Language Processing and Learning Models for Community Question Answering in Arabic

Salvatore Romeo\textsuperscript{a}, Giovanni Da San Martino\textsuperscript{a}, Yonatan Belinkov\textsuperscript{b}, Alberto Barrón-Cedeño\textsuperscript{a}, Mohamed Eldesouki\textsuperscript{a}, Kareem Darwish\textsuperscript{a}, Hamdy Mubarak\textsuperscript{a}, James Glass\textsuperscript{b}, Alessandro Moschitti\textsuperscript{a}

\textsuperscript{a}Qatar Computing Research Institute, HBKU, Doha, Qatar
\textit{email: \{sromeo, gmartino, albarron, mohamohamed, kdarwish, hmubarak, amoschitti\}@qf.org.qa}

\textsuperscript{b}MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA
\textit{email: \{belinkov, glass\}@mit.edu}

\textbf{Abstract}

In this paper we focus on the problem of question ranking in community question answering (cQA) forums in Arabic. We address the task with machine learning algorithms using advanced Arabic text representations. The latter are obtained by applying tree kernels to constituency parse trees combined with textual similarities, including word embeddings. Our two main contributions are: (i) an Arabic language processing pipeline based on UIMA—from segmentation to constituency parsing—built on top of Farasa, a state-of-the-art Arabic language processing toolkit; and (ii) the application of long short-term memory neural networks to identify the best text fragments in questions to be used in our tree-kernel-based ranker. Our thorough experimentation on a recently released cQA dataset shows that the Arabic linguistic processing provided by Farasa produces strong results and that neural networks combined with tree kernels further boost the performance in terms of both efficiency and accuracy. Our approach also enables an implicit comparison between different processing pipelines as our tests on Farasa and Stanford parsers demonstrate.

\textit{Keywords:} community question answering, constituency parsing in Arabic, tree-kernel-based ranking, long short-term memory neural networks, attention models.
1. Introduction

Community-driven question answering (cQA) on the web typically refers to popular forums in which users ask and answer questions on diverse topics. The freedom to post practically any question and answer in virtual anonymity promotes massive participation. The large amount of posts resulting from this environment demands the implementation of automatic models to filter relevant from irrelevant contents. This scenario has received attention from researchers in both the natural language processing and the information retrieval areas. However, for several reasons, languages other than English—including Arabic—have received relatively less attention.

In this research, we focus on the problem of improving the retrieval of questions from an Arabic forum with respect to a new user question. Our task is formally defined as follows. Let \( q \) be a new user question and \( D \) the set of question–answer pairs, previously posted in a forum. Rank all \( \rho \in D \) according to their relevance against \( q \). The main purpose of the ranking model is to improve the user’s experience by (i) performing a live search on the previously-posted questions, potentially fulfilling the user’s information need at once and (ii) avoiding the posting of similar questions, particularly if they have already been answered. From the natural language processing point of view this can also be the source of a collection of question paraphrases and near-duplicates, which can be further explored for other tasks.

Our model for question ranking uses Support Vector Machines. We use a combination of tree kernels (TKs) applied to syntactic parse trees, and linear kernels applied to features constituted by different textual similarity metrics computed between \( q \) and \( \rho \). We build the trees with the constituency parser of Farasa—which we introduce in this paper for the first time—and compare it against the well-consolidated Stanford parser [1]. Additionally, we integrated Farasa in a UIMA-based cQA pipeline\(^1\) which provides powerful machine learning features for question similarity assessment and reranking. Furthermore, we design word embeddings to complement the feature vectors.

In contrast to other question-answering (QA) tasks, forum questions tend to be ill-formed multi-sentence short texts with courtesy fragments, context, and elaborations. As TKs are sensitive to long (irrelevant) texts, we focus on the automatic selection of meaningful text fragments to feed TKs. To

\[^1\text{It should be noted that our UIMA pipeline with Farasa will be made available to the research community.}\]
do so, we design a selection model based on the weights assigned to each word in the texts by an attention mechanism in a long short-term memory network (LSTM). Such a model can filter out irrelevant or noisy subtrees from the question syntactic trees, significantly improving both the speed and the accuracy of the TKs-based classifier.

The rest of the paper is organized as follows. Section 2 offers the necessary background on general QA and cQA, both in Arabic and in other languages. In Section 3 we take a brief diversion from QA to describe Farasa, the technology we use for Arabic natural language processing. We turn back to QA in Section 4, where we present our question ranking model. Section 5 describes our neural network model designed to improve our tree representation by selecting the most relevant text fragments. Section 6 discusses our experiments and obtained results. Section 7 concluded with final remarks.

2. Background

As models for QA require linguistic resources, work focused on the Arabic language is relatively humble compared to other better-resourced languages, such as English [2]. Obviously, the scarceness of language resources is not the only issue. In Arabic, characteristics such as a rich morphology, the interaction among multiple dialects, and the common lack of diacritics and capitalization in informal language, pose unprecedented challenges for a QA system to succeed [3]. cQA is one specific scenario of QA. Most of the research work carried out for the Arabic language is focused on standard QA: the search for an answer over a collection of free-text documents. Therefore, this section is divided in three parts. Firstly, we overview some of the literature on Arabic QA. Secondly, we describe the three main stages of a cQA system, including a review of the approaches available to tackle each task, mainly for English. Thirdly, we overview the relatively-scarce literature on cQA for Arabic.

2.1. Question Answering in Arabic

Here we overview some of the most representative models proposed to address the three components of a QA system in Arabic: question analysis, passage retrieval, and answer extraction.

In question analysis, the task consists of generating the best possible representation for a question $q$ in order to retrieve a subset of relevant documents and, eventually, passages. The question pre-processing applied by Rosso et al.
[4] consists of stopword removal and named entity recognition. Afterwards, they classify \( q \) by means of its intended information need —whether \( q \) is asking for a name, a date, a quantity, or a definition— in order to look for the required information in the retrieved passages. Other approaches also try to extract the question’s focus (i.e., the main noun phrase) as well as named entities [5, 6, 7].

The resulting representation of \( q \) is used for retrieving text passages, \( p \), that might answer the question. One alternative is retrieving those \( p \) that include a certain amount of the words or phrases in \( q \). Besides computing a similarity function \( \text{sim}(q, p) \) [7], the ranking function can be based on the positional distance among the matching terms in the document [8, 9], i.e., the closer the terms in the document, the more likely it may represent a good answer for \( q \). A semantic expansion on the basis of resources such as the Arabic WordNet can come into play as well [9].

Once the most promising text passages have been retrieved, it is time to extract specific answers. Most approaches rely on manually-defined patterns, heuristics, rules, and semantic similarities between question focus and candidate answers; for instance, using \( n \)-grams [6, 10].

