ExCiting Project

Trevor Killeen
Graeme Britz
Jesse Warren
{killeent, grbritz, jessewar}@cs.washington.edu

1. Project Motivation & Goals

Traditionally, the abstract of an academic paper provides a detailed summary of that paper. For certain use cases, such as giving a detailed summary of the paper in full, the abstract is an optimal source of information. In other situations, one may only care about the main takeaway of the paper. For example, when writing an academic paper, authors commonly cite previous works they have built upon. In this situation, a single concise summary of the work in question is more useful. Such concise summaries are not a built-in component of academic works.

By using the in-text citations of a paper to generate such a summary, we can leverage the work of other academics to determine the important aspects of the cited paper. This is especially useful to researchers, because they are concerned with a paper's importance to subsequent research.

As a result, our goal was to build a pipeline that takes as input an academic paper and a collection of papers that contain an in-text citation of that paper, and use those in-text citations to generate a concise and useful summary of the paper in question.

2. Data Set

We utilized the ACL Anthology Network corpus (Radev et al., 2013) to inform development of our pipeline and to generate test summaries for evaluation purposes. The ACL corpus contains over 20k papers from the field of Computational Linguistics, with more than 100k citations between papers in the data set.

3. Implementation Details

The main steps of our pipeline are:

1. Acquiring and preparing a data set of academic papers
2. Setting up and populating a database with the paper collection and relevant metadata
3. Extracting citation contexts
4. Extracting the sentences surrounding citation literals
5. Summarizing the extractions
6. Evaluating the summaries

Our pipeline was designed to work well with our example data set, but also to be relatively extensible so that other data sets could be used. Additionally, by decoupling each step in the process, one could easily swap in a new extraction or summarization mechanism. For example, one could use a different extraction technique that was tailored to a particular data set. We took advantage of this decoupling by running a number of different clustering and summarization algorithms over the same data.

3.1 Preparing the data set

A paper data set must contain two forms of metadata in order to support our pipeline. First, papers in the data set must be assigned a unique identifier (a "paper ID"). In the ACL dataset, the provided papers were labeled with unique 7- character identifiers, e.g. A12-3456. The other necessary component is a citation graph of the papers in the dataset. For every paper in the dataset, this graph contains an edge to every other paper in the dataset that cites this paper. The ACL dataset already contained such a citation graph, which mapped paper IDs to the papers IDs of papers that cited them. This allowed us to count the number of times a particular paper was cited by other papers in the set, and to identify those particular papers.

Because our goal was to generate summaries of papers from their citations in other works, we needed to ensure that we would have enough samples to generate useful summaries. We decided to only generate summaries for papers with over 100 in-citations, i.e. papers in our dataset that were cited by more than 100 other papers in the data set. We wrote and ran scripts to determine the subset of papers meeting this threshold and to extract the subset of papers in the data set that contained the citations. From our initial collection of 20k papers, we were left with 91 papers to summarize and around 8200 papers (the "citing papers"), which cited the original papers roughly 9000 times.

We did not experiment with different cutoffs. Because there is loss at each step of the pipeline, it is important to start with a sizable set of in-citations for each paper.

3.2 Picking and populating a database with papers and metadata

We stored our data in a MongoDB database. MongoDB is a document-based NoSQL DBMS with a rich query language and support for unstructured data. Both our initial uncertainty about the format of the data we wanted to store and what eventual data we would need to store motivated our decision to use MongoDB, as it supports flexible schemas that can be changed over time. Additionally, MongoDB provides the ability to store multiple collections (groups of documents) within the same database instance. This allowed us to quickly access both our raw data and generated summaries from the same command-line interface.

We initially populated our database with the papers in our dataset. We stored each paper as a MongoDB document containing the paper title, paper ID, publication year and author names.

3.3 Extracting citation contexts for papers in the data set

Next, we needed to extract the citations from the raw paper text. We used ParsCit (Councill et al., 2008), an open source library for extracting citation contexts from academic papers. Given a raw file containing the paper text, ParsCit will generate an XML document containing the chunk of text surrounding (and thus containing) each citation in the paper. These "citation contexts" are accompanied by metadata, including the title of the paper cited in that context. An example of an extracted chunk can be found in Sup. Fig. 1.

We wrote scripts to run ParsCit on the 8200 citing papers. The output XML files contained over 50,000 citation contexts. We had to match these contexts with one of the 91 papers we wanted to summarize, and discard those that were associated with papers outside of this subset. First, we used XPath to parse the XML files to extract the XML elements associated with citations.

