Learning to Rank Aggregated Answers for Crossword Puzzles

Massimo Nicosia[1,2], Gianni Barlacchi[2] and Alessandro Moschitti[1,2]

[1] Qatar Computing Research Institute
[2] University of Trento
m.nicosia@gmail.com, *gianni.barlacchi@gmail.com*, *amoschitti@qf.org.qa*

Abstract. In this paper, we study methods for improving the quality of automatic extraction of answer candidates for automatic resolution of crossword puzzles (CPs), which we set as a new IR task. Since automatic systems use databases containing previously solved CPs, we define a new effective approach consisting in querying the database (DB) with a search engine for clues that are similar to the target one. We rerank the obtained clue list using state-of-the-art methods and go beyond them by defining new learning to rank approaches for aggregating similar clues associated with the same answer.

1 Introduction

CPs are among the most popular language games. Automatic solvers mainly use AI techniques for filling the puzzle grid with candidate answers. The basic approach is to optimize the overall probability of correctly filling the grid by exploiting the likelihood of each candidate answer, fulfilling the grid constraints. Previous work [4] clearly suggests that providing the solver with an accurate list of answer candidates is vital. These can be (i) partially retrieved from the Web and (ii) most importantly they can be recuperated from a DB of previously solved CPs (CPDB). The latter contains clues from previous CPs, which are often reused: querying CPDB with the target clue may allow for recuperating the same (or similar) clues. It is interesting to note that all previous automatic CP solvers use standard DB techniques, e.g., SQL Full-Text query, for querying CPDBs. In [2], we showed that IR techniques can improve clue retrieval but our approach was limited on providing better ranking of clues whereas CP solvers require the extraction of the answer. In other words, given the list of similar clues retrieved by an IR system, a clue aggregation step and a further reranking process is needed to provide the list of answer candidates to the solver. More specifically, each clue c_i in the rank is associated with an answer a_{c_i}. A typical approach to select or rerank answers is to consider c_i as a vote for a_{c_i}. However, this is subject to the important problem that clues are relevant to the query with different probability. Trivially, a clue very low in the rank is less reliable than clues in the first position. One solution may use the score provided by the learning to rank algorithm (LTR) as a vote weight but as we show, its value is not uniformly distributed with respect to the probability of the correctness of c_i: this makes voting strategies less effective. In this paper, we study and propose different techniques for answer aggregation and reranking with the aim of solving the problem above. First of all, we apply logistic regression (LGR) to the scores produced by LTR algorithms for transforming them into probabilities. This way, we can apply a voting approach with calibrated probabilities, which improves on previous work. Secondly, we propose an innovative machine learning model for learning to combine the information that each c_i bring to their a_{c_i}: we define a representation of each a_{c_i} based on aggregate features extracted from c_i, e.g., their average, maximum and minimum reranking score. We experiment with this new answer representation with both LGR as well as SVM^{rank} [7]. Thirdly, another important contribution is the construction of the dataset for clue retrieval: it is constituted by 2,131,034 of clues and associated answers. This dataset is an interesting resource that we made available to the research community. Eventually, using the above dataset, we carried out two sets of experiments on two main tasks: (i) clue reranking, which focuses on improving the rank of clues c_i retrieved for a query; and (ii) answer reranking, which targets the list of a_{c_i}, i.e., their aggregated clues. The results of our experiments with the above dataset demonstrate that (i) standard IR greatly improves on DB methods for clue reranking, i.e., BM25 improves on SQL query by 6 absolute percent points; (ii) kernel-based rerankers using several feature sets, improves SQL by more than 15 absolute percent points; and (iii) using our answer aggregation reranking methods, the improvement on Recall (Precision) at rank 1, increases by additional 2 points absolute over the best results.

