Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

FAKULTÄT
FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN

Bachelor Thesis

Comparative Query Suggestion

Hauke Heller

Studiengang Informatik
Matr.-Nr. 6004435

Erstgutachter: Prof. Dr. Chris Biemann
Zweitgutachter: Dr. Alexander Panchenko

Abgabe: 23.07.2019

Inhaltsverzeichnis

1 Abstract

This work aims to provide an overview of different query suggestion approaches and explains its relevance. By adding a feature that makes comparative suggestions to the Comparative Argument Machine, it shows how query suggestion may enhance user experience in practice. To test the usefulness of the comparative suggestions, they are compared with suggestions from the Google Suggest API. In a second approach, suggestions are tested for usefulness by an annotator.

2 Introduction

To help users comparing two objects independent from a specific domain, the Comparative Argument Machine (CAM) was developed in a former project. It enables users making domain-independent comparative queries with two comparison objects A and B, as well as zero or more aspects.

As users often have little information about the objects they want to compare, they might even lack examples of good comparison objects they could compare comparison object A to.

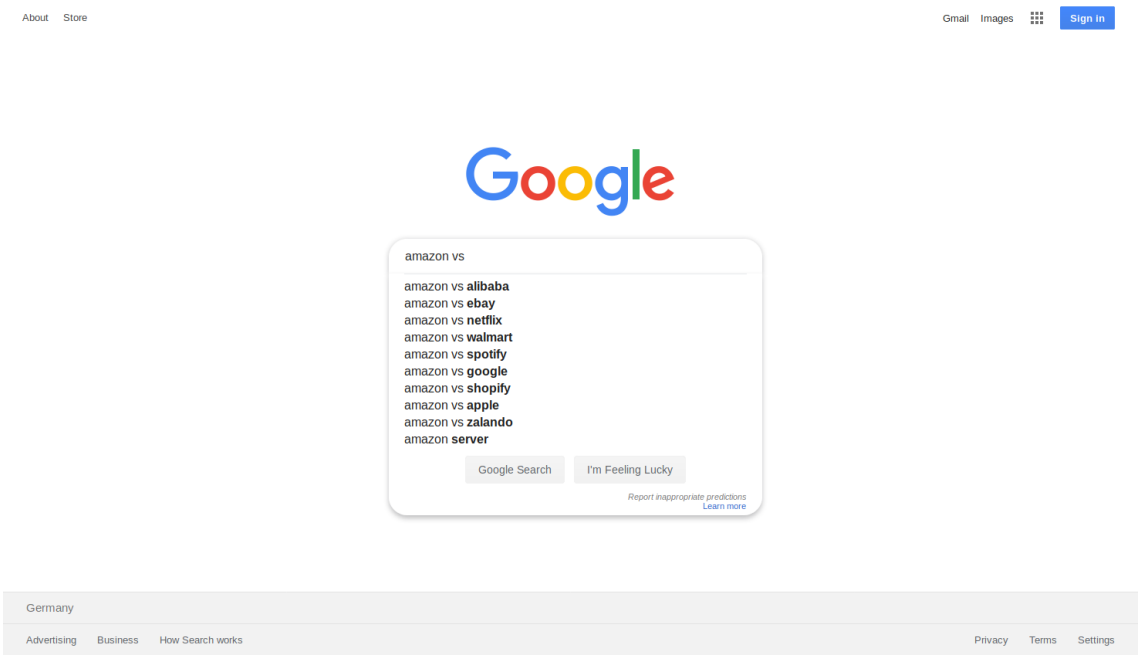

Abbildung 2.1: Possible approach to gather comparison objects for *amazon* [1]

A possible approach here would be to use the Google Suggest feature of Google Search by typing «*comparison object A*» *vs* in its text field like shown above. Adding a similar functionality to CAM would be an example of query suggestion. This work will discuss different aspects of query suggestion and finally, show how CAM is enhanced with a comparative query suggestion feature.

[1] https://www.google.com/

3 Background

As this work aims to explore query suggestion in the context of the comparative argument machine (CAM), this chapter will briefly explain CAM and introduce some important concepts and techniques of query suggestion.

3.1 The Comparative Argument Machine (CAM)

CAM is a system that aims to be able to process domain-independent comparative queries and return a preferred object based on a large collection of text documents"[SBZ+19]. It extracts argumentative textual statements from web resources to answer questions asking to compare two objects A and B, optional with respect to zero or more aspects {C}. Its result contains all sentences that contain A, B taking {C} to account in order of their usefulness.[SBZ+19]

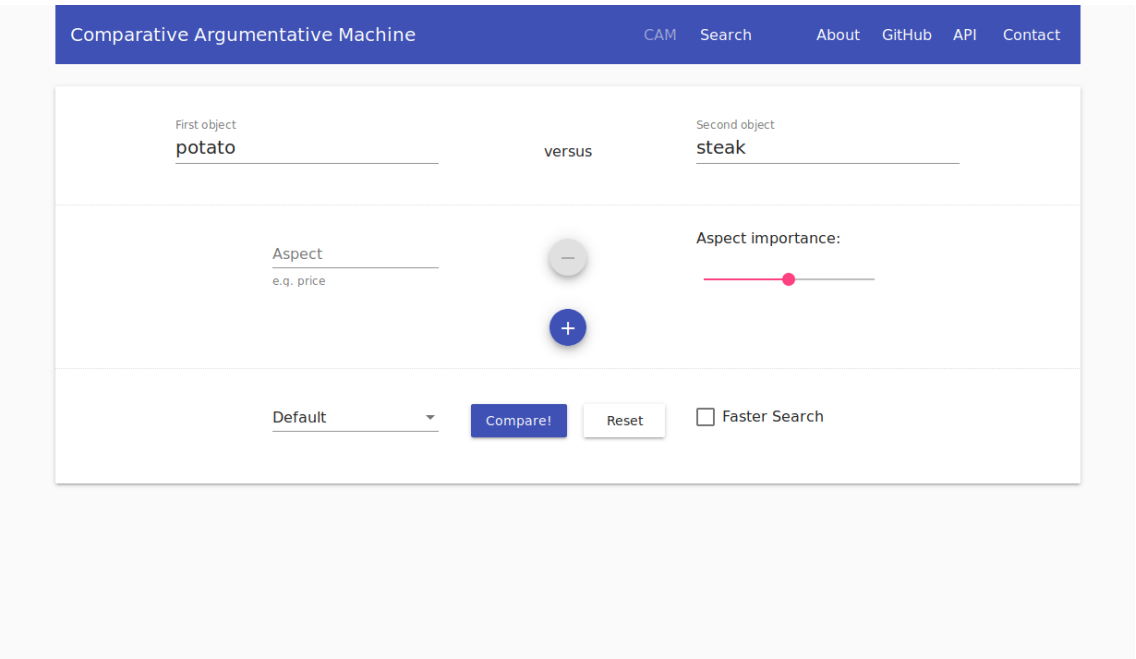

Abbildung 3.1: CAM frontend [1]

