Just.Ask – A multi-pronged approach to question answering

Ana Cristina Mendes*, Luísa Coheur†, João Silva‡, Hugo Rodrigues§

L2F, INESC-ID Lisboa/IST
Av. Prof. Cavaco Silva
2780-990 Porto Salvo Tagus Park, Portugal
Tel.: +351-214 233 577
Fax: +351-214 233 252

In the last decades, several research areas experienced key improvements due to the appearance of numerous tools made available for the scientific community. For instance, Moses plays an important role in recent developments in machine translation and Lucene is, with no doubt, a widespread tool in information retrieval. The existence of these systems allows an easy development of baselines and, therefore, researchers can focus on improving preliminary results, instead of spending time in developing software from scratch. In addition, the existence of appropriate test collections leads to a straightforward comparison of systems and of their specific components.

In this paper we describe Just.Ask, a multi-pronged approach to open-domain question answering. Just.Ask combines rule- with machine learning-based components and implements several state-of-the-art strategies in question answering. Also, it has a flexible architecture that allows for further extensions. Moreover, in this paper we report a detailed evaluation of each one of Just.Ask components. The evaluation is split into two parts: in the first one, we use a set of questions gathered from the TREC evaluation forum, having a closed text collection, locally indexed and stored, as information source; in the second one, we use a manually build test collection – the GoldWebQA – that intents to evaluate Just.Ask performance when the information source in use is the Web, without having to deal with its constant changes. Therefore, this paper contributes with a benchmark for research on question answering, since both Just.Ask and the GoldWebQA corpus are freely available for the scientific community.

Keywords: Question Answering; Evaluation Framework; Question Interpretation; Passage Retrieval; Answer Extraction

1. Introduction

With the advent of the Internet and the World Wide Web (WWW) in the early 1990s, massive amounts of textual information have become widespread available to the general public, making it a highly attractive place for searching information. However, as the amount of information keeps growing at a staggering pace, it is becoming more and more difficult to find specific information. The traditional information retrieval approach to this problem – web search engines –, require users

*ana.mendes@l2f.inesc-id.pt
†luisa.coheur@l2f.inesc-id.pt
‡joao.silva@l2f.inesc-id.pt
§hugo.rodrigues@l2f.inesc-id.pt

to pose their queries in terms of a set of keywords, which are then used to return a plethora of documents that the engine deems as relevant. While this approach works well in many cases, sometimes what a user really wants is to get a succinct answer to a given question, instead of searching for it in a collection of thousands of documents. Question Answering (QA) systems deal with this problem, by providing interfaces in which users can express their information needs in terms of a natural language question, and retrieve the exact answer to that very same question instead of a set of documents.

In the last years, if several evaluation forums, such as the Cross-Language Evaluation Forum (CLEF) and the Text REtrieval Conference (TREC), have testified many improvements in QA research, it is also true that QA tasks are becoming more demanding, making it difficult for newcomers to participate in the competition. In addition, although a global comparison of the participating systems is straightforward in these evaluation forums, it is still very difficult (or impossible) to compare the specific components of each system, since the results achieved by these are typically not reported or obtained under different conditions.

In this paper we present Just.Ask, an open-domain question answering system that implements several Artificial Intelligence techniques and takes advantage of different available information sources and tools. Just.Ask already attains state of the art results in Question Classification [1] and some of its modules can be easily replaced/enhanced, allowing straightforward comparisons of (new) techniques. Just.Ask is specially focused in factoid-like questions, defined as short-sized, fact-based questions. Nonetheless, non factoid-like questions (like definitions) are also addressed. This paper also contributes with a detailed (intrinsic and extrinsic) evaluation of Just.Ask by using a local corpus as information source and also by using the Web. Just.Ask performance is tested with a set of questions from the TREC evaluation campaigns and also with a corpus partially built with questions from CLEF and snippets retrieved from the Web. In a point of fact, due to the continuous changes in the Web, we decided to manually build this corpus, the GoldWebQA, based on the Multisix English corpus [2], made up of questions that were plausible at a certain point in time, and the snippets retrieved from the Web by that time, containing possible answers to that questions. Moreover, we have manually added to that corpus all the correct answers stated in the snippets, independently of the way these are stated. Both Just.Ask and the GoldWebQA are freely available for the scientific community.[a] We hope in this way to make a relevant contribution to research in QA, as Just.Ask can be used as a benchmark in future research work and GoldWebQA allows to easily test different answer extraction/selection techniques.

The organization of the paper is the following: Section 2 describes related work, Section 3 details Just.Ask question interpretation component, Section 4 explains how passage retrieval is made in Just.Ask and Section 5 presents the answer extraction step. Then, Section 6 details the experimental setup, including the evaluation

[a]Both are freely available at `http://qa.l2f.inesc-id.pt/wiki/index.php/Resources`.

corpora. Sections 7 and 8 report the evaluation of each component, as well as an end to end evaluation of Just.Ask, and Section 9 concludes and points to future improvements. Finally, Appendix A focus on the parametrization possibilities of Just.Ask

2. Background

Although, in recent years, a great deal of attention has been given to question answering by the research community, QA is by no means a new field of research. In 1965, [3] published a survey article describing no less than fifteen question answering systems for the English language that were built in the preceding five years. Among the reviewed systems there was `BASEBALL` [4], which handled questions about baseball games played in the American League over a period of one year. A few years later, [5] – sponsored by NASA – developed `LUNAR`, which answered questions about lunar rock and soil samples that were collected by the Apollo Moon missions. The system was demonstrated at the Lunar Science Conference in 1971, where it answered correctly to 78% of the questions posed by the attending geologists [6].

Both `LUNAR` and `BASEBALL` were essentially natural language interfaces to databases (NLIDB), where a user's question is translated into a database query, and the query's output is returned as the answer. Many more systems were developed along the lines of `LUNAR` and `BASEBALL`, being the majority of these systems still limited to restricted-domains, having its knowledge stored in a database, and being very hard to port to different domains. A detailed survey on NLIDBs can be found in [7].

More recently, with the advent of the WWW in the early 1990s, and the resultant explosion of electronic media, many research groups began to exploit the web as a large text corpus, creating the so called web-based question answering systems, such as `START` [b] [8,9].

Despite these initial efforts, QA was in some way forgotten for some years, and it was not until 1999, with the launch of the QA track [10] in the renowned TREC[c], that QA became a very hot research area. In fact, in the last years, several evaluation forums, like CLEF and TREC, have promoted and testified many of the improvements of QA systems. Nevertheless, although the participating systems implement an extensive panoply of techniques, the general architecture of a modern QA system has become standardized [11], consisting of a pipeline composed of three main components dedicated to question interpretation, passage retrieval and answer extraction, respectively. Figure 1 depicts the typical QA pipeline.

Question interpretation is the task responsible for understanding the posed questions. It strongly varies from system to system and can involve several subtasks, like **question classification** or the extraction of different types of information

[b] `http://start.csail.mit.edu/`
[c] `http://trec.nist.gov/`

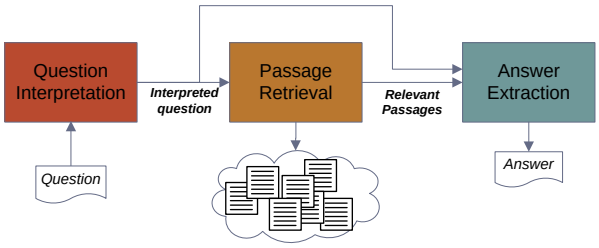

Fig. 1: Typical pipelined architecture of a QA system.

