UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

An ontology-based approach to model qualitative world knowledge extracted from product reviews for use in QA systems

Thomas Huber

Matriculation number: 67287

"Knowledge Extraction and Representation for Text Entailment Inference"
Seminar Topic, SS 2018
Chair of Digital Libraries and Web Information Systems
University of Passau

Table of Contents

Chapter1 Introduction

One approach to find or reject textual entailment between two texts is to use world knowledge infer the entailment. World knowledge refers to specific knowledge such as *An apple is a fruit.* or *Fire is hot.* Using this information it becomes possible to infer entailment between the text pairs.

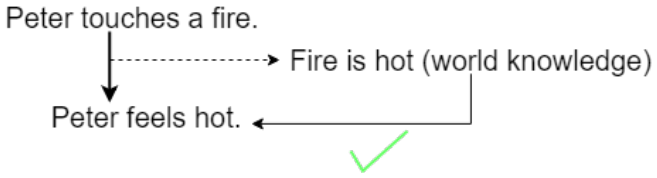

Figure 1.1: Example usage of world knowledge to infer textual entailment

To have this sort of knowledge available for use in a textual entailment system it is necessary to extract it and then represent the knowledge in some way.

An example of such a work is [SFH18]. The work makes use of WordNet [Mil95] to create a *knowledge graph* over the definitions contained therein. A graph navigational algorithm is then used to find paths between pairs of nodes, where the nodes represent knowledge extracted from the WordNet definitions, to find textual entailment between pairs of texts and use the path taken in the knowledge to create a human readable justification of why there is an entailment, if an entailment is found.
The knowledge found in WordNet however is limited. The definitions are rather short and do not contain instance specific knowledge. The definition for *cellphone* from WordNet is as follows:

> a hand-held mobile radiotelephone for use in an area divided into small sections, each with its own short-range transmitter/receive

This definition does not include that modern phones have built-in cameras or the capability to run apps, among other things. Even if such information was included, it does not help when working with specific instances of a type, such as a *specific* cellphone.
To alleviate this problem I have built a system that extracts instance specific, qualitative world knowledge from product reviews over specific products based on [SBC+07] and have added the extracted information into the knowledge graph used in [SFH18] to answer simple queries over specific product instances.
The category of products that is used to showcase the system is *cellphones*. The goal of this seminar topic is not to build a full-fledged system but rather show how the definitional knowledge graph can be extended with information extracted from product reviews and how this knowledge can be used in a textual entailment context, by showcasing a very basic Question Answering system.

Chapter2 Literature Review

2.1 Recognizing and Justifying Text Entailment through Distributional Navigation on Definition Graphs

My work primarily builds on the work described in [SFH18]. A detailed explanation follows in chapter 3 as part of the project description.

2.2 Red Opal: Product-Feature Scoring from Reviews

In [SBC+07] the authors describe a system that is capable of extracting *features* from reviews of products on online retailer platforms. The extracted features are ranked according to their own formula. The general idea is that features *F* of products can only be nouns. The features F have to occur more often in all reviews of category *C* than they do in written text of the language of the reviews. A more detailed explanation of how I use this work in my own follows in chapter 3.

2.3 Effective Collaboration in Product Development via a Common Sharable Ontology

In [MBB05] the authors describe how to build an ontology over products in a product development environment. They build an ontology that includes information about which specific parts a product contains to aid in the development of new products. As one of the goals of my work was to extend the already existing WordNet knowledge graph the parts of the paper describing the idea behind an ontology and how to build one were not very useful to this work.
The authors describe how they use the query mechanism of the Protégé [1] platform for simple queries over their ontology. Indeed this works for very basic queries such as *What are the design features of the "tee" part?* as used in the paper.
The authors give an example of a complex query but mention that it is not straightforward to perform. Furthermore their approach fails however for more complex queries or use cases, such as *Which phones take the nicest pictures?* where there is no direct or obvious path to the target. In this particular case the knowledge of which part of a phone can take pictures (the camera) has to be connected to the act of *taking a picture*. Using Protégé's query mechanism it is possible to find a connection between *phone* and *camera* but one has to know which feature can satisfy the user's request. The main difference between the query mechanism used by the authors and my approach is that the user does not need to have background knowledge to formulate the query in such a way that the system can work with it.