Initially we tried to do exact string matching to map each extracted citation context to a paper within our set. However, we quickly realized that none of the titles within the contexts matched any title names within our set due to syntax errors associated with ParsCit. We contemplated calculating the Levenshtein distance between each context title and every title within our set. We would then map the title back to the paper in the set that had the shortest calculated distance. Upon closer inspection we realized that case-insensitive comparison with whitespace and punctuation trimming was sufficient and would allow us to map roughly 90% of the 9000 relevant citation contexts back to papers within our set.

We populated a MongoDB collection with these citation contexts. We stored each citation context as a MongoDB document containing the cited paper ID, the citer paper ID (i.e. the ID of the paper containing the citation context), the raw text and the citation literal (e.g. "Smith, 2009").

3.4 Extracting the sentence surrounding the citation literals in the citation context

The citation contexts generated by ParsCit are typically 7 to 10 sentences in length, with 3 to 4 sentences before and after the sentence containing the citation literal. Though these surrounding sentences may (and often do) contain additional information relevant to the particular citation the context is associated with, we decided to use only the sentence surrounding the citation literal.

We chose to use this limited scheme for a couple of reasons. First and foremost, determining the relevance of the surrounding sentences to the particular citation associated with a context is difficult to do automatically. Additionally, because our goal was to generate concise summaries of papers, lengthier, more detailed information would likely be discarded in our summarization process.

We wrote a Python script to extract the sentences surrounding the citation literal. This script lightly preprocessed the input citation contexts and used a naïve extraction algorithm to extract sentences. This script was not perfect, but was able to extract useful sentences roughly 90% of the time. An example of a sentence extraction is found in Sup. Fig. 2.

We populated a MongoDB collection with these extracted sentences. We stored each citation sentence as a MongoDB document containing the sentence, associated citation context and paper ID of the paper associated with this citation.

3.5 Summarization

Next we needed to summarize the groups of extracted sentences associated with each paper. For most of our summaries we used the same core summarizer, Sum_c. Sum_c is a naïve summarizer taken from Glowing Python that utilizes NLTK. It is based upon the idea that sentences containing the most recurrent words in a body of text (henceforth referred to as a document) probably cover many of the topics in that document. First, the algorithm transforms each sentence into a bag of words representation. Then, the word counts are normalized to be relative to the most frequent word in that document and ordered by the sum of these normalized frequencies. Sum_c then returns the top N sentences by this ranking. For our project, we were only looking at single sentence summaries, so we took the top ranked sentence each time. Sum_c uses the NLTK python library for tokenization and stop word filtering.

3.5.1 Evaluated Methods

We used Sum_c in three places in our evaluation:

1. To select a sentence from the abstract of a paper. This sentence was used as a baseline measure for comparison.
2. To summarize all of the extractions we had of citations for a paper. We did this by concatenating every extraction together into a single document and then feeding that into Sum_c. We named the sentence returned the "naïve-summary".
3. To summarize the "best" sentence containing our best n-gram. More information on how n-grams were used can be found below.

In each of these cases, we used the same summarizer to generate new summaries by feeding it different documents as input.

For our evaluated methods, we used n-grams as a method to filter our extractions and create a more focused document to feed into Sum_c. We created n-grams, with n ranging 1 to 5, using a Node.js script. We then computed TF-IDF scores for these n-grams and selected the n-gram with the highest TF-IDF score. Preference was given to longer n-grams by starting our selection process at the n=5 and selecting the n-gram with the highest TF-IDF score if it passed a threshold of 10.0. We then worked down from n=5 to n=1 until an n-gram met this threshold and thus was selected. If there were still no n-grams left at this point, we removed that document from evaluation, as it was not comparable to the others.

Once we had selected the highest n-gram, we filtered all of our extractions to create a document of only the extractions that contained that n-gram. Then we created two different summaries from this document.

1. The "best-ngram-summary" was taken from feeding this document into Sum_c and returning the highest ranked sentence.
2. The "longest-ngram-summary" was the longest sentence in this document.

Please refer to figure 3 in the appendix for screenshots of our evaluated summary methods and the baseline measure.

3.5.2 Other methods

We tried a number of other methods for summarization that did not make it into our final round of evaluation. With more time and refinement we would have liked to evaluate them. We think they could perform well. The most significant class of methods we tried concerned using clustering to identify different topics that citers would cite a paper for.