(named entities or main verbs, for example). Question classification is, however, transversal to most systems. During this step a semantic category representing the type of the answer that is being sought after is assigned to the given question. This process is extremely important for two reasons: the attained classification can help narrowing down the number of possible candidate answers and, depending on the question category, different strategies can be chosen to find the correct answer. Moreover, a misclassified question can hinder the ability of the system to reach the right answer, because it can cause downstream components to make wrong assumptions about the question. Hence, it is of crucial importance to have a precise question classifier, in order to achieve good overall results. The set of question categories is referred to as question type taxonomy. Li and Roth's [12] two-layer question type taxonomy is widely used in QA, specially by the machine learning community, probably due to the fact that the authors have published nearly 6,000 labelled questions and made them freely available. In fact, there has been much research in question classification exploring the use of machine learning techniques and evaluated against Li and Roth corpus [12,13,14,15].

After question interpretation, the next step is to find relevant passages where the answer to the question might be found. In QA parlance, this step is usually referred to as **passage retrieval**. It should be noticed that some systems, like [16], prefer to retrieve documents instead of passages, and analyze the whole document in search for the answer. If, on the one hand, this approach can introduce noisy information to be later filtered in the answer extraction module, on the other hand, the usage of advanced NLP techniques can potentiate the detection of answers hidden in anaphoric and elliptic expressions, which is barely impossible when the system is dealing merely with passages. The trade-off between passage retrieval and document retrieval for the purpose of QA has been matter of study [17,18]. Nevertheless, the passage retrieval component is often neglected, in the sense that most systems rely on already existing solutions of information retrieval, like Google or Lucene search engines. However, there are still some systems, such as the one from `PRIBERAM` [19], `QA@L2F` [20] or `QRISTAL` [21], that endeavor in a stage of pre-processing the corpus, by collecting and organizing information contained in thousands of documents in relational databases. The passage retrieval module can also make use of the question

category that is obtained by the question classification component, by using this information to retrieve only those documents that contain at least one occurrence of the expected question type, allowing the answer extraction module to only search documents that may potentially have the correct answer. Another fundamental task in passage retrieval has to do with the **query formulation**, in which a question is translated into a suitable representation that can be manipulated by the information source to retrieve relevant passages. Note that this representation is not necessarily an unstructured set of keywords – indeed, it can be based on a structured query language.

The **answer extraction** is the final component in the QA pipeline. Its goal is to extract and select the final answer from the relevant passages returned by the passage retrieval and present it to the user of the system. As an example, consider the question "*When was Mozart born?*", classified as NUMERIC:DATE, and for which the following passages were retrieved:

- *Mozart Wolfgang Amadeus Mozart (born in Salzburg, Austria, on **January 27, 1756** – died in Vienna, Austria, on **December 5, 1791**) was a great composer.*
- *Mozart was born **January 27, 1756** in Salzburg, Austria.*

Given the above information, the answer extraction component would be responsible for extracting candidate answers from the passages – e.g., the instances of dates in bold face –, and then choose the correct answer, which is *January 27, 1756* and, finally, to present it to the user.

Traditional methods in answer extraction are based on pattern matching [22], noun phrases and named entity extraction [23,24], especially if the target are factoid questions. However, many other approaches exist. For instance, [25] implements a Support Vector Machines-based answer extractor of definition questions, based on tree kernel technology; [26] also presents a machine learning approach, focused on parameter learning for entity answer; [27] targets on relating candidate answers before the answer selection step.

Considering recent efforts towards the availability of QA resources, both Aranea [24] and Ephyra [28] should be mentioned. Aranea is a QA system based on two different approaches, one that searches the answer in structured resources, the other that exploits data redundancy of unstructured information sources. In the later, the used Natural Language Processing (NLP) techniques are reduced to the count of n-grams for the purpose of candidate answers extraction, while the selection of the final answer is based on several heuristics. OpenEphyra is a framework for QA derived from the Ephyra system [28]. This framework follows a more semantic approach, namely when it comes to the extraction of candidate answers: several similarity measures are calculated between the semantic structures of the questions and sentences where to extract the answer, and the candidate answers are extracted depending on their semantic type or the semantic role missing in the question.

3. Question interpretation in Just.Ask

The **Question Interpretation** component of Just.Ask receives as input a natural language question and outputs an "interpreted question", which is a structure composed of the different types of information extracted from the question, namely: the question tokens, the question syntactic components, the question headword and the question classification. Therefore, this task is responsible for two main steps: question analysis and question classification. For this, it makes use of available NLP tools/resources and techniques. Moreover, Just.Ask implements a module that, depending on the question, determines its headword or focus. A detailed view of the Question Interpretation component of Just.Ask is shown in Figure 2.

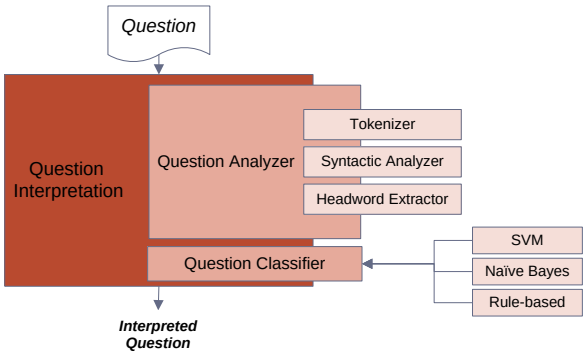

Fig. 2: Detailed view of the Question Interpretation component of Just.Ask.

In order to allow a straightforward comparison of results between systems, Just.Ask classifier exploits the widely used Li and Roth's [12] two-layer question type taxonomy, consisting of 6 coarse grained categories (ABBREVIATION, DESCRIPTION, ENTITY, HUMAN, LOCATION, and NUMERIC), which are further divided into 50 finer grained categories.

The classifier of Just.Ask allows the usage of different classification techniques. Particularly, classification can be made with recourse to hand-built rules or to machine-learning techniques, namely Support Vector Machines (SVM) and Naïve Bayes. State of the art results were attained by the classifier of Just.Ask when modeling the task of question classification as a supervised learning classification problem (using SVM), although the most successful features used to train the model are generated by the rule-based classifier. In fact, after several experiments, we realized that the usage of unigrams, headwords and its semantic classification resulted in the highest accuracy, being the last two set of features created by the rule-based classifier.

In what concerns the question headword, Just.Ask adopts a similar notion to that described in [15]: the headword is a word in a question that *represents* the information that is being sought after. For instance, considering the question *What*

is the capital of Spain?, *capital* is its headword. Note that many questions do not have a headword. For example, in *How many plays has Shakespeare written?* we are searching for the number of plays written by the poet, not the plays themselves (for that, the question could be *Which plays has Shakespeare written?*, with headword *plays*). Our approach to find the question headword uses the parse tree of a question, traversed top-down by a pre-defined set of rules – the head-rules. For the purpose of parsing the input question, we use the Berkeley Parser [29], trained on the QuestionBank [30], a treebank of 4000 parse-annotated questions. When it comes to the head-rules, they are a heavily modified version of those given in [31], specifically tailored to extract headwords from questions. Nevertheless, there are a few exceptions to the head-rules and, for that, we use a set of *non-trivial rules* that determine the correct headword for situations that cannot be covered by the *head-rules*.

In order to enrich the question headword with semantics, WordNet [32], the lexical database of English, was incorporated in Just.Ask. Therefore, sets of related WordNet synsets were manually grouped into fifty clusters, each representing a question category. Then, the question headword was translated into a WordNet synset using a set of heuristics. Finally, a breadth-first search on the translated synset's hypernym tree was employed, in order to find a synset that pertains to any of the pre-defined clusters; several heuristics were used to aid word sense disambiguation.

A detailed description of the classification task in Just.Ask, as well as an extensive report on the experiments we conducted to find the most promising features to train the machine-learning classifier, can also be found in [1]. The set of hand-built rules used in Just.Ask rule-based classifier is described in the same paper and is publicly available for research purposes in `http://qa.l2f.inesc-id.pt/wiki/index.php/Resources`.