2.4 Informed Recommender: Basing Recommendations on Consumer Product Reviews

In [AZSD07] the authors describe how they extract features from product reviews. They build an ontology of the products and how good or bad the customers think specific features (called *concept* in the paper) for those products are. This ontology is then used in a recommender system
The difference to the feature extraction component used by me, based on [SBC+07], is that the approach in this paper is based on machine learning. They train a classifier that labels sentences in the reviews into three categories: *good*, for sentences that describe a feature that the

[1] https://protege.stanford.edu/

reviewer thinks is good, *bad* for sentences that describe a feature that the reviewer thinks is bad, and *quality*, for sentences that describe the reviewers skill level. They use the "Rule Induction Kit for Text" (RIKTEXT) to extract rules for the three categories: *fast* indicates a "good" text for example. While the general idea is very similar to the Red Opal ([SBC+07]) system, the general approach is very different. To make use of their work, one has to create and label training data. The Red Opal approach only takes into account the frequency of the words and as such works without re-training or the need to create new training data to account for previously unknown features.
The labelled sentences are then the basis for the "concept extraction", which is comparable to the feature extraction in Red Opal. For their approach the authors manually extract the feature and define synonyms. The review sentences are then classified using the trained classifier and for each feature an entry in the ontology is created, showing how the reviewer likes specific product features.
The approach described here is not nearly as flexible as the Red Opal approach for feature extraction as it requires the creation of training data and creation of a synonym database which is very expensive. For this reason, I have used the Red Opal approach to extract both features and qualitative knowledge of the features from product reviews.

Chapter3 Project Description

3.1 Crawler for product data

To collect sufficient amounts of data for the feature extraction I have built a webcrawler using the Scrapy [1] framework. The target was the *drills* category on Amazon. 36375 reviews were crawled over the course of about 10 hours from 109 products.
The speed of crawling has to be slow, otherwise there is the risk of getting banned from Amazon [2].
Due to the relatively slow speed and the limited number of products extracted during that time, I have looked for a different source of data to be able to build the feature extraction faster.
I have found a suitably large dataset of product reviews on Kaggle [3]. It includes 450212 total reviews over 4518 products in the *phone* category.

Table 3.1: Datasets overview

Dataset	number of reviews	number of products
Drills, using own crawler	36375	109
Kaggle phone dataset	450212	4518

For my work I have used only the Kaggle dataset. Both the crawler I have built and the extracted reviews for products in the drills category are included in the code, see apendix A.

3.2 Feature extraction using Red Opal

The core ideas of the feature extraction as discussed in [SBC+07] are the following:

[1] https://scrapy.org/, accessed 2018-06-20
[2] https://doc.scrapy.org/en/latest/topics/practices.html#avoiding-getting-banned, accessed 2018-06-20
[3] https://www.kaggle.com/PromptCloudHQ/amazon-reviews-unlocked-mobile-phones/data, accessed 2018-06-20

- a feature of a product can only be a noun
- the positions of the occurrences of a feature in a product review are irrelevant (bag-of-words assumption)
- the features discussed in the reviews occur more often than they do in other written or spoken text

The authors of the paper then develop a formula that can calculate the probability that a certain word occurs randomly the amount of times it does in all product reviews. The lower this probability, the higher the probability that it is a feature of the products.
The texts are processed using spaCy. It has good performance for POS-tagging which is the main component used in this work [4]. I use the Wikipedia frequency counts made available by the University of Leipzig [5] to have a frame of reference for the word frequency in written English as required by the Red Opal component.
Further I have made a major adjustment to the feature extraction: Only words that occur in the Wikipedia frequency corpus are considered. There are a lot of spelling errors in the reviews. Generally the algorithm of Red Opal is supposed to be resistant to them but in my experience this is not always the case when a lot of high scoring features are considered (the top 100 or even top 1000). As the goal is to extract features from products to extend an existing knowledge graph it makes sense to exclude words that are incorrectly spelled and thus not actually words. Another possibility would be to perform spelling correction on incorrectly spelled words but due to the time limitations of the seminar project this option was not explored. Table 3.2 shows the three highest scoring features and their score for the phone dataset.