[1] http://ltdemos.informatik.uni-hamburg.de/cam/

3.2 Query suggestion

"Query suggestion has widely been used in most commercially web search engines which facilitates the interaction between users and search engines."[SZH11] The query suggestions are offered to search engine users as an additional option next to the results shown to them from their initial submitted query. Hence, if the user is not satisfied with the results, they may choose to click on one of the suggested queries to refine the search. Research works have indicated that query suggestion greatly improves user satisfaction rate, especially for information queries.[SZH11]

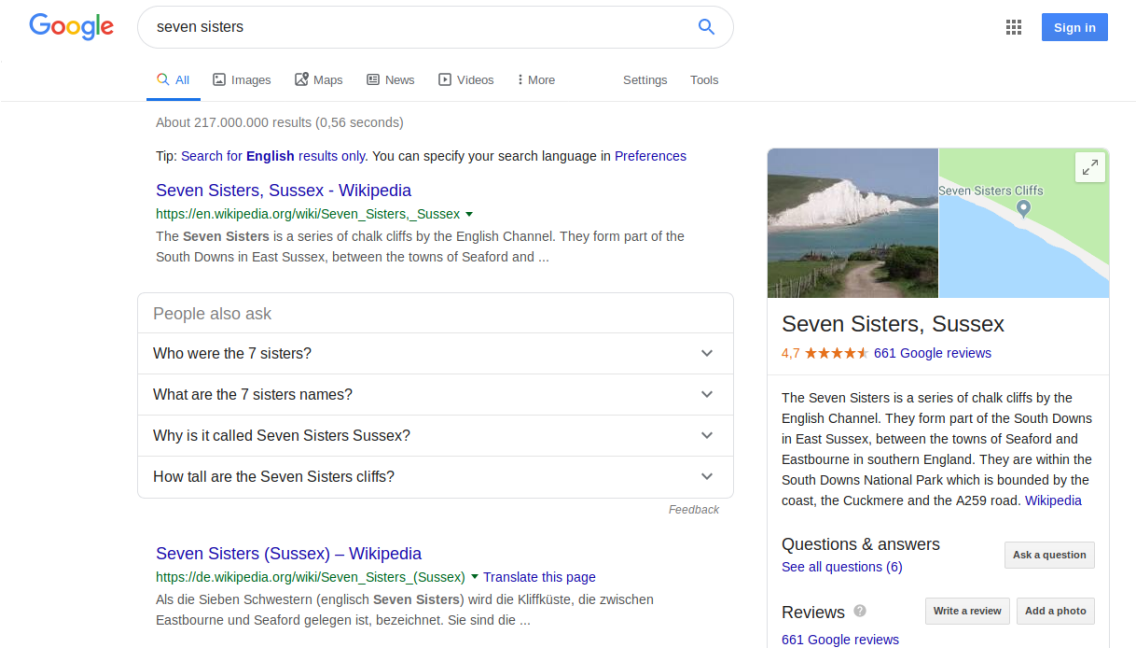

Abbildung 3.2: Google query suggestions for the term *seven sisters* [2]

To present the best query suggestions, Andrew et al. describe a system that makes it possible to rank query suggestions by employing a machine-learned model of user behavior that outputs an indication of usefulness to generated query suggestions.[APR+11] In this scenario, it is crucial that the suggested queries offer a good variety of suitable queries. Hao Ma et al. describe a technique for generating alternative queries to Web users to improve the user search experience. Their method thrives to suggest both semantically relevant and diverse queries to Web users. Based on Markov random walk and hitting time analysis on the query-URL bipartite graph, it aims to prevent semantically redundant queries from receiving a high rank and hence leads to a more diverse result set.[MLK10] This is backed by Makoto P. Kato et al. who analyzed different kinds of data sets comprising millions of unique queries, query suggestions, and patterns of users. Their analysis suggested that search engines provide better assistance when rare or single-term queries

[2] https://www.google.com/

are input and that they dynamically provide query suggestions according to the searcher's current state.[KST13]

3.3 Term relevance feedback

Term relevance feedback describes a feature that may be part of an information retrieval (IR) system. It allows users to mark documents as relevant to their needs and present them to the IR system which may use this information to retrieve more documents that are similar to the marked documents. The user may in a second iteration mark relevant documents, present them to the IR system and so on. [RL03] The major problem with this approach is that users have difficulties selecting good terms from a list of candidates and are therefore reluctant to make use of this feature. Query suggestions as discussed in the previous subsection on the other hand often rely on past queries that are similar to the user's current query or might have difficulties determining these similarities. To address these problems Diane Kelly et al. seek to assist users to formulate and reformulate queries by combining both approaches by automatically creating query suggestions using term relevance feedback techniques.[KGB09]

3.4 Dynamic query suggestion

Dynamic query suggestion, also called auto-completion, was initially designed as a prosthesis for people with limited abilities to express themselves vocally. Swiffin et al. introduced this when they described their Predictive Adaptive Lexicon (PAL) in 1987, a software designed to propose words to entered prefixes to reduce the number of character inputs necessary to enter any given text. It already gave word predictions that adapted to users vocabulary by automatically capturing words which were not already in its dictionary. It operates based on a dictionary that can adapt when the user enters words which are not already in it. Each entry in the dictionary contains statistical data on the usage of that word. To minimize search time, the dictionary is stored in a tree where every node is the prefix of their subtree. Hence, from any node the set of possible predictions is contained in its sub-tree.[SAPN87]

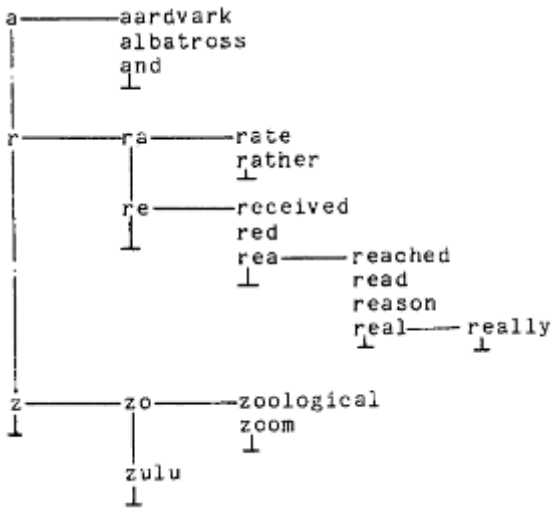

Abbildung 3.3: The Dictionary tree structure [SAPN87]

Similar to this, Mysen et al. describe a technique of dynamic query suggestion based on a prefix, received from a user device. A query prefix is received from a user device. A user category is determined based on the user identifier which itself is determined based on the user device. A node that represents the query prefix is located in a query graph and descendent child nodes representing queries are located. Each node has one or more user categories and each user category is associated with user-category specific frequency measures. The located nodes are ranked base on the associated user-category specific frequency measure and sent to the user device.[MS11]