4. Passage retrieval in Just.Ask

The **Passage Retrieval** component of Just.Ask receives as input the interpreted question from the preceding component and outputs the set of relevant passages for the posed question.[d] Figure 3 shows a zoom-in perspective of this component.

Just.Ask employs a multi-strategy approach to passage retrieval, with each strategy tailored to a specific question category or groups of question categories. Different strategies involve the use of different **information sources** and different **query formulations**. Once the information source to be used has been identified, and the queries have been formulated, the final step is to submit the queries to the information source's endpoint and retrieve the results. These queries are all submitted and processed in parallel, using multiple threads, in order to maximize the performance of the system. The results of each query are aggregated (with duplicates removed),

[d]Throughout this paper, we will refer to the output of the Passage Retrieval component as *passage*, regardless of it being a paragraph from a text or the short summary retrieved by a search engine to a query (snippet).

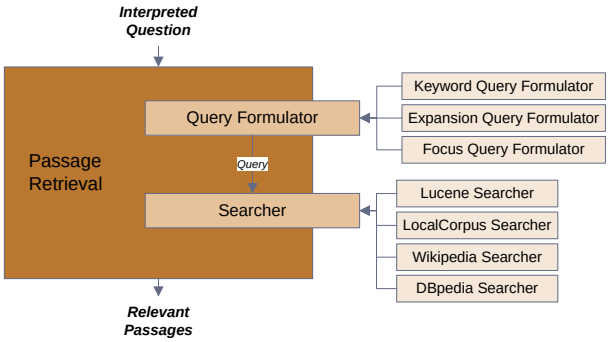

Fig. 3: Detailed view of the Passage Retrieval component of Just.Ask.

and sent to the answer extraction module for further processing. Metadata about the results, such as the rank of each search result, is also stored and used in the answer filtering module. The following subsections describe the information sources and the query formulation strategies used in Just.Ask.

4.1. *Information sources*

At the moment, and although this can be easily extended, Just.Ask uses Lucene[e] search engine to retrieve relevant passages from unstructured sources; Wikipedia[f] and DBpedia [33] are repositories of semi-structured content also used by the system.

The search engines are used to retrieve passages for factoid-like questions. These passages correspond to the snippets/paragraphs that the search engine associates with each returned search result. The reason we use passages instead of the content of the corresponding web page/document is that a higher accuracy is usually attained with passages, since the full document will provide much more (incorrect) candidate answers, which are difficult to handle. We have conducted some informal experiments on this topic, where we compared the use of paragraphs with the use of the full documents in answering 1000 factoid-like questions. The approach based on passages attained 266 correct answers, against the approach based on documents, where 288 where retrieved.

In what concerns the semi-structured sources, due to their encyclopaedic nature, Just.Ask uses them to answer non factoid-like questions that require longer answers. In particular, Wikipedia is utilized to answer DESCRIPTION:DEFINITION and HUMAN:DEFINITION questions. Also, for questions whose cardinality is greater than one – i.e., list questions that require more than one answer –, Wikipedia is also used. DBpedia is used in conjunction with Wikipedia. Briefly, Wikipedia is used to locate the article where the answer to a given question might be found,

[e] `http://lucene.apache.org/`.
[f] `http://www.wikipedia.org/`.

while DBpedia is used to extract the actual answer from the article in a structured manner, without having to access the full-text of the article's web page. We discuss this strategy in the following subsection.

4.2. *Query formulation*

The usage of a certain information source depends on the classification attributed to each question. This, the formulation of the query is made depending on the desired information source.

The most simple query formulation strategy is based on keywords, and consists in generating a query that comprises all the words in a given question, except the stopwords, question words and punctuation. For instance, given the question *When was Beethoven born?*, a query with the keywords `Beethoven born` would be generated. The keyword query formulator is applied to generate queries to use in search engines and, therefore, is it applied to every question whose category is associated with this information source – i.e., factoid-like questions.

Another query formulation strategy is based on the focus of the question, and it is used in a combined strategy that uses both Wikipedia & DBpedia. The general idea is to use Wikipedia's search to locate the title of the article where the answer to a given question might be found, and then use DBpedia to retrieve the *abstract*[g] of the article, which is returned as the answer. For that purpose, the focus query formulator builds the query uniquely with the focus of the question. As an example, consider the question "*Who was Afonso Henriques ?*". First, a query with the questions focus *Afonso Henriques* is generated and sent to Wikipedia's API. Second, the first result returned – *Afonso I of Portugal*, in this case – is transformed into a DBpedia resource – `http://dbpedia.org/resource/Afonso_I_of_Portugal`. At last, we create the SPARQL query to retrieve the abstract of the article from DBpedia.

5. Answer extraction in Just.Ask

The **Answer Extraction** component receives the interpreted question originated in the Question Interpretation component, as well as the relevant passages retrieved by the Passage Retrieval component. The answer extraction can be divided in two stages – **candidate answer extraction** and **answer selection**. For candidate answer extraction, we take advantage of the rich question type taxonomy utilized in this work to devise strategies for each particular question category or groups of question categories. For instance, for NUMERIC type questions, we employ an extensive set of regular expressions to extract candidate answers, whereas for HUMAN type questions, we use a machine learning-based named entity recognizer. In what regards the answer selection, our strategy is to first normalize candidate answers, aggregate answers by lexical equivalence, apply a clustering algorithm to group

[g]The abstract of a Wikipedia article roughly corresponds to the first paragraph of the article.

together similar answers and then filter out unwanted candidate clusters. Finally, since each resulting cluster is scored, the final answer is decided by *ballot* – i.e., the representative answer within the cluster with highest score is chosen.

The Answer Extraction component and its sub components are depicted in Figure 4.

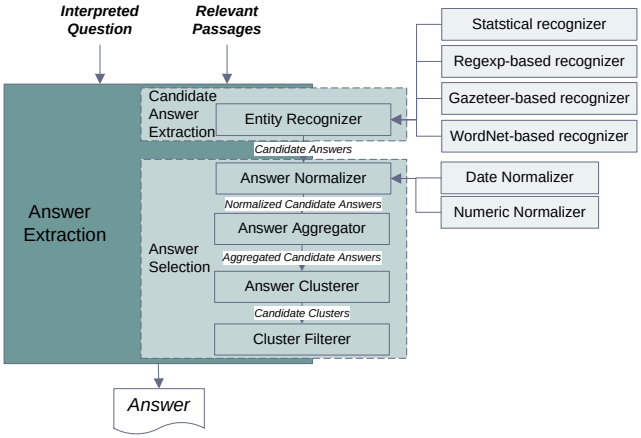

Fig. 4: Detailed view of the Answer Extraction component of Just.Ask.

5.1. *Candidate answer extraction*

In the following we describe the strategies developed to extract candidate answers from relevant passages (namely, Just.Ask uses Regular Expressions, Named Entities and WordNet-based recognizers and Gazetteers), as well as the question categories that each strategy is dedicated to.

An extensive set of regular expressions was created to extract candidate answers for questions of category NUMERIC. Regular expressions provide a very concise way to describe candidate answers for this type. For instance, to identify potential answers to NUMERIC:TEMPERATURE questions, the regular expression `/[0-9]+(K|R|°C|°F)/` could be used. Expressions were also created in a modular manner, with a *numerical basis* being shared among several categories. For example, both NUMERIC:DISTANCE and NUMERIC:TEMPERATURE share the same numerical basis, with the only difference being in the units that follow the numbers – linear measures and temperature units, respectively. Moreover, we developed a set of regular expressions that are made specific to each question, since they are built with the question focus. For instance, given *What does NEO stand for?*, we dynamically create a group of expressions in order to match answers to that question, such as `/.* +(NEO)/` or `/.* +, NEO,/`[h]. Currently, these regular expressions exist uniquely

[h]It is worth mentioning that the regular expressions utilized in this work are far more complex

for questions that belong to categories ABBREVIATION and NUMERIC:COUNT, but can be further extended. Each regular expression is paired with a numeric score, which is assigned to every candidate answer that is matched and extracted by it.