Table 3.2: Highest scoring features

Feature	ln(P(feature))
phone	-15126199.8514816
screen	-1790234.63804627
battery	-1685233.74694203

Phone is found because it is a noun and, as can be expected of reviews of phones, occurs very often and is thus the highest scoring feature by far. The others make sense as features of phones: Phones can have a screen and a battery.
The feature extraction has been implemented from scratch based on the cited paper using Python 3.6 and the results are stored in a database using Django [6], a Python web framework with database support.

3.2.1 Insertion into WordnetGraph

As currently the only extracted features come from phones, the only node in the original WordnetGraph is *wnn:cellular_telephone__cellular_phone__cellphone__cell__mobile_phone*.
A new RDF property is added, *ns1:has_feature* and triples in the known subject, predicate, object format added, where the object is one of the features found in the feature extraction. For the insertion into the existing graph rdflib [7] was used. The idea of shaping the triples this way comes from [**SHF18**].

[4] https://spacy.io/models/en#en_core_web_sm, accessed 2018-06-20
[5] http://wortschatz.uni-leipzig.de/en/download/, accessed 2018-06-20
[6] https://www.djangoproject.com/
[7] http://rdflib.readthedocs.io/en/stable/

Table 3.3: RDF example

Subject	Predicate	Object
wnn:cellular_telephone__cellular_phone__cellphone__cell__mobile_phone	ns1:has_feature	camera, battery, screen...

3.3 Product scoring

The found features (see section 3.2) are used to calculate so called *Product Feature Scores*. They show how good a certain feature is given a specific product. This is used later to perform the product recommendation, refer to section 3.4.3.
I have added an additional score, based on the one that was defined in the paper.

$$\text{Let } l(p) = \text{number of reviews of product p} \tag{3.1}$$

$$m(p) = \begin{cases} 1, \text{if } l(p) \text{ top } 50\% \text{ \# reviews} \\ 0.75, \text{if } l(p) \text{ top } 90\% \text{ \# reviews} \\ 0.5, \text{if } l(p) \text{ top } 95\% \text{ \# reviews} \\ 0.25, \text{if } l(p) \text{ top } 99\% \text{ \# reviews} \\ 0.1, \text{otherwise} \end{cases} \tag{3.2}$$

Table 3.4: Products ordered by number of reviews

Product	# reviews	modifier
A	30	0.75
B	40	0.75
C	50	1

$$mod\_s(p,f) = m(p) * s(p,f) \tag{3.3}$$

Product A has a modifier of 0.75 because it is not in the upper half of products in terms of number of reviews (the top 2), but it is in the top 90% (top 3, 0.9 * 3 rounded up).
This means that the 1% of reviews with the lowest number of reviews get a modifier of 0.1 and thus have their score lowered drastically. The idea behind this is to reward products that have a high number of reviews. A high number of reviews that assign a positive rating to a certain feature indicate that the score for this feature of this particular product is more meaningful or consistent than a product with only one review that assigns a positive rating to the same product.
As mentioned in the previous section 3.2, the extracted product feature scores are stored in the database.

3.4 Implementation of graph navigation using WordnetGraph

In this section I will outline how I have implemented the graph navigational algorithm as described in [**SFH18**] and how the triples added in section 3.2 play a role. For the sake of brevity the explanation of the original algorithm will be kept short and I will instead focus on the details of my implementation.
To find and justify textual entailment the algorithm attempts to find a path in the knowledge graph of the Wordnet definitions between two nodes, where nodes represent sets of synonyms, so called *synsets*. The edges between the nodes come from the knowledge graph, and can be *supertype* (apple has supertype fruit), *has differentia quality* (oil furnace is a furnace with differentia quality 'burns oil'), *has origin location* and others.
From the start node the algorithm finds all connected nodes and checks the semantic relatedness between the target node and the connected nodes. To calculate the semantic relatedness, I am using Indra [**SSB+18**]. Indra is a framework that can calculate word embeddings and semantic relatedness between pairs of words. A similar work is [**PP06**] but due to familiarity with