By addressing these three generic steps, different kinds of questions can be answered. For instance, Al Chalabi [11] focused on factoid QA by first determining if \( q \) is of kind who, what, when, how, etc. QASAL (Question-Answering System for Arabic Language) [5] goes beyond factoid QA by exploiting the linguistic annotation system of NooJ [12] to deal with definitional questions as well. Salem et al. [13] focused on why and how questions by means of the Rhetorical Discourse Structure (RST) formalism.

2.2. The Architecture of a Community Question Answering System

The cQA scenario is slightly different: a new question \( q \) formulated by the forum user tends to be less factual and more elaborated, often including contextual information, elaborations, multiple questions, and even irrelevant text fragments. The reference collection \( D \) is not composed of free-text documents, but of previously-posted forum questions, together with their answers provided by other users (if any). This leads to building a system architecture as the one represented in Figure 1, which is inspired by Potthast et al. [14].

The first step in the cQA architecture is that of heuristic retrieval. Given question \( q \) and a relatively-large collection of forum question–answer pairs \( \langle \rho, \alpha \rangle \in D \), an inexpensive mechanism is applied to retrieve the most similar (related) questions \( \rho \). Standard information retrieval technology (e.g., a
search engine based on inverted indexes), can be applied to solve this task.

The creators of the corpus [15] we use for our experiments (Section 6) used
Solr² to deal with this stage. This step results in the subset of potentially-
relevant candidate pairs $D_q \subset D$.

Having $q$ and $D_q$ as input, the knowledge-based re-ranking stage is in
charge of performing a more refined ordering of the questions. The objective
is locating those pairs $\langle \rho, \alpha \rangle \in D$ such that $\rho$ are semantically-equivalent
(or at least highly relevant) to $q$. The relatively-small size of $D_q$ allows for
the use of more sophisticated —generally more expensive— technology. This
is the task we address in this research work, by applying a combination of
kernels on both structural and deep learning features (cf. Section 4).

Extensive work has been carried out to design models for this crucial stage
of cQA. Although most of them have been devised for English forums, it is
worth mentioning some of the approaches. Cao et al. [16] tackled this problem
by judging topic similarity, whereas Duan et al. [17] searched for equivalent
questions by considering the question’s focus as well. Zhou et al. [18] dodged
the lexical gap³ between $q$ and $\rho$ by assessing their similarity on the basis of a
(monolingual) phrase-based translation model [19], built on question–answer
pairs in a similar fashion to Jeon et al. [20]. Wang et al. [21] computed the

\footnote{https://lucene.apache.org/solr}

\footnote{The classical IR problem of matching the few query terms in relevant documents.}
similarity between $q$ and $\rho$ on top of syntactic-tree representations: the more substructures the trees have in common, the more similar the questions are. The recent boom in neural network approaches has also impacted question re-ranking. dos Santos et al. [22] applied convolutional neural networks to retrieve semantically-equivalent questions’ subjects. They had to aggregate a bag-of-words neural network when dealing with whole questions; that is, subject and (generally long) body. Support vector machines have shown to be highly competitive in this task. For instance, Franco-Salvador et al. [23] used SVMrank [24] on a manifold of features, including distributed representations and semantic information sources, such as BabelNet [25] and Framenet [26]. Both Barrón-Cedeño et al. [27] and Filice et al. [28] achieved a good performance using KeLP [29] to combine various kernels with different vectorial and structural features.

Once the most promising questions $\rho$ in the forum are retrieved, potential answers to the new query $q$ are selected. The answers $\alpha$ attached to $\rho$ are compared against $q$ in order to estimate their relevance. This is not a trivial problem because the anarchy of Web forums allows users to post irrelevant contents. One of the first approaches to answer selection relied completely on the website’s metadata [30], such as an author’s reputation and click counts. Agichtein et al. [31] explored a graph-based model of contributors relationships together with both content- and usage-based features. These approaches depend heavily on the forum’s meta-data and social features. Still, as Surdeanu et al. [32] stress, relying on these kinds of data causes the model portability to be difficult; a drawback that disappears when focusing on the content of the questions and answers only. Tran et al. [33] applied machine translation in a similar fashion as Jeon et al. [20] and Zhou et al. [18], together with topic models, embeddings, and similarities. Hou et al. [34] and Nicosia et al. [35] applied supervised models with lexical, syntactic and meta-data features. Some of the most recent proposals aim at classifying whole threads of answers [36, 37] rather than each answer in isolation.

This cQA architecture assumes $q$ is a newly-posted question. A hybrid scenario is that of question deduplication. In this case, $q$ is just another question in the forum, together with its corresponding thread of answers. As a result, the information of both the question and its thread of comments can be used to determine if two posts are asking the same or similar questions. Both Ji et al. [38] and Zhang et al. [39] used LDA topic modeling to learn the latent semantic topics that generate question–answer pairs and used the learned topic distribution to retrieve similar historical questions.
It is worth noting that many of the aforementioned approaches [23, 27, 28, 33, 34, 35] were applied during the two editions of SemEval Task 3 on cQA [40, 15]. In this work we take advantage of the evaluation framework developed for Arabic in the 2016 edition [15] (cf. Section 6.1).

2.3. Community Question Answering for Arabic

As the reader can observe, most of the work on cQA has been carried out for other languages than Arabic, including LiveQA [41], which allowed participants to provide answers to real user questions, live on the Yahoo! Answers site. To the best of our knowledge, the first effort to come out with a standard framework for the evaluation of cQA models for Arabic is precisely that of [40, 15].

This resource promoted the design of five models for question re-ranking in Arabic. The most successful approach [42] included text similarities at both word and sentence level on the basis of word embeddings. Such similarities were computed both between $q$ and $\rho$, new and retrieved question, respectively, and between $q$ and $\alpha$, with $\alpha$ being the answer linked to the forum question $\rho$ after performing term selection as a pre-processing step.

Barrón-Cedeño et al. [27] used tree kernels applied to syntactic trees together with some features in common with [42]. A combination of rule-based, text similarities, and word embeddings has shown to give some benefit in Arabic cQA [43]. Our cQA system reuses ideas and some of the models we developed in [27, 42].

Magooda et al. [44] applied language models enriched with medical terms extracted from the Arabic Wikipedia. Finally, Malhas et al. [45] exploited embeddings in different ways, including the computation of average word vectors and covariance matrices. The performance of these models is included in Table 7, as they represent the state-of-the-art in the testbed we use for our experiments.

3. The Farasa Arabic NLP Toolkit

For our Arabic processing, we used our in-house pipeline of Arabic tools called Farasa 4 —insight or chivalry in Arabic. The pipeline includes a segmenter, a POS tagger, a named entity recognizer, a dependency parser, a

---

4Available at http://farasa.qcri.org
constituency parser, and a diacritizer. The syntactic parser is a new contribution, introduced in this paper for the first time. Farasa is tuned for the news domain and for Modern Standard Arabic (MSA). Still, Farasa can handle other genres along with classical and dialectal Arabic, but at reduced accuracy. This is possible because of the large overlap between MSA and other varieties of Arabic. Farasa fills an important gap in the span of available tools. It is the only comprehensive suite of Arabic tools that is both open source and whose internal subcomponents are competitive with the state of the art. Here we focus on the relevant components for our current task: segmenter, POS tagger, and constituency parser. We pose both segmentation and POS tagging as ranking problems, using kernel-based machines. We pose constituency parsing as a sequence labeling problem, where we use a CRF labeler that uses features from the segmenter and POS tagger. Both SVM and CRF have the advantage of being robust and computationally efficient.

3.1. UIMA Architecture for Arabic Natural Language Processing

Our Arabic natural language processing pipeline is based on UIMA.\(^5\) UIMA is a framework that allows for the integration of systems to analyze unstructured information (e.g., text documents) whose aim is to extract new knowledge relevant to the particular application context.

UIMA enables to compose applications with self-contained components. Each UIMA component implements an interface defined by the framework and both the input and output structures are described by means of XML descriptor files. The framework is in charge of managing these components, connecting the analysis engines and controlling the data flow. An analysis engine (AE) is a software module that analyzes artifacts (e.g., text) and infers information from them. The analysis engines are built starting from building units called annotators. An annotator is a component that analyzes artifacts and produces additional data and/or metadata (e.g., annotation on the analyzed artifact). An AE can contain a single annotator (primitive AE) or multiple annotators (aggregate AE).

Figure 2 shows the architecture of our pipeline, composed of four AEs. The modularity and flexibility of UIMA allows us for opting for different software modules to perform each of the tasks painlessly. The first AE uses OpenNLP\(^6\) for sentence splitting, besides performing tokenization. We

\(^5\)https://uima.apache.org
\(^6\)https://opennlp.apache.org
Figure 2: Our UIMA-based Arabic natural language processing architecture. Each block represents an analysis engine and includes the (alternative) technology it encompasses.

trained the sentence splitting model on 5k sentences from the AQMAR Arabic Wikipedia Supersense corpus [46] and NIST’s MT06 corpus. For the rest of the AEs, we can opt for using either Farasa’s or Stanford’s [1] technology. They are in charge of segmentation into clitics, Part of Speech (POS) tagging, and parsing. In Section 6, we will show the impact of using Farasa or Stanford to process the texts, by comparing different question rankers, each using one of the two parsing systems.

In the following subsections we describe the Farasa segmenter, POS tagger, and parser.

3.2. Farasa Segmenter

The Farasa segmenter is described in detail in [47, 48]. The segmenter breaks words into their underlying clitics. For example, the word \texttt{wktAbhm} (and their book) is segmented into \texttt{w+ktAb+hm}. We pose segmentation as a ranking problem, where the ranker attempts to rank possible segmentations of a word. The segmenter uses SVM\textsuperscript{rank} [49] with a linear kernel to determine the best segmentation for each word. We used a linear kernel with a trade-off factor between training errors and margin equal to 100 (parameters tuned on offline experiments carried out over a development set). The ranker uses a dense vector of features which is able to generalize well beyond the cases that are observed during training. Additionally, decoding using SVM\textsuperscript{Rank} is computationally efficient as it involves simple vector multiplication, where speed is highly desirable in processing large amounts of data. We also experimented with using CRF-based sequence labeling [50], and our SVM\textsuperscript{Rank} approach yields better segmentation results with higher speed. Further, we

\footnote{https://www.nist.gov/programs-projects/machine-translation}
conducted offline experiments to compare our approach to bidirectional Long Short Term Memory (bi-LSTM) over CRF and the results were comparable. It was trained on parts 1 (v. 4.1), 2 (v. 3.1), and 3 (v. 2) of the Penn Arabic Treebank (ATB) [51]. Instead of testing the segmenter on a subset of ATB (which may lead to artificially-high results due to its limited lexical diversity), we tested our segmenter on a corpus of seventy WikiNews articles from 2013 and 2014 [48]. It contains 18,300 manually-segmented and POS tagged words from articles on seven domains: politics, economics, health, science and technology, sports, arts, and culture.\footnote{The corpus is available at \url{https://github.com/kdarwish/Farasa}.}

Table 1 reports on the segmentation accuracy of Farasa and compares it to that of Madamira [52] —a popular state-of-the-art system— on the WikiNews corpus. The performance of the Farasa segmenter is competitive.

### 3.3. Farasa Part-of-Speech Tagger

Our Arabic part-of-speech tagger uses the simplified PATB tag set proposed by [50]. Table 2 includes the tags. The POS tagger attempts to find the optimal tag for each clitic produced by the segmenter, as well as determining the gender (masculine or feminine) and number for nouns and adjectives (singular, dual, or plural). Like the segmenter, the POS tagger uses SVM\textsuperscript{Rank} to find the best tag for each clitic. We decided to adopt SVM\textsuperscript{Rank} for POS tagging for the reasons mentioned earlier for segmentation. Additionally, our SVM\textsuperscript{Rank} outperforms a CRF sequence labeling model [50] and is on par with using a bi-LSTM model [53]. Thus we construct a feature vector for each possible POS tag for each clitic. We supply these vectors to SVM\textsuperscript{Rank} indicating which vector should be ranked the highest given the weights. We then used SVM\textsuperscript{Rank} [49] to learn feature weights. As for the segmenter, we used a linear kernel with a trade-off factor between training errors and margin equal to 100 (parameters tuned on offline experiments carried out over a development set). All possible POS tags for a clitic are scored using the classifier, and the POS with the highest score is picked.

Given a sentence composed of the clitics $c_{-n} \ldots c_0 \ldots c_m$, where $c_0$ is the current clitic and its proposed POS tag, we train the classifier using the following features, computed by maximum-likelihood estimation on our training corpus:

- $p(\text{POS} \mid c_0)$ and $p(c_0 \mid \text{POS})$. 
<table>
<thead>
<tr>
<th>Task—System</th>
<th>Farasa</th>
<th>Madamira</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segmentation</td>
<td>98.9%</td>
<td>98.8%</td>
</tr>
<tr>
<td>POS tagging</td>
<td>94.9%</td>
<td>95.3%</td>
</tr>
</tbody>
</table>

Table 1: Accuracy of segmentation and POS tagging for Farasa and Madamira.

- \( p(\text{POS} \mid c_{-i} \ldots c_{-1}) \) and \( p(\text{POS} \mid c_{1} \ldots c_{j}) \mid i, j \in [1, 4] \).
- \( p(\text{POS} \mid c_{-i}\text{POS} \ldots c_{-1}\text{POS}) \) and \( p(\text{POS} \mid c_{1}\text{POS} \ldots c_{j}\text{POS}); i, j \in [1, 4] \). Since we don’t know the POS tags of these clitics a priori, we estimate the conditional probability as

\[
\sum p(\text{POS} \mid c_{-i}\text{possible}_\text{POS} \ldots c_{-1}\text{possible}_\text{POS}) .
\]

For example, if the previous clitic could be a NOUN or an ADJ, then

\( p(\text{POS} \mid c_{-1}) = p(\text{POS} \mid \text{NOUN}) + p(\text{POS} \mid \text{ADJ}) \).

If the clitic is a stem, we also compute the following features:

- \( p(\text{POS} \mid \text{stem}_\text{template}) \). Arabic words are typically derived from a closed set of roots that are placed in so-called stem templates to generate stems. For example, the root \( \text{k}\text{t}\text{b} \) can be fit in the template \( \text{CCAC} \) to generate the stem \( \text{k}\text{t}\text{Ab} \) (book). Stem templates may conclusively have one POS tag (e.g., \( \text{yCCC} \) is always a verb) or favor one tag over another (e.g., \( \text{CCAC} \) is more likely a NOUN than an ADJ).

- \( p(\text{POS} \mid \text{prefix}) \) and \( p(\text{POS} \mid \text{suffix}) \). Some prefixes and suffixes restrict the possible POS tags for a stem. For example, a stem preceded by DET is either a NOUN or an ADJ.

- \( p(\text{POS} \mid \text{prefix}, \text{prev}_\text{word}_\text{prefix}), p(\text{POS} \mid \text{prev}_\text{word}_\text{suffix}) \) and \( p(\text{POS} \mid \text{prev}_\text{word}_\text{POS}). \) Arabic has agreement rules for noun phrases and idafa constructs (Noun+Noun relation) that cover definiteness, gender, and number. Both these features help capture agreement indicators.

In case we could not compute a feature value during training (e.g., a clitic was never observed with a given POS tag), the feature value is set to \( \epsilon = 10^{-10} \).

If the clitic is a prefix or a suffix, stem-specific features are assigned the same \( \epsilon \) value.
Table 2: Part-of-speech tag set of Farasa.

<table>
<thead>
<tr>
<th>POS</th>
<th>Description</th>
<th>POS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADV</td>
<td>adverb</td>
<td>ADJ</td>
<td>adjective</td>
</tr>
<tr>
<td>CONJ</td>
<td>conjunction</td>
<td>DET</td>
<td>determiner</td>
</tr>
<tr>
<td>NOUN</td>
<td>noun</td>
<td>NSUFF</td>
<td>noun suffix</td>
</tr>
<tr>
<td>NUM</td>
<td>number</td>
<td>PART</td>
<td>particles</td>
</tr>
<tr>
<td>PREP</td>
<td>preposition</td>
<td>PRON</td>
<td>pronoun</td>
</tr>
<tr>
<td>PUNC</td>
<td>punctuation</td>
<td>V</td>
<td>verb</td>
</tr>
<tr>
<td>ABBREV</td>
<td>abbreviation</td>
<td>CASE</td>
<td>alef of tanween fatha</td>
</tr>
<tr>
<td>FOREIGN</td>
<td>non-Arabic as well as non-MSA words</td>
<td>FUT_PART</td>
<td>future particle “s” prefix and “swf”</td>
</tr>
</tbody>
</table>

In order to improve efficiency and reduce the choices the classifier needs to pick from, we employ some heuristics that restrict the possible POS tags to be considered by the classifier: (i) If the clitic is a number (composed of digits or spelled in words), restrict to “NUM”. (ii) If all the characters are Latin, restrict to “FOREIGN”. (iii) If it is a punctuation mark, restrict to “PUNCT”. (iv) If the clitic is a stem and we can figure out the stem-template, restrict to POS tags that have been seen for that stem-template during training. (v) If the clitic is a stem, restrict to POS tags that have been seen during training, given the prefixes and suffixes of the word.

We trained the POS tagger using the same partitions of the ATB that we used for the segmenter (cf. Section 3.2). Table 1 shows the accuracy of our POS tagger on the WikiNews dataset [48] and compares it to Madamira. Madamira edges Farasa by 1.6%. A manual inspection on a random sample of 100 errors showed that 54% of the miss-classifications come from the confusion between adjectives and nouns, whereas 13% are between verbs and nouns. Errors in the preliminary segmentation step cause 21% of the POS mistakes. In such cases, any assigned POS would be incorrect. Table 3 lists the observed error types (covering 95% of errors) including examples.

The POS tagger also assigns gender and number tags to nouns and adjectives. This module is carried over from the Qatara POS tagger [50] and uses the random forest classifier from Weka [54]. The classifier generated 10 trees, with 5 attributes for each tree with unlimited depth, and was trained using 8,400 randomly selected unique nouns and adjectives from ATB. The classifier uses the following features: (i) stem template; (ii) stem template length; (iii) POS tag; (iv) attached suffix(es); (v) whether the word ends with
<table>
<thead>
<tr>
<th>Error Type</th>
<th>%</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJ → NOUN</td>
<td>29</td>
<td>“Al&lt;ElAm Albdyl” (<em>alternative media</em>)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Albdyl” recognized as NOUN</td>
</tr>
<tr>
<td>NOUN → ADJ</td>
<td>25</td>
<td>“m$AryE wykymAnyA” (<em>Wikimania projects</em>)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“wykymAnyA” recognized as ADJ</td>
</tr>
<tr>
<td>Segment Error</td>
<td>21</td>
<td>“blgp AlbAyvwn” instead of “Al+bAyvwn”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(in <em>Python</em> language)</td>
</tr>
<tr>
<td>V → NOUN</td>
<td>10</td>
<td>“hw Elm AlErbyp” (<em>he taught Arabic</em>)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Elm” recognized as NOUN (<em>science</em>)</td>
</tr>
<tr>
<td>Function words</td>
<td>7</td>
<td>“mnhA” (from it) recognized as ADJ</td>
</tr>
<tr>
<td>NOUN → V</td>
<td>3</td>
<td>“k$f Avry” (<em>archaeological discovery</em>)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“k$f” recognized as V (<em>discovered</em>)</td>
</tr>
</tbody>
</table>

Table 3: POS tagging error types and examples; covering 95% of the errors.

For testing, 20-fold cross validation was used. The average accuracy for gender and number classification were 95.6% and 94.9% respectively [50].

3.4. Farasa Constituency Parser

The Farasa constituency parser is an in-house re-implementation of the Epic parser [55]; the best-performing Arabic parser in the SPMRL 2013 multilingual constituency parsing shared task [56]. The parser uses a CRF model trained on features derived from the Farasa POS tagger. In compliance with the ATB segmentation, we attached determiners and noun suffixes to the stems. For each clitic, we obtain the information provided by the POS tagger, namely the POS, gender, number, whether the clitic has a determiner, and whether the clitic ends with *ta-marbouta* — the feminine singular noun

---

10We crawled the gazetteer from a list of Palestinian high school graduates including names and genders and Arabic Wikipedia articles (snapshot from September 28, 2012) that have English equivalents and belong to the Wikipedia categories containing the words ‘person’, ‘birth’, and ‘death’ if it has gender information.
suffix. Given such information, the parser generates surface features for each clitic \( c_0 \). Some of these features include the leading and trailing letters in a clitic. The parser uses the leading \( n \) letters in the clitic as features \((n \in [1,5])\). For example, given the clitic \( \text{AlktAb} \) (the book), these features would be \{A,Al,Alk,Alkt,AlktA\}. Similarly, the parser uses the trailing \( l \) letters in each clitic as features, \((l \in [1,5])\). A constraint is placed on the leading and trailing letters: the resulting sequence needs to occur 100+ times in the training data. Furthermore, the parser considers span features, where a span is a bracketed sub-tree (e.g., \("(NP (\text{NOUN AlktAb}))\"\)). The span features include the span’s first word, last word, and length; the words before and after the span; split point feature; and span shape feature. To ensure a well-formed nested tree, the parser deduces a minimal probabilistic context-free grammar (PCFG). The parser depends primarily on surface features (i.e. derived only from the clitics in the sentence) to provide context and deep syntactic cues.

Depending primarily on the surface features gives the parser two advantages. Firstly, it greatly simplifies the structural components of the parser, which would not affect the parser’s efficiency since so many deep syntactic cues have surface manifestations. Secondly, it allows for an easy adaptation to new languages.

We used the SPMRL 2013 shared task dataset [57] considering the same training/dev/test partitions for evaluation. In our first experiment, we used the original gold POS tags from the dataset. In our second experiment, we use the segmentation and POS tagging as generated by Farasa. Table 4 compares Farasa (with the two setups) and the Epic parser [55]. Although the Farasa parser is a re-implementation of EPIC, the obtained results differ. Farasa parser when trained with the same dataset as the EPIC parser outperforms it on the dev set, but lags behind on the test with a 1.74 drop in \( F_1 \) measure. When using the Farasa segmenter and POS tagger to tag
words instead of the gold tags we observe a drop of 2.76 and 0.67 for the dev and test sets respectively. The drop can be attributed to tagging errors that are propagated to the parser. However, the drop of 0.67 on the test is an affordable cost for the automation process.

As aforementioned, the Farasa tools are trained on the news genre written in Modern Standard Arabic (MSA), whereas Web forums commonly contain texts written in informal or Dialectal Arabic (DA). Farasa recognizes most of the dialectal words as out of vocabulary (OOV), which affects negatively POS tagging, NER, and syntactic parsing. For a sample of 100 random questions and answers from the Altibbi question-and-answering medical forum,\(^\text{11}\) we found that 20% of questions contain at least one dialectal word while answers are written in MSA by professional doctors. In this domain, we found that the majority of the DA words are function words, whereas content words and terms, such as diseases and body parts, are written in MSA. At the semantic level, this is less important compared to the effect at the syntactic level.

A small degradation in accuracy in Arabic QA systems may occur when using Farasa, designed for MSA, when dealing with DA. Nevertheless, as our results in Section 6 show, this degradation is not important.

4. Kernels for Question Re-Ranking

Now we focus on the re-ranking step of cQA, having as input a query question and a set of question-answer pairs, previously retrieved from a Web forum (cf. Section 2.2). Let \( Q \) and \( A \) be the set of questions and answers (passages) from the forum, respectively. Let \( q \) be a new question. Our task is to model a scoring function, \( r : Q \times Q \times A \to \mathbb{R} \), which reranks \( k \) question–answer pairs, \( \langle \rho, \alpha \rangle \), where \( \rho \in Q \), \( \alpha \in A \), with respect to their relevance to \( q \). Please note that \( Q \times A = D \), which we used in other sections for a more compact reference. We design our scoring function as:

\[
r(q, \rho, \alpha) = \vec{w} \cdot \phi(q, \rho, \alpha) .
\]

We can use implicit representations in kernel-based machines, e.g., SVMs, by expressing \( \vec{w} \) as

\[
\vec{w} = \sum_{i=1}^{n} \tau_i y_i \phi(q_i, \rho_i, \alpha_i) ,
\]

\(^{11}\)http://www.altibbi.com; this is the source of the corpus we use in this research.
where $n$ is the number of training examples, $\tau_i$ are weights, $y_i$ are the example labels (Relevant and Irrelevant), and $\phi(q_i, \rho_i, \alpha_i)$ is the representation of the question pairs. This leads to the following scoring function:

$$r(q, \rho, \alpha) = \sum_{i=1}^{n} \tau_i y_i \phi(q, \rho, \alpha) \cdot \phi(q_i, \rho_i, \alpha_i)$$

(3)

where the kernel $K(\cdot, \cdot)$ intends to capture the similarity between pairs of objects constituted by the query and the retrieved question answer pairs.

To any $\phi()$ whose codomain is finite corresponds a kernel function $K(x, x')$, defined on the input space such that $\forall x, x', K(x, x') = \phi(x) \cdot \phi(x')$ [58]. We used three types of representations: parse trees, features derived from word embeddings (word2vec), and text similarity metrics. We combine them as follows:

$$K(\langle q, \rho, \alpha \rangle, \langle q_i, \rho_i, \alpha_i \rangle) = \phi_{tk}(q, \rho) \cdot \phi_{tk}(q_i, \rho_i)$$

(4)

$$+ \phi_{w2v}(q, \rho, \alpha) \cdot \phi_{w2v}(q_i, \rho_i, \alpha_i)$$

(5)

$$+ \phi_{bow}(q, \rho, \alpha) \cdot \phi_{bow}(q_i, \rho_i, \alpha_i)$$

(6)

4.1. Tree kernels

We define Eq. (4) as follows

$$\phi_{tk}(q, \rho) \cdot \phi_{tk}(q_i, \rho_i) = TK(t(q, \rho), t(q_i, \rho_i)) + TK(t(\rho, q), t(\rho_i, q_i))$$

(7)

where $TK$ is a tree-kernel function; e.g., the SubSet Tree (SST) Kernel [59], which measures the similarity between trees. This way, we do not need to extract syntactic feature vectors from the text pairs (i.e., engineering $\phi_{tk}$ is unnecessary). We just need to apply TKs to the pairs of syntactic trees, which provides a score representing the structural similarity. We opt for the state-of-the-art TK model proposed by Severyn and Moschitti [60] and previously used for question ranking in cQA by Barrón-Cedeño et al. [61] and Romeo et al. [62]. As described in Eq. (4), we apply TKs to pairs of questions rather than questions with their answers.

The function $t(x, y)$ in Eq. (7) is a string transformation method that returns the parse tree from the text $x$ — the tree computed with Farasa—
What are the symptoms of depression in children and adolescents?

(What are depression symptoms?)

Figure 3: Constituency trees of two questions connected by REL links. The questions correspond to ids 200430 and 47524 in the CQA-MD corpus [15] (cf. Section 6.1).
Table 5: Overview of string, lexical, and syntactic similarity measures.
coded in a binary SVM [69].

This formulation combines two of the best models presented at SemEval 2016 Task 3 [27, 42, 71] (cf. Section 6.1).

5. Text Selection based on Neural Networks

As shown in Section 2, several neural network approaches have been successfully applied to QA tasks. Unfortunately, question retrieval in cQA is heavily affected by a large amount of noise and a rather different domain, which make it difficult to effectively use out-of-domain embeddings to pre-train neural networks. Figure 4 illustrates some of the difficulties in cQA questions: long greetings and introductions, spelling errors, and incorrect or missing punctuation marks. Correct grammar and usage of punctuation marks is important for sentence splitting and syntactic parsing. This probably prevented the participants to SemEval tasks from achieving satisfactory results with such models [15]. Inspired by [72], in [62] we tried to exploit neural models using their top-level representations for the \((q, \rho)\) pair and fed them into the TK classifier. Nevertheless, this combination proved to be ineffective as well.

Instead of trying to combine the models, we use neural networks to identify the most important pieces of text in both \(q\) and \(\rho\). We use an LSTM [73, 74], augmented with an attention mechanism. LSTMs have proven to be useful in a number of language understanding tasks. Recently Rocktäschel, et al. [75] adapted an attentional LSTM model [76] to textual entailment, and a similar model has been applied to cQA [77]. We follow the same setup of the latter (Section 5.1). Then, we use the attention weights for our text selection algorithm, which aims at removing subtrees containing useless or noisy information (Section 5.2).

5.1. Learning Word Importance with LSTM

The main idea of learning the importance of words for a task is to use the data and labels about the task itself. Given a pair \((q, \rho)\), we learn two serial LSTM models: LSTM\(_q\) reads the word vectors of \(q\), one by one, and records the corresponding memory cells and hidden states; the final memory cell is used to initialize LSTM\(_\rho\), which reads the word vectors of \(\rho\).

---

\(^{12}\)Binary SVMs showed comparable results to SVM\(^{\text{rank}}\) [70].
Formally, an LSTM computes the hidden representation for input $x_t$ with the following iterative equations:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{mi}m_{t-1} + b_i)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{mf}m_{t-1} + b_f)$$

$$m_t = f_t \odot m_{t-1} + i_t \odot \tanh(W_{xm}x_t + W_{hm}h_{t-1} + b_m)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{mo}m_t + b_o)$$

$$h_t = o_t \odot \tanh(m_t)$$

where $\sigma$ is the sigmoid function, $\odot$ is element-wise multiplication, and $i$, $f$, $o$, and $m$ are input, forget, output, and memory cell activation vectors. The crucial element is the memory cell $m$ that is able to store and reuse long term dependencies over the sequence. The $W$ matrices and $b$ bias vectors are learned during training.

The final hidden state of LSTM$_\rho$, $\vec{h}_\rho,N$, is used as a feature vector to feed a multi-layer perceptron (MLP) with one hidden layer, followed by a softmax classifier. The objective function is the cross-entropy objective over binary relevant/irrelevant target labels.

Given the hidden states produced by LSTM$_q$, we compute a weighted representation of $q$:

$$\vec{h}_q = \sum_{i=1}^{L} \beta_i \vec{h}_{q,i},$$

where $\vec{h}_{q,i}$ are the hidden states corresponding to the words of $q$, and the attention weights $\beta_i$ are computed as:

$$\beta_i = \frac{\exp(a(\vec{h}_{q,i}, \vec{h}_\rho,N))}{\sum_{j=1}^{L} \exp(a(\vec{h}_{q,j}, \vec{h}_\rho,N))}.$$
Function PruneTree \((T, th)\);

Input: a tree \(T\);

a pruning threshold \(th\);

Output: a pruned version of \(T\)

1. `pruneNode(root(T), th);`

Function pruneNode \((o, th)\);

3. If \(|children(o)| > 0\) then
4. 
5. For \(ch \in children(o)\) do
6.  
7. `pruneNode(ch, th);`
8.  
9. End
10. If \(|children(o)| = 0 \&\& !REL_Node(o)\) then
11.  
12. `remove (o, T);`
13.  
14. End
15. Else
16.  
17. If \(o.weight < th \&\& !REL_Node(o)\) then
18.  
19. `remove (o, T);`
20.  
21. End
22. End

Algorithm 1: Function PruneTree for pruning a tree according to attention weights.

Here \(a()\) is parameterized as a MLP with one hidden layer and a \textit{tanh} non-linearity [75]. The input to the MLP is then a concatenation of \(\vec{h}_q\) and \(\vec{h}_{\rho,N}\).

Intuitively, \(\beta_i\) assigns a higher weight to words in \(q\) if they are useful for determining the relation to \(\rho\). As we will see, these attention weights turn out to be useful for selecting important parts of the questions for the TK models. Note also that the attention here is one-sided — only on \(q\). In practice, we train another model, with attention on \(\rho\), and use its weights as well.

5.2. Parse Tree Pruning based on Neural Networks

Our tree-pruning approach to text selection is illustrated in Algorithm 1. Its main idea is to filter out the leaf nodes of the parse tree corresponding to words associated with weights lower than a user-defined threshold, where the word weights are provided by Eq. (9). The most important step of Algorithm 1 is the recursive function \textit{pruneNode}, which is initially invoked.

21
for the root node of the tree. Function \textit{pruneNode} checks whether the node $n$ is a leaf (Line 4) and then applies the appropriate strategy: (i) for non-leaf nodes, \textit{pruneNode} is invoked for the children of $o$, then $o$ is removed if all of its children are removed and (ii) a leaf node is removed if its weight is lower than the user-defined threshold, $th$. REL-tagged nodes are never removed, regardless of their weight. Different thresholds determine different percentages of pruned nodes, and we explore various thresholds as part of our experiments.

6. Evaluation of Question Re-Ranking Models

In this section, we aim at analyzing the impact of the different representation components in the cQA question re-ranking task. Section 6.1 describes the experimental settings. Section 6.2 illustrates the experimental methodology. Our experiments evaluate four aspects: (i) the impact of the NLP processors, (ii) the performance of kernels on vectorial features and tree kernels used in isolation, (iii) the performance of kernel combinations, and (iv) the impact of text selection using tree pruning. We analyze and discuss the results in Section 6.3.

6.1. Evaluation Framework

We perform our experiments using the evaluation framework released in the SemEval 2016 Task 3-D [15]. The framework consists of a corpus in Arabic from the medical domain—the CQA-MD corpus—and a set of evaluation metrics. Nakov et al. [15] queried different Web forums to build up a collection of query questions linked to a set of 30 candidate forum questions–answer pairs. The outcome: a total of 45,164 question–answer forum pairs attached to one of 1,531 query questions. The relevance of each $\rho \in D$ was manually annotated by means of crowdsourcing considering three labels: \textit{Direct} if $\rho$ contains a direct answer to $q$; \textit{Related} if $\rho$ covers some of the aspects asked by $q$; and \textit{Irrelevant} if $\rho$ and $q$ are unrelated. An ideal ranking should place all direct and relevant $\rho \in D$ on top, followed by the irrelevant pairs. Table 6 shows some statistics of the dataset. The answer associated with each of the 30 forum questions was provided by a professional physician and it is considered correct.
<table>
<thead>
<tr>
<th>Category</th>
<th>Train</th>
<th>Dev</th>
<th>Test</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Questions</td>
<td>1,031</td>
<td>250</td>
<td>250</td>
<td>1,531</td>
</tr>
<tr>
<td>QA Pairs</td>
<td>30,411</td>
<td>7,384</td>
<td>7,369</td>
<td>45,164</td>
</tr>
<tr>
<td>– Direct</td>
<td>917</td>
<td>70</td>
<td>65</td>
<td>1,052</td>
</tr>
<tr>
<td>– Related</td>
<td>17,412</td>
<td>1,446</td>
<td>1,353</td>
<td>20,211</td>
</tr>
<tr>
<td>– Irrelevant</td>
<td>12,082</td>
<td>5,868</td>
<td>5,951</td>
<td>23,901</td>
</tr>
</tbody>
</table>

Table 6: Statistics about the CQA-MD corpus (borrowed from [15]).

The official evaluation measure is Mean Average Precision (MAP); a standard evaluation metric in information retrieval computed as

\[
MAP = \frac{\sum_{q} |Q| AveP(q)}{|Q|},
\]

where \(Q\) is the set of test questions and \(AveP\) is the average precision value for each query, computed as

\[
AveP(q) = \frac{\sum_{k=1}^{|D_q|} (P(k) \times rel(k))}{\{|\text{relevant documents}\}|},
\]

where \(|D_q|\) is the number of retrieved pairs in the ranking, \(rel(k)=1\) if \(\rho\) at position \(k\) is relevant, and \(P(k)\) is computed as

\[
P(k) = \frac{|\{|\text{relevant documents}\} \cap \{|\text{retrieved documents}\}|_k}{k};
\]

that is, the size of the intersection between relevant and retrieved documents up to rank \(k\) divided by \(k\).

6.2. Experiments and Methodology

Our experiments address the question re-ranking stage in the architecture for community question answering (cf. Section 2). That is, given a query \(q\), re-rank a collection of related question–answer pairs in \(D_q\). In order to do that, we stick to the same training/development/test partition defined by Nakov et al. [15] for the SemEval 2016 cQA challenge. Regarding the implementation of the models, for the word2vec representations, we trained the embeddings on 26M words of unsupervised data, provided together with the CQA-MD corpus.

We designed four follow-up experiments of increasing complexity:
Table 7: MAP scores of the official submissions to the SemEval 2016 Task 3-D. In addition we report MAP values for the development set of our systems.

<table>
<thead>
<tr>
<th>Submission</th>
<th>Dev.</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 SLS [42]</td>
<td>47.31</td>
<td>45.83</td>
</tr>
<tr>
<td>2 ConvKN [27]</td>
<td>42.67</td>
<td>45.50</td>
</tr>
<tr>
<td>3 RDI_team [44]</td>
<td>—</td>
<td>43.80</td>
</tr>
<tr>
<td>4 QU-IR [45]</td>
<td>—</td>
<td>38.63</td>
</tr>
<tr>
<td>5 UPC_USMBA [78]</td>
<td>—</td>
<td>29.09</td>
</tr>
<tr>
<td>Random Baseline</td>
<td>—</td>
<td>29.79</td>
</tr>
</tbody>
</table>

Experiment 1: Impact of NLP Processors. Our first experiment uses only a tree-kernel SVM on parse trees. The difference between our two runs is that we either use Farasa or Stanford’s [1] technology to generate the parse-tree representations. This allows for an implicit comparison of these two parsers.

Experiment 2: Isolated Models. We perform tests on our three re-ranking models in isolation. Beside the tree-kernel SVM on parse trees from Experiment 1, we experiment with a linear-kernel SVM on word2vec and similarity representations and with the attentional LSTM neural network.

Experiment 3: Kernel Combination. We combine two SVM kernels on different features: tree kernels on the parse trees and the linear kernel on the word2vec and similarity representations.

Experiment 4: Tree Pruning. We explore different thresholds to prune the parse trees on the basis of the LSTM attention weights before learning the scoring function with an SVM. Specifically, we perform experiments combining tree kernels with the linear kernel on word2vec and similarity features.

6.3. Results and Discussion

In order to provide a more comprehensive perspective of our experimental results, Table 7 reports the MAP values obtained by the participant systems on the test set of SemEval 2016 Task 3-D. It should be noted that we designed both the two top systems, SLS and ConvKN. The first one was based on a committee of four different systems using different embedding versions as well as methods for filtering the initial word representation, whereas the second applied tree kernels and similarity metrics. In this paper, we only used one system from SLS, corresponding to our linear kernel, which performs relatively more stably with respect to both development and test sets.
Although committees are rather effective and typically produce higher accuracy than a single system, they tend to obscure the contribution of the different representations, which are the main target of our study.

It is worth noting that the test set results in Table 7 are obtained by models trained on the training data merged with the development set. Thus, such results are generally higher than those we obtain in this paper on the test set, where we only use the training set for learning all our models. We preferred this approach for our experiments so that we can better compare the results between development and test sets and, at the same time, have a faster training and test processing.

6.3.1. Experiment 1: Impact of NLP Processors.

As a way to compare Farasa and Stanford parsers, we ran a set of experiments in which the only difference was the processor used to generate the trees. We used an SVM with $C = 1$ and the normalized SST kernel [79] as TK in Eq. (7) with the following values for the parameter $\lambda = \{0.001, 0.01, 0.05, 0.1, 0.2\}$, which provide different weights to subtrees of different size. Changing $\lambda$, we can emphasize different portions of the parse trees and thus carry out a more systematic comparison between the parsers.

Figure 5 shows the MAP evolution for the two models, with respect to the $\lambda$ parameter of the kernel. The highest MAP values on development

![Figure 5: MAP as a function of the $\lambda$ parameter of the SST kernel. We compare the performance of our tree-kernel model when the parse-tree representation is built with either Farasa or Stanford.](image)
Table 8: MAP performance for our ranking models when applied in isolation on the development and test partitions.

<table>
<thead>
<tr>
<th>Model</th>
<th>Dev.</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear-kernel SVM on Word2vec and sims.</td>
<td>44.94</td>
<td>40.73</td>
</tr>
<tr>
<td>Tree-kernel SVM on Farasa Parse trees</td>
<td>42.53</td>
<td>40.87</td>
</tr>
<tr>
<td>NN (attention on q)</td>
<td>34.85</td>
<td>33.40</td>
</tr>
<tr>
<td>NN (attention on ρ)</td>
<td>37.47</td>
<td>35.09</td>
</tr>
</tbody>
</table>

(39.93) and test (38.49) sets are obtained when using Farasa. In such cases the increment with respect to Stanford is of 1.44 and 0.88 MAP points, respectively. This is an interesting result as it is in line with our linguistic expert of Arabic who, analyzing some of the trees generated on our data by both parsers, observed a better quality of the Farasa POS-tagger than the one used in the Stanford parser. This different quality also affects chunk definition and their dependencies. It seems that using the entire structure of the parse tree allows TKs to benefit from an overall better quality of Farasa parser to produce better rankings.

6.3.2. Experiment 2: Isolated Models.

Table 8 shows the performance of our ranking models when applied in isolation. The linear- and the tree-kernel models perform on par with each other on the test set, both obtaining competitive results. Still, they lie behind the top 2 systems included in Table 7, at MAP values of ∼ 40.8 on the test set.

As aforementioned, the neural network does not reach a competitive performance, maybe due to the small amount of data available for training. However, this is not the only contribution the network model can provides as we can use its weights for text selection.

6.3.3. Experiment 3: Kernel Combination.

The first row of Table 9 reports the performance of the combination of the tree kernel on parse trees built with Farasa and the linear kernel on word2vec and similarity features. Note that the combination improves over tree kernel and linear kernel in isolation. With respect to our previous systems, i.e., SLS and ConvKN, we got lower values for the test set: as previously pointed out, (i) SLS is a combination of four different systems; and (ii) in this paper, we only use the training data, whereas we trained SLS and ConvKN on both the training and development sets to obtain the test set results.
Table 9: MAP performance for our ranking models when applied in combination and after pruning. The latter was applied with two different thresholds, 0.74 and 0.82, which obtained the highest MAP on development and test sets, respectively.

<table>
<thead>
<tr>
<th>Model</th>
<th>Dev.</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree-kernel (no pruning) + Word2vec and sims.</td>
<td>46.58</td>
<td>41.09</td>
</tr>
<tr>
<td>Tree-kernel (pruning ratio 0.74) + Word2vec and sims.</td>
<td>46.78</td>
<td>41.93</td>
</tr>
<tr>
<td>Tree-kernel (pruning ratio 0.82) + Word2vec and sims.</td>
<td>46.01</td>
<td>42.20</td>
</tr>
</tbody>
</table>

Figure 6: Experiments with pruned trees. From top to bottom the plots show the prediction time, the learning time and MAP as a function of the ratio of pruned nodes.

6.3.4. *Experiment 4: Tree Pruning.*

While combining feature vectors and tree kernels improves the MAP scores in our experiments, the use of tree kernels has a negative impact on the running time. Thus, we prune parse trees as described in Section 5.2.

In this experiment, we evaluate the combination of the linear kernel on word2vec and similarity features with the SST kernel over syntactic trees. Both kernels are not normalized. The top two plots show prediction and learning time (in minutes) as a function of the ratio of pruned nodes. As expected both learning and prediction times decrease roughly linearly with respect to the number of pruned tree nodes.
The plot at the bottom shows the corresponding MAP values, again as a function of the ratio of pruned nodes. Rather than decreasing due to the reduced representation, the MAP scores increase, reaching 46.78 (+0.20 with respect to no pruning) on the development set and 42.20 (+1.11) on the test set. This occurs because our pruning model manages to filter out irrelevant fragments from the trees. For instance, discarding the phrase “in children and adolescents” in Figure 3 would allow a model to better determine that the two questions are practically equivalent.

The threshold maximizing MAP on the development set is the one corresponding to 0.74 pruning ratio (see second line of Table 9). Its MAP score on the test set is 41.93 (+0.84) and the learning and prediction times decrease from 887 to 295 minutes and from 98 to 20 minutes, respectively, with respect to the unpruned data. This means that learning and prediction processes are 3 and 4.9 times faster than the kernel combination without pruning.

7. Conclusions

Recently, community-driven question answering in websites (cQA) has seen a renewed interest both from natural language processing and information retrieval researchers. Most work in cQA has been carried out for the English language, resulting in a lack of techniques and resources available to deal with other languages, such as Arabic. Motivated by this aspect, in this paper we addressed the problem of cQA in an Arabic forum. In particular, we focused on the task of question re-ranking: given a newly-posted question, retrieve equivalent or similar questions already in the forum. If similar questions have been addressed in the past, the users can quickly obtain an answer to their question.

In order to deal with the necessary processing of the Arabic texts, for the first time, we introduced some components of our in-house pipeline of Arabic NLP tools called Farasa. This includes a segmenter, a POS tagger, a named entity recognizer, a dependency parser, a constituency parser, and a diacritizer. We integrated Farasa into our cQA architecture using the UIMA-based framework. This way, we could extract effective features, such as lexical and syntactic information from Arabic text, and feed them into our machine learning models. Our evaluation on a realistic collection of forum questions in the medical domain allowed us to test Farasa’s capabilities when dealing with a real-world application.
In particular, we addressed the task of question re-ranking as a binary classification problem, where each example represents a pair \{user-question, forum-question\}. We proposed an effective combination of tree kernels built on top of the constituency parse trees provided by Farasa and Arabic word embeddings based on neural networks. This combination allowed for better capturing the semantic relatedness between two short pieces of text, i.e., questions and pairs of questions and answers, and achieved state-of-the-art performance for Arabic question re-ranking.

Additionally, we designed models for selecting meaningful text in order to reduce noise and computational cost. For this purpose, we applied long short-term memory neural networks to identify the best subtrees in the syntactic parsing of questions, which are then used in our tree-kernel-based ranker. We combined the text selection approach with word embeddings based on neural networks, boosting the performance. With thorough experiments we showed that (i) syntactic information is very important for the question ranking task, (ii) our model combining tree kernels, word embeddings and neural networks for text selection is an effective approach to fully exploit advanced Arabic linguistic processing and (iii) our reranker based on tree kernels can be used to implicitly evaluate the performance of different syntactic parsers.

Finally, our UIMA pipeline for Arabic NLP as well as for cQA will be made available to the research community.

References


\bibinfo{doi}{{http://dx.doi.org/10.1007/s10579-009-9114-z}}.


[82] Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on
Natural Language Processing, ACL-HLT ’15, Association for Computational Linguistics, Beijing, China, 2015.