Although regular expressions are a very powerful tool, they can become cumbersome to use when what we are trying to search for is not in a *rigid* format. For instance, some of the questions require particular names as candidate answers – e.g., HUMAN:INDIVIDUAL questions call for person names –, which can occur in many different formats, and are therefore difficult to express using regular expressions. Moreover, some of these names can refer to different entities, depending on the context in which they occur, thus aggravating the problem. An example of this situation is the name *Washington*, which can either refer to a city, a state, or a person. To cope with the above problems, we used a machine learning-based named entity recognizer, which is able to automatically learn a model to extract entities, based on a set of annotated examples. In particular, we employed Stanford's Conditional Random Field-based named entity recognizer [34], which is able to recognize four entity types: PERSON, LOCATION, ORGANIZATION, and MISCELLANEOUS. The latter serves as a container for named entities that do not fit in the first three categories, such as book and song titles.

So far, we have considered numerical answers – for which regular expressions are a good fit –, and entity-based answers – dealt with by a machine learning-based named entity recognizer. There is, however, another type of answer that we consider in this work, and which does not fall in either of the above categories – *type of* questions. Consider the ENTITY:ANIMAL question "*Which **animal** is the fastest ?*". For this question, the answer is not a particular instance of an animal, but rather a *type of* animal – cheetah. These answers are very difficult to extract from natural language text, as they can be easily confused with other nouns that are present in relevant passages. Therefore, we suggest a new approach for extracting answers for *type of* questions, using WordNet's hyponymy relations. We exploit the fact that candidate answers for these questions are often hyponyms of the question's headword to construct a dictionary in *run time*[i] with the entire hyponym tree of the headword. The dictionary is then used by an exact dictionary matcher algorithm to extract candidate answers. For this work, LingPipe's implementation of the Aho-Corasick [35] algorithm was used. Also, as a corollary of this strategy, particular instances of a given word can also be extracted, if they exist in WordNet. For example, in the questions "*What is the largest **planet** in the Solar System ?*" and "*What is the world's best selling **cookie** ?*", both *Jupiter* (⇒ *planet*) and *Oreo* (⇒ *cookie*) are extracted. This algorithm is utilised for every question that pertains to the ENTITY category, with the exception of ENTITY:EVENT, ENTITY:LETTER,

than the expressions presented in the examples we have provided so far and take into consideration a wide range of formats and numeric units.

[i]We use the term *run time* to refer to the fact that the dictionaries are not constructed *a priori*, but rather when they are needed, in run time.

ENTITY:TERM, and ENTITY:WORD. Moreover, it is also used for the categories HUMAN:TITLE, LOCATION:MOUNTAIN, and LOCATION:OTHER.

Finally, certain question categories, such as LOCATION:COUNTRY, have a very limited set of possible answers – names of all the countries in the world, in this case. For these situations, a gazetteer[j] can help [36] to accurately extract candidate answers, as it can be used to assure that only candidate answers of the expected type are extracted. We used a gazetteer for both LOCATION:COUNTRY and LOCATION:CITY categories. The gazetteers are utilized in a similar way as the exact dictionary matcher described previously in this section, with the difference being in the fact that gazetteers are not constructed in *run time*, but they already exist when the system starts up.

As a final note, we should mention that the answer extraction strategies are applied in parallel, using multiple threads (one thread per passage), in order to maximize the performance of the system.

5.2. *Answer selection*

After candidate answers have been extracted, the last step is to choose the final answer to be returned. Four tasks can be performed before this decision (**normalization**, **aggregation**, **clustering** and **filtering**), as in Just.Ask they are optional and can be easily activated or deactivated.

5.2.1. *Normalization*

We start by normalizing candidate answers that belong to categories NUMERIC:COUNT and NUMERIC:DATE. In these cases, we attempt to diminish the variation of the answers by reducing them to a canonical representation. Since our recognizer is able to extract entities written with numeric and alphabetic characters (and both), this representation allows comparisons between answers: for instance, *one thousand* and *1000* are both reduced to *1000.0*.

5.2.2. *Aggregation*

After being normalized, candidate answers are aggregated by lexical equivalence. The goal is to reduce the number of candidate answers by merging those that are lexicographically equal (insensitive case) into a single candidate answer. The score of the new answer is the sum of the scores[k] of all answers it comprises.

[j] A gazetteer is a geographical dictionary, typically used to identify places. However, we use the term in a more broader sense to refer to a dictionary of any type of entities.

[k] With the exception of candidate answers that were extracted using regular expressions, every other candidate answer has a score of 1.0. Being so, the score of new answer boils down to the number of answers it aggregates, in most scenarios.

5.2.3. *Clustering*

Once equal candidate answers have been aggregated, we perform a clustering step. For that purpose, it is required the definition of a distance measure, which determines how the similarity of two candidate answers is calculated. In Just.Ask, we have the possibility to chose from the *overlap distance* and the *Levenshtein distance* [37] normalized to the maximum length of the two answers being compared (notice, however, that other measures can be easily integrated in Just.Ask).

The Levenshtein distance measures the least number of edit operations to transform one string into another, and the overlap distance is defined as:

$$\text{overlap}(X, Y) = 1 - \frac{|X \cap Y|}{\min(|X|, |Y|)}\,, \tag{1}$$

where $|X \cap Y|$ is the number of tokens shared by candidate answers X and Y, and $\min(|X|, |Y|)$ is the size of the smallest candidate answer being compared. This metric returns a value of 0.0 for candidate answers that are either equal or one is contained in the other (without taking into account the order of the tokens).

In either cases, the lower the distance, the similar the strings are. The chosen distance is used in conjunction with a standard single-link agglomerative clustering algorithm, which works as follows. Initially, every candidate answer starts in its own cluster. Then, at each step, the two closest clusters, up to a specified threshold distance, are merged. The distance between two clusters is considered to be the minimum of the distances between any members of the clusters, as opposed to complete-link clustering, which uses the maximum.

To illustrate the clustering algorithm at work, used in conjunction with the overlap distance with a threshold of 0.0, consider the following set of candidate answers: {John Kennedy, Kennedy, John F. Kennedy, John McCarthy}. In the first step, *John Kennedy* and *Kennedy* are merged together. In the second step, *John F. Kennedy* is merged with the resulting cluster from the previous step. Finally, since the minimum distance from *John McCarthy* to the cluster {John Kennedy, Kennedy, John F. Kennedy} is 0.5, and this value is greater than the threshold, the algorithm halts.

In addition, the answer representative of each cluster is defined as the most informative answer. Just.Ask uses a set heuristics to choose the representative answer among the candidate answers in the cluster. First, it uses the score of the answer (recall that, if the answers have been aggregated, their score is the sum of the scores of the answers it aggregates). In case of a tie, the system uses the most informative answer, assumed as the longest answer within each cluster. For instance, in the cluster {John Kennedy, Kennedy, John F. Kennedy}, *John F. Kennedy* is selected as representative answer of this cluster. Again, in the case of a tie, Just.ask uses the alphabetical order of the answers.

Moreover, a score is assigned to each cluster, which is simply the sum of the scores of all candidate answers within it.

5.2.4. *Filtering*

After the clusters of candidates have been built, and in order to remove undesired answers, we apply a simple filter to our clusters. If any of the answers present in any of the clusters is contained in the original question, than the whole cluster is discarded. To understand the importance of this filter, consider the question *Who assassinated John F. Kennedy?*, classified as HUMAN:INDIVIDUAL. For this question, *John F. Kennedy* and *John Kennedy* are extracted as candidate answers, since both answers match the named entity type associated with the question's category (PERSON), and, due to their similarity, they are clustered together. However, it is clear that none is the answer that is sought. Moreover, since the *John F. Kennedy* answer appears in almost every passage, as the formulated query itself contains *John F. Kennedy*, this will result in a very high score for it. Thus, in order to prevent this unwanted answer to be returned, the entire cluster is discarded.