Indra I have opted to use that. As the path between the source node and the target node is not known the semantic relatedness between pairs of nodes is used to measure the validity of an edge. If an edge leads to a node that has a low semantic relatedness with the target then most likely following this edge will not eventually lead to the target.
The next nodes to be visited come from the nodes with high enough relatedness' WordNet definitions. From those, the so called headwords are extracted (refer again to [**SFH18**]). They are the most relevant words in the definition. I extract them using dependency parsing using the previously mentioned spaCy framework [8].
Here I have made an experimental change to how the algorithm works: For supertype relations instead of extracting the main noun of the definition of the connected node I instead use the node itself as the next node.
An example:

- *apple* has supertype *fruit*
- fruit has the definition: *the ripened reproductive body of a seed plant*
- the extracted noun would be *seed plant* or *plant* and therefore the next node
- instead, I use *fruit* as the next node

This change was made as a direct connection is the supertype relation indicates that all the properties and relations of the supertype node, the less specific node, apply to the subtype, the less specific node. In the example above, this would mean that all properties that hold for 'fruit' also hold for 'apple', as 'apple' is more specific than 'fruit', and all apples are fruit.
The currently visited node and the next possible nodes are concatenated and added to a stack. This represents the paths taken so far in the navigation. The algorithm continues from the last node in the path and continues as outlined until it reaches the target node. It terminates if a maximum number of paths between source and target is found or alternative if the maximum depth of the search is reached.

3.4.1 Usage of product features in the graph

As I have mentioned in subsection 3.2.1 I have added product features for phones into the graph. They added triples are considered in the navigational algorithm and open up new paths or shorten existing paths. Figure 3.1 an example.

[8] https://spacy.io

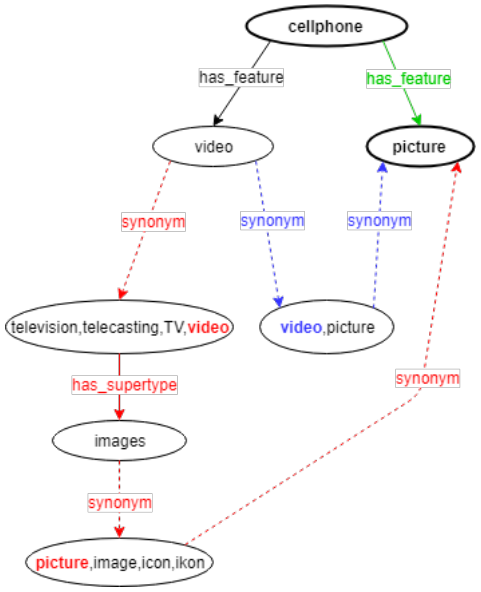

Figure 3.1: Example of three paths found between 'cellphone' and 'picture'

The trivial case is the path cellphone → picture. *picture* was found as a feature of cellphones by the feature extraction (see section 3.2).
A non-trivial case is the path cellphone → video → images → picture.

- *video* was found as a feature of cellphones...
- ... and is part of the synset *television,telecasting,TV,video*...
- ... which has the supertype *images* in the original knowledge graph...
- which is a synonym of *picture*, the target node

The parameters for the navigational algorithm are as follows:

- Start node: cellphone.n
- End node: picture.n
- Minimum relatedness: 0.33
- Maximum number of paths: 3
- Maximum path depth: 6

With the parameters specified my implementation of the navigational algorithm has found no path between *cellphone* and *picture*. From the WordNet definition for *cellphone*[9] it becomes clear that the trivial case of a direct connection is not possible regardless of implementation of the algorithm, as neither *picture* or a synonym thereof appear in the definition. It can therefore be concluded that the extension of the knowledge graph with features extracted from product reviews is a valid approach that opens up new paths.

3.4.2 Evaluation of the navigational graph algorithm

To compare my implementation in Python with the original implementation I have used the BPI [10] dataset, as it is also used in the original paper. The metrics are taken directly from [**SFH18**].