We used the K-Means algorithm to generate all of our clusters. K-Means requires numeric vectors as its input. To satisfy this requirement, we tried two different techniques. The first technique was to transform each extracted sentence into a bag of words representation. We then clustered the matrix created by these transformed sentences using the Node.js clusterfck library. Unfortunately the results of this method were too poor to consider for evaluation, which is why we did not include these clusters in our evaluation round.

Later, we appended TF-IDF scores of unigrams to the word count vectors and experimented with different numbers of clusters in the K-Means algorithm. This time we used Python's scikit-learn library to perform the clustering and transformations. We were able to achieve better results when this was done, but were not able to get any evaluation of these results before the end of the quarter.

We also explored using Latent Dirichlet Allocation and Suffix Tree clustering as ways to generate clusters and summaries, but were not able to achieve useful results in our timeline. We encourage future researchers to explore these and other methods more.

The clusters we generated were also stored in our MongoDB database.

4. Results

Our experiments were designed to answer two core questions:

1. Are our summaries preferable to what exists today?
2. How good are our summaries?

4.1 Experimental Setup

To evaluate preference, we used the paper abstract as a baseline to compare our summaries to against. To evaluate quality, we wanted to be able to rate the summaries for precision and recall. In our research, we found no automated way to evaluate our summaries that would work for our project, so several manual methods were considered. Crowdsourcing was deemed not viable due to the highly technical nature of our content – academic research papers. We considered two other options more strongly:

- Hiring graduate students to read the target papers and then rate the summaries we generated
- Contacting the original authors of the papers and ask them to evaluate our work

We decided to use the latter approach in this project, and leave the option of hiring evaluators to future research.

We were able to find 44 email addresses of authors of the papers we were interested in. We then created an email script and template and used them to contact each author. The emails explained our project and directed them to a web form (Sup. Fig. 4) used to capture their evaluations. The webpages were created using Express.js and the data was piped into our MongoDB database.

In the web forms we displayed four summary methods side by side:

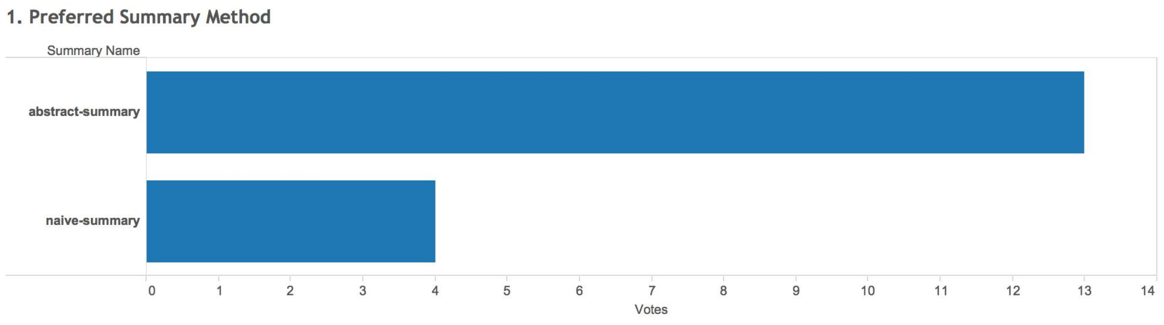
1. A sentence taken from the abstract (the “abstract-summary”)
2. The most probable sentence from all extractions (the “naïve-summary”),
3. The most probable sentence containing the best n-gram (the “best-ngram-sentence”)
4. The longest sentence containing the best n-gram (the “longest-ngram-sentence”)