5.2.5. *Selection*

Finally, after these intermediate steps, the representative answer of the cluster with highest score is returned. Furthermore, in case two clusters happen to have the same score, the tie is settled by returning the answer of the cluster with highest search rank – i.e., the cluster whose answers were in the first results returned by the information source.

6. Experimental setup

6.1. *Evaluation measures*

In order to assess the performance of Just.Ask and its components, we made use of several measures that have been proposed and extensively reported in the literature, namely for the evaluation of QA systems [38]. The measures used in the evaluation of Just.Ask are the following:

Accuracy, defined as the proportion of questions answered correctly:

$$Accuracy = \frac{\#Correctly\ answered\ questions}{\#Questions\ in\ the\ test\ corpus}. \tag{2}$$

Precision, defined as the proportion of questions answered correctly in the answered questions:

$$Precision = \frac{\#Correctly\ answered\ questions}{\#Answered\ questions}. \tag{3}$$

Recall, defined as the proportion of questions answered:

$$Recall = \frac{\#Answered\ questions}{\#Questions\ in\ the\ test\ corpus}. \tag{4}$$

***F*-measure**, used to combine the above two measures into a single metric, and it is defined as a weighted harmonic mean of precision and recall:

$$F_\beta = \frac{(\beta^2 + 1) \times Precision \times Recall}{\beta^2 \times Precision + Recall}\,, \tag{5}$$

where the parameter β is used to adjust the weights that are given to each measure. A value of $\beta > 1$ favours recall, whilst a value of $\beta < 1$ favors precision. When precision and recall are equally weighted (i.e., $\beta = 1$), the measure usually goes by the name of F_1.

Mean Reciprocal Rank (MRR), used to evaluate systems that return ranked lists of items to a query. The reciprocal **rank** of an individual query is defined to be the multiplicative inverse of the rank of the first relevant item, or zero if no relevant items are returned. Thus, the mean reciprocal rank is the average of the reciprocal rank of every quey in a test corpus. More formally, for a test collection of N queries, the MRR is given by:

$$MRR = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{rank_i}\,. \tag{6}$$

6.2. *Evaluation corpora*

6.2.1. *GoldWebQA*

The Web is constantly changing and, due to this reason, results attained by a Web QA system might differ in every run. In addition, questions that were plausible during a certain time period, can make no sense some time after. For instance, the question *How old is Deng Xiaoping?*, from the Multisix corpus [2], makes no sense anymore, since Deng Xiaoping died in 1997. Moreover, some question/answer pairs become obsolete through time. As an example, the answer to the question *Who is the Japanese Prime Minister?* was, in 1994, *Morihiro Hosokawa* and now is *Noda Yoshihiko*. Furthermore, each correct answer can be stated in multiple forms, posing problems in a QA system evaluation if all the possible reformulations of an answer are not present in the reference. For instance, if a list of correct answers contains uniquely the answer *"March 21, 1961"* to the question *When was Lothar Matthaeus born?*, the answers *"March 21st 1961"*, *"Tuesday, March 21, 1961"*, *"in March of 1961"*, *"21-Mar-1961"*, *"21st March 1961"* and "on 21st March 1961" will be considered incorrect. Hence, we decided to build a gold corpus – the GoldWebQA – that tries to puzzle out all these problems.

The GoldWebQA is constituted by 200 questions, a set of snippets retrieved from the Web containing possible answers to these questions, all the correct answers occurring in the snippets independently of the way they are stated, and the category of the questions, according to Li and Roth's question type category. By the time when the snippets were retrieved (representing a snapshot of the Web), all the questions were plausible and valid.

The set of questions from the GoldWebQA is an updated and extended version of the Multisix corpus [2], composed also by 200 English questions and the respective correct answers, manually collected from the Los Angeles Times corpus of the year of 1994, that was initially built for the cross-language tasks at CLEF QA-2003. Questions from the GoldWebQA were obtained as follows:

(1) Each question from Multisix was manually validated, to eliminate questions that no longer make sense. With this, 12 questions were removed from the corpus.
(2) Each answer from Multisix was manually updated, as there were many obsolete answers.
(3) The first 12 questions belonging to categories DESCRIPTION:DEFINITION and HUMAN:DEFINITION from the UIUC dataset (see Section 7.1 for details) were added to the corpus, in order to deal with the lack of questions from these categories in Multisix (non factoid-like questions).
(4) The 200 questions were manually classified according to Li and Roth's question type taxonomy.

The number of questions per category and question word in the GoldWebQA corpus is presented in Tables 1 and 2.

Table 1: Number of questions per category in the GoldWebQA corpus.

Category	# Quest.		Category	# Quest.	
DESCRIPTION:DEFINITION	6		LOCATION:CITY	8	
DESCRIPTION:DESCRIPTION	2	**9**	LOCATION:COUNTRY	4	
DESCRIPTION:MANNER	1		LOCATION:MOUNTAIN	1	**39**
ENTITY:ANIMAL	1		LOCATION:OTHER	25	
ENTITY:CREATIVE	10		LOCATION:STATE	1	
ENTITY:MEDICINE	2		NUMERIC:COUNT	24	
ENTITY:OTHER	1	**21**	NUMERIC:DATE	30	
ENTITY:PLANT	1		NUMERIC:DISTANCE	2	**63**
ENTITY:SUBSTANCE	3		NUMERIC:MONEY	2	
ENTITY:TERM	3		NUMERIC:PERCENT	1	
HUMAN:DESCRIPTION	6		NUMERIC:PERIOD	4	
HUMAN:GROUP	21	**64**	ABBREVIATION:ABBREVIATION	1	**4**
HUMAN:INDIVIDUAL	37		ABBREVIATION:EXPANSION	3	

Table 2: Number of questions per question word in the GoldWebQA corpus.

Question word	Who	When	Where	What	Which	How	Name	Other
# Questions	36	25	22	50	8	29	8	22

In what concerns the retrieved snippets, they allow to truly simulate the functioning of an online web question answering system, that resorts to a web search engine. Moreover, besides protecting against the changes occurring in the Web, their

use has several advantages when compared to the use of the *real* Web, namely: it reduces the latency time of the system when waiting for an external component; and, it allows benchmarking of systems that are, by nature, very difficult to evaluate, in particular in the answer extraction/selection steps.

In order to build the set of snippets plus the reference:

(1) From each question of the GoldWebQA, we built keyword-based queries (as described in Section 4.2), and collected 64 snippets using the search engines Google and Yahoo.
(2) We manually updated the GoldWebQA by adding variations of each correct answer that were found in the snippets to the existing correct answers.

To mention some statistics of this test collection, there is an average of 3.89 correct answers per question with standard deviation of 5.82. The top 6 questions with higher number of correct answers (from 21 to 50) start with the question word *Name*, like *Name a German philosopher*.

6.2.2. *TREC-QA_2002/2007*

The second evaluation is made with a corpus of factoid-like questions and their correct answers, built from the freely available data of the TREC QA tracks, years 2002 to 2007, where the answers were given by the competing systems and judged as correct by human assessors. From the original corpus, we collected the questions whose answers can be found in the New York Times (NYT) and discarded those with pronominal anaphora and ellipsis. The resulting corpus contains 1000 questions and their respective answers. Lucene[1] was used to index and retrieve the paragraphs from the NYT from 1987 to 2007.

7. Evaluation – JustAsk@GoldWebQA

In this section, we present a detailed evaluation of Just.Ask using the GoldWebQA corpus.

7.1. *Question interpretation evaluation*

In what concerns the Question Interpretation component, the evaluation is focused on the question classifier, due to its vital importance for the QA task.