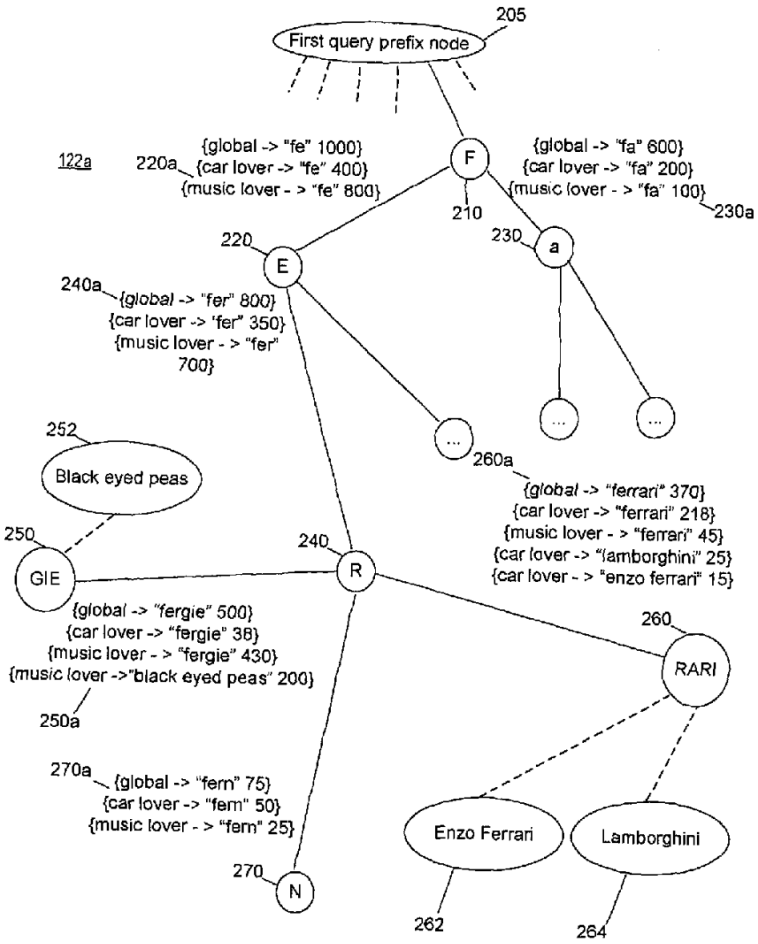

Abbildung 3.4: Dynamic query suggestion tree [MS11]

As this feature assists in query creation and completion, reduces keystrokes and helps avoid typos, it can be found in a growing number of text boxes.

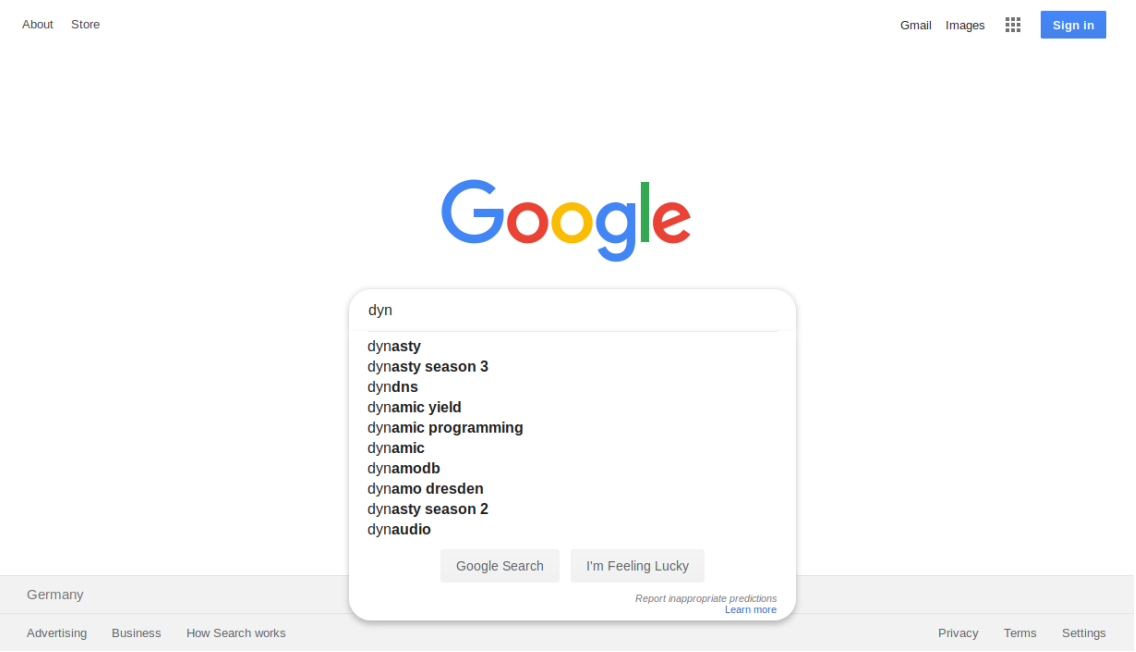

Abbildung 3.5: Dynamic query suggestions on google.com for the prefix *dyn* [3]

Abbildung 3.6: Dynamic query suggestions on wikipedia.org for the prefix *dyn* [4]

