 8).

[8] Vít Novotný. *Finding similar documents with Word2Vec and Soft Cosine Measure*. 2019. URL: https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/soft_cosine_tutorial.ipynb (besucht am 30. 09. 2019) (siehe S. 6).

[9] Grigori Sidorov u. a. „Soft similarity and soft cosine measure: Similarity of features in vector space model“. *Computación y Sistemas* 18.3 (2014), S. 491–504 (siehe S. 5).