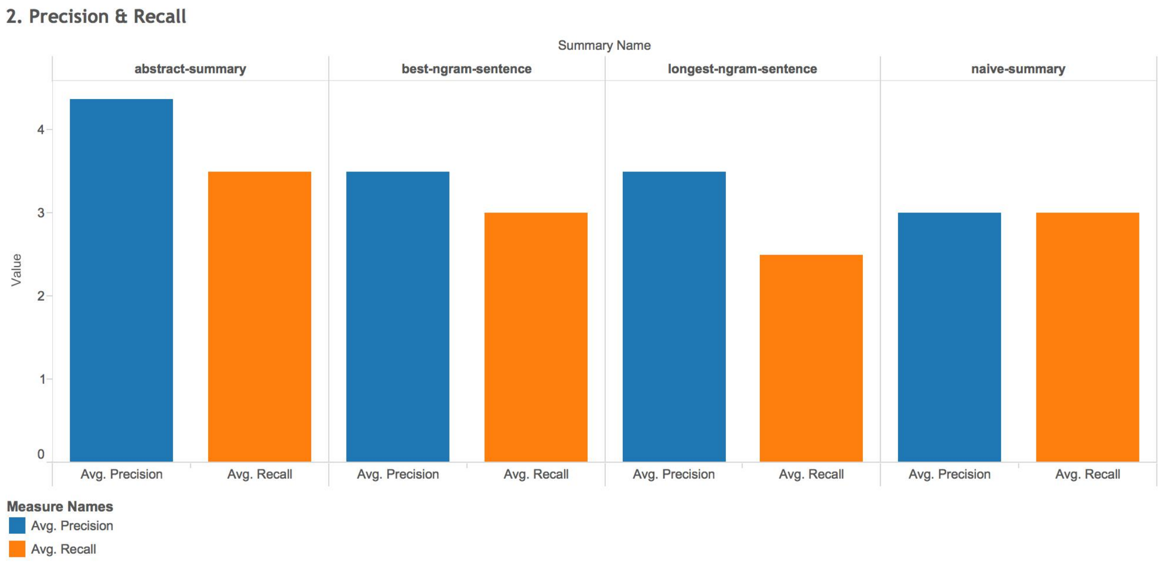
The first form asked respondents to select which method they preferred the best, and both forms asked respondents to rate summaries for perceived precision and recall score on a 1-5 scale.

4.2 Result Data

We had a 38.6% response rate for the first form (17 out of 44 authors responded). 8 respondents continued on to the second form, giving us a 47% carryover rate from the first and second form.

As you can see below in *Preferred Summary Method*, the respondents preferred the summaries generated from the abstracts the most. You can also see from *Precision & Recall,* that *abstract-summary* had the highest scores in both precision and recall of these methods.





4.3 Discussion

We were not surprised that our respondents preferred the "abstract-summary"– they were the authors of the abstracts after all. It also made sense that the "abstract-summary" performed the highest on precision, since it was taken directly from the paper. Likewise, we expected the n-gram methods to perform well on precision, since they targeted key phrases that were heavily referenced by citers. It is somewhat interesting that neither of the n-gram methods got votes on the first form given that they had similar precision and recall scores to the "naïve-summary", but this could be due to the small sample size of our dataset.

Overall, we did not have a large sample size to evaluate our methods, and we recommend more research to verify that these findings are meaningful. We also only evaluated our results on single sentence summaries; it is very possible that larger summaries would be found to be more meaningful.

5. Summary & Future Direction

Our initial goal was to create concise yet meaningful summaries of papers. We realized over time that the naïve extraction methods we had to use given our limited time and resources would generate suboptimal results.

For example, we chose to extract the sentences associated with a citation from citation contexts using simple regular expressions. As a result, the majority of our extractions, while containing useful and relevant information, were sentence fragments. Thus the algorithms we used to determine the most meaningful of these extractions from the group yielded sentences that usually contained meaningful information, but did not make much sense a complete description.

This led us to pivot to focusing on extracting the most important n-grams within a group of extractions. And while this diverged from the original goal of creating meaningful summaries, we came closing to achieving the related goal of identifying the main topics contained within a paper. Despite these shortcomings, we were still able to generate useful summaries in certain cases, and also set ourselves up well for future improvements.

Along the way, we learned a lot. This includes learning that data processing is often the majority of the work in a Machine Learning project. We spent more than half of our time acquiring and processing the data set before we were able to run algorithms. This led us to write numerous scripts to automate various data processing tasks. We also spent a lot of time in research, prototyping various algorithms and workflows, many of which we ultimately tossed away.

Looking forward, we believe the best next step would be to improve the way we extracted the text for a citation from its citation context. This would address our problem of attempting to cluster citation contexts that were at best a sentence fragment containing relevant information and at worst a group of words conveying no information about the paper whatsoever.

One way to achieve this goal would be to establish a crowdsourcing workflow where we would have a set of workers identifying what they consider to be the text associated with the citation and then a separate set of workers choosing their favorite between multiple citation texts that were previously identified. This process could be repeated until we believed the citation contexts were of significantly high quality. Applying clustering algorithms such as k-means on these extractions would likely yield better results. Experimenting with different vector representations to use for the k-means algorithm would also likely improve our results.

