In-tool Learning for Selective Manual Annotation in Large Corpora

Erik-Lân Do Dinh[†], **Richard Eckart de Castilho**[†], **Iryna Gurevych**[†‡]
[†]Ubiquitous Knowledge Processing Lab (UKP-TUDA)
Department of Computer Science, Technische Universität Darmstadt
[‡]Ubiquitous Knowledge Processing Lab (UKP-DIPF)
German Institute for Educational Research and Educational Information
http://www.ukp.tu-darmstadt.de

Abstract

We present a novel approach to the selective annotation of large corpora through the use of machine learning. Linguistic search engines used to locate potential instances of an infrequent phenomenon do not support ranking the search results. This favors the use of high-precision queries that return only a few results over broader queries that have a higher recall. Our approach introduces a classifier used to rank the search results and thus helping the annotator focus on those results with the highest potential of being an instance of the phenomenon in question, even in low-precision queries. The classifier is trained in an *in-tool* fashion, except for preprocessing relying only on the manual annotations done by the users in the querying tool itself. To implement this approach, we build upon CSniper[1], a web-based multi-user search and annotation tool.

1 Introduction

With the rapidly growing body of digitally available language data, it becomes possible to investigate phenomena of the language system that manifest themselves infrequently in corpus data, e.g. non-canonical constructions. To pinpoint occurrences of such phenomena and to annotate them requires a new kind of annotation tool, since manual, sequential annotation is not feasible anymore for large amounts of texts.

An *annotation-by-query* approach to identify such phenomena in large corpora is implemented in the recently published open-source tool CSniper (Eckart de Castilho et al., 2012).

To enable a selective manual annotation process, a linguistic search engine is used, allowing the creation of queries which single out potential instances of the phenomenon in question. Those potential instances are then displayed to the user, who annotates each one as being an instance of the phenomenon or not. This process of searching and annotating can be performed by multiple users concurrently; the annotations are stored for each user separately. In a subsequent evaluation step, a user can review the annotations of all users, e.g. to discard a query if it yields unsatisfying results. Finally, the annotations of multiple users can be merged into a gold standard.

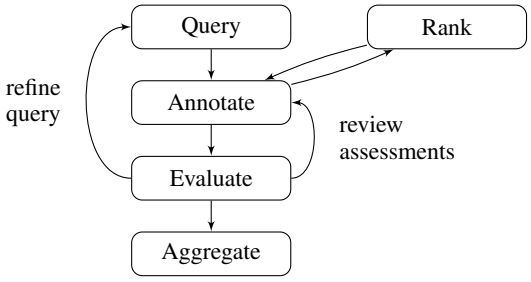

Figure 1: *Annotation-by-query* workflow extended with a ranking step.

This approach relieves the annotator from having to read through the corpus from the beginning to the end to look for instances of a phenomenon. However, the search may yield many results that may superficially appear to be an instance of the desired phenomenon, but due to ambiguities or due to a broadly defined query only a small subset may be actual instances. This still leaves the annotator with the tedious task of clicking through the search results to mark the true instances.

[1] https://dkpro.github.io/dkpro-csniper

Proceedings of ACL-IJCNLP 2015 System Demonstrations, pages 13–18,
Beijing, China, July 26-31, 2015.

To reduce the time and effort required, we present an extension of the *annotation-by-query* approach (Figure 1) that introduces a ranking of the query results (Section 2) by means of machine learning; we order the results by confidence of the used classifier. To obtain a model for the classifier, we employ an *in-tool* learning approach, where we learn from the annotations that are made by users in the tool itself. This makes our ranking approach useful for highly specific tasks, since no pre-trained models are needed.

Finally we demonstrate the viability of our concept by the example task of finding non-canonical constructions in Section 3.

2 Ranking linguistic query results

Our approach employs machine learning to facilitate — but not to completely replace — the manual annotation of query results. A query expresses the intention of the user to find a specific linguistic phenomenon (*information need*). An information retrieval search engine would provide the user with a list of results that are ranked according to their relevance, fulfilling the information need. However, linguistic search engines such as CQP (Evert and Hardie, 2011) — which is used by CSniper — are basically pattern-matching engines, operating on different lexical or morphosyntactic features like part-of-speech (POS) tags and lemmata and do not have a concept of relevance. Thus, if the query provided by the user overgeneralizes, relevant results are hidden among many irrelevant ones, ultimately failing to satisfy the user's information need.