Table 3.5: Confusion matrix for BPI dataset

	Precision	Recall	F1
Original	0.65	0.54	0.59
My system	0.54	0.68	0.6

Table 3.6: Evaluation on BPI dataset

	Yes	No
Yes	67	58
No	31	93

The way textual entailment is checked is based on how it is described in the cited paper: Overlapping words for both text and hypothesis are removed and for the remaining set of words the k-highest scoring pairs (in terms of semantic relatedness, calculated using Indra) are kept and paths searched between them using the navigational algorithm. k is the maximum of the size of the two sets of non-overlapping words of text and hypothesis. Words with a low inverse document frequency (idf) are removed as well, as they can be reached from almost any node in the graph (see [**SFH18**]).
I have added an additional restriction: I keep only verbs, nouns and proper nouns. Adjectives

[9] WordNet: a hand-held mobile radiotelephone for use in an area divided into small sections, each with its own short-range transmitter/receive

[10] http://www.cs.utexas.edu/users/pclark/bpi-test-suite/

appear in the graph but only as the object in the triples and not as their own nodes with supertypes, differentia quality etc. To have a clear start and end they should either be verbs or nouns as both of them appear as *WordNetNounSynset* and *WordNetVerbSynset* in the graph, respectively, with supertypes and properties.
Because of this added limitation some of the pairs used for the evaluation do not have pairs of words that can be used for the algorithm. Those pairs are then considered to have no entailment. The confusion matrix with those empty pairs excluded from the evaluation can be seen in table 3.7.

Table 3.7: Confusion matrix, BPI dataset without empty pairs

Precision	Recall	F1
0.71	0.68	0.7

Table 3.8: Evaluation BPI dataset without empty pairs

	Yes	No
Yes	67	27
No	31	28

21 text-hypothesis pairs with entailment were unable to be checked due to not having pairs of words after removing overlapping words and keeping only nouns and verbs. Meanwhile 71 pairs without entailment were excluded. This indicates that my restriction of keeping only nouns and verbs in addition to the other restrictions might be valid as significantly more pairs without textual entailment were excluded than pairs with textual entailment. The metrics for this confusion matrix can be seen in table 3.8. The metrics improve by a significant amount. One way this could be used is to use the navigational algorithm to check for textual entailment if there are pairs of words that the algorithm can use and to use a different method for those text-hypothesis pairs where too many words are filtered out.

3.4.3 Product recommendation

As an example of how textual entailment in conjunction with product feature scores can be used in a real system I have built a simple product recommender. One example query is *Which cellphones take the nicest pictures?*.
I have analysed a few possible queries for product recommendation and also taken into account the queries mentioned in **[MBB05]** (see section 2.3). By doing this, I have noticed that the two main ways queries are formulated are those that focus on an action, such as *charging* in the case of phones, and those that focus on a noun, such as *pictures* or *apps*.
Using the dependency parsing of spaCy I have analysed those two types of queries to extract suitable word pairs for use in the navigational algorithm. Figure 3.2 shows an example parse tree.

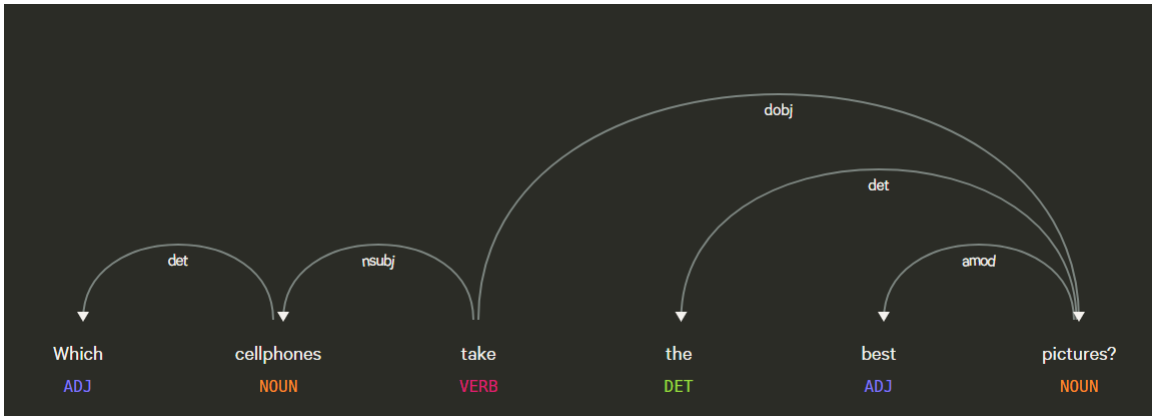

Figure 3.2: Parse tree example using https://explosion.ai/demos/displacy

Based on the analysis using dependency parsing, I make the following assumptions about the queries:

