LIMSI-CNRS@CLEF 2015: Tree Edit Beam Search for Multiple Choice Question Answering

Martin Gleize[1,2] and Brigitte Grau[1,3]

[1] LIMSI-CNRS, Rue John von Neumann, 91405 Orsay CEDEX, France
[2] Université Paris-Sud, Orsay
[3] ENSIIE, Evry
gleize@limsi.fr, bg@limsi.fr

Abstract. This paper describes our participation to the Entrance Exams Task of CLEF 2015's Question Answering Track. The goal is to answer multiple-choice questions on short texts. Our system first retrieves passages relevant to the question, through lexical expansion involving WordNet and word vectors. Then a tree edit model is used on graph representations of the passages and answer choices to extract edit sequences. Finally, features are computed from those edit sequences and used in various machine-learned models to take the final decision. We submitted several runs in the task, one of which yielding a c@1 of 0.36, which makes our team the second best on the task.

Keywords: Question Answering, Passage Retrieval, Textual Entailment

1 Introduction

The task focuses on the reading of single documents and identification of the correct answer to a question from a set of possible answer options. The identification of the correct answer requires various kinds of inference and the consideration of previously acquired background knowledge. Japanese University Entrance Exams include questions formulated at various levels of complexity and test a wide range of capabilities. The challenge of "Entrance Exams" aims at evaluating systems under the same conditions humans are evaluated to enter the University. Previously the evaluation campaign Question Answering For Machine Reading Evaluation (QA4MRE at CLEF) [8] focused on multiple-choice questions designed to evaluate computer systems, but this relatively new task takes on challenges typically offered to humans. It naturally translates into more complex inference phenomena to solve [1].

2 System Architecture

The overarching goal of our system is essentially to validate correct answers without invalidating them, and invalidate wrong answers without validating them. The architecture of our multiple-choice question-answering system is described

in Figure 1. Its pipeline is composed of mainly five modules: preprocessing, passage retrieval, graph enrichment, beam search with tree edit model and final classifiers for validation/invalidation. The remaining of this section is dedicated to the detailed description of those modules.

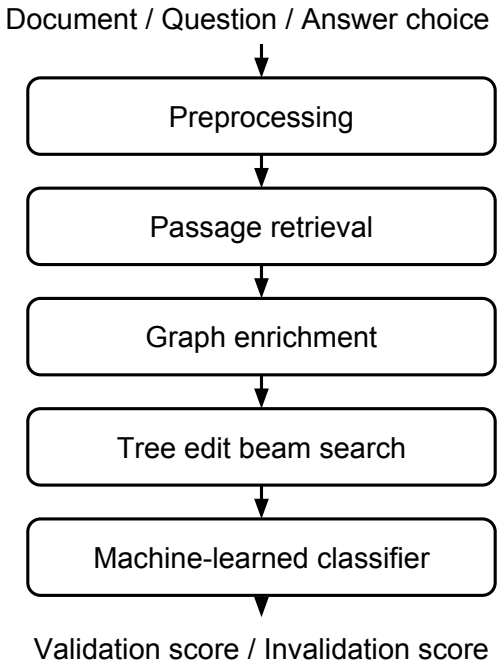

Fig. 1. System Architecture

2.1 Preprocessing

We use Stanford CoreNLP as the main Natural Language annotation tool. Each sentence from the document, questions or answer choices is tagged with Part-Of-Speech [9] and syntactically parsed [4]. In addition, a coreference resolution system [7] is applied on the whole document as well as question-answer pairs. We add to this coreference resolution process manual rules derived from the data, like replacing first person pronouns in non-dialogue context with "the author" or "the writer", depending on which is used in the questions. Named Entity Recognition was not used, due to not being very helpful in past editions of Entrance Exams. NER is very important to factoid QA, because it produces annotations which correspond roughly to the expected type of answer, but on complex multiple-choice questions which rarely use entities as an answer type,

it is intuitively less crucial.