To tackle this problem, we use the annotations already created by users on the search results to train a classifier. Unannotated query results are then fed to the classifier whose output values are then used as relevance ratings by which the results are ranked. The classifier producing the ranking can be invoked by the user at any time; it can be configured in certain characteristics, e.g. the annotations of which users should be used as training data, or how many of the selected users have to agree on an annotation for it to be included.

2.1 Workflow and ranking process in CSniper

Currently, we train the classifier on features derived from the constituency parse tree, which makes it useful for tasks such as locating sentences containing infrequent ambiguous grammatical constructions (cf. Section 3). Since parsing the query results is too time-intensive to be done during runtime, we parsed the corpora in advance and stored the parse trees in a database. To train the classifier, we employed SVM-light-tk (Moschitti, 2006; Joachims, 1999), a support vector machine implementation which uses a tree kernel to integrate all sub-trees of the parse tree as features.

Consider the following typical scenario incorporating the ranking: A user constructs a query based on various features, such as POS tags or lemmata, which are used to search for matching sentences, e.g.

"It" [lemma="be"] [pos="AT0"]? [pos="NN."]*[2]

The result is a list of sentences presented in a *keywords-in-context* (KWIC) view, along with an annotation field (Figure 2).

Then the user starts to annotate these sentences as *Correct* or *Wrong*, depending whether they truly represent instances of the phenomenon in question. Clicking on the *Rank results* button (Figure 2) invokes our ranking process: The SVM-light-tk classifier is trained using the parse trees of the sentences which the user previously annotated. The resulting model is then used to classify the remaining sentences in the query results. We rank the sentences according to the output value of the decision function of the classifier (which we interpret as a relevance/confidence rating) and transiently label a sentence as either *(Correct)* (output value > 0) or *(Wrong)* (output value ≤ 0). The results in the KWIC view are then reordered according to the rank, showing the highest-ranking result first. Repeatedly annotating those highest-ranking results and re-ranking allows for quickly annotating instances of the phenomenon, while also improving the classifier accuracy at the same time.

2.2 *Find* mode

After annotating instances based on simple queries and ML-supported ranked queries, we considered the natural next step to be searching automatically for the phenomenon in question utilizing machine learning, using arbitrary sentences from the corpus as input for the classifier instead of only the results returned by a query. Such an automatic search could address two concerns: 1) it removes

[2] It-cleft example query: "It", followed by a form of "to be", an optional determiner and a common noun.

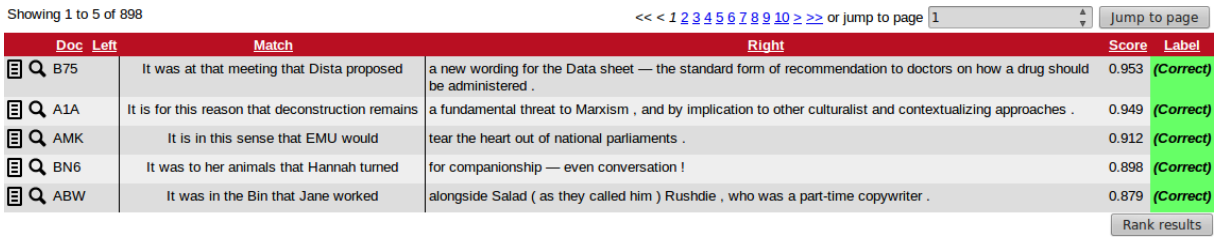

Figure 2: A screenshot showing the results table after the ranking process, with sentences sorted by confidence of the classifier (*Score*). The results are shown in a *keywords-in-context* (KWIC) view, separating left context, query match and right context (within a range of one sentence). Clicking on *(Correct)* changes the label to *Correct*.

the need for the user to design new queries, allowing users less experienced in the query language to annotate more effectively side-by-side with advanced users; 2) it could optimally generalize over all the queries that users have already made and potentially locate instances that had not been found by individual high-precision queries.