[3] https://www.google.com/
[4] https://en.wikipedia.org/wiki/Main*page*

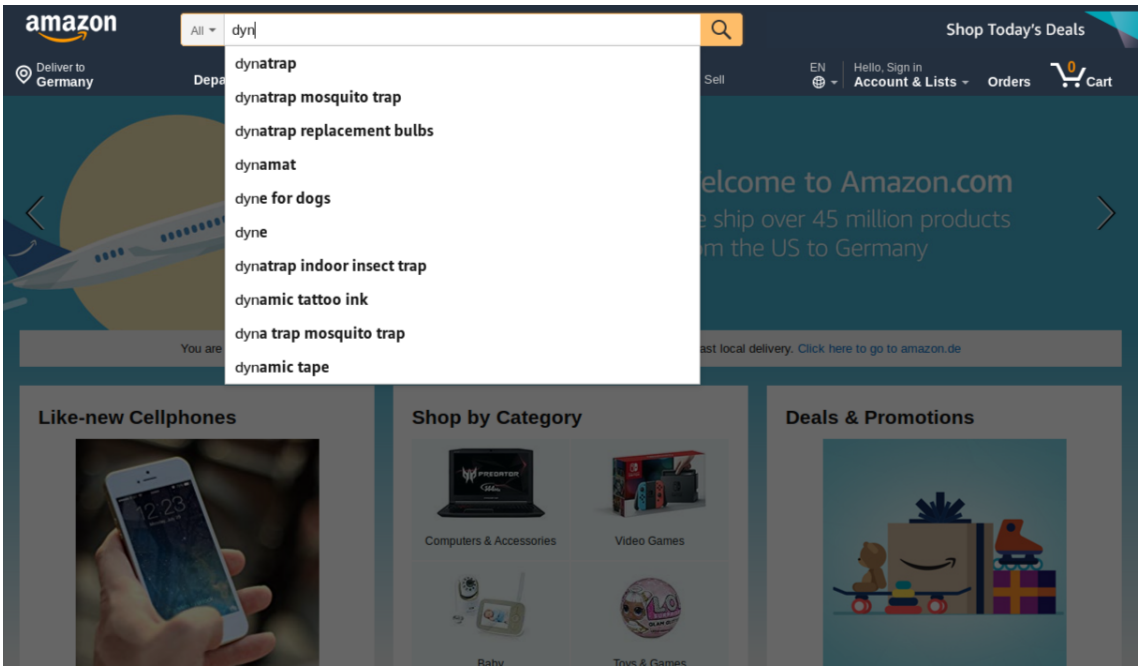

Abbildung 3.7: Dynamic query suggestions on amazon.com for the prefix *dyn* [5]

[5] https://www.amazon.com/

4 The Initial Experiment

4.1 The Comparison Candidate Retrieval (CCR) machine

In his master thesis, Matthias Schildwächter already implemented the Comparison Candidate Retrieval machine (CCR). It takes a list of comparison objects as input and outputs comparison candidates for each comparison object. As this work ultimately intends to extend CAM as a kind of auto-completion feature, CCR was adapted to explore different strategies of doing so. Its basic functionality is described hereafter:

1. DepCC is searched for sentences that contain the comparison object as well as the word *vs*. "DepCC is the largest to date [...] linguistically analyzed corpus in English [...] from a web-scale crawl"[PRF+17]. The result is saved in in a json object *es_json*:

```
def retrieve_sentences(comparison_object, vs='vs'):
    esHostname = 'http://ltdemos.informatik.uni-hamburg.de/
        depcc-index/'
    index = 'depcc'
    crawlDataRepos = '/_search?q='
    url = esHostname + index + crawlDataRepos
            + 'text:(\"{}\"%20AND%20\"{}\")&from=0&size=10000'
            .format(comparison_object, vs)
    es_json = requests.get(url,
                auth=HTTPBasicAuth(sys.argv[1], sys.argv[2]))
    return es_json
```

   As an example, the resulting json (shortened) for the comparison object *python*:

```
{
    "took": 251,
    "timed_out": false,
    "_shards": {
        "total": 32,
        "successful": 32,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": 5106,
        "max_score": 24.678474,
        "hits": [{
```

```
            "_index": "depcc",
            "_type": "text",
            "_id": "AdePGGYB6G1JbSfZCWuA",
            "_score": 24.678474,
            "_source": {
                "sentence_hash": 1487222179,
                "document_id": "http://bugs.python.org/issue9807",
                "insert_id": "",
                "text": "# (\"python-debug\" vs \"python\").",
                "sentence_id": 60
            }
        }, {
            ...
        }]
    }
}
```

A list of sentences is populated with the text from each hit in the json object:

```
def extract_sentences(es_json):
    hits = es_json.json()['hits']['hits']
    sentences = []
    for hit in hits:
        text = hit['_source']['text']
        sentences.append(text)
    return sentences
```

A JSON object is returned and a list is populated with those sentences.

2. Each sentence is searched for the pattern <comparison object> - *vs* - <noun phrase> or the other way around. Each unique noun phrase is stored as comparison candidate in a dictionary along with and sorted by the number of patterns in which it occurred.

```
def extract_candidates(comp_obj, sentences):
    unique_candidates = {}
    for sentence in sentences:
        blob = TextBlob(sentence)
        for candidate in blob.noun_phrases:
            if candidate not in [comp_obj, 'vs', 'vs.']
            and is_candidate(candidate, comp_obj, sentence):
                if candidate in unique_candidates:
                    unique_candidates[candidate] += 1
                else:
                    unique_candidates[candidate] = 1
    unique_candidates = sorted(unique_candidates.items(),
    key=operator.itemgetter(1), reverse=True)
    return unique_candidates
```

3. A list of candidates is retrieved from the Google Suggest API in order to compare them to those retrieved from CCR.

```
def get_suggestions(s):
    ggl_suggestions = lambda s:get("http://google.com/
        complete/search?client=gma&q="+s).json()[1]
    return ggl_suggestions(s)
```

4.2 Results

For the initial experiment, comparison candidates were calculated by CCR for a list of 30 unique words. In parallel, comparison candidates were queried from the Google Suggest API for the same words to see how CCR holds up to a widely used tool. The accumulated calculation time was 6:14 Minutes, that is a mean time of 12.47 seconds for each word. To classify CCR's comparison candidates, some statistical measures of performance were calculated:

- True positives (TP) as the size of the intersection of the comparison candidates from the Google Suggest API and CCR.

  $$TP = GoogleSuggestions \cap CCRSuggestions$$

  It describes how many words that were identified by the Google Suggest API as comparison candidates were also identified as such by CCR.

- Precision as the ration of TP to the amount of all of CCR's comparison candidates.

  $$Precision = \frac{TP}{CCRSuggestions}$$

  It puts the TP into perspective, as we want CCR to suggest only comparison candidates that are also comparison candidates of the Google Suggest API.

- Recall as the ratio of TPs to the amount of all of Google's comparison candidates, as we want CCR not to miss any of Googles suggestions.

  $$Recall = \frac{TP}{GoogleSuggestions}$$

- F_1 score as the harmonic average of precision and recall as we want both the TP to be the size of the comparison candidates of the Google Suggest API and also the CCR to suggest nothing else.