The classifier of Just.Ask was tested using the GoldWebQA corpus where it was able to correctly classify 185 questions, resulting in an accuracy of 92.5% for the fine-grained categories. The 15 questions that were not correctly classified belonged to categories DESCRIPTION (3), ENTITY (4), HUMAN (4), NUMERIC (1) and LOCATION (3). From the erroneously classified questions that should have been classified with the coarse-grained category HUMAN, all belonged to the fine-grained category

[1] `http://lucene.apache.org`

GROUP (for instance, the question *Which fast-food chain cut the price of the Big Mac hamburger* was classified as ENTITY:FOOD, instead of HUMAN:GROUP). This is not a surprise as this type of questions are known to be particularly difficult to classify.

Moreover, for benchmarking purposes, we have previously carried out an intrinsic evaluation of Just.Ask question classifier [1] on the UIUC corpus, the publicly available data set of the Cognitive Computing Group at University of Illinois at Urbana-Champaign[m], which consists of a training corpus of nearly 5500 questions and a test corpus with 500 questions, each question labeled according to Li and Roth's taxonomy for question classification. The question classifier of Just.Ask uses the `LIBSVM` [39] implementation of a SVM classifier with a linear kernel, trained with all 5500 questions from the referred training corpus, using the one-versus-all multi-class strategy.

Table 3: Comparison of accuracy results attained by this work, against other results reported in the literature that use the same train and test datasets.

	Category granularity	
Author	Coarse	Fine
This work	**95.0%**	**90.4%**
[12]	91.0%	84.2%
[40]	90.0%	80.2%
[41]	93.4%	86.2%
[14]	91.8%	86.6%
[13]	94.0%	-
[15]	93.6%	89.2%

Since we made use of the same corpora, either for training and testing, we can compare Just.Ask question classifier performance with other works. The attained results are summarized in Table 3, and discussed in detail in [1], where we show that by training a machine learning classifier with unigrams and the information manipulated by our rule-based classifier, higher accuracy is achieved for coarse- and fine-grained categories than the ones mentioned in state of the art literature.

Depending on the category of the posed question, Just.Ask uses distinct information sources in the Passage Retrieval component and adopt different strategies in the Answer Extraction component. Therefore, in the following we split the evaluation into two major subsections: the first is dedicated the results achieved for non factoid-like questions (belonging to categories DESCRIPTION:DEFINITION and HUMAN:DESCRIPTION); the second is focused on the results achieved for the factoid-like questions.

[m] `http://l2r.cs.uiuc.edu/~cogcomp/Data/QA/QC/`.

7.2. *Non factoid-like questions*

Regarding the non factoid-like questions, Just.Ask correctly answered 9 of the 12 questions categorized as DESCRIPTION:DEFINITION or HUMAN:DESCRIPTION.

The system was not able to find the correct answer in DBPedia to two DESCRIPTION:DEFINITION questions (*What are liver enzymes?* and *What is the nature of learning?*) and to one HUMAN:DESCRIPTION question (*Who is Coronado?*). In the formers, Just.Ask returned a wrong answer; in the latter, Just.Ask did not return any answer.

7.3. *Factoid-like questions*

7.3.1. *Passage retrieval evaluation*

Regarding the Passage Retrieval component, the goal was to study the influence of different settings to the retrieval of passages. Specifically, we wanted to understand if there was any significant difference in using different search engines and if there was any significant difference in dealing with a different number of retrieved passages.

Being so, in several runs of Just.Ask, we varied different parameters in the Passage Retrieval component, namely:

- the used search engine: either Google or Yahoo.
- the number of passages retrieved from the search engine: 8, 32 and 64.

With this, we could analyze the impact of each parameter in the performance of the Passage Retrieval component, particularly in the number of questions for which the retrieved passages contain at least one correct answer. For simplicity reasons, we will denote these passages as *positive passages*.

Table 4 present, for different number of retrieved passages, the number of factoid-like questions for which there is at least one positive passage (# $\text{Questions}_{1+PosPassages}$), the mean reciprocal rank of the first positive passage for all factoid-like questions (MRR_{AllQ})[n] and the mean reciprocal rank of the first positive passage only for the factoid-like questions for which at least one positive passage exists ($\text{MRR}_{QPosPassages}$).

The first consideration after the analysis of the table is that the passage retrieval results when using Google surpass the results attained when we use Yahoo.

Regardless of the search engine, the number of questions for which there is at least one positive passage increases as the number of retrieved passages also increase, but this amount does not pass the barrier of 84% (in the total number of factoid-like questions). Also, if all questions are considered, the increase in the number of passages retrieved by the search engine leads to an increase of the MRR of the first positive passage, even if always lower than 0.5. However, if only the questions with positive passages are considered, the MRR values decrease, indicating that, in

[n] Recall that there are 188 factoid-like questions in the GoldWebQA corpus.

Table 4: Passage retrieval intrinsic evaluation, while varying the number of passages retrieved from the web search engine.

Google			
	# Questions$_{1+PosPassages}$	MRR$_{AllQ}$	MRR$_{QPosPassages}$
8	129 (68.6%)	0.43	0.62
32	152 (80.9%)	0.44	0.54
64	158 (84.0%)	0.44	0.52
Yahoo			
	# Questions$_{1+PosPassages}$	MRR$_{AllQ}$	MRR$_{QPosPassages}$
8	117 (62.2%)	0.34	0.55
32	146 (77.6%)	0.36	0.46
64	153 (81.4%)	0.36	0.44

average, the first positive passage is encountered in the tail of the set of passages. Thus, the results attained suggest that, for some cases, the first correct answer that solves a question does not appear in the top first results retrieved by search engines.

We have also inspected the questions for which neither of the search engines retrieved a positive passage. An example of such occurs, for instance, in the question *Which Elvis Presley's cover did Gilmore sing?*, where none of retrieved passages contain any of its correct answers. From the 30 questions with no retrieved positive passages when using Google and the 35 questions with no retrieved positive passages when using Yahoo, 19 overlap.

The results achieved lead us to two main considerations. On the one hand, the usage of both search engines in parallel could boost the results of the accuracy of the Passage Retrieval component to the mark of 90% (169 positive passages retrieved for 188 queries). Indeed, it agrees with the principle behind data redundancy in which "more is better", also supported by other works in the literature, like [24]. On the other hand, the queries to 11 different questions could not retrieve any positive passage, regardless of the engine in use, which suggests an open research line when it comes to the understanding of questions and their translation into queries.

7.3.2. *Answer extraction evaluation*

The evaluation of the Answer Extraction component of Just.Ask is important to understand if the system successfully extracts and selects the correct final answer.

Here we particularly analyze its sub-components: the Candidate Answer Extraction (CAE) and the Answer Selection (AS). We use the setting in which the Passage Retrieval component of Just.Ask yielded the best results, that is, when 64 passages were retrieved from Google. Moreover, to calculate the distance between candidate answers we used the normalized Levenshtein distance, with a threshold of 0.2.

Table 5 presents the number of questions for which a certain amount of answers was extracted. For those questions, the table shows the amount in which the CAE sub-component was successful, and the amount in which the AS sub-component was

successful. We consider the CAE stage as being successful when it extracts at least one correct answer. By the same token, the AS stage is successful when a correct answer is selected to be in the list of 3 answers that the system consider as final.

Table 5: Answer extraction intrinsic evaluation, using 64 passages from the search engine Google.

	Google – 64 passages				
# Extracted Answers	Questions (#)	CAE success	AS success	CAE acc.	AS acc.
0	10	0	–	0.00%	–
1 to 10	12	4	4	33.33%	100.00%
11 to 20	10	7	5	70.00%	71.43%
21 to 40	23	16	12	69.57%	75.00%
41 to 60	40	34	25	85.00%	73.52%
61 to 100	32	29	26	90.63%	89.66%
> 100	31	27	21	87.10%	77.78%
All	158	117	93	74.05%	79.49%

Several considerations can be made from the analysis of the given table. When the number of extracted candidate answers is low (between 1 and 10), if it happens to contain the correct answer (that is, the CAE stage is successful), then the answer selection stage chooses it as final answer. When the number is higher, some questions for which a correct answer exists in the passages are being filtered out in both sub-components of the Answer Extraction component. However, we can not discriminate any tendency of results to improve or deteriorate with the increase of the number of extracted answers.