A few other areas of improvement would be to apply multiple iterations of evaluation, using the results of our evaluations to improve our techniques, and to reduce noise at the various stages in the pipeline.

6. Acknowledgements

The authors would like to thank Dan Weld and Jonathan Bragg for advising us throughout the course of the project. Furthermore, we would like to thank all of the authors involved in evaluating our summaries and providing helpful feedback. In particular, thanks to Marti Hearst and Jason Eisner with written feedback on how to improve our methods.

7. Appendix

7.1 Work Allocation

A separable breakdown of work is as follows:

- Trevor performed significant work on data preparation and context extraction, including extracting the subset of papers to summarize and the associated citing papers and writing the scripts to do sentence extractions. Trevor investigated the use of Suffix Tree Clustering of the sentence extractions, which we ultimately didn't use. He also authored the majority of this written report!

- Jesse performed significant work on parsing the XML output from ParsCit into citation contexts and on clustering the extractions using K-Means. Jesse wrote scripts to determine the most descriptive n-grams from citation contexts using a TF-IDF weighting scheme. He also wrote the email script used to contact authors.

- Graeme performed significant work on clustering using the K-Means algorithm and TF-IDF scores and summarization using a probabilistic approach. Graeme was responsible for sanitizing the dataset and aggregating and preparing the results for evaluation. Graeme was responsible for all web aspects of the project, including setting up an MTurk server (later discarded) and the forms used for collecting the final evaluation results. He also served as our MongoDB DBA.

All of us were involved in guiding the development of the project. We spent countless hours meeting and discussing ideas, researching open source libraries and prototyping ideas that we ended up throwing away.

7.2 External Code

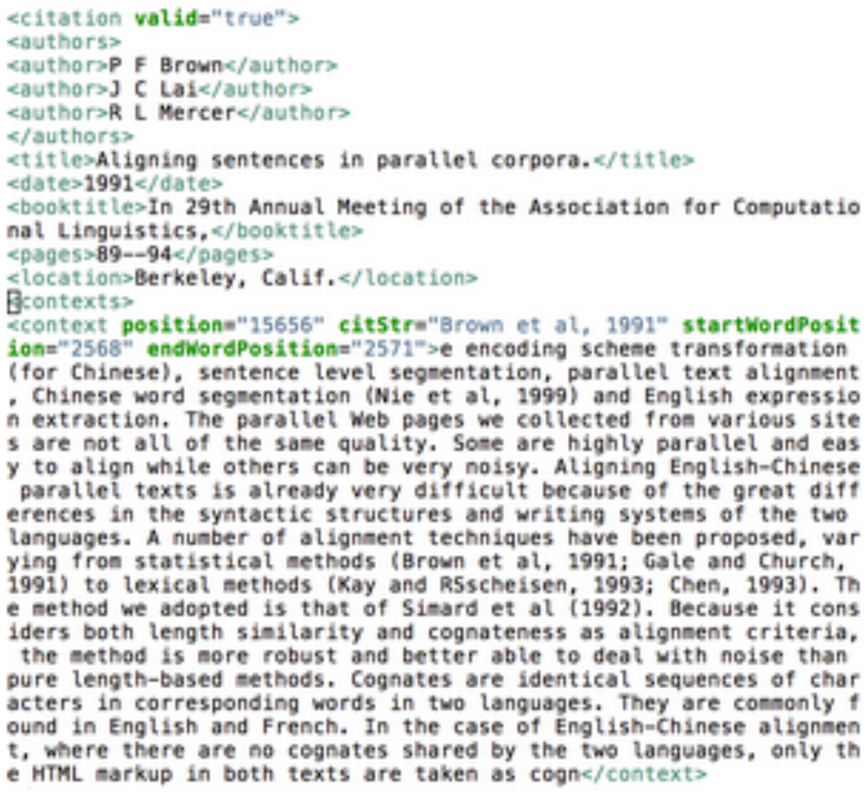
We used a number of open source libraries and tools, including:

- MongoDB: https://github.com/mongodb/mongo
- ParsCit: https://github.com/knmnyn/ParsCit
- Many Node.js packages: express, xpath, clusterfck, mongo, mathjs, common-node, mongoose, natural, stopwords, mongodb, underscore, string, xml2js, xmldom, xpath. See https://www.npmjs.com/
- Multiple Python libraries:
  - Scikit-learn (http://scikit-learn.org/)
  - NLTK (http://www.nltk.org)
  - SciPy (http://www.scipy.org/)
- Glowing Python NTLK summarizer (http://glowingpython.blogspot.com/2014/09/text-summarization-with-nltk.html)