  $$F_1 score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

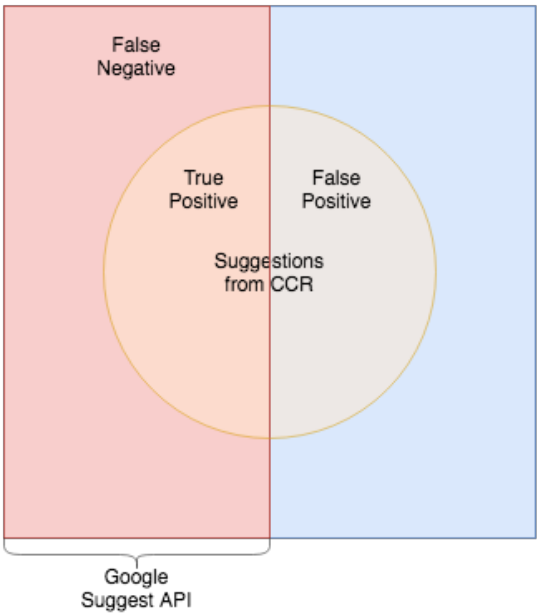

Abbildung 4.1: Statistical measures of the performance

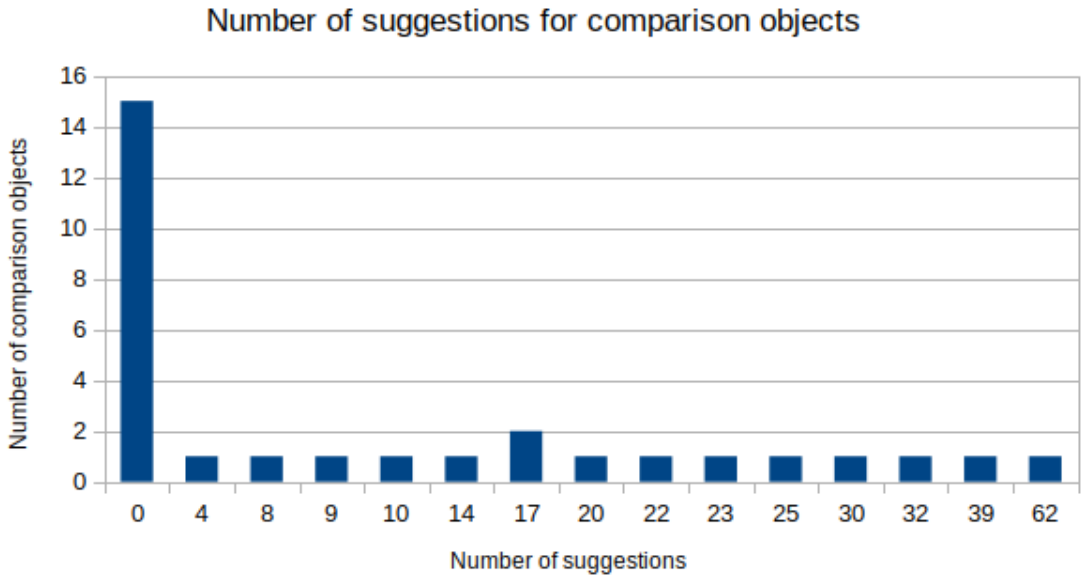

Abbildung 4.2: Initial experiment results I

For 16 out of 30 comparison objects, CCR failed to suggest comparison candidates. On the other hand, taking into account the fairly basic approach of the comparison candidates retrieval, the results are not too bad:

Mean true positives:	1.0667
Mean false negatives:	16.6
Mean false positives:	10.0
Mean precision:	0.0643
Mean recall:	0.0585
Mean F_1 score:	0.0539

In comparison with suggestions from the Google Suggest API, CCR suggested on average 1.0667 true positives, with a maximum of 6 for the comparison object *truck*. Mean precision is 0.0643, mean recall is 0.0585 and mean F_1 score is 0.0539.

It is, of course, debatable whether comparison candidates selected by the Google Suggest API hold up for a kind of gold standard. Comparing CCR's suggestions to them says little about their real usefulness.

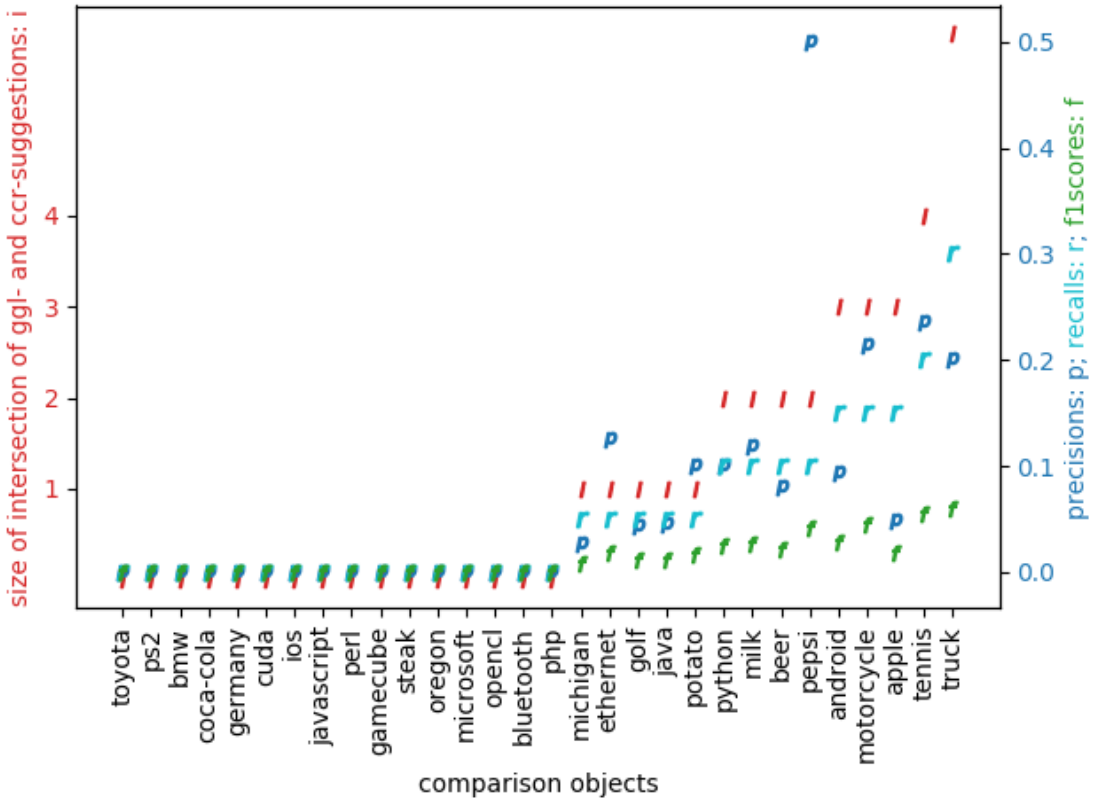

Abbildung 4.3: Initial experiment results

5 Integration of CAM with CCR

To demonstrate the benefits of the comparative candidate retrieval machine (CCR), we enhanced the comparative argument machine (CAM) with CCR's main functionality. This feature suggests comparative objects to the user's entries in the input element 'First object'. As on the fly computation of suggestions by CCR with the current setup, unfortunately, takes around 8 seconds, we pre-computed suggestions for a dictionary of words.

```
co_suggestions_dict = {}

def requestSuggestions(comparison_object):
    ccr_suggestions = requests.get("http://127.0.0.1:5000/ccr/"+ '{}'
        .format(comparison_object)).json()
    data = {
        "comparison_object": comparison_object,
        "suggestions": ccr_suggestions
        }
    return data

with open(./comparison_objects_filename) as json_file:
    comparison_objects = json.load(json_file)
    co_suggestions_dict = Pool(4).map(requestSuggestions,
    comparison_objects)

with open(./outfile.json, 'w') as outfile:
    json.dump(co_suggestions_dict, outfile)
```

