Answer Extraction with Multiple Extraction Engines for Web-Based Question Answering

Hong Sun[1], Furu Wei[2], and Ming Zhou[2]

[1] School of Computer Science and Technology, Tianjin University, Tianjin, China
kaspersky@tju.edu.cn
[2] Microsoft Research Asia, Beijing, China
{fuwei,mingzhou}@microsoft.com

Abstract. Answer Extraction of Web-based Question Answering aims to extract answers from snippets retrieved by search engines. Search results contain lots of noisy and incomplete texts, thus the task becomes more challenging comparing with traditional answer extraction upon offline corpus. In this paper we discuss the important role of employing multiple extraction engines for Web-based Question Answering. Aggregating multiple engines could ease the negative effect from the noisy search results on single method. We adopt a Pruned Rank Aggregation method which performs pruning while aggregating candidate lists provided by multiple engines. It fully leverages redundancies within and across each list for reducing noises in candidate list without hurting answer recall. In addition, we rank the aggregated list with a Learning to Rank framework with similarity, redundancy, quality and search features. Experiment results on TREC data show that our method is effective for reducing noises in candidate list, and greatly helps to improve answer ranking results. Our method outperforms state-of-the-art answer extraction method, and is sufficient in dealing with the noisy search snippets for Web-based QA.

Keywords: Web-based Question Answering, Answer Extraction, Rank Aggregation, Learning to Rank.

1 Introduction

Question Answering (QA) aims to give exact answer to questions described in natural language. Some QA systems directly employ well-built search engines for this task which are called Web-based QA systems [1]. This kind of systems contain three modules: 1) question analysis module to analyze question and generate queries; 2) passage retrieval module to retrieve relevant passages from search engine; 3) answer extraction module to extract the final answer. Comparing with traditional QA, Web-based QA can take advantage of the tremendous data resource provided by Web and eliminate the efforts to store and index huge amount of documents. Current search engines are becoming more and more sophisticated, so Web-based QA can also benefit from the optimized search results provided by search engines.

C. Zong et al. (Eds.): NLPCC 2014, CCIS 496, pp. 321–332, 2014.

We focus on Answer Extraction task for Web-based QA. The task is to generate exact answers with search snippets. It contains two steps: extract candidates such as noun phrases and later rank them based on ranking function, e.g., similarity between question and sentence bearing the candidate. Traditional web-based answer extraction is conducted on search snippets [1] instead of plain texts in the retrieved web pages, so that it can utilize the high-quality summarizations generated by search engines without parsing or mapping sentences into the original web pages. The side-effect is that answer extraction results are influenced by the incomplete and noisy search snippets. On the one hand, using search snippets containing many negative sentences results with large amount of noisy candidates; on the other, state-of-the-art answer extraction methods rely on syntactic information [2,3] could be seriously affected by the incomplete structure of search text. Previous work about Web-based QA have discussed using n-grams as candidates and rank them based on redundancies [10]. This method eliminates the need of deep understanding of the noisy search snippets and leverages substantial redundancy encoded in the large amount of search texts. However, enumerating n-grams results with even more negative candidates in the hypothesis space which poses too much address on ranking [4].

Single extraction method is easily affected by the noisy search snippets, in order to ease the problem, we discuss an effective way by adopting multiple extraction engines for Web-based QA. Different engines analyze texts from different aspects, the chance for them all being wrong is small, thus the consensus information among them is useful for alleviating the impact from the noisy texts. With multiple engines, we can perform more strict pruning to filter noisy candidates and perform more accurate ranking. Specifically, we first aggregate multiple extraction engines' results with a Pruned Rank Aggregation method. We employ a modified Supervised Kemeny Ranking model for rank aggregation and at the same time perform pruning on each engine's list based on redundancies within and acrose different extraction engines. After generating the list, we use Learning to Rank with similarity, redundancy, quality and search features to rank the candidate list. Experiments on TREC dataset show that our pruning and ranking algorithm is efficient for reducing noises in candidate list and achieving better ranking results than state-of-the-art answer extraction method. This result makes Web-based QA more accurate, robust and applicable in real application scenarios.