7.3 Project Code

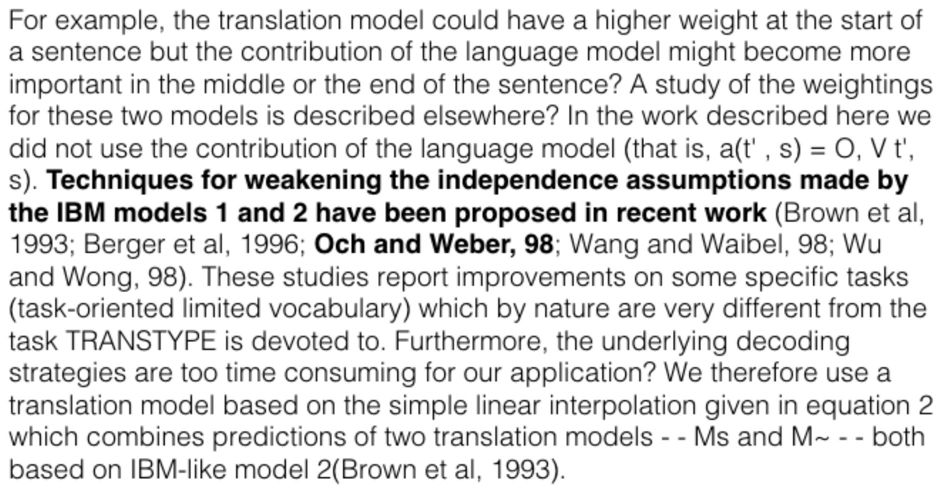
Project code is open source and available at https://github.com/jessewar/ExCiting. The majority of this code is associated with running the pipeline over our particular dataset, but can serve as an example implementation to guide development. The GitHub repository contains a README that gives a brief overview of the project and points users to this report for further reading.

7.4 Supplemental Figures



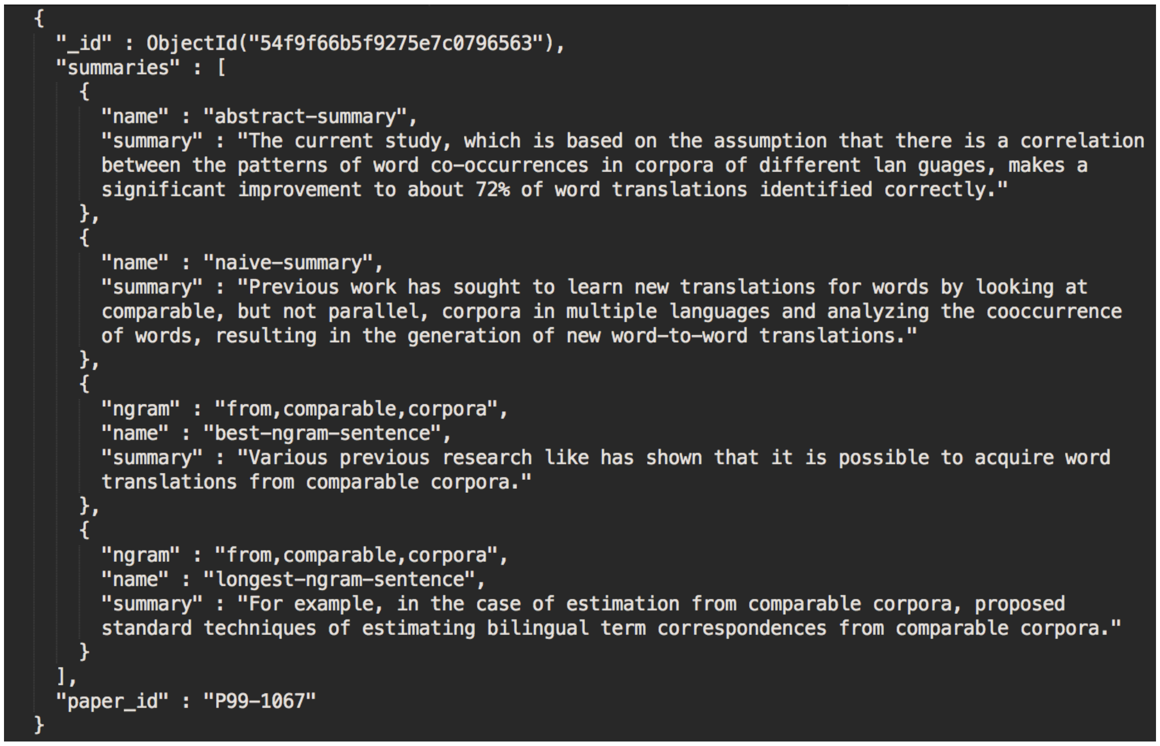
```
<citation valid="true">
<authors>
<author>P F Brown</author>
<author>J C Lai</author>
<author>R L Mercer</author>
</authors>
<title>Aligning sentences in parallel corpora.</title>
<date>1991</date>
<booktitle>In 29th Annual Meeting of the Association for Computatio
nal Linguistics,</booktitle>
<pages>89--94</pages>
<location>Berkeley, Calif.</location>
<contexts>
<context position="15656" citStr="Brown et al, 1991" startWordPosit
ion="2568" endWordPosition="2571">e encoding scheme transformation
(for Chinese), sentence level segmentation, parallel text alignment
, Chinese word segmentation (Nie et al, 1999) and English expressio
n extraction. The parallel Web pages we collected from various site
s are not all of the same quality. Some are highly parallel and eas
y to align while others can be very noisy. Aligning English-Chinese
 parallel texts is already very difficult because of the great diff
erences in the syntactic structures and writing systems of the two
languages. A number of alignment techniques have been proposed, var
ying from statistical methods (Brown et al, 1991; Gale and Church,
1991) to lexical methods (Kay and RSscheisen, 1993; Chen, 1993). Th
e method we adopted is that of Simard et al (1992). Because it cons
iders both length similarity and cognateness as alignment criteria,
 the method is more robust and better able to deal with noise than
pure length-based methods. Cognates are identical sequences of char
acters in corresponding words in two languages. They are commonly f
ound in English and French. In the case of English-Chinese alignmen
t, where there are no cognates shared by the two languages, only th
e HTML markup in both texts are taken as cogn</context>
```