For comparison purposes, Table 6 shows the results achieved by the Answer Extraction component when 8 and 32 passages from Google are used.

Table 6: Answer extraction intrinsic evaluation, using Google and two different amounts of retrieved passages.

	Google – 8 passages			Google – 32 passages		
# Extracted Answers	Questions (#)	CAE success	AS success	Questions (#)	CAE success	AS success
0	16	0	–	12	0	–
1 to 10	73	59	51	18	8	8
11 to 20	34	29	23	21	14	10
21 to 40	5	4	4	57	52	41
41 to 60	1	1	1	26	22	18
61 to 100	0	–	–	14	11	10
> 100	0	–	–	4	4	3
All	129	93	79	152	111	90

As expected, the number of candidate answers per question decreases when the number of passages available also diminish. Achieved results show a compromise

between the number of passages from where to extract candidate answers and the accuracy of the Answer Extraction component. Indeed, if the number of passages is too low, there is the possibility that the passages do not contain true positives (correct answers) in sufficient amount than can lead to their selection as the final answer. If the number of passages is too high, there is a chance that the system will extract too many false positives (wrong answers), leading to mistakes in the selection of the final answer.

Regarding the influence of the decrease of the number of passages on the performance of each of the stages, there is an increase of 2% on the accuracy of the candidate answer extraction stage (from 72.09% when using 64 passages to 74.05% when using 8), and a decrease of 5% on the accuracy of the answer selection stage (from 84.95% when using 64 passages to 79.79% when using 8).

In Table 7 we present the results of intrinsic evaluation of the Answer Extraction component of Just.Ask according to the question category. We show the total number of questions existing for each category and, for the different amounts of retrieved passages from Google, the number of questions that resulted in the extraction of at least one candidate answer.

Table 7: Answer extraction intrinsic evaluation, for different amounts of retrieved passages, according to the question category.

		Google – 8 passages			Google – 32 passages			Google – 64 passages		
	Total	Q.	CAE	AS	Q.	CAE	AS	Q.	CAE	AS
ABB	4	3	1	1	3	2	2	4	3	3
DES	3	2	0	–	2	0	–	2	0	–
ENT	21	15	3	3	16	3	2	17	5	2
HUM	58	42	35	32	50	41	33	51	42	35
LOC	39	27	24	21	34	28	25	35	29	24
NUM	63	40	30	22	47	37	28	49	38	29
	188	129	93	79	152	111	90	158	117	93

This table shows the categories that had higher impact on the results achieved by the CAE and AS stages of Just.Ask. For instance, when it comes to the coarse-grained category ENTITY, the system only extracts a correct answer for five questions in the 21 possible. We notice also that increasing the number of passages does not always lead to better results. For example, for the category ENTITY, the best absolute results are achieved when only 8 passages are used. For categories HUMAN and LOCATION, there is a drop in the performance of the Answer Extraction component (measured by the number of questions that returned a correct answer divided by the number of questions for which a positive passage exists).

The achieved results clearly evince the categories in which more efforts should be invested. For instance, the main improvement for the category ENTITY should be in the extraction of candidate answers, while for categories HUMAN and NUMERIC it should rather be in the selection of the final answer.

7.3.3. *Overall evaluation of the factoid-like questions*

The best result of Just.Ask was achieved when we used the first 64 passages from Google. Table 8 presents the system's overall results. Since Just.Ask returns a list with 3 possible final answers to a given question, the accuracy, precision, recall and MRR are measured and shown for the top 3 answers returned by the system. In this table, we also present the accuracy results for the top one answer returned (Accuracy@1).

Table 8: Best results achieved in the evaluation Just.Ask@GoldWebQA.

# Questions		Correct	Wrong	No Answer
188		93	77	18
Accuracy	Precision	Recall	MRR	Accuracy@1
49.5%	54.7%	90.4%	0.40	31.9%

In a total of 188 questions, Just.Ask gave answers to 170. From these, 93 were correct and 77 wrong. Being so, the system attained an accuracy of 49.5%. Moreover, the results in Accuracy@1 show that, from the 93 correctly answered questions, the system was able to push the correct answer to the first position of the list in 60 questions.

Detailed results Table 9 details the achieved results according to the question category. The result that first pops out is that for the category DESCRIPTION the system had an accuracy of 0.0%. The second worst result is that of category ENTITY, for which the system failed to answer 13 questions and skipped 6, from a total of 21. For instance, the system did not gave any answer to the question *What is the Chicken Boy sculpture made of?* and gave the wrong answer to *What flower was named after the first lady Hillary Rodham Clinton?*: *"orchid"* instead of *"tulip"*.

Table 9: Just.Ask results according to the different question categories.

Category	Correct	Incorrect	No answer	Accuracy
ABBREVIATION	3	1	0	75.00%
DESCRIPTION	0	0	3	0.00%
ENTITY	2	13	6	9.50%
HUMAN	35	20	3	60.34%
LOCATION	24	15	0	61.54%
NUMERIC	29	28	6	46.03%
Total	93	77	18	49.47%

In Table 10 we show the system's results depending on the question word. Whereas Just.Ask demonstrates a good performance in questions that typically involve the name or definition of a person (question word "Who"), the worst results happen for questions that start with "How".

Table 10: Just.Ask results according to the different question words.

Question word	Correct	Incorrect	No answer	Accuracy
Who	19	11	0	63.33%
When	18	7	0	72.00%
Where	12	10	0	54.55%
What	17	15	12	38.64%
Which	4	3	1	50.00%
How	8	16	5	27.59%
Name	6	2	0	75.00%
Other	9	13	0	40.91%
Total	93	77	18	49.47%

Impact of the components Just.Ask follows the typical pipelined architecture. Thus, it is relevant to understand which components are failing, since this failure will reflect on the following components. Being so, we consider that the question interpretation fails when it does not attribute the correct category to a question, the passage retrieval fails if it does not retrieve at least one positive passage for a question, and the answer extraction fails when it does not extract and selects the correct answer for a question.

We analysed, for each of the 95 wrong and unanswered questions, which component is the first one failing. The results on Table 11 show the impact on the final answer of each component independently, and do not account for the failures in cascade that occur in the system due to its pipelined architecture.

Table 11: Number of times each component is the first to fail, avoiding the system to return the correct answer in 95 wrong or unanswered questions.

Component	# Questions
Question Interpretation	15
Passage Retrieval	24
Answer Extraction	56
	95

As it can be seen, there is still work to be done in Just.Ask components. The Question Interpretation was properly tuned, reflecting in a small number of failures due to this component. When it comes to the Passage Retrieval component, we believe that one way to improve our results may imply the recourse to query expansion. Regarding the Answer Extraction, it requires deeper improvements, and probably new strategies, as most of our failures originate in this component.

Indeed, it is important to realize which components contribute to the success (or not) of the final answer, since that, whenever a component fails, the system is not able to retrieve the correct answer to the posed question. In a total of 188 factoid-like questions, the Question Interpretation component succeeded in 173 (92.02% of the questions), the Passage Retrieval in 158 (84.04%), and the Answer Extraction

in 93 (49.47%).

8. Evaluation – JustAsk@TREC-QA_2002/2007

In this section, we present a detailed evaluation of Just.Ask using the TREC-QA_2002/2007 corpus.