To support this, we implemented the *Find* mode, to locate instances of a phenomenon while abstracting from the queries. In this mode, the SVM is first trained from all previously (manually) labeled instances for a given phenomenon, without taking the queries into account that were used to find those instances. Then the corpus is partitioned into smaller parts containing a predefined amount of sentences (we used 500). One of these partitions is chosen at random, and the sentences therein are ranked using the SVM. This step is repeated, until a previously defined number of sentences have been classified as *Correct*. Those sentences are then shown to the user, who now can either confirm a sentence as containing the phenomenon or label it *Wrong* otherwise.

2.3 Related work

Existing annotation tools include automation functionality for annotation tasks, ranging from rule-based tagging to more complex, machine-learning-based approaches.

Such functionalities can be found in the annotation software WordFreak (Morton and LaCivita, 2003), where a plug-in architecture allows for a variety of different taggers and classifiers to be integrated, for example part-of-speech taggers or coreference resolution engines. Those require pretrained models, which limits the applicability of the automation capabilities of WordFreak to tasks for which such models are actually available. In addition to assigning annotations a single label, WordFreak allows plugins to rank labels for each annotation based on the confidence of the used classifier. Note that this is different to our ranking approach, where we instead perform a ranking of the search results which shall be annotated.

Another tool incorporating machine learning is WebAnno (Yimam et al., 2014), which implements features such as custom labels and annotation types. In addition, WebAnno supports automatic annotation similar to our approach, also employing machine learning to build models from the data annotated by users. Those models are then used to annotate the remainder of the documents. To accomplish this, WebAnno uses a split-pane view, showing automatic suggestions in one pane and manually entered annotations in another. The user can accept a suggested annotation, which is transferred to the manual pane. Lacking the search capability, WebAnno lists automatic annotations in the running corpus text, which makes it unsuited for selective annotation in large corpora. The approach that we implemented on top of CSniper instead ranks the search results for a given query by confidence of the classifier.

Yet another form of in-tool learning is *active learning*, as is implemented, e.g., in Dualist (Settles, 2011). In an active learning scenario the system aims to efficiently train an accurate classifier (i.e. with as little training data as possible) and thus repeatedly asks the user to annotate instances from which it can learn the most. Such an approach can work well for reducing the amount of training data needed to produce a model which achieves high accuracy, as has been — amongst others — shown by Hachey et al. (2005). However, they also learn in their experiments that those highly informative instances are often harder to annotate and increase required time and effort of annotators. Our approach is different from active

learning as our goal is not to improve the training efficiency of the classifier but rather to allow the user to interactively find and label as many true instances of a phenomenon as possible in a large corpus. Thus, the items presented to the user are not determined by the expected information gain for the classifier but rather by the *confidence* of the classifier, presenting the user with those instances first which are most likely to be occurrences of the phenomenon in question.

3 Case study: Finding non-canonical constructions

We demonstrate our novel approach on the task of locating *non-canonical constructions* (NCC) and conduct an intrinsic evaluation of the accuracy of the system augmented with machine learning output on the data annotated by expert linguists. The linguists annotated sentences for occurrences of certain NCC subtypes: *information-packaging constructions* (Huddleston and Pullum, 2002, pp. 1365ff.), which present information in a different way from their canonical counterparts without changing truth conditions; specifically *It-clefts* ("It was Peter who made lunch."), *NP-preposing* ("A treasure, he was searching."), and *PP-inversion* ("To his left lay the forest.") clauses.

For our experiments, we used the British National Corpus (2007), comprising 100 million words in multiple domains[3]. Constituency parsing was conducted using the factored variant of the Stanford Parser (Klein and Manning, 2003), incorporated into a UIMA pipeline using DKPro Core (Eckart de Castilho and Gurevych, 2014).

As a baseline we use queries representing the experts' intuition about the realization of the NCCs in terms of POS tags and lemmata. We show that our system improves the precision of the query results even with little training data. Also we present run times for our ranking system under real-world conditions for different training set sizes. Further, we compare Krippendorff's α coefficient as an inter-annotator agreement measure among only annotators to the α which treats our system as one additional annotator.

We conducted the experiments based on the manually assigned labels of up to five annotators. If a sentence has been annotated by multiple users, we use the label that has been assigned by the majority; in case of a tie, we ignore the sentence. These so created gold standard annotations were used in an iterative cross-validation setting: for each query and the corresponding annotated sentences we ran nine cross-validation configurations, ranging from a $10/90$ split between training and testing data to a $90/10$ split, to investigate the reliability of the classifier as well as its ability to achieve usable results with little training data.