From the pre-computed suggestions, an Elasticsearch index was created:

```
es = Elasticsearch(hosts = [{"host" : "localhost", "port" : 9200}],
    timeout=300)

request_body = {
    "settings" : {
        "number_of_shards": 1,
        "number_of_replicas": 0
    }
}

res = es.indices.create(index = "suggestions-index",
        body = request_body)
```

```
13
14 counter = 0
15 document_name = './suggestions/outfile.json'
16     .format(str('%05d' % i))
17
18 with open(document_name) as json_file:
19     data = json.load(json_file)
20     for obj in data:
21         es.index(
22             index = "suggestions-index",
23             doc_type = "suggestions",
24             id = counter,
25             body = obj)
26         counter += 1
```

When using CAM's frontend, each time the input element 'First object' loses focus, a request is made to the Elasticsearch index:

```
1  requestSuggestions() {
2      this.httpRequestService.getEsSuggestions(
3          this.urlBuilderService.buildEsUrl(),
4          this.object_A).subscribe(
5              data => {
6                  this.options = data['hits']['hits'][0]
7                      ['_source']['suggestions'];
8                  console.log('Suggestions found: ' + this.options);
9              }
10     );
11 }
```

The array of pre-computed suggestions (this.options) is presented to the user as a drop-down list when they focus the input element 'Second-object' as shown in the following examples:

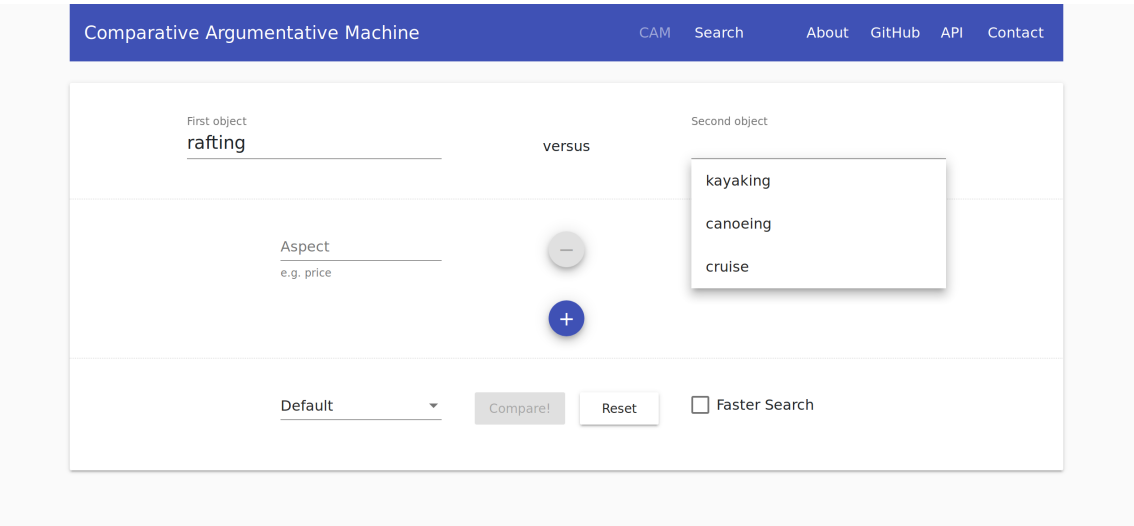

Abbildung 5.1: Presentation of Suggestions for the word *rafting* as drop-down list

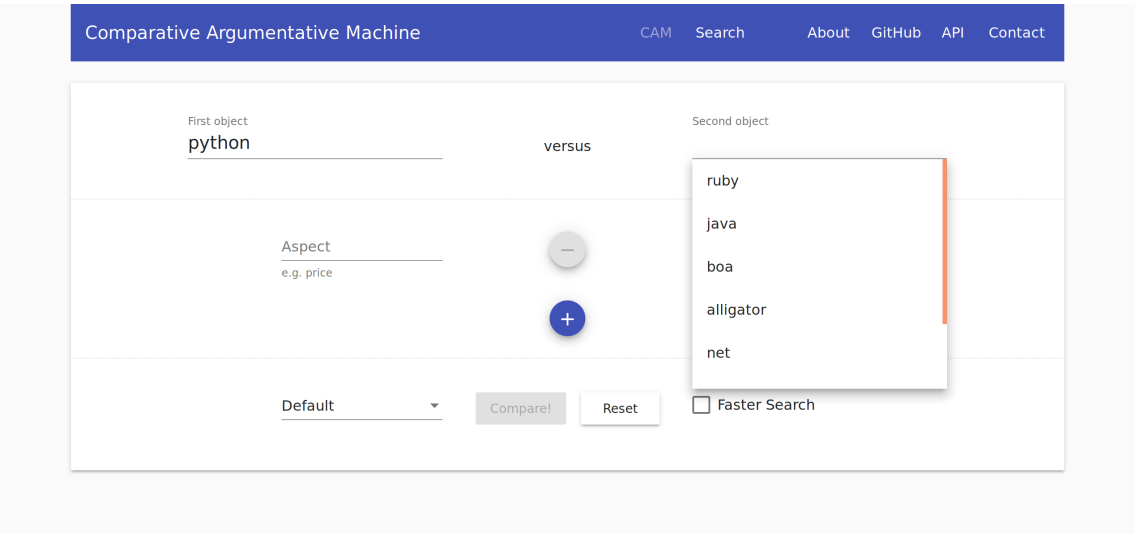

Abbildung 5.2: Presentation of Suggestions for the word *python* as drop-down list

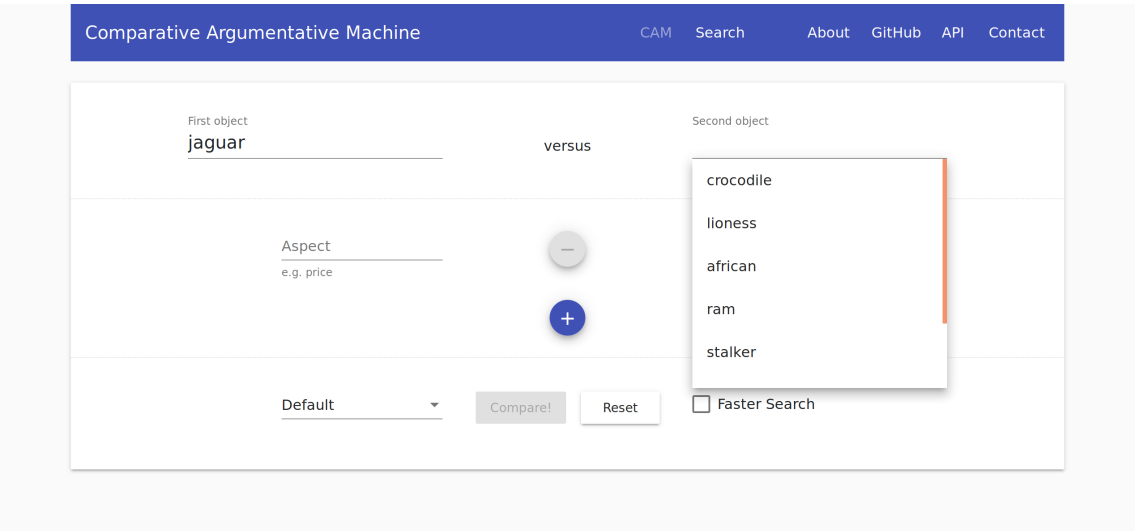

Abbildung 5.3: Presentation of Suggestions for the word *jaguar* as drop-down list