2.2 Passage Retrieval

The passage retrieval module aims at extracting relevant short snippets from the document to pass to the more computationally expensive modules further down the pipeline. Words of the question and the answer choice act as the query. However, it is very rare that words of the question exactly appear in the relevant passage of the document, so we have to use some form of query expansion.
We enrich the lemmas with coreference information, WordNet relations (synonyms, antonyms, hypernyms, hyponyms), and weigh the words by the IDF score of the original word in the document.
If the words of the query are not found using the previous expansion methods, we use a vector-based representation of words to compute a similarity measure. Word vectors are those found in [3]. To each word, we assign a vector of 50 values. [3]'s resource actually provides multiple vectors for each word, to account more accurately for polysemy, so we use the same window-based disambiguation method as the author to compute the right one. We then pair the query word vectors with the document word vectors with the highest cosine similarity. We also take into account bigram vectors, by summing 2 vectors, which means that we can effectively handle 1-to-2, 2-to-1 and 2-to-2 scored alignments.
Passages are ranked according to the scoring function defined by Equation 1 and are then naturally extended to the full sequence of sentences they span.

$$score(passage) = \frac{\#matchedWords}{\#queryWords} \times \sum_{i=1}^{n-1} \frac{score(w_i) + score(w_{i+1})}{dist(i, i+1)^2} \quad (1)$$

We take into account the potential absence of query words by multiplying the passage score by the fraction of query words the passage contains. Each $w_i \in \{w_1, ..., w_n\}$ a document word matching a query word is given a simple alignment *score* (1 if they have same lemmas, 0.9 if they are WordNet synonyms, 0.8 if they are in another WordNet relation, and their word vector cosine similarity otherwise), weighted with the IDF of the word, and the formula is normalized by the square of the distance between the words in the sentence.
Overall, this passage retrieval method retrieves a lot of short passages, most of which will overlap or won't be correct, but the beam search which uses them is designed to handle numerous source passages.

2.3 Graph Enrichment

The passages were syntactically parsed with Stanford CoreNLP to obtain the initial dependency graphs. We fuse those graphs together by linking their roots with a *followed-by* arc which materializes in the single remaining graph that a sentence is followed by another in the passage. Then we use ConceptNet [5] to

enrich the graph.
ConceptNet is a semantic triplet base containing relations about common-knowledge of the world, designed to be used especially for machine understanding of text written by people. It is built from nodes representing words or short phrases of natural language, and labeled relationships between them (the nodes are called "concepts" for tradition, but they'd be better known as "terms".) For example, ConceptNet contains everyday basic knowledge, like *MotivatedByGoal(learn, knowledge)*: you would learn because you want knowledge. It also contains cultural knowledge, like *UsedFor(saxophone, jazz)*: a saxophone is used for jazz. Our assumption is that understanding the documents in the Entrance exams corpus requires a lot of human common-sense, easily acquired by human readers of that level, but difficult to grasp for computers. So we want to enrich the text with relations which attempt to fill that gap.
Concepts from ConceptNet are mainly single words, like "saxophone" or "jazz", so they are easy to link to our original graph. However, it is not easy to integrate *relations* to our graph, because they have labels that are potentially composed of several words, like UsedFor or MotivatedByGoal. We could split those labels into words and use those in the graph, but we preferred attaching to the original graph the parse tree of the *surfaceText* element of the relations. Surface texts are the original natural language text that expressed the statement, like "a saxophone is used for jazz". We attach the parse tree of these sentences to any concept whose head word is in the original graph. We only retrieve from ConceptNet relations that are indicative of an entailment relation of any kind, namely: IsA, PartOf, MemberOf, UsedFor, CapableOf, Causes, HasPrerequisite, MotivatedByGoal, Desires.

2.4 Tree edit beam search

Tree edit model Our goal is to characterize a sequence of transformations applied to the passage to obtain the answer choice. Those transformations affect the graph built in the previous section, which is made of parse trees, and the transformations will be called *edits*, hence a tree edit model. Basically, we apply different edits iteratively to the tree, modifying it each time, so that the edited tree is closer to the tree of the answer choice. When we find an *edit sequence* to turn the passage into the answer choice, we look at the nature of edits that were effectively applied, and if they are elements of proof that the passage is indeed close to the answer choice, or if it is too far to conclude anything. This will be done in the subsequent sections.
Table 1 presents the supported edit operations. Figure 2 presents an example of successive applications of three of them.

Beam search The main problem is that there are many choices to make when applying an edit. Which edit to choose? Where to apply changes in the tree? What new elements must be added? What to do next? Any of these choices is an easy source of error, so rather than picking one each time and hoping to find

Table 1. Edit operations on trees

Edit operation	Description
Delete(d: Tree)	Delete the node d and replace it with its children.
Insert(i: Word, p: Tree)	Insert the word i under its new parent p.
Rename(t: Tree, w: Word)	Replace the word attached to the node t with w.
Move(m: Tree, op: Tree, np: Tree)	Move the subtree m from under op to under np.

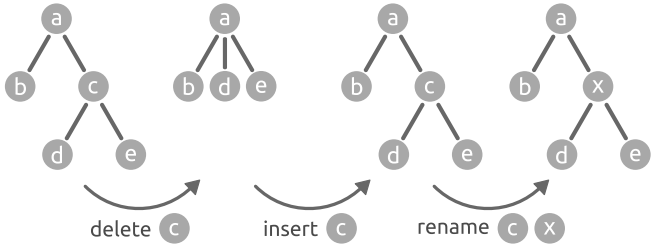

Fig. 2. Examples of successive edit operations

the right edit sequence, we apply a lot of the possible edits, and explore only the most promising using a satisfyingly good heuristic.
Beam search is an optimization of best-first search that reduces its memory requirements. It uses breadth-first search to build its search space. At each step of the search, we generate all possible edits of the trees at the current step, sorting them in increasing order of heuristic cost defined below. However, we only store a predetermined number of best trees at each level, called the *beam width*. Only those trees are edited next and the rest is discarded. This method allows to fine-tune via the beam width the probability to find useful edit sequences and the memory and time costs.

Partial Tree Kernel as a heuristic We need a heuristic measure of how far a tree at the current step is from the target tree (the dependency tree of an answer choice). We implement the Partial Tree Kernel defined in [6] to compute the similarity between the current tree and the target tree. As a tree kernel, it classically computes the number of common subtrees between 2 trees, but this particular version of the tree kernel is adapted to n-ary trees, which is what we have in dependency structures. The kernel computation is normalized with $\tilde{K}(x,y) = \frac{K(x,y)}{\sqrt{K(x,x)}\sqrt{K(y,y)}}$, for K the kernel and x, y the trees.

Algorithm At the start of the algorithm, the working set consists solely of the enriched dependency graphs of all the retrieved passages. The target tree is the answer choice (or the question plus the answer choice when the answer is an end to the question sentence as can be the case in the CLEF dataset). In our experiments, we keep at most 10 passages in the retrieval step. Then, every possible relevant edit operation is applied to each passage. Inserts and renames

can only add a node that is present in the target tree. Moves can only move a node under a parent so that the link parent → child is present in the target tree. Those edited trees are added to the working set, and the partial tree kernel with the target tree is computed for all of them. The working set is then filtered to only the top 50 trees with the best kernel score (50 is our beam width) and the algorithm can now start again with the application of the edit operations on the new working set.
It stops when 10 different edit sequences have been found (some filtering is done to ensure that we do not obtain mere variations of the first sequence found), or after 200 edit steps, whichever comes first.

2.5 Feature extraction

The goal is to classify an edit sequence with two different machine-learned classifiers, one to decide if the related answer choice is validated, and one to decide if it is invalidated. The design of features is thus primordial. In practice, we will use the same features and the same machine learning algorithm for the two classifiers, so the only difference will be the training data, discussed in the next section.
Most features are counts of specific edit unigrams or bigrams in the edit sequence, and are summarized in Table 2. Pre-processing informations that were not used in the beam search are used at this point, like dependency relations in the parse tree, coreferences, and whether what we edit was part of the ConceptNet additions or can be linked in WordNet.

3 Experiments and results

3.1 CLEF 2015 QA Track: Entrance Exams data and evaluation

Our data consist of the trial and test sets at CLEF 2015 Question Answering Track, Task 2: Entrance Exams. The trial data is composed of the test sets at CLEF 2013 and 2014, each containing a series of 12 texts, and for each of them, 4 to 6 multiple-choice questions to answer, for about 120 questions in total. In the 2015 test set, there are 19 documents, and a total of 89 questions. There are 4 answer choices possible for each of the questions. This corpus has been extracted from the Tokyo University Entrance Exam in English as a foreign language.
Systems are evaluated according to their c@1, defined in equation 2.

$$C@1 = \frac{1}{n}(n_R + n_U \frac{n_R}{n}) \tag{2}$$

with n the total number of questions, n_R the number of correctly answered questions, n_U the number of unanswered questions.

Table 2. Features of an edit sequence

Feature	Description
`editTotal`	Total number of edits in the sequence

`deleteTotal`
`deleteVerb`
`deleteNoun`
`deleteProperNoun`
`deleteSubject`
`deleteObject`
`deleteRoot`
`deleteNegation`
`deleteConceptNet`
	Number of total delete edits, edits which delete a verb, a noun, a proper noun, a subject (indicated by the `subj` Stanford dependencies), an object, the root of the tree, a negation (indicated by the `neg` dependency), and something added to the graph through ConceptNet

`insertTotal`
`insertVerb`
`insertNoun`
`insertProperNoun`
`insertNegation`
	Analogous to the above, for insert edits

`renameTotal`
`renameVerb`
...
`renameSyn`
`renameAnt`
`renameHypHyp`
`renameStrongWordVectorSim`
`renameCoref`
`renameNonCoref`
	Analogous to the above, for rename edits + edits which rename a word into its synonym in WordNet, or into its antonym in WordNet, or into a hypernym/hyponym in Wordnet, edits which rename a word into another with strong word vector similarity (above a threshold, defined empirically), edits which rename a pronoun into its referent according to the Stanford coreference resolution, and edits which rename a pronoun into some other referent

`moveTotal`
`moveVerb`
...
`moveConceptNet`
`moveMoreThan2Nodes`
	Analogous to the above, for move edits + edits which move more than 2 nodes

All bigram combinations of the above	Number of pairs of the successive given edits in the sequence
`dependencyEditSequence`	Number of pairs of successive edits applied to 2 nodes in a dependency relation

`originalTotal`
`originalVerb`
...
	Fraction of the original words, verbs, nouns, proper nouns, that was **not** edited in the sequence

3.2 Learning classifiers

The classifier pair, for validation and invalidation, uses the feature set defined in the previous section. We experimented with two models, logistic regression and random forest, both implemented in Weka [2], and results are presented in the next subsection. We focus here on how we built our training data.
What we want to avoid is trying to learn how to transform any random text snippet in the document into any random answer choice, because it serves no purpose. Indeed, as readers, we cannot validate the right answer choice by looking at a couple of arbitrary sentences in the text, nor can we invalidate a wrong answer choice if the passage we are reading is not even related to the question. Thus, we annotate the relevant passages in the training data manually, and our algorithm runs on them, without a passage retrieval phase. A *relevant passage* is roughly the sufficient text snippet which expresses both the question and the elements of the answer choice. Of course, sometimes the answer choice is not exactly expressed by the passage, as commonly happens for wrong answer choices, and sometimes, albeit rarely, the answer choice is not even expressed at all in the document. Two answer choices to the same question can share a relevant passage, as we annotate complete sentences.
We create the learning (passage, answer choice) pairs by annotating them following the semantics described in Figure 3. In this figure, RP stands for right passage, RA for right answer, WA for any wrong answer, WAx for the wrong answer choice x, WPx for the passage expressing it, OP for any other passage than the one expressing the paired answer choice. To summarize, the only time we can either validate or invalidate are when we operate on passages relevant to some answer choice: we annotate as validated only if we have both the right passage and the right answer, and invalidated if we have a wrong answer choice with either the passage which expresses it in the document or the right passage. This follows the intuition that as readers, given a question, a passage and an answer choice, we can probably tell if the provided passage is self-sufficient in expressing the right answer to the question or if there is a mismatch between an answer choice and the passage in the text it refers to.
Then the edit sequences for this data are computed, their features are extracted, and sequences for both classifiers are labeled using the aforementioned semantics. Implicitly, as this is not visible in Figure 3, if an edit sequence is labeled 1 (valide/invalidate) for one classifier, it is labeled 0 for the other. The thin dashed arrows simply symbolize that the label is 0 for both classifiers.

For the test run, the algorithm runs on the test data, and the answer is chosen based on the regression numbers output by the two classifiers. First, for each answer choice, the edit sequence with the highest max(`validationScore`, `invalidationScore`) is selected. Ideally we want an edit sequence which is characteristic of either a high confidence validation, or a high confidence invalidation, so that we may classify the answer choice confidently as either correct or incorrect in the next step. Then, the answer choice whose selected sequence has the highest `validationScore` − `invalidationScore` is finally picked: we want in

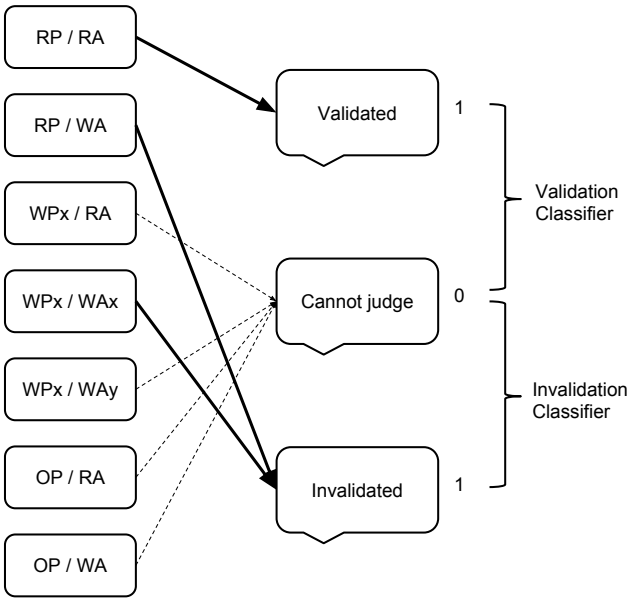

Fig. 3. Semantics of (passage, answer) pairs

correctly classified answer choices as much separation between their validation and invalidation scores as possible. If they have a similar validation and invalidation score, the system just ends up guessing. We acknowledge that this is a very basic decision process and address this point more in detail in Section 3.5.

3.3 Results

In this section, we report the results of our two learning models on the testing set of questions. Table 3 presents those results. A c@1 of 0.36 gave us the second place among teams which participated in the task.

Table 3. Results on test

	Random forest	Logistic regression
Questions answered	89	89
Errors	57	61
Accuracy	0.360	0.314
# of tests with c@1 ≥ 0.5	8	4
c@1	0.360	0.314

The random forest model performed better on the test run, which confirms what we expected during development.

3.4 Error analysis

Qualitative analysis A pertinent qualitative analysis is always delicate to do for machine learning systems with such low performances. It is indeed always possible to draw examples that look like the system is obviously supposed to correctly handle but end up as errors. Conversely, it is always possible to find a complex instance on which the system somewhat miraculously worked (i.e. made a lucky guess).
Nevertheless, we first report some of the simple errors that our system made. In the following passage/question pair, our system got lured by answer 3, closest in surface form to the relevant passage. ConceptNet does not link "held" to "trapped", and "its original nature" from the correct answer could not be linked to anything in the passage (it is however found further in the text).

Several years ago, certain scientists developed a way of investigating the nature of the atmosphere of the past by studying air caught in the ice around the North or South Pole. According to their theory, when snow falls, air is trapped between the snowflakes. The snow turns to ice with the air still inside.
Certain scientists claimed that 1) atmospheric gases increase the yearly amount of snow 2) falling snowflakes change the chemical balance of the air 3) the action of atmospheric gases causes snow to turn into ice **4)** the air held between snowflakes keeps its original nature (correct)

In the following passage/question pair, our system picked the answer choice 3. It would have been easy to pick the correct answer 1 if "wrong" could have been linked to "mistake", but in ConceptNet, this is a *RelatedTo* relation, which we did not consider. We realize that there is actually a lot of information in those *RelatedTo* relations, and ideally our system should handle them, but we decided in the design phase to remove them because they are not semantically precise.

Everyone stared. That was embarrassing enough, but it was worse when I finished my coffee and got ready to leave. My face went red - as red as his hair - when I realized I'd made a mistake.
The woman's face turned red **1)** because she realized that she had been quite wrong about the boy (correct) 2) because she realized that the boy was poor and hungry 3) because she saw everyone staring at her 4) because she hated being shouted at

In both those cases, a more precise characterization of correct passages would have been useful, because in the first case, our answer choice skips over the sentence which contains the correct answer, and in the second case, the sentence containing our answer choice appears way before the sentence containing both question and correct answer.
Finally we report an example of correctly answered question through mostly

invalidation. In the following passage/question pair, our system frankly invalidated answer choice 1 (due to the added negation) and answer choice 4 (due to the first sentence of the passage saying the opposite). Then, answer choice 2 had edit sequences which hinted at both validation and invalidation, so it was still a risky pick (but with slightly more invalidation). In the end, the remaining answer choice (3), for which the system found neither validation nor invalidation, was correctly picked by default.