For *It-clefts*, we observe that elaborate queries already have a high precision, on which the SVM improves only marginally. The query

"It" /VCC[] [pos="NP0"]+ /RC[][4] (it17)

already yields a precision of 0.9598, which does not increase using our method (using 10% as training data, comparing the precision for the remaining 90%). However, while broader queries yield lower precision, the gain by using the SVM becomes significant (Table 1), as exemplified by the precision improvement from 0.4919 to 0.7782 for the following *It-cleft* query, even at a $10/90$ split.

"It" /VCC[] /NP[] /RC[][5] (it2)

For other inspected types of NCC, even elaborate queries yield a low baseline precision, which our approach can improve significantly. This effect can be observed for example in the following *NP-preposing* query, where precision can be improved from 0.3946 to 0.5871.

[pos="N."]{1,2} [pos="PNP" & word!="I"] [pos="V.*"]*[6] (np55)

We conducted a cursory, "real-world" test regarding the speed of the ranking system.[7] Training the SVM on differently sized subsets of the 449 sentences returned by a test query, we measured the time from clicking the *Rank results* button until the process was complete and the GUI had updated to reorder the sentences (i.e. including database queries, training, classifying, GUI update). The process times averaged over five "runs" for each training set size (20%, 50% and 90%) amount to 5 seconds, 7 seconds, and 14 seconds respectively. This leaves us with the preliminary impression that our system is fast enough for small to medium

[3] CSniper and the used SVM implementation are language independent, which allowed us to also run additional preliminary tests using German data.

[4] "It", verb clause, one or more proper nouns, relative clause. *VCC*, *NC*, and *RC* are macros we defined in CQP, see Table 2.

[5] "It", verb clause, noun phrase, relative clause.

[6] One to two nouns, personal pronoun other than "I", verb.

[7] System configuration: Intel i5 2,4 GHz, 2GB RAM, SSD 3GB/s, Linux in a VM

	it2	it17	it33	np34	np55	np76	pp42	pp99	pp103
Baseline	0.4919	0.9598	0.7076	0.4715	0.3946	0.4985	0.7893	0.4349	0.2365
SVM, 10/90	0.7782	0.9598	0.7572	0.5744	0.5871	0.5274	0.8152	0.8357	0.8469
SVM, 50/50	0.8517	0.9608	0.8954	0.6410	0.6872	0.6193	0.8657	0.8769	0.8720
SVM, 90/10	0.8634	0.9646	0.9261	0.6822	0.7723	0.6806	0.8865	0.8820	0.8796

Table 1: Precision for various NCC queries (*Baseline*) and for using the SVM with 10%, 50% and 90% training data.

sized training sets; as the last result suggests, for larger sets it would be desirable for our system to be faster overall. One way to achieve this is to pre-compute the feature vectors used in the training phase once — this could be done at the same time with the parsing of the sentences, i.e. at the setup time of the system.

Krippendorff's α, an inter-annotator agreement (IAA) measure which usually assumes values between 0 (no reliable agreement) and 1 (perfect agreement), amounts to 0.8207 averaged over all manually created *It-cleft* annotations. If we interpret the SVM as an additional annotator (α_{svm}), the IAA drops to 0.5903. At first glance this seems quite low, however upon closer inspection this can be explained by an overfitting of the classifier. This effect occurs for the already precise baseline queries, where in some cases less than 5% of the query results were labeled as *Wrong*. The same holds for *NP-preposing* (α: 0.6574, α_{svm}: 0.3835) and *PP-inversion* (α: 0.9410, α_{svm}: 0.6964). We interpret this as the classifier being successful in helping the annotators after a brief training phase identifying additional occurrences of particular variants of a phenomenon as covered by the queries, but not easily generalizing to variants substantially different from those covered by the queries.

4 Conclusion

With automatic ranking we introduced an extension to the *annotation-by-query* workflow which facilitates manual, selective annotation of large corpora. We explained the benefits of *in-tool* learning to this task and our extension of an open-source tool to incorporate this functionality. Finally, we showed the applicability of the concept and its implementation to the task of finding non-canonical constructions.