5.1 Comparing word lists for the precomputation of comparison suggestions

In order to compare the suggestions computed by CCR, we used the Google Suggest Api to generate suggestions in parallel for the comparison objects dictionary:

```
1 with open(./comparison_objects_filename) as json_file:
2     comparison_objects = json.load(json_file)
3     for comparison_object in comparison_objects:
4         ggl_suggestions = get_suggestions(comparison_object)
5         data = {
```

```
            "comparison_object": comparison_object,
            "suggestions": ggl_suggestions
        }
        co_ggl_suggestions_dict.append(data)

def get_suggestions(s):
    ggl_suggestions = lambda s:get(
        "http://google.com/complete/search?client=gma&q="+s).json()[1]
    return ggl_suggestions(s)
```

5.1.1 Dictionary A

Dictionary A is taken from a GitHub project as *words_dictionary.json*[Dwy19] and contains 369,759 English words. It was used as it was the biggest dictionary of English words I came across as JSON format.

CCR failed to make suggestions for 349,964 (94,6%) of the 369,759 English words, hence, compared to the results of the initial experiment, numerics are more disappointing:

Mean true positives:	0.0219
Mean false negatives:	2.5294
Mean false positives:	0.1410
Mean precision:	0.0074
Mean recall:	0.0022
Mean F_1 score:	0.0034

Only one out of fifty words that Google suggests equals a suggestion from CCR. Mean precision and recall, as well as the F_1 score, are very small.

For those comparison objects that CCR succeeds in making suggestions, 8040 words (40%) produce only one comparative suggestion, 3836 words (19%) produce seven suggestions and more:

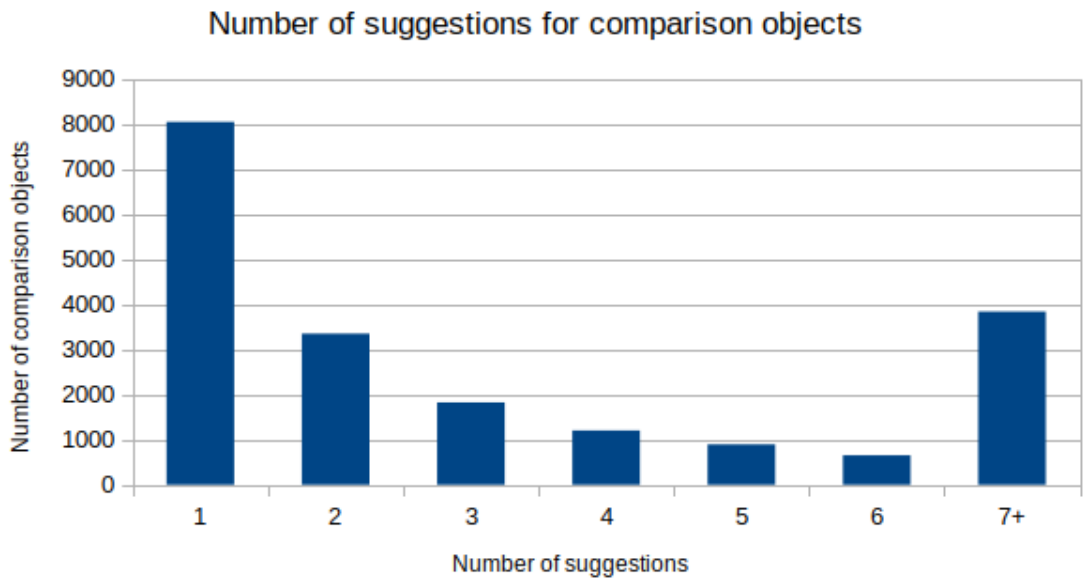

Abbildung 5.4: CCR's results for Dictionary A

Mean true positives:	0.4094
Mean false negatives:	9.4623
Mean false positives:	2.6343
Mean precision:	0.1376
Mean recall:	0.0411
Mean F_1 score:	0.0632

5.1.2 Dictionary B

Dictionary B is taken from a data set containing 190 words used in the paper *Categorizing Comparative Sentences* as they are better examples of comparative queries. [PBF+18]

For this dictionary, CCR succeeds in making suggestions for more than half of the comparison objects and for most of those, CCR makes seven or more suggestions:

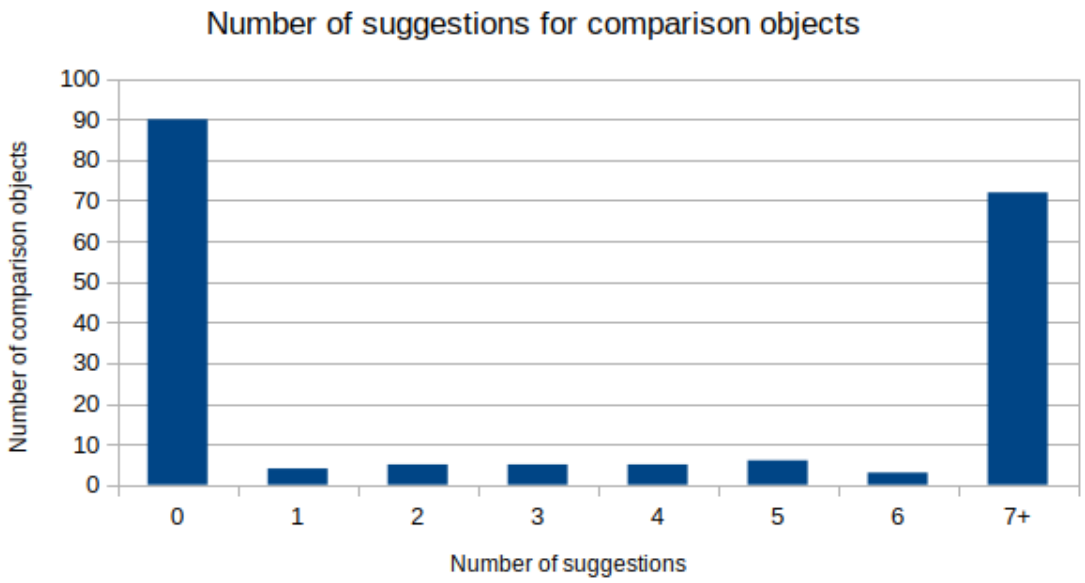

Abbildung 5.5: CCR's results for Dictionary B

Hence, comparing these results to Suggestions from the Google Suggest API, numerics look much better than with dictionary A:

Mean true positives:	0.5
Mean false negatives:	11.3
Mean false positives:	2.6632
Mean precision:	0.0835
Mean recall:	0.0460
Mean F_1 score:	0.0149

A different approach to comparing suggestions from CCR with those from the Google Suggest API is annotating CCR's suggestions manually. For example, CCR's suggestions for the comparison object *Word* are *latex*, *pages*, *letterpress*, *photo*, *inflection*, *advertising* or *paper*. Interpreting the comparison object as the text editor from Microsoft Office, *latex* and *pages* would be useful suggestions. *Letterpress* is a game and comparing *Word* to *photo, inflection, advertising or paper* would also not make much sense.