Kate was an energetic woman who expected people always to be doing something, and she found plenty of jobs for Fred to do. This made him feel part of the household, but now he really wanted to be able to sit and reflect on the events of his life. If he had continued to live alone, he would have had the time to do this to his heart's content. One afternoon he felt he simply had to get away from the house. "I'm going for a walk," he said, closing the door behind him. Leaving the town, he walked across the fields and followed a slow-moving stream toward the hills. After a while he came to a pool in the stream under some trees. Here, he thought, was a place he could come to when he needed to reflect on the past. Although the stream seemed unlikely to have any fish, he would simply tell Kate he had found a place to go fishing. When he mentioned the stream that night, his son-in-law, Jim, said in disbelief, "There aren't any fish there. That stream runs dry half the summer."
Why did Fred tell Kate that he had found a place to go fishing? 1) He didn't feel part of the household with Kate and Jim. 2) He enjoyed fishing very much and was glad to be able to do it again. **3)** He wanted a way to leave the house without hurting Kate's feelings. 4) He was bored in the house because there were few things to do.

Quantitative analysis The general trend was that our system performed better when edit sequences remained short, with over 40% accuracy when the chosen edit sequences are shorter than 6 edits (on average on all the answer choices). We considered this was still not significant enough of an advantage to choose not to answer questions based on a length threshold of edit sequences.

3.5 Future work

We did not take advantage of the possibility to choose not to answer a question. In our experience, every missed answer adds variance when running on the test set (we are evaluated on even fewer questions, when there are not many to begin with), so we did not prioritize exploiting this feature of the evaluation. However, we believe our learning method has the potential to handle it. In future works, it would be interesting to design a meta-classifier working on the output of the two current classifiers.

4 Conclusion

Our system has been developed to answer multiple-choice questions. We extract features from edit sequences obtained from our tree edit beam search method, and learn two classifiers for validation and invalidation of answer choices. In the CLEF 2015 evaluation campaign, Question Answering track, Entrance Exams task, our best submitted run obtained the second performance among teams.
In further works, we plan to improve our graph enrichment method, which seems to be a promising avenue. We are considering adding paraphrases to the graph. Moreover, we plan to develop a meta-classifier dealing with the final decision, based on the individual validation/invalidation scores per answer choice, instead of relying on manually crafted rules.

References

1. Gleize, M., Grau, B.: A hierarchical taxonomy for classifying hardness of inference tasks. In: Chair), N.C.C., Choukri, K., Declerck, T., Loftsson, H., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., Piperidis, S. (eds.) Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14). European Language Resources Association (ELRA), Reykjavik, Iceland (may 2014)
2. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. ACM SIGKDD explorations newsletter 11(1), 10–18 (2009)
3. Huang, E.H., Socher, R., Manning, C.D., Ng, A.Y.: Improving word representations via global context and multiple word prototypes. In: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1. pp. 873–882. Association for Computational Linguistics (2012)
4. Klein, D., Manning, C.D.: Accurate unlexicalized parsing. In: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1. pp. 423–430. Association for Computational Linguistics (2003)
5. Liu, H., Singh, P.: Conceptneta practical commonsense reasoning tool-kit. BT technology journal 22(4), 211–226 (2004)
6. Moschitti, A.: Efficient convolution kernels for dependency and constituent syntactic trees. In: Machine Learning: ECML 2006, pp. 318–329. Springer (2006)
7. Recasens, M., de Marneffe, M.C., Potts, C.: The life and death of discourse entities: Identifying singleton mentions. In: Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 627–633 (2013)
8. Sutcliffe, R., Peñas, A., Hovy, E., Forner, P., Rodrigo, Á., Forascu, C., Benajiba, Y., Osenova, P.: Overview of qa4mre main task at clef 2013. Working Notes, CLEF (2013)
9. Toutanova, K., Klein, D., Manning, C.D., Singer, Y.: Feature-rich part-of-speech tagging with a cyclic dependency network. In: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1. pp. 173–180. Association for Computational Linguistics (2003)