2 Related Work

Our work focus on Answer Extraction of Web-based QA. Methods for this task can be classified as two types.

The first type of methods consider information from question when generating candidates and use relatively simple ranking functions. For example, in traditional QA, one most commonly adopted method is to extract Named Entities (NE) matching with answer type with Named Entity Recognition (NER) tools [5]. For this method, errors in answer type classification will propagate to

the stage of answer extraction, and performance of answer extraction will be limited by performance of NER. In Web-based QA, search snippets are different from training texts of NER, this makes results of NER significantly degraded [4]. Pattern-based method is another commonly used method [6]. It has high precision, but those patterns are defined on predicates which are very fine-grained and not easy to be adapted to new data. Recent work has studied to apply machine learning in answer extraction. The first attempt is to view answer extraction as a process to extract question-biased terms [7], each token in passage is classified as is or is not an answer token. Within this direction, factor graph [4] is used to consider the relations among different passages, and Tree Edit Distance is considered in later work [2] for a more accurate similarity measurement between question and passages.

The second type of method is to generate candidates based on text's structure or dictionaries. Noun Phrase (NP) is one commonly used unit [3], and ranking score of an NP can be calculated based on syntactic structure of passage. In the state-of-the-art method [3], each NP considered as a candidate is aligned to Focus [8] on the parsing tree; thus for each alignment, a similarity score could be calculated based on tree kernel. This method relies on syntactic structure and is affected by the noisy search snippets. We'll show in our experiments that the performance of this method seriously degraded in the search contexts of Web-based QA. Besides, some work use dictionaries to generate candidates, for example, state-of-the-art QA system Watson [9] extracts Wikipedia titles from passages, and employs Learning to Rank with hundreds of features in answer ranking. This method requests lots of human effort for feature engineering has low adaptability to new domains [3]. Besides, n-grams in texts are commonly employed in Web-based QA [10]. Such method uses answer type to filter out candidates and perform tiling on candidates with overlaps. Frequencies weighted with prior score of query are viewed as the ranking score. Extracting n-grams results with large amount of candidates containing more noises thus requires more sophisticated ranking function and features [4].

3 Method

Our method for answer extraction contains two steps: 1) joint aggregation and pruning of candidate lists generated by different extraction engines; 2) rank the candidate list generated by previous step.

3.1 Pruned Rank Aggregation

The first step could be viewed as a rank aggregation task of candidate lists generated by multiple extraction engines. As search snippets provide various of texts from Web, they contain many noises. For example, in our statistics 19% sentences in TREC corpus [2] contain correct answer, while such ratio is around 10% in search result. This indicates more negative snippets without containing correct answer will bring many noisy candidates to answer extraction. Thus it's

necessary to perform strict pruning during or after the aggregation to reduce the amount of noises.

Algorithm 1. Pruned Rank Aggregation with Multiple Extraction Engines

Input: $C_i = \{c_{i1}, ..., c_{il}\}$ as ordered candidate list from engine i, $l_i = |C_i|$, each list ordered by frequency of candidate $n_{c_{ij}}$; $\{w_1, ..., w_k\}$ are weights of each engine, $0 \leq w_i \leq 1, \sum_i w_i = 1$; $\{t_i, ...t_k\}$ are single engine pruning thresholds, $0 \leq t_i \leq 1$; p, m, n are pruning thresholds after aggregation.