This is of course a weakness of this approach as different annotators might certainly annotate suggestions differently. Some examples of how suggestions were annotated in in this case are:

Comparison Object	CCR Suggestions: **bold** words are annotated as useful
Word	**latex**, **pages**, letterpress, photo, inflection, advertising, paper
Amazon	**apple**, **bookstores**, publishers, norse, target, android, borders
Ibuprofen	**acetaminophen**, **naproxen**, **tylenol**, **aleve**, **aspirin**, **celebrex**, **codeine**
Nissan	**jeep**, **tesla**, **corvette**, **ford**, **volt**
Birthday	bathing
Camping	survival, **hotels**, spaniels, **lodges**
Fox	cyborg, **duck**, trump, ness, **abc**, link, dish
Pasta	pesach, **potatoes**, **noodles**, **mac**, **rice**
Pen	**pencil**, **sword**, **stroke**, **brush**, **finger**, **pc**, **keyboard**
Rat	**pigeon**, **hawk**, **rats**, **squirrel**, **bandicoot**, **fish**, **hamster**

Doing this for all 190 words from dictionary B and taking the useful suggestions as the true positives and the rest as the false positives, the mean precision can be calculated:

Mean true positives:	2.1
Mean false positives:	1.0632
Mean precision:	0.6639

The precision of 66 % is pretty good, a mean of only 3.1632 suggestions per comparison object weakens this though.

6 Conclusion and Future Work

CCR can be used as a feature that provides CAM with comparative query suggestions. Even though the quantities of suggestions are very different comparing dictionary A and B, the quality for more common comparison objects, like those of dictionary B, is quite OK when compared to suggestions from the Google Suggest API and even better when tested for usefulness by an annotator.

As discussed, the precision calculated from CCR's annotated results might depend strongly on the annotators' idea of usefulness. To tackle this, the annotation work should be done by more annotators, who could also be asked to come up with a list of comparison suggestions to be able to calculate recall and F_1 score.

Apart from further improving the quality and quantity of CCR's suggestions, it should be possible to use CCR's basic functionality to not only provide CAM with a list of comparative suggestions but also a list of aspects for every specific comparison object combination.

Literaturverzeichnis

[APR+11] ANDREW, Galen ; PARK, Sooho ; ROUNTHWAITE, Robert L. ; CUCERZAN, Silviu-Petru ; BUCKLEY, Jamie P. ; CHAN, Joanna: *Query suggestion generation*. Juli 19 2011. – US Patent 7,984,004

[Dwy19] DWYL: *dwyl/english-words*. `https://github.com/dwyl/english-words`. Version: Apr 2019

[Fra] FRANZEK, Mirco: Comparative Argument Mining.

[KGB09] KELLY, Diane ; GYLLSTROM, Karl ; BAILEY, Earl W.: A comparison of query and term suggestion features for interactive searching. In: *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* ACM, 2009, S. 371–378

[KST13] KATO, Makoto P. ; SAKAI, Tetsuya ; TANAKA, Katsumi: When do people use query suggestion? A query suggestion log analysis. In: *Information retrieval* 16 (2013), Nr. 6, S. 725–746

[MDL+13] MISHNE, Gilad ; DALTON, Jeff ; LI, Zhenghua ; SHARMA, Aneesh ; LIN, Jimmy: Fast data in the era of big data: Twitter's real-time related query suggestion architecture. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* ACM, 2013, S. 1147–1158

[MLK10] MA, Hao ; LYU, Michael R. ; KING, Irwin: Diversifying Query Suggestion Results. In: *AAAI* Bd. 10, 2010

[MS11] MYSEN, Clarence C. ; SCHWARTZ, Scott E.: *Dynamic query suggestion*. September 27 2011. – US Patent 8,027,990

[MSC+12] MYLLYMAKI, Jussi ; SINGLETON, David P. ; CUTTER, Al ; LEWIS, Matthew ; EBLEN, Scott: *Location based query suggestion*. Oktober 30 2012. – US Patent 8,301,639

[PBF+18] PANCHENKO, Alexander ; BONDARENKO, Alexander ; FRANZEK, Mirco ; HAGEN, Matthias ; BIEMANN, Chris: Categorizing Comparative Sentences. In: *arXiv preprint arXiv:1809.06152* (2018)

[PRF+17] PANCHENKO, Alexander ; RUPPERT, Eugen ; FARALLI, Stefano ; PONZETTO, Simone P. ; BIEMANN, Chris: Building a web-scale dependency-parsed corpus from CommonCrawl. In: *arXiv preprint arXiv:1710.01779* (2017)

[RL03] RUTHVEN, Ian ; LALMAS, Mounia: A survey on the use of relevance feedback for information access systems. In: *The Knowledge Engineering Review* 18 (2003), Nr. 2, S. 95–145

[SAPN87] SWIFFIN, Andrew ; ARNOTT, John ; PICKERING, J A. ; NEWELL, Alan: Adaptive and predictive techniques in a communication prosthesis. In: *Augmentative and Alternative Communication* 3 (1987), Nr. 4, S. 181–191

[SBF+04] SMYTH, Barry ; BALFE, Evelyn ; FREYNE, Jill ; BRIGGS, Peter ; COYLE, Maurice ; BOYDELL, Oisin: Exploiting query repetition and regularity in an adaptive community-based web search engine. In: *User Modeling and User-Adapted Interaction* 14 (2004), Nr. 5, S. 383–423

[SBZ+19] SCHILDWÄCHTER, Matthias ; BONDARENKO, Alexander ; ZENKER, Julian ; HAGEN, Matthias ; BIEMANN, Chris ; PANCHENKO, Alexander: Answering Comparative Questions: Better than Ten-Blue-Links? In: *arXiv preprint arXiv:1901.05041* (2019)

[SZH11] SONG, Yang ; ZHOU, Dengyong ; HE, Li-wei: Post-ranking query suggestion by diversifying search results. In: *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval* ACM, 2011, S. 815–824

[WD08] WANG, Jian ; DAVISON, Brian D.: Explorations in tag suggestion and query expansion. In: *Proceedings of the 2008 ACM workshop on Search in social media* ACM, 2008, S. 43–50

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient 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.
Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

Hamburg, den _____ Unterschrift: _____