- the *NSUBJ* is always the category of product
- if the *DOBJ* is a noun then that is the target feature
- if the *DOBJ* is not a noun then the *ROOT* is the target and always a verb

I consider the problem of finding features that match the query to be a textual entailment problem. The query and the product features can be transformed into text and hypothesis pairs like so:

- Text: A *PRODUCT-CATEGORY* has feature *SOME-FEATURE*.
- Hypothesis: A *PRODUCT-CATEGORY* can *TARGET-VERB*.
- Hypothesis: A *PRODUCT-CATEGORY* has *TARGET-NOUN*.

Using the example from figure 3.2, this would become:

- Text: A *cellphone* has feature *camera*.
- Hypothesis: A *cellphone* has *picture*.

Neither text nor hypothesis are explicitly created like this, but the idea behind this brought me to my current implementation. I search for paths between the category of the query (phone) and the target (target verb or target noun) and collect all paths that I find. Since the product features were added to the knowledge graph and as has been shown in 3.4.1 they can be used to find new path it makes sense to search for a path from the product category and the target feature. It is possible that there is a path that does not make use of the product features, but this case implies that none of the products in that category have the features that were used in that path.

I keep all the paths that have a *has_feature* relation in them: Those features are the ones that are relevant to the query. Once the features that match the query are extracted, showing products that have those features and their scores as calculated in 3.3 is simple. The data is stored in the database and the matching products simply have to be retrieved. Figure 3.3 shows an example query. The tabs are matching features that were found and one example path from start node to end node is shown, together with matching products and information about their scores for that particular feature.