2 Related Work

There have been many attempts to build automatic CP solving systems. Their goal is to outperform human players in solving crosswords, more accurately and in less time. Knowledge about previous CPs is essential for solving new ones as clues often repeat in different CPs. Thus, all systems contain at least a module for clue retrieval from CPDBs. Proverb [8] was the first system for automatic resolution of CPs. It includes several modules for generating lists of candidate answers. These lists are merged and used to solve a Probabilistic-Constraint Satisfaction Problem. Proverb relies on a very large crossword database as well as several domain-specific expert modules. WebCrow [4] extends Proverb by applying basic linguistic analysis such as POS tagging and lemmatization. It uses semantic relations contained in WordNet, dictionaries and gazetteers. To exploit the database of clue-answer pairs, WebCrow applies MySQL match and Full-Text search functions. We used WebCrow as baseline as its CPDB module is one of the most accurate among CP resolution systems. This makes it one of the best system for Automatic CP resolution. The authors kindly made it available to us. It should be noted that, to the best of our knowledge, the state-of-the-art system is Dr. Fill [6], which targets the crossword filling task with a Weighted-Constraint Satisfaction Problem. However, its CPDB module is comparable to the one of WebCrow.

3 Advanced Learning to Rank Algorithms

We used the reranking framework applied to CPs described in [2]. This uses a preference reranking approach [7] exploiting structural kernels [10] and feature vectors.

Structural kernels. The model described in [11] are fed with a textual query and the list of related candidates, retrieved by a search engine (used to index a DB) according to some similarity criteria. Then, the query and the candidates are processed by an NLP pipeline, which contains many text analysis components: the tokenizer[3], sentence detector[1], lemmatizer[1], part-of-speech (POS) tagger[1], chunker[4] and stopword marker[5]. The output of these processors are used for building tree representations of clues. We use kernels applied to syntactic trees and feature vectors to encode pairs of clues in SVMs, which reorder the candidate lists. Since the syntactic parsing accuracy can impact the quality of our trees, and thus the accuracy of SVMs, we used shallow syntactic trees.

3.1 Feature Vectors

In addition to structural representations, we also used features for capturing the degrees of similarity between clues.

iKernels features (iK). these are a set of similarity features taking into account syntactic information captured by n-grams, and using kernels:
– *Syntactic similarities*. Several cosine similarity measures are computed on n-grams (with $n = 1, 2, 3, 4$) of word lemmas and part-of-speech tags.
– *Kernel similarities*. These are computed using (i) string kernels applied to clues, and tree kernels applied to structural representations

DKPro Similarity (DKP). We used similarity features used in Semantic Textual Similarity (STS) tasks, namely features in DKPro from the UKP Lab [1]. These features were effective in predicting the degree of similarity between two sentences:
– *Longest common substring measure* and *Longest common subsequence measure*. They determine the length of the longest substring shared by two text segments.
– *Running-Karp-Rabin Greedy String Tiling*. It provides a similarity between two sentences by counting the number of shuffles in their subparts.
– *Resnik similarity*. The WordNet hypernymy hierarchy is used to compute a measure of semantic relatedness between concepts expressed in the text.
– *Explicit Semantic Analysis* (ESA) similarity [5]. It represents documents as weighted vectors of concepts learned from Wikipedia, WordNet and Wiktionary.
– *Lexical Substitution* [3]. A supervised word sense disambiguation system is used to substitute a wide selection of high-frequency English nouns with generalizations. Resnik and ESA features are computed on the transformed text.

WebCrow features (WC). We included the similarity measures computed on the clue pairs by WebCrow and the Search Engine as features:
– *Lucene Score*. BM25 score of the target candidate.
– *Clue distance*. It quantifies how dissimilar the input clue and the retrieved clue are. This formula is mainly based on the well known Levenshtein distance.

[3] http://nlp.stanford.edu/software/corenlp.shtml
[4] http://cogcomp.cs.illinois.edu/page/software_view/13
[5] Stopwords: https://github.com/mimno/Mallet/blob/master/stoplists/en.txt

4 Aggregation Models for Answer Reranking

CP resolution is a sort of question answering task: it requires extracting the answer rather than a set of ranked clues. Groups of similar clues retrieved from the search engine can be associated with the same answers. Since each clue receives a score from the reranker, a strategy to combine the scores is needed. We aim at aggregating clues associated with the same answer and building meaningful features for such groups. We designed two different strategies: (i) apply LGR to the scores of our reranker to obtain probabilities and then sum together those referring to the same answer candidates; and (ii) represent each answer candidate with features derived from all the clues associated with it, i.e., their aggregation using standard operators such average, min. and max.