Output: $A = a_1, ..., a_n$ is candidate list after aggregation and pruning

1: Initialize $A = \emptyset$
2: **for all** C_i **do**
3: **for all** $c_{ij} \in C_i$ **do**
4: **if** !ContainContentWord(c_{ij}) or ($j > t_i \cdot l_i$ and !Exists(c_{ij}, C'_i),$\forall i' \neq i$) **then** Remove c_{ij} from C_i
5: **end if**
6: **end for**
7: Update $l_i = |C_i|$
8: **end for**
9: **function** MergeListWithModifiedSKR($\{C_i\}$)
10: Initialize $M_{i,j} \Leftarrow 0$, $C' = \bigcup_i C_i$, update frequency $n_{c_k} = \Sigma_j n_{c_{ij}}$, if $c_k = c_{ij}$
11: **for all** candidate list C_i **do**
12: **for all** $j = 1$ to $l_i - 1$ **do**
13: **for all** $l = j + 1$ to l_i **do** $M_{c_{ij},c_{il}} \Leftarrow M_{c_{ij},c_{il}} + w_i \cdot \log(n_{c_{ij}} - n_{c_{il}} + 0.1)$
14: **end for**
15: **end for**
16: **end for**
17: Quick sort C' with $M_{C'_i,C'_j}$, candidate with larger value gets prior order
18: **return** C'
19: **end function**
20: Compute word frequency f_{w_j} for all $w_j \in c_i$ and !IsStopword(w_j), where $c_i \in C'$
21: **for all** $c_i \in C'$ **do**
22: **if** $i \leq |C'| \cdot p$ or $\Pi f_{w_i} \geq n, w_i \in c_i$ or $n_{c_i} \geq m$ **then** Add c_i to A
23: **end if**
24: **end for**

Thus we propose a Pruned Rank Aggregation method for joint aggregating and pruning candidate lists generated by multiple extraction engines. In search results, texts are more redundant than off-line corpus, so redundancy provides us with useful information in distinguishing between good and low-quality candidates. In addition, as different extraction engines perform analysis differently, if a candidate is supported by multiple engines, it is unlikely that it's a noisy candidate. Thus we design the method for pruning based on inner and inter redundancies of different extraction engines during rank aggregation. The algorithm is shown in Algorithm 1. Given different candidate lists pre-ordered by frequencies of candidates, line 1-8 first prune each candidate list to filter candidates existing in only one list and with low rank[1]. Line 9-19 employ a modified

Supervised Kemeny Ranking (SKR) [11] to aggregate different lists, element of matrix in SKR is weighted by difference between frequencies of two candidates in the same list (line 13). Later, the aggregated list is further pruned at line 20-24 based on global candidate-based and word-based frequencies. Pruning parameters $\{t_i\}$, p, m, n in the algorithm are tuned with development set; w_i is calculated as accuracy of each engine in development set. After Pruned Rank Aggregation, we get a candidate list with less noisy candidates, and at the same time, recall of answer is not hurt comparing to aggregation without pruning. Such a result helps to increase answer ranking result in the later stage.

3.2 Ranking

Ranking task is that given question Q, search snippets $S = \{s_1, s_2, ..., s_n\}$, external Knowledge Base K, candidates $A = \{a_1, a_2, ..., a_m\}$, perform ranking with scoring function:

$$\begin{aligned} f(a_i) &= P(\ a_i \mid Q,\ S,\ K,\ A) \\ &= \Sigma_j\ \lambda_j \cdot h_j(a_i,\ Q,\ S,\ K,\ A) \end{aligned} \tag{1}$$

where $\{h_j(\cdot)\}$ is a set of ranking features and λ_j is the corresponding feature weight. Evidences for ranking in this work come from question, search results, Knowledge Base as well as the whole hypothesis space.

After generating the score, answer extraction ranks candidates with the score and selects candidate answer with the highest score as the final output:

$$\widehat{a} = arg \max_{a_i \in A} f(a_i) \tag{2}$$

Previous work has showed that Learning to Rank works well for answer ranking of factoid question [12]. In this work, we follow the strategy and adopt *Rank SVM* [13] as our answer ranking model. It converts ranking problem to a binary classification problem during training, each pair of candidates is viewed as a positive training sample if correct candidate's score is higher than negative ones' and vice versa. During predication, each candidate's score is estimated with features and the weights.

We define 5 different feature sets to capture quality of a candidate from different aspects(number in () indicates number of the features):

Consensus Features (9). This set measures candidates' agreement across different extraction engines. Such feature set includes: number of engines generating this candidate; variance of frequencies of the three engines; reciprocal rank of the candidate in each engine's ranking list (pre-ranked by the same ranking model in this work without consensus features); score of candidate in each engine's list; number of occurrences identified by two/three engines at the same time.

[1] We collect stopword list from `http://www.ranks.nl/stopwords`. Besides those stopwords, all words are viewed as content words.

Redundancy Features (7). This set measures redundancy for each candidate in the whole hypothesis space and includes: frequency of the candidate; number of different search snippets and passages containing the candidate; n-gram based redundancy score:

$$f(a_j, A) = \Sigma_i^T p(tk_i) \tag{3}$$

where tk_i is n-gram in a_j, T is the number of the n-grams. Score of an n-gram is estimated with:

$$p(tk_i) = \Sigma_{j=1}^N \frac{n_{a_j} * \delta'(tk_i, a_j)}{n_j} \tag{4}$$

where $\delta'(tk_i, a_j)$ is the indicator function equals to 1 if tk_i appears in candidate a_j and 0 otherwise, n_{a_j} is the frequency of a_j; N is the number of candidates, n_j is the number of n-grams in a_j. We compute the above scores of unigram and bigram, in addition with such scores normalized by candidate's length.

Similarity Features (41). This set measures similarity between candidate and question, including text similarity of candidate's context and semantic similarity between candidate and answer type. For text similarity, to avoid syntactic parsing we use term level similarities such as: Longest Common Sequence (LCS), sequential LCS, edit distance, number overlapped content words in sentences, and number of overlapped content words in contexts around candidate and question focus (phrase in question could be replaced by answer [14]) respectively. Those similarities are calculated both in passage level and sentence level (each search snippet contains about 3 sentences separated by "..."), and normalized by question length and sentence/passage length respectively. In addition, 9 similarity features are based on word embddding [15] such as average similarity between question phrases and search snippet phrases.

For semantic similarity, we measure whether the candidate matches with answer type or Lexical Answer Type (LAT) [14]. LAT is more specific concept than answer type, such as *Doctor, city*, etc. We build answer type dictionary from FreeBase [2] and adopt NeedleSeek [16] as LAT dictionary. We also build regular expressions to identify quality and date candidates.

Candidate Quality Features (9). This set measures the candidate's own quality, including: whether the candidate is capitalized; number of content words in the candidate and the value normalized by total token number of the candidate; number of candidate's tokens in question and the value normalized by total token number; length of the candidate.

Search Features (7). This set includes average, worse and best rank of snippets bearing the candidate given by search engine; the candidate is extracted from search snippets or titles; whether the candidate is extracted from Wikipedia site.

[2] `www.freebase.com`

3.3 Extraction Engines

In this work we employ three extraction engines:

Sequential Labeling Method (CRF). The first method is sequential labeling method similar with previous work [2,4,7]. Each token in passage is labeled as is or is not an answer token. Question-answer pairs are used to generate training data for this method. Given a question Q, answer A and search snippets S, sequential tokens in S are labeled with 1 if they match with A and 0 otherwise. We adopt 15 features similar to previous work [4] but without syntactic-based features, and employ CRF [17] as our labeling model. In order to increase recall of this method, we follow the setting of forced CRF [2], that for each passage, beside tokens with positive labels, we change top k (k is set to 2 based on results in development set) negative tokens' labels to positive (those tokens are ranked based on their scores of positive label). If continuous tokens' labels are 1, they are merged to form a single candidate answer.

Wikipedia Title-Based Method (Wiki). Following state-of-the-art system's method [9], we extract Wikipedia title entries appeared in search snippets. Specifically, we build a dictionary of 7.8 million entries consisting of all Wikipedia title entries. Given a search snippet, we scan the snippet with forward maximum matching and extract all the matched entries case-insensitively.

Noun Phrase-Based Method (NP). As we focus on factoid questions, Noun Phrases cover most of the correct answers. Noun phrases are extracted from search snippets, we adopt Stanford parser [3] to identify noun phrases.

4 Experiments

4.1 Experiment Setup

Our data is collected from TREC data. We build training and development set from TREC. There are two test sets in our work. The firs one (Test-1) is the one used in previous work [2,3], previous work have provided documents for answer extraction for this set. The second one (Test-2) is a larger test set, but there's no documents provided for this set. For all the training, development and testing sets, we collect search snippets for each question. The snippets are retrieved by five queries: the question; verb, noun, adj, adv words in question; Named Entities (if any) in question; noun phrase and verbs; verb and its dependents in question. For each query we collect top 20 snippets returned by search engine, and select 60 search snippets most similar to the question from them. Summary of the data is shown in Table 1. *Avg. Passage per Q* indicates in average for each question, there are how many passages available for answer extraction; *Rate of Positive Passages Per Q (%)* indicates among all the passages, the rate of positive passages that contains the correct answer.

Table 1. Answer extraction training and testing data used in this work

Data	Questions	Passage Type	Avg. Passage Per Q	Rate of Positive Passages Per Q (%)
Train	1200	Search	60	9.82
Test-1	75	Search	60	10.33
	89	Document	17	18.72
Test-2	293	Search	60	10.11

Table 2. Comparative results on testing set

Testset	Passage	Method	Top 1 Acc.	Top 5 Acc.	MRR
Test-1	Document	Tree Kernel	70.79	82.02	73.91
		Our Method	69.66	79.78	72.12
Test-1	Search	Tree Kernel	52.00	78.67	58.17
		Our Method	66.67	84.00	70.71
Test-2	Search	Tree Kernel	51.19	72.35	59.81
		N-gram	50.85	72.70	60.78
		Our Method	66.55	79.52	69.93

Our CRF-based answer extraction method and feature weights of ranking model is trained on our training set with search texts. Parameters of pruning are tuned with development set, $w_{crf} = 0.34, w_{wiki} = 0.33, w_{np} = 0.33$, $t_{crf} = 0.9$, $t_{wiki} = 0.75$, $t_{np} = 0.9$, $p = 0.9, m = 2$ and $n = 2$. We compare our method with state-of-the-art answer extraction method using Tree Kernel[3]. We re-implement the method and perform training and testing on our dataset. The author didn't mention which chunker or parser they employ, so we use Stanford parser to generate parsing trees and extract NP chunks from the trees.

We report Top 1 Accuracy (ratio of correctly answered questions with the first candidate in the ranking list), Top 5 Accuracy (ratio of correctly answered questions with the first 5 candidates) and MRR (multiplicative inverse of the rank of the first correct answer) metrics.

4.2 Experiment Results

Compare to State-of-the-art Method

The comparative result with Tree Kernel method is shown in Table 2[4]. In document set, Tree Kernel based method slightly outperforms ours. When adapting both methods from regular document to search text, they all degrade. In search set, Tree Kernel method's performance seriously drops from 71% accuracy to 52%, while in contrast, our method is less affected and outperforms Tree Kernel

[3] `http://www-nlp.stanford.edu/software/`

method on search texts. This indicates our method is more efficient for Web-based answer extraction.

For both methods, results on document sets are superior to the one on search sets, this is because search snippets contain more noises than documents. As it's shown in Table 1, rate of positive passages containing the correct answer in document set is 18.72%, while in contrast, such rate is much lower in search set. So both of the answer extraction methods have to deal with more negative candidates in the hypothesis space. Specifically, positive and negative candidates' rate of our method is 1: 27 in document set, while such rate is 1: 59 in search set before pruning. In addition, search snippets are often incomplete, and sometimes key words in question are in sub-sentence apart from the correct answer. We observe that about half of the positive sentences containing the correct answer don't contain any key words in question or aren't complete sentences. Under such condition, syntactic similarity between question and answer bearing sentences will be reduced. The result is that traditional method such as Tree Kernel method relies on such similarity will degrade on the ill-formed search snippets. Previous work [10] discuss to use n-gram for answer extraction, as a result, number of negative samples in the hypothesis space of that method will also be lager. As Table 2 shows, the n-gram based method also performs worse than our method.

Single Extraction Versus Multiple Extractions

Using multiple extraction engines brings great contributions. There are two positive effects of the multiple extraction engines-based method: 1) redundancies among different engines enables us to perform more restrict pruning which reduces noises in candidate list; 2) consensus information across different engines is useful for improving the ranking result.