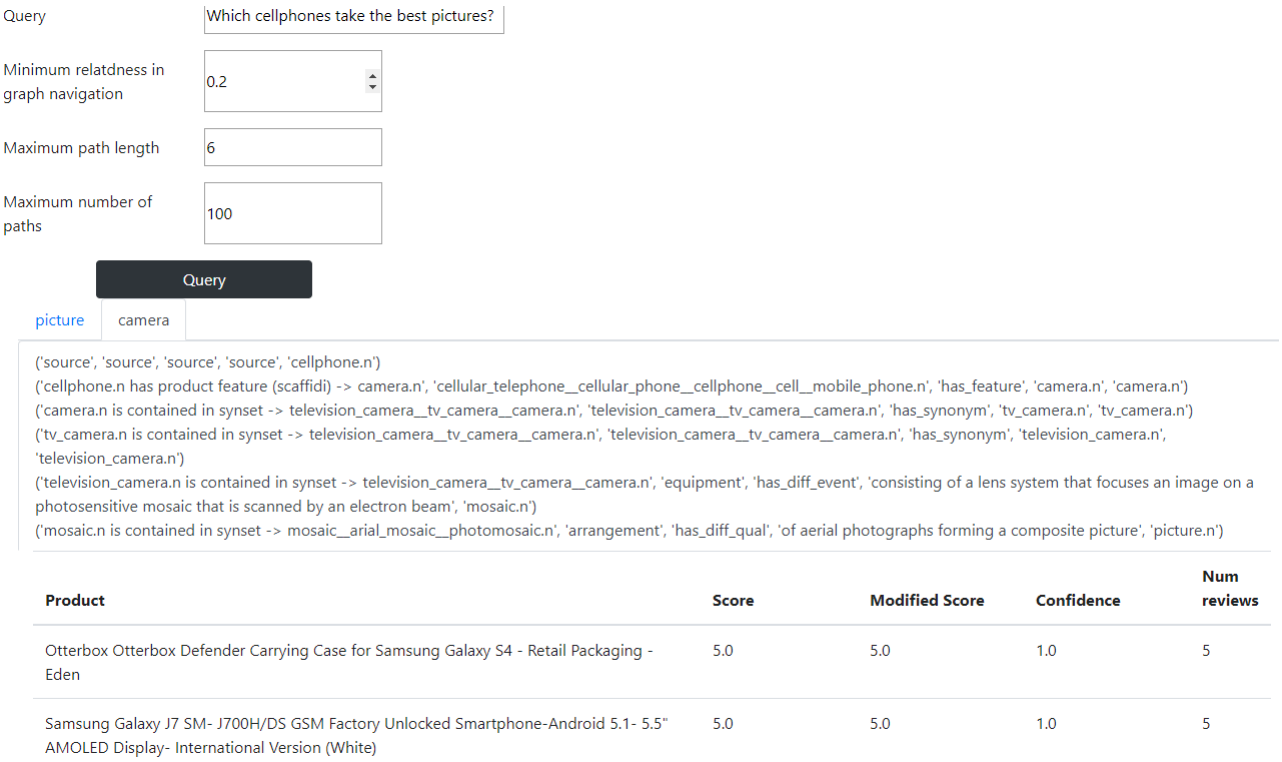

Figure 3.3: Example query and result

Chapter4 Summary

4.1 Discussion and recommendations for further work

The developed system is meant to be a proof-of-concept of augmenting an existing knowledge graph with knowledge about product categories based on features of those products extracted from product reviews. The biggest issue is that my implementation has a lower precision when performing an evaluation than the original implementation (see 3.5). The problem here is most likely an error in the implementation but nonetheless when excluding the entailment pairs from the evaluation that do not produce checkable word pairs the results of the system are quite robust (see 3.8). As the product recommendation is very basic right now it is a good basis for an extension of my work. One option is to explicitly create entailment pairs from the extracted product features as mentioned in section 3.4.3 and find all product features that match the query first instead of looking for paths between the category and the target of the query.

4.2 Summary

As part of this seminar I have extended the existing WordNet knowledge graph based on [SFH18] with features extracted from product reviews using an approach by [SBC+07]. As a proof-of-concept features were extracted from 450212 reviews of products in the cellphone category on Amazon and new triples added to the knowledge graph to augment the navigational algorithm with new paths. I have shown that this is a valid approach by showcasing entirely new paths in the graph in section 3.4.1. Furthermore I have built a basic product recommender that makes use of the graph navigational algorithm to perform basic product recommendation based on user queries.

AppendixA Code

A.1 Git repo

The code for the system developed as part of this thesis can be found at https://github.com/Madjura/dragn.

A.2 Documentation

The documentation for the system can be found at https://madjura.github.io/dragn/. Installation instructions are included as part of the documentation.

Bibliography

[AZSD07] Silvana Aciar, Debbie Zhang, Simeon Simoff, and John Debenham. Informed recommender: Basing recommendations on consumer product reviews. *IEEE Intelligent systems*, 22(3), 2007.

[MBB05] Sihem Mostefai, Abdelaziz Bouras, and Mohamed Batouche. Effective collaboration in product development via a common sharable ontology. *International journal of computational intelligence*, 2(4):206–212, 2005.

[Mil95] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[PP06] Siddharth Patwardhan and Ted Pedersen. Using wordnet-based context vectors to estimate the semantic relatedness of concepts. In *Proceedings of the Workshop on Making Sense of Sense: Bringing Psycholinguistics and Computational Linguistics Together*, 2006.

[SBC+07] Christopher Scaffidi, Kevin Bierhoff, Eric Chang, Mikhael Felker, Herman Ng, and Chun Jin. Red opal: product-feature scoring from reviews. In *Proceedings of the 8th ACM conference on Electronic commerce*, pages 182–191. ACM, 2007.

[SFH18] Vivian S Silva, André Freitas, and Siegfried Handschuh. Recognizing and justifying text entailment through distributional navigation on definition graphs. 2018.

[SHF18] Vivian S Silva, Siegfried Handschuh, and André Freitas. Categorization of semantic roles for dictionary definitions. *arXiv preprint arXiv:1806.07711*, 2018.

[SSB+18] Juliano Efson Sales, Leonardo Souza, Siamak Barzegar, Brian Davis, André Freitas, and Siegfried Handschuh. Indra: A word embedding and semantic relatedness server. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA).