Hunter Gatherer: UdeM at 1CLICK-2

Pablo Duboue
Université de Montréal, Canada
dubouep@iro.umontreal.ca

Jing He
Université de Montréal, Canada
hejing@iro.umontreal.ca

Jian-Yun Nie
Université de Montréal, Canada
nie@iro.umontreal.ca

Team Name

UdeM

Subtasks

1CLICK-2 English

ABSTRACT

We describe our hunter-gartherer system for the NTCIR-10 1CLICK-2 task. We inspire ourselves on the DeepQA framework looking to adapt it for the 1CLICK task. Several techniques can be integrated naturally in this framework. The hunter component generates candidates based on the passage retrieval for the original query, the gartherer component collects evidence for each candidate and score them based on the collected evidence, and finally a summarization system is utilized to organized the high-quality candidates. The evaluation results show the effectiveness of this framework in the 1CLICK search task.

1. INTRODUCTION

Information retrieval (IR) aims to find relevant information for information needs. Most existing IR techniques consider the relevant information at document levels and rank documents according to their relevance to users' queries. Alternatively, 1CLICK search [16, 9] defines the information retrieval task in a finer granularity, i.e., information units (iUnits). In particular, it requires systems to return short answering text that contains these relevant information units. Figure 1 shows an example query and relevant iUnits, as provided by the organizers.

The 1CLICK Task itself involves then, given a query, produce a 1,000 characters summary (desktop version) or a 280 characters summary (mobile version) that are to be extracted from given 200 ranked pages provided by the organizers. The queries themselves belong to 8 known query types (celebrities, how to, location, etc), but the type is not explicitly stated with the query.

In this work, we inspire ourselves on the DeepQA framework looking to adapt it for the 1CLICK task. DeepQA framework has been successfully used in IBM Watson QA system for both Jeopardy Challenge and TREC QA task [5]. Generally, 1CLICK task is different from QA task, because it usually does not contain question word (except queries in QA category) and it is usually more general than a question. However, they both heavily rely on the search component, and they both need to score and organize the information candidates (iUnits in 1CLICK task and relevant nuggets in QA literatures). The advantage of DeepQA framework is that it can integrate a large number of knowledge learnt by diverse techniques to improve the answering performance. In our work we wanted to simplify the DeepQA architecture and apply an unified approach without query type identification.

The main idea of DeepQA framework is that each component is responsible for generating knowledge and corresponding confidence about the answer, and then we need to integrate these information to get the final result. There are three main components in the framework:

- Candidate Generation: generate candidate iUnits for a specific query
- Candidate Scoring: we can score candidate iUnits according to its features such as type, evidence strength, etc.
- Candidate Organization: organize the candidate in a piece of compact text

In the following three sections, we will describe Hunter Gatherer system[1] in detail.

2. HUNTER: CANDIDATE GENERATION

1CLICK is evaluated based on the iUnits in the return text, so the main task is to detect the relevant iUnits. However, it is difficult to determine the relevance between queries and iUnits directly. Thus we consider the relation between iUnits and queries in the context of passages. In particular, we need to identify candidate iUnits from query relevant passages, [2] and and we also need to use passages that contain both the query and the candidate iUnit as evidence to estimate the reliability of each candidate iUnit. We took the passages as they were returned by the search component, i.e., no attempt was made to further qualify them as answer-bearing passages [10] or their perceived relevance [7].

In the component of candidate generation, we first identify possible relevant iUnits as candidates for a specific query.

Generally, relevant iUnits should appear in relevant passages, so we need to acquire relevant passages by main search.

[1]Source code for Hunter-Gatherer is available at `https://github.com/jinghe/hunter-gatherer`

[2]In our work passages are simply defined as overlapped fixed windows in documents, since it shows that retrieval on fixed length passages can usually perform better than retrieval on semantic units such as paragraphs [1, 20]. We empirically set the passage length as 120 terms, and overlapping length as 50 terms in this work.

Query: `Whitney Houston Death`

Relevant information: *On February 11, 2012, Houston was found dead in suite 434 at the Beverly Hilton Hotel, submerged in the bathtub.*

V001001 February 11, 2012
V001002 Beverly Hilton Hotel
V001003 suite 434
V001004 submerged in the bathtub

the cause of Houston's death was drowning and the "effects of atherosclerotic heart disease and cocaine use".

V002001 drowning
V002002 atherosclerotic heart disease
V002003 cocaine

Figure 1: Query Example.

Table 1: Indri Query Setting

Text	Query	Phrase Type
A B	#1(A B)	named entity
A B	#combine(0.5 #1(A B) 0.5 #combine(A B))	pattern phrase

In main search, we first identify the phrases in the original query. Specifically, we use NTLK named entity recognizer to identify the named entities in the query (in the terminology of [3], we employ only "document search" candidates). If a phrase is a named entity, we require the words to appear in the exact order in the passages; The rebuilt query is represented in Indri query language [19], each token and phrases are connected by the "combine" operator. The details are presented in Table 1. For example, for the query "Whitney Houston death", we can get the token "death" and named entity "Whitney Houston" after parsing the query, so the Indri query can be expressed as

#combine[passage120:50](#1(Whitney Houston) death)

where we set retrieved passage length as 120 and overlap between candidate passages as 50. The "#1" operator means that "Whitney" and "Houston" should appear together in order and "#combine" combines the matching score of the two components.

The top passages of "main search" are likely to be relevant to the query, and we detect candidate iUnits from a relevant passage pool containing top K passages. In this stage, we pursue high recall of iUnits, so a large number of candidate iUnits are kept. We expect most of the irrelevant iUnits can be filtered by the candidate scoring component. We treat tokens and named entities appearing in the passage pool as candidate iUnits.

We also consider to extract some "key information" from the retrieval passages as candidate iUnits. For example, for some queries about persons or locations, their properties such as career and birthday can be considered as "key information". To extract such information, we need a training data containing labels about these information, and learn an extractor based on the training data. In our work, we simply consider the infobox in Wikipedia pages as "key information". The infobox in an wikipedia article is a fixed-format table designed to be added at the top right corner of the article, presenting a summary of some aspect the article share. For example, for a wikipedia article about a city, the corresponding infobox usually contains the information about its area, population, time zone, etc. In this way, we can get the training data by matching the infobox property text with the corresponding Wikipedia article. We use mallet [14] to train a CRF model [11] for the extraction purpose.

3. GATHERER: CANDIDATE SCORING

In this section, we will introduce the Candidate scoring component. In this component, we first gather the evidence for each candidate iUnit, and then integrate the evidence information to score the iUnit.

We use evidence search to gather evidence information for each candidate iUnit. In this step, we construct a new Indri language query containing both original query and iUnit. The original query part was built in the same way as described in the main search. For an iUnit, it can be a token, a named entity or a pattern based phrase extracted by CRF model. The query is created according to the rules defined in Table 1. Then the query is submitted to the Indri search engine and get a list of evidence passages. For the query "Whitney Houston death", we can get candidate iUnits such as "Beverly Hilton Hotel". In this step, we need to get evidence by searching with a query containing original query and iUnit information. For the iUnit "Beverly Hilton Hotel", our system can find that it is a named entity again, so the evidence search query is built as

#combine[passage120:50](#1(Whitney Houston) #1(Beverly Hilton Hotel) death)

For a candidate iUnit with the corresponding passages acquired from main search and evidence search, it can be scored by integrating these information. Intuitively, an iUnit should be more likely to be relevant if it appears in many high relevant passages in the main search, and there are many high relevant evidence passages to support it. Therefore, we can define a heuristic method to measure the relevance of an iUnit as follows:

$$R(q,u) = \lambda_1 \cdot \sum_{p \in MS, u \in p} (R(q,p) + \alpha) + \lambda_2 \cdot \sum_{p \in ES} (R(q,p) + \beta) \quad (1)$$

where q,u and p represent a query, an iUnit and a passage respectively, $R(q,p)$ is the relevance score for passage p according to query q, MS is the passage set from the main search and ES is the passage set from the evidence search. λ_1, λ_2, α and β are free parameters to control the importance of each components.

In a more sophisticated manner, we can integrate the above features in a learning framework. If we have a set of (query, iUnits) pairs, in which each iUnit is relevant to the query, we can learn a iUnit ranking model. However, since this is the first year for 1CLICK English task, no training data is available. Alternatively, we select 60 Wikipedia articles as the training data, in which the title is treated as the query, and iUnits in the first paragraph are treated relevant iUnits. Besides, for each query in this training data set, we also need passages from main search and evidence search to extract required features, and some irrelevant iUnits as negative examples. In this work, we use ClueWeb09 dataset for both main search and evidence search, and other candidate iUnits from the main search are treated as irrelevant iUnits. Given the training data, we learn an iUnit ranking model with gradient boosting tree method [12].

4. CANDIDATE ORGANIZATION

The required output of 1CLICK task is a compact piece of text instead of a list of relevant iUnits. Given the limited length of the text, it is still challengeable to contain these iUnits. Each sentence from the main search can be considered as a candidate for the final output, and we need to maximize the overall relevance information within the length constraint. In an ideal setting, given some training output, we could have used supervised learning to determine the best target sentences [6]. Without training data, however, the overall relevance information can be estimated by the sum of relevance score from the non-redundant iUnits. Thus, this problem is casted as an Integer Linear Programming problem [15].

We express selection of sentences as an optimization problem. Given NC nuggets (our candidates) and NS sentences, where some sentences contain some nuggets (expressed as a binary matrix M) and each nugget has an score (expressed by the vector W), we want to select sentences up to a certain length (the length of each sentence is contained in the vector L) so to maximize the scores of the contained nuggets, which in GLPK [13] is expressed as:

```
param NS; param NC; param K;
param M{1..NS, 1..NC}, binary;
param L{1..NS}, integer;
param W{1..NC};

var s{1..NS}, binary;
var e{1..NC}, binary;
maximize z: sum { i in 1..NC } e[i]*W[i];
subject to l: sum { i in 1..NS } L[i]*s[i] <= K;
subject to m {j in 1..NC}:
    sum { i in 1..NS } M[i,j]*s[i] >= e[j];
```

Using ILP produced the best results in our test runs, but we submitted also runs without using ILP due to time execution limitations. In such runs we realized duplicate passages were a big issue. We thus filtered with MMR method [2], using a bigram model and compute a distance between sentences as the Jaccard distance [8] between their bigrams, when taken as a set.

5. FINAL REMARKS

We submitted four runs in total:

UDEM-E-D-MAND-1 Basic Desktop Hunter Gatherer: 50 passages in main search, 20 top documents per candidate, 200 documents for evidence search.

UDEM-E-M-MAND-2 Basic Mobile Hunter Gatherer: same as above, but mobile version.

UDEM-E-D-MAND-3 Wiki Extractors Hunter Gatherer: same as UDEM-E-D-MAND1 but with Wikipedia-trained CRF extractors.

UDEM-E-D-MAND-4 ILP Hunter Gatherer: same as UDEM-E-D-MAND1 but with ILP.

We submitted three desktop runs: a baseline run using the MMR system described at the end of Sec. 4 (Run 1), an improved run using the CRF-based candidate extractors trained on Wikipedia (Run 3) and an ILP-based run (Run 4).[3] The results are in Table 2, compared also to the maximum, minimum and averages for each query category and overall for desktop mandatory runs. The first thing we can see from the table is that we had our runs as the top scoring and lower scoring submissions. This highlights the importance of adapting DeepQA ideas intelligently. We can also see Run 4 was one of the better performing runs, even though it makes no explicit distinction between query types. Having its roots on Question Answering, it is also good to see that it performs so much better in the QA category (it is 30% better than the second best submission in that category). The value of the CRF-based candidate extractors can be seen in some improvement in the celebrity-oriented queries (the first four query-type columns). We are intrigued about the behavior in the POLITICIAN category; we look forward studying other participant submissions to further elucidate the difference.

We sadly had no time to do an ILP-based ran for the mobile version nor the CRF extractors.

An after-submission error analysis showed we were negatively affected by:

- Spam, i.e., sentences and text with the intention to deceive search engines.
- Keywords in meta-tags in the head of a page (different from spam itself and easier to filter).
- Sentences unusually long but where only small segment was relevant.

All these issues should be addressable with further work. In particular, we are interested in exploring breaking apart multi-clause sentences leveraging work in text simplification [18] or sentence compression [4].

Other aspects we are interested in leveraging in future work is the use of unsupervised parsing [17] for phrase-based candidate generation.

[3]Run 4 was not using the CRF-based candidate extractors, as we wanted to see the different contribution of the two approaches. Run 3 had 8% more relevant candidates.

RUN	Category								
	All	ACTOR	ATHLE	ARTIST	POLIT	FACIL	GEO	DEFIN	QA
Run 1	0.047	0.040	0.028	0.039	0.037	0.060	0.025	0.066	0.068
Run 3	0.050	0.058	0.016	0.038	0.086	0.058	0.016	0.077	0.053
Run 4	0.080	0.068	0.084	0.074	0.025	0.079	0.062	0.076	0.146
MAX	0.080	0.068	0.084	0.074	0.086	0.083	0.080	0.088	0.146
MIN	0.047	0.040	0.016	0.018	0.025	0.005	0.016	0.055	0.053
AVRG	0.059	0.053	0.034	0.032	0.049	0.070	0.044	0.067	0.096
MEDIAN	0.055	0.053	0.028	0.027	0.039	0.076	0.035	0.066	0.089

Table 2: Evaluation results.

6. REFERENCES

[1] J. P. Callan. Passage-level evidence in document retrieval. In *Proceedings SIGIR '94*, pages 302–310, 1994.

[2] J. G. Carbonell and J. Goldstein. The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries. In A. Moffat and J. Zobel, editors, *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 335–336, Melbourne, Australia, 1998.

[3] J. Chu-Carroll and J. Fan. Leveraging wikipedia characteristics for search and candidate generation in question answering. In W. Burgard and D. Roth, editors, *AAAI*. AAAI Press, 2011.

[4] J. Clarke and M. Lapata. Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research*, 31(1):399–429, 2008.

[5] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, et al. Building Watson: an overview of the DeepQA project. *AI magazine*, 31(3):59–79, 2010.

[6] D. C. Gondek, A. Lally, A. Kalyanpur, J. W. Murdock, P. A. Duboue, L. Zhang, Y. Pan, Z. M. Qiu, and C. Welty. A framework for merging and ranking of answers in DeepQA. *IBM Journal of Research and Development*, 56(3.4):14:1 – 14:12, 2012. Digital Object Identifier: 10.1147/JRD.2012.2188760.

[7] J. He, P. Duboue, and J.-Y. Nie. Bridging the gap between intrinsic and perceived relevance in snippet generation. In *Proceedings of COLING 2012*, pages 1129–1146, Mumbai, India, December 2012. The COLING 2012 Organizing Committee.

[8] P. Jaccard. ÃL'tude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin de la SociÃl'tÃl' Vaudoise des Sciences Naturelles*, (37):547–579, 1901.

[9] M. P. Kato, M. Ekstrand-Abueg, V. Pavlu, T. Sakai, T. Yamamoto, and M. Iwata. Overview of the NTCIR-10 1CLICK-2 task. *Proceedings of the 10th NTCIR Conference*, 2013.

[10] E. Krikon, D. Carmel, and O. Kurland. Predicting the performance of passage retrieval for question answering. In X. wen Chen, G. Lebanon, H. Wang, and M. J. Zaki, editors, *CIKM*, pages 2451–2454. ACM, 2012.

[11] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.

[12] P. Li, C. J. C. Burges, and Q. Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. In *NIPS*, 2007.

[13] A. Makhorin. The GNU Linear Programming Kit (GLPK), 2000. Available online from `http://www.gnu.org/software/glpk/glpk.html`.

[14] A. K. McCallum. Mallet: A machine learning for language toolkit. 2002.

[15] R. T. McDonald. A study of global inference algorithms in multi-document summarization. In *ECIR*, pages 557–564, 2007.

[16] T. Sakai, M. P. Kato, and Y.-I. Song. Overview of NTCIR-9 1CLICK. *Proceedings of NTCIR-9*, pages 180–201, 2011.

[17] Y. Seginer. Fast unsupervised incremental parsing. In *Annual Meeting of the Association for Computational Linguistics*, volume 45, page 384, 2007.

[18] A. Siddharthan. Syntactic simplification and text cohesion. *Research on Language and Computation*, 4(1):77–109, 2006.

[19] T. Strohman, D. Metzler, H. Turtle, and W. B. Croft. Indri: A language model-based search engine for complex queries. In *Proceedings of the International Conference on Intelligent Analysis*, volume 2, pages 2–6. Citeseer, 2005.

[20] J. Zobel, A. Moffat, R. Wilkinson, and R. Sacks-Davis. Efficient retrieval of partial documents. *Inf. Process. Manage.*, 31:361–377, May 1995.