Table 3 (from this section to save space we show results in Test-2 set as it contains more testing questions) shows pruning results on single and multiple extraction engines with the same pruning parameters. When combining all the engines' results, recall is increased. Pruning helps to reduce noises, but it also hurts recall. When independently prune each engine's candidate list, their recalls all degrade, and the combination's (labeled as Combine with Single Prune) recall also degrades seriously. As for our method (labeled as Combine with Joint Prune), when we consider the global appearances from different engines, the impact from pruning on answer recall is eased.

Table 4 shows ranking results of single engine. Single engine's ranker is trained on their own candidate list with the same model and features except consensus features. After combining results from different engines, accuracy of answer extraction is improved comparing to any single one. But if we remove consensus features during combination, performance of answer ranking drops a lot. Further, if we remove pruning based on redundancy among different engines, the

[4] We didn't compare our method with Watson's extraction method, as Watson employs lots of manual efforts and hundreds of features, which makes directly comparison impossible. It is also why previous work [2,3] don't make such comparison.

Table 3. Results of pruning on single and multiple extraction engines on Test-2. Recall is the binary recall of correct answer, N:P indicates ratio of negative cases' number to positive ones'.

	NP		Wiki		CRF		Combine (Single Prune)		Combine (Joint Prune)	
	Recall	N:P	Recall	N:P	Recall	N:P	Recall	N:P	Recall	N:P
No Pruning	93.52	23	85.32	70	86.69	15	94.54	59	94.54	59
Pruning	82.26	15	77.37	37	79.18	9	85.67	29	93.86	31

Table 4. Performance of ranking results on Test-2 set

Method	Top 1 Acc.	Top 5 Acc.	MRR
NP	53.24	73.72	62.04
Wiki	51.88	71.67	60.36
CRF	54.27	76.45	63.67
All	66.55	79.52	69.93
All -Consensus Features	56.31	75.43	63.97
All -Joint Pruning	54.61	77.82	64.39

result also becomes worse. In all, employing multiple extraction engines helps to increase the performance of answer extraction for Web-based QA.

Table 5 shows an example in testing set with top 5 candidates. Tree kernel and NP method tend to extract complete units from search texts, so they share some common candidates; CRF method tends to extract and give high rank to short candidates related to question, such as years in this example; Wikititle-based method extracts entities in snippets, such as years, titles in this case. They perform different analysis on the texts thus the results present with different trends. Although none of the engine correctly answer the question, after combining the three lists, our method outputs the correct answer.

Feature Ablation Test

We compare contributions from different feature sets by removing one feature set at a time and performing same training and testing. Results are shown in Table 6, removing features in ranking does not have impact on recall, so we only show accuracy measurements. Most of the features are similarity features, so that set has the most contribution. We only design shallow similarity features, if we employ more syntactic features, ranking performance can be further improved. Besides, consensus as illustrated before also has very important role. Follow is redundancy features, from the results we see that redundancy with only 7 features can greatly improve ranking results. Last, quality and search features, although with very few number of features, are also useful.

Table 5. Example of different extraction engines' results

Question	When did Jack Welch retire from GE?
Answer	2001
Method	Top 5 Candidates
Tree kernel	general electric; chairman and ceo; his younger wife; oct 05, 2012;2001
NP	general electric; jack welch 's; 2001; one; chairman and ceo
CRF	2012; 2002; 2001; one; 1999
Wiki	general electric; retirement; 2012; 2001; ceo
Ours	2001;general electric; 2012; retirement; ceo

Table 6. Testing results of ablating different feature sets on Test-2 set

Feature Set	Number of Features	Top 1 Acc.	Top 5 Acc.	MRR
All	73	66.55	79.52	69.93
-Similarity	73-41=32	54.27	74.06	62.05
-Consensus	73-9=64	56.31	75.43	63.97
-Redundancy	73-7=66	57.34	76.79	64.74
-Quality	73-9=64	63.83	77.47	68.46
-Search	73-7=66	64.51	77.82	68.51

Error Analysis

We randomly select 50 wrongly answered questions and analyze the errors. 5 questions' answers are missed in the hypothesis space, most of them are Quality or Time questions. 15 questions are correctly answered, but the labeled answer is different from our output. For example, for question *Where did the Battle of the Bulge take place?*, the given answer is *Luxembourg*, but our output *Ardennes* is also correct. In traditional QA, answers are set based on given corpus, so the expression of the correct answer is fixed; but in Web, the way to express the answer are various, strictly judge answer's correctness based on TREC data will underestimate performance of Web-based QA. Further, 30 questions' answers are wrongly ranked which suggests using more sophisticated ranking features.