Figure 1. A single extracted citation context outputted by ParsCit.



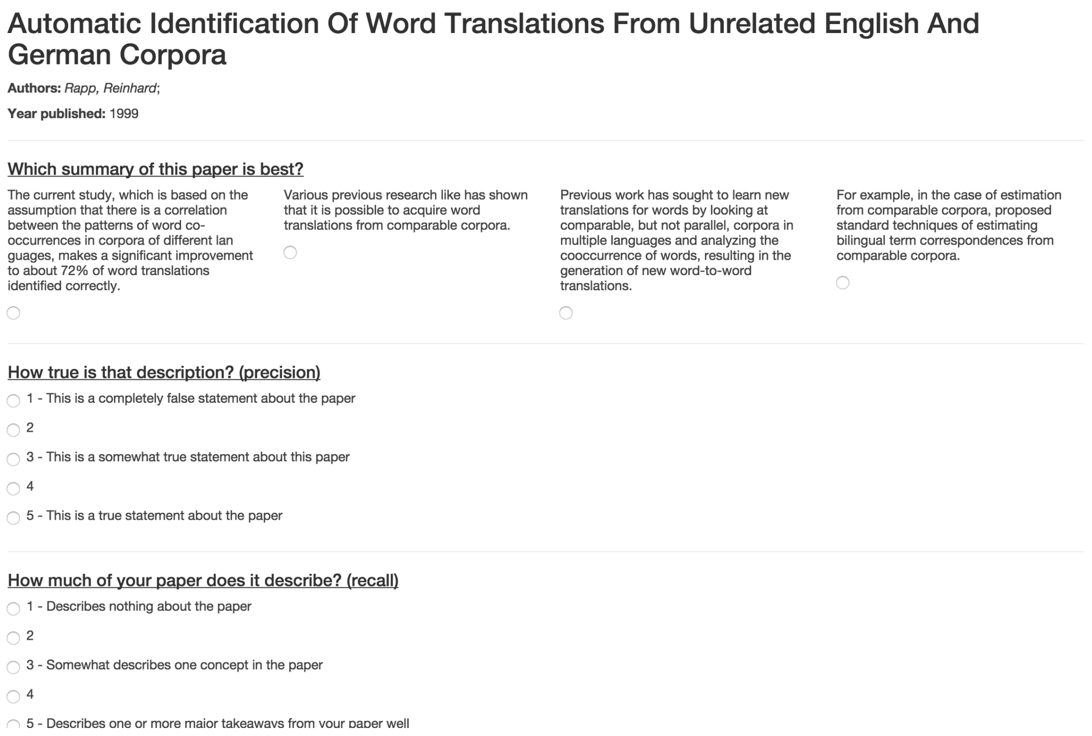
For example, the translation model could have a higher weight at the start of a sentence but the contribution of the language model might become more important in the middle or the end of the sentence? A study of the weightings for these two models is described elsewhere? In the work described here we did not use the contribution of the language model (that is, a(t' , s) = O, V t', s). **Techniques for weakening the independence assumptions made by the IBM models 1 and 2 have been proposed in recent work** (Brown et al, 1993; Berger et al, 1996; **Och and Weber, 98**; Wang and Waibel, 98; Wu and Wong, 98). These studies report improvements on some specific tasks (task-oriented limited vocabulary) which by nature are very different from the task TRANSTYPE is devoted to. Furthermore, the underlying decoding strategies are too time consuming for our application? We therefore use a translation model based on the simple linear interpolation given in equation 2 which combines predictions of two translation models - - Ms and M~ - - both based on IBM-like model 2(Brown et al, 1993).