Logistic Regression Model. The search engine or the reranker associate clues with scores that are not probabilities and have their own distributions. In contrast, LGR assigns probabilities to answer candidates. Such probabilities, learned using also additional features, are more effective for aggregation. We apply the following formula: $Score(G) = \frac{1}{n}\sum_{c \in G} \frac{P^{LR}(y=1|\boldsymbol{x}_c)}{rank_c}$ to obtain a single final score for each different answer candidate, where c is the answer candidate, G is the set of clue answers equal to c, and n is the size of the answer candidate list. $\boldsymbol{x}_c$ is the feature vector associated with $c \in G$, $y \in \{0, 1\}$ is the binary class label ($y = 1$ when c is the correct answer). $rank_c$ is the rank assigned from the reranker to the word c. Eventually, we divide the probability by the rank of the answer candidate to reduce the contribution of bottom candidates. The conditional probability computed by the linear model is the following: $P^{LR}(y = 1|c) = \frac{1}{1+e^{-y\boldsymbol{w}^T\boldsymbol{x}_c}}$, where $\boldsymbol{w} \in \mathbb{R}^n$ is a weight vector [12].

Learning to rank aggregated answers. We apply SVM^{rank} to rerank each set of clues having the same answer candidate. To build the feature vectors associated with such groups, we average the features used for each clue by the first reranker, i.e., those described in Sec. 3.1. We call these features **FV**. Additionally, we compute the sum and the average of the scores, the maximum score, the minimum score and the term frequency of the word in the CPDB Dataset. We call them (**AVG**). Eventually, we model the occurrences of the answer instance in the list by means of positional features: we use n features, where n is the size of our candidate list (i.e., 10). Each feature corresponds to the positions of the answer instance in the list. We call them (**POS**).

5 Experiments

The experiments compare different ranking models. i.e., WebCrow, BM25 and several rerankers, for the task of clue retrieval. Most importantly, we show innovative models for aggregating and reranking answers based on LGR and SVM^{rank}.

5.1 Database of previously resolved CPs (CPDB)

We compiled a crosswords corpus combining (i) the downloaded CPs from the Web[6] and (ii) the clues database provided by Otsys[7]. We removed duplicates, fill-in-the-blank

[6] http://www.crosswordgiant.com
[7] http://www.otsys.com/clue

Model	MRR	SUC@1	SUC@5
WebCrow (WC)	64.65	57.14	74.98
BM25	75.17	63.78	90.40
RR (iK)	78.01	67.34	92.32
RR (iK+DKP)	80.89	71.62	93.14
RR (iK+DKP+WC)	**81.70**	**72.50**	**94.02**

Table 1: Similar Clue Reranking

Model	MRR	SUC@1	SUC@5
Raw voting	41.33	17.44	78.48
LGR voting	83.16	73.18	96.68
SVM (AVG+POS)	83.49	73.82	96.78
SVM (AVG+POS+FV)	**83.95**	**74.60**	**96.78**
LGR (AVG+POS+FV)	81.70	73.54	96.74

Table 2: Answer reranking

clues (which are better solved by using other strategies) and clues representing anagrams or linguistic games. The resulting compressed dataset, called CPDB, contains 2,131,034 unique and standard clues, with associated answers.

5.2 Experimental Setup

We used SVM-light-TK[8] to train our models, with default parameters. It enables the use of structural kernels [10] in SVM-light [7]. We applied a polynomial kernel of degree 3 to the explicit feature vectors. To measure the impact of the rerankers as well as the baselines, we use: success at rank 1 (SUC@1), which is the percentage of questions with a correct answer in the first position; Mean Reciprocal Rank (MRR), which is computed by $\frac{1}{|Q|}\sum_{q=1}^{|Q|}\frac{1}{rank(q)}$, where *rank(q)* is the position of the first correct answer in the candidate list; and success at rank 5 (SUC@5), which is the percentage of questions with at least one correct answer in the first 5 reranked clues.