5 Conclusion

Web-based Question Answering is to generate answer from search snippets returned by search engines. Search snippets contain many noises and many incomplete sentences, which lowers down performance of traditional methods of answer extraction. In this paper we discuss about using multiple extraction engines for Web-based QA. We adopt a Pruned Rank Aggregation method to prune noisy candidates with redundancies among different engines during rank aggregation. The resulted candidate list is ranked with a Learning to Rank method with similarity, redundancy, search and quality features. Our method improves performance of Web-based answer extraction, and outperforms state-of-the-art answer extraction method.

Acknowledgments. We'd like to thank Yajuan Duan for her contribution of question analysis components. We also want to thank all the reviewers for their valuable comments.

References

1. Brill, E., Lin, J., Banko, M., Dumais, S., Ng, A.: Data-intensive question answering. In: TREC, pp. 393–400 (2001)
2. Yao, X., Van Durme, B., Callison-Burch, C., Clark, P.: Answer extraction as sequence tagging with tree edit distance. In: HLT-NAACL, pp. 858–867 (2013)
3. Severyn, A., Moschitti, A.: Automatic feature engineering for answer selection and extraction. In: EMNLP, pp. 458–467 (2013)
4. Sun, H., Duan, N., Duan, Y., Zhou, M.: Answer extraction from passage graph for question answering. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, pp. 2169–2175. AAAI Press (2013)
5. Xu, J., Licuanan, A., May, J., Miller, S., Weischedel, R.: Answer selection and confidence estimation. In: 2003 AAAI Symposium on New Directions in QA (2003)
6. Ravichandran, D., Ittycheriah, A., Roukos, S.: Automatic derivation of surface text patterns for a maximum entropy based question answering system. In: Proceedings of HLT-NAACL (2003)
7. Sasaki, Y.: Question answering as question-biased term extraction: A new approach toward multilingual qa. In: Proceedings of ACL, pp. 215–222 (2005)
8. Bunescu, R., Huang, Y.: Towards a general model of answer typing: Question focus identification. In: Proceedings of the 11th International Conference on Intelligent Text Processing and Computational Linguistics, RCS Volume, pp. 231–242 (2010)
9. Chu-Carroll, J., Fan, J.: Leveraging wikipedia characteristics for search and candidate generation in question answering. In: Proceedings of AAAI (2011)
10. Lin, J.: An exploration of the principles underlying redundancy-based factoid question answering. ACM Transactions on Information Systems 25(2), 6 (2007)
11. Subbian, K., Melville, P.: Supervised rank aggregation for predicting influence in networks. arXiv preprint arXiv:1108.4801 (2011)
12. Agarwal, A., Raghavan, H., Subbian, K., Melville, P., Lawrence, R.D., Gondek, D.C., Fan, J.: Learning to rank for robust question answering. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, pp. 833–842. ACM (2012)
13. Joachims, T.: Training linear svms in linear time. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 217–226. ACM (2006)
14. Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A.A., Lally, A., Murdock, J.W., Nyberg, E., Prager, J., et al.: Building watson: An overview of the deepqa project. AI Magazine 31(3), 59–79 (2010)
15. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.P.: Natural language processing (almost) from scratch. Journal of Machine Learning Research 12, 2493–2537 (2011)
16. Shi, S., Liu, X., Wen, J.R.: Pattern-based semantic class discovery with multi-membership support. In: Proceedings of the 17th ACM Conference on Information and Knowledge Management, pp. 1453–1454. ACM (2008)
17. Lafferty, J., McCallum, A., Pereira, F.C.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data (2001)