For future work, we plan to speed up the learning process (e.g. by saving feature vectors instead of re-calculating them), and also add the ability for users to configure the features used to train the classifier, e.g. incorporating lemmata or named entities instead of only using the parse tree. Integrating such configuration options in an easily understandable and user-friendly fashion may not be trivial but can help to generalize the approach to support additional kinds of sentence level annotation.

Acknowledgements

We would like to thank Pia Gerhard, Sabine Bartsch, Gert Webelhuth, and Janina Rado for annotating and testing. Furthermore we would like to thank Janina Rado for creating the CQP macros used in the tests.

This work has been supported by the German Federal Ministry of Education and Research (BMBF) under the promotional reference 01UG1416B (CEDIFOR), by the German Institute for Educational Research (DIPF) as part of the graduate program "Knowledge Discovery in Scientific Literature" (KDSL), and by the Volkswagen Foundation as part of the Lichtenberg-Professorship Program under grant No. I/82806.

References

Richard Eckart de Castilho and Iryna Gurevych. 2014. A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In *Proceedings of the Workshop on OIAF4HLT at COLING 2014*, pages 1–11.

Richard Eckart de Castilho, Iryna Gurevych, and Sabine Bartsch. 2012. CSniper: Annotation-by-query for Non-canonical Constructions in Large Corpora. In *Proceedings of ACL 2012*, System Demonstrations, pages 85–90, Stroudsburg, PA, USA. ACL.

Stefan Evert and Andrew Hardie. 2011. Twenty-first century corpus workbench: Updating a query architecture for the new millennium. In *Proceedings of CL2011*, Birmingham, UK.

Shortcut	Expansion
VCC	([pos="VBB" | pos="VBD" | pos="VBZ"]* [lemma="be"]) | ([pos="V.*"]* [pos="VBG" | pos="VBI" | pos="VBN"]* [lemma="be"])
NP	[pos="AT0"]? []? [pos="AJ.*"]* [pos="N.*"]
RC	([pos="DTQ" | pos="PNQ" | pos="CJT"] /VCF[] []?) | ([pos="CJT"]? /NP[] /VCF[] []?) | ([pos="PR.*"]* [pos=".Q"] /NP[] /VCF[] []?)
VCF	[pos="V.?B" | pos="V.?D" | pos="V.?Z" | pos="VM0"] [pos="V.*"]*

Table 2: CQP macro expansions for self-defined macros. BNC uses the CLAWS5 tagset for POS tags (http://www.natcorp.ox.ac.uk/docs/c5spec.html).

Ben Hachey, Beatrice Alex, and Markus Becker. 2005. Investigating the Effects of Selective Sampling on the Annotation Task. In *Proceedings of CoNLL 2005*, pages 144–151, Stroudsburg, PA, USA. ACL.

Rodney D. Huddleston and Geoffrey K. Pullum. 2002. *The Cambridge Grammar of the English Language*. Cambridge University Press.

Thorsten Joachims. 1999. Making large-scale support vector machine learning practical. In Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods*, pages 169–184. MIT Press, Cambridge, MA, USA.

Dan Klein and Christopher D. Manning. 2003. Accurate Unlexicalized Parsing. In *Proceedings of ACL 2003*, pages 423–430, Stroudsburg, PA, USA. ACL.

Thomas Morton and Jeremy LaCivita. 2003. WordFreak: An open tool for linguistic annotation. In *Proceedings of NAACL HLT 2003*, NAACL-Demonstrations, pages 17–18, Stroudsburg, PA, USA. ACL.

Alessandro Moschitti. 2006. Making Tree Kernels Practical for Natural Language Learning. In *Proceedings of EACL 2006*, pages 113–120, Trento, Italy.

Burr Settles. 2011. Closing the Loop: Fast, Interactive Semi-supervised Annotation with Queries on Features and Instances. In *Proceedings of EMNLP 2011*, pages 1467–1478, Stroudsburg, PA, USA. ACL.

The British National Corpus, version 3 (BNC XML Edition). 2007. Distributed by Oxford University Computing Services on behalf of the BNC Consortium. URL: http://www.natcorp.ox.ac.uk/.

Seid Muhie Yimam, Richard Eckart de Castilho, Iryna Gurevych, and Chris Biemann. 2014. Automatic Annotation Suggestions and Custom Annotation Layers in WebAnno. In *Proceedings of ACL 2014*, System Demonstrations, pages 91–96, Stroudsburg, PA, USA. ACL.