5.3 Ranking results

To build the reranking training and test set, we used the clues contained in CPDB for querying the search engine, which retrieves a list of candidates from the indexed clues excluding the input clue. For each input clue, similar candidate clues are retrieved and used to form a first list for the reranker. The training set is composed by 8,000 unique pairs of clue/answer that have at least one correct answer in the first 10 candidates retrieved by the search engine. We also created a test set containing 5,000 clues that are not contained in the training set. We tested two different models: (i) BM25 and (ii) reranking models (RR). Since WebCrow includes a database module, we also report its accuracy. We used the BM25 implementation of Lucene [9] as the IR Baseline: lists are ordered using Lucene scores. To rerank these lists, we tried different combinations of features for the rerankers, described in Section 3.1. The results in Tab. 1 show that: (i) BM25 produces an MRR of 75.17%, which improves on WebCrow by more than 6.5 absolute percent points, demonstrating the superiority of an IR approach over DB methods; (ii) RR (iK) achieves a higher MRR, up to 4 percent absolute of improvement over BM25 and thus about 10.5 points more than WebCrow. With respect to this model, the improvement on MRR of (iii) RR (iK+DKPro) is up to 1.2 percent points and finally, (iv) RR (iK+DKP+WC) improves the best results of another full percent point. Tab. 2 shows the results for answer reranking: (i) voting the answer using the raw score of the reranker is not effective; (ii) voting, after transforming scores into probabilities with LGR, improves on the best clue reranking model in terms of SUC@1 and MRR; (iii)

[8] http://disi.unitn.it/moschitti/Tree-Kernel.htm

the SVM^{rank} aggregation model using AVG and POS feature sets improves on the LGR voting model; (iv) when FV are added we notice a further increase in MRR and SUC@1; (v) LGR on the same best model AVG+POS+FV is not effective, showing that ranking methods are able to refine answer aggregation better than regression methods.

6 Conclusions

In this paper, we improve the answer extraction from DBs for automatic CP resolution. We design innovative learning to rank aggregation methods based on SVMs on top of state-of-the-art rerankers designed for clue reordering. Our approach first retrieves clues using BM25, then applies SVMs based on several features and tree kernels and eventually, collapses clues with the same answers, thus modeling answer reranking. The latter uses innovative aggregation features and positional features. The comparisons with state-of-the-art CP solvers, i.e., WebCrow, show that our model relatively improves it by about 30% (16.4 absolute percent points) in SUC@1 and even more on MRR. For our study, we collected over 6 millions of English clues and we created a dataset for clue similarity with over 2 millions of English clues. This is an important resource for IR research that we make available to the community.

References

1. Bär, D., Zesch, T., Gurevych, I.: Dkpro similarity: An open source framework for text similarity. In: Proceedings of ACL (System Demonstrations) (2013)
2. Barlacchi, G., Nicosia, M., Moschitti, A.: Learning to rank answer candidates for automatic resolution of crossword puzzles. In: Proceedings of the Eighteenth Conference on Computational Natural Language Learning. Association for Computational Linguistics (June 2014)
3. Biemann, C.: Creating a system for lexical substitutions from scratch using crowdsourcing. Lang. Resour. Eval. 47(1), 97–122 (Mar 2013)
4. Ernandes, M., Angelini, G., Gori, M.: Webcrow: A web-based system for crossword solving. In: In Proc. of AAAI 05. pp. 1412–1417. Menlo Park, Calif., AAAI Press (2005)
5. Gabrilovich, E., Markovitch, S.: Computing semantic relatedness using wikipedia-based explicit semantic analysis. In: Proceedings of the 20th International Joint Conference on Artifical Intelligence. pp. 1606–1611. IJCAI'07 (2007)
6. Ginsberg, M.L.: Dr.fill: Crosswords and an implemented solver for singly weighted csps. J. Artif. Int. Res. 42(1), 851–886 (Sep 2011)
7. Joachims, T.: Optimizing search engines using clickthrough data. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 133–142. KDD '02, ACM, New York, NY, USA (2002)
8. Littman, M.L., Keim, G.A., Shazeer, N.: A probabilistic approach to solving crossword puzzles. Artificial Intelligence 134(12), 23 – 55 (2002)
9. McCandless, M., Hatcher, E., Gospodnetic, O.: Lucene in Action, Second Edition: Covers Apache Lucene 3.0. Manning Publications Co., Greenwich, CT, USA (2010)
10. Moschitti, A.: Efficient convolution kernels for dependency and constituent syntactic trees. In: ECML. pp. 318–329 (2006)
11. Severyn, A., Moschitti, A.: Structural relationships for large-scale learning of answer reranking. In: Proceedings of ACM SIGIR. New York, NY, USA (2012)
12. Yu, H.F., Huang, F.L., Lin, C.J.: Dual coordinate descent methods for logistic regression and maximum entropy models. Mach. Learn. 85(1-2), 41–75 (Oct 2011)