Figure 2. An example of a sentence extracted from the citation context generated by ParsCit. The relevant citation literal is "Och and Weber, 98." The extracted sentence, in bold, will be used to summarize Och and Weber's paper.



```
{
  "_id" : ObjectId("54f9f66b5f9275e7c0796563"),
  "summaries" : [
    {
      "name" : "abstract-summary",
      "summary" : "The current study, which is based on the assumption that there is a correlation
      between the patterns of word co-occurrences in corpora of different lan guages, makes a
      significant improvement to about 72% of word translations identified correctly."
    },
    {
      "name" : "naive-summary",
      "summary" : "Previous work has sought to learn new translations for words by looking at
      comparable, but not parallel, corpora in multiple languages and analyzing the cooccurrence
      of words, resulting in the generation of new word-to-word translations."
    },
    {
      "ngram" : "from,comparable,corpora",
      "name" : "best-ngram-sentence",
      "summary" : "Various previous research like has shown that it is possible to acquire word
      translations from comparable corpora."
    },
    {
      "ngram" : "from,comparable,corpora",
      "name" : "longest-ngram-sentence",
      "summary" : "For example, in the case of estimation from comparable corpora, proposed
      standard techniques of estimating bilingual term correspondences from comparable corpora."
    }
  ],
  "paper_id" : "P99-1067"
}
```

Figure 3. The baseline abstract summary and our three generated summaries for the paper titled Automatic Identification of Word Translations from Unrelated English and German Corpora (Rapp 1999).



Automatic Identification Of Word Translations From Unrelated English And German Corpora

Authors: *Rapp, Reinhard*;

Year published: 1999

Which summary of this paper is best?

The current study, which is based on the assumption that there is a correlation between the patterns of word co-occurrences in corpora of different lan guages, makes a significant improvement to about 72% of word translations identified correctly. ○	Various previous research like has shown that it is possible to acquire word translations from comparable corpora. ○	Previous work has sought to learn new translations for words by looking at comparable, but not parallel, corpora in multiple languages and analyzing the cooccurrence of words, resulting in the generation of new word-to-word translations. ○	For example, in the case of estimation from comparable corpora, proposed standard techniques of estimating bilingual term correspondences from comparable corpora. ○

How true is that description? (precision)

○ 1 - This is a completely false statement about the paper
○ 2
○ 3 - This is a somewhat true statement about this paper
○ 4
○ 5 - This is a true statement about the paper

How much of your paper does it describe? (recall)

○ 1 - Describes nothing about the paper
○ 2
○ 3 - Somewhat describes one concept in the paper
○ 4
○ 5 - Describes one or more major takeaways from your paper well

Figure 4. A screenshot of a sample evaluation form for paper authors to complete.

8. References

Councill, Isaac G. and Giles, C. Lee and Kan, Min-yen. 2008. ParsCit: An open-source CRF reference string parsing package. In *International Language Resources and Evaluation*, Marrakech.

Radev, Dragomir R. and Muthukrishnan, Pradeep and Qazninian, Vahed and Abu-Jbara, Amjad. 2013. The ACL anthology network corpus. *Language Resources and Evaluation*, 47(4): 919-944.

Rapp, Reinhard. 1998. Automatic Identification of Word Translations from Unrelated English and German Corpora. In *ACL '99 Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics,* pages 519-526.