8.1. *Overall evaluation*

To allow a straight comparison with the results achieved in the evaluation with the GoldWebQA, when submitting a query to Lucene, we asked for 64 paragraphs from the NYC. Table 12 presents the system's overall results. Again, as in the previous evaluation, since Just.Ask returns a list with 3 possible final answers to a given question, the accuracy, precision, recall and MRR are measured and shown for the top 3 answers returned by the system. In this table, we also present the accuracy results for the top one answer returned (Accuracy@1).

Table 12: Results achieved in the evaluation Just.Ask@TREC-QA_2002/2007.

# Questions		Correct	Wrong	No Answer
1000		266	582	152
Accuracy	**Precision**	**Recall**	**MRR**	**Accuracy@1**
26.6%	31.4%	84.8%	0.21	15.6%

In a total of 1000 factoid-like questions, Just.Ask gave answers to more than 84% (848). However, it was successful only in 31% of them. From the 266 successfully answered questions, in 156 the system was able to push the correct answer to the first position of the list of answers.

8.2. *Detailed results*

In Table 13 we show the system's results according to the different question words. In all question words, the accuracy of the system does not reach the 50%. Whereas the best results are achieved for questions starting with the word "Where" (LOCATION questions), the worst results were attained for questions starting with "Name", "How" and "What".

Comparing with the results achieved in the evaluation performed using the GoldWebQA corpus, the system's overall performance dropped significantly. One of the reasons for this situation is the fact that we are using as reference answers only the correct answers returned from the participating systems at the TREC competition (we discarded the inexact and unsupported answers). Moreover, the TREC-QA_2002/2007 corpus does not include the variations of the correct anwers, which can be particularly critical for questions belonging to the coarse category NUMERIC (for example, the question *When was NATO established?* has as answers *"1949"*,

Table 13: Just.Ask results according to the different question words.

Question word	Correct	Incorrect	No answer	Accuracy
Who	35	67	2	33.65%
When	36	73	3	32.14%
Where	26	29	2	45.61%
What	115	252	87	25.33%
Which	9	10	2	42.86%
How	20	99	49	11.90%
Name	0	2	0	0.00%
Other	25	50	7	30.49%
Total	266	582	152	26.60%

"April 4, 1949" and *"April 4 1949"*. An answer which is a variation of that date is evaluated as wrong). The number of correct answers per question is also smaller than that verified in the GoldWebQA corpus (an average of 2.63 and 3.89 correct answers per question in the TREC-QA_2002/2007 and the GoldWebQA corpora, respectively). This situation directly reflects in the results achieved by the Passage Retrieval and Answer Extraction components of Just.Ask, which we present in the following.

Regarding the Passage Retrieval component, Table 14 shows the number of positive passages returned by Lucene, where it can be seen that (at least) one correct answer was present in the passages to only 662 of the total 1000 question.

Table 14: Passage retrieval intrinsic evaluation.

	Lucene		
	# Questions$_{1+PosPassages}$	MRR$_{AllQ}$	MRR$_{QPosPassages}$
64	662 (66.2%)	0.16	0.25

From the total amount of questions for which Lucene retrieved at least one positive passage, the Answer Extraction component was not able to extract correct answers in almost 38% of the questions (see Table 5). The performance of the Answer Selection sub-component remains rather stable, regardless of the number of extracted candidate answers.

In the evaluation using the TREC-QA_2002/2007 corpus, Just.Ask attained an overall accuracy of 26.6%. A detailed analysis of the system's revealed an accuracy of each component (and sub-component) of around 60%[o], showing the effects of the system's architecture to its overall results: errors occurring in the first components directly reflect in the components that follow in the pipeline.

[o]However, assuming an accuracy of the Question Interpretation component of around 90%.

Table 15: Answer extraction intrinsic evaluation, using 64 passages from the search engine Lucene.

	Lucene – 64 passages				
# Extracted Answers	Questions (#)	CAE success	AS success	CAE acc.	AS acc.
0	65	0	–	0.00%	–
1 to 10	79	30	18	37.97%	60.00%
11 to 20	62	28	16	45.16%	57.14%
21 to 40	134	90	58	67.16%	64.44%
41 to 60	121	100	64	82.64%	64.00%
61 to 100	118	94	60	79.66%	63.83%
> 100	83	73	50	87.95%	68.49%
All	662	415	266	62.68%	64.10%

9. Conclusions and future work

In this paper we presented Just.Ask, a modular and extensible QA system. Just.Ask searches for the answer to a question using a multitude of state-of-the-art strategies and techniques, from machine-learning to others with their roots in NLP. Moreover, Just.Ask allows the use of several information sources.

Along with Just.Ask, we have built a test collection (GoldWebQA), which we also make available. We split the evaluation of Just.Ask in two parts: in the first we use the GoldWebQA corpus; in the second, we use a set of 1000 questions and their answers collected from the QA tracks of TREC (TREC-QA_2002/2007). The evaluation allowed us to identify the strong and weak points of Just.Ask. Some considerations that resulted from the evaluations are stated in the following:

- Just.Ask question classifier attains state-of-the-art results when evaluated against the UIUC corpus. When using the GoldWebQA, it reaches 92.5% of accuracy. This step is surely one of the basis of Just.Ask, since no question incorrectly classified is correctly answered.
- Regarding the Passage Retrieval component, results suggest that the increase on the number of passages does not always lead to better overall performance. Indeed, for some question categories, increasing from 8 to 32 passages compromises the relative accuracy of the Answer Extraction component. This is mostly due to the fact that many more wrong answers are involved in the answer selection stage. Another important conclusion is that a considerable number of questions resulted in the retrieval of no passages with a correct answer (around 15% in the evaluation with the GoldWebQA corpus and 40% with the TREC-QA_2002/2007 corpus). In this context, query expansion is a desirable research direction.
- The Answer Extraction component also requires improvements, since even when Just.Ask extracts the correct answer from the available relevant passages, often it is not able to select it from the set of candidate answers.
- In overall terms, considering the different categories, the questions that belong

to category ENTITY as well as questions that start with the word "How" need urgent improvements.

Just.Ask and the test collection are available for research purposes, and we believe that they can be used as a baseline for other groups in this line of research. Moreover, and since the results achieved in the Passage Retrieval component also require improvements, Just.Ask could also be useful for the purpose of evaluating IR systems: how can we tune the Passage Retrieval component in order to maximize the number of positive passages returned for a question, while avoiding to retrieve the passages that do not contain any correct answer?

As future work, besides the previously mentioned improvements, we intend to evaluate the adaptation of Just.Ask to other languages, as well as the applicability of the system in closed domains, such as cinema or art. Moreover, and following recent tendencies where the output of many (unavailable) QA systems are combined [42,43], we consider that Just.Ask can also be used in combination with such systems. Indeed, combining Just.Ask with the other available QA software – Ephyra [28] and Aranea [24] – will have the advantage of being able to integrate systems that are no longer black boxes. That is, instead of combining the output of QA systems, researchers will be able to compare their components and plug and play with them.

Appendix A. Parameterizing Just.Ask

Just.Ask is an open source Question-Answering system for the English language, freely available for the research community. It is entirely implemented in Java.

At the present moment, Just.Ask depends on Li and Roth's question type taxonomy. Therefore, the implemented strategies for passage retrieval and answer extraction can be made general or specific to a certain question category(ies).

The parametrization of Just.Ask is primarily done in an XML file. Among others, in this file one can parameterize the different components of Just.Ask, as well as the system general configuration:

General: the input file and the maximum number of final answers that the system can return.

```
<config>
  ...
  <qa>
    <questionsFile>REF200_24-11-2010.txt</questionsFile>
    <answers>3</answers>
  </qa>
  ...
</config>
```

Question Interpretation The classification model, the type of tokenizer and the grammar.

```
<config>
  ...
  <interpretation>
    <classification>
      <model>resources/classification/english/models
           /L2F.QC.en.fine.SVM</model>
    </classification>
    <analysis>
      <tokenizerType>PTB</tokenizerType>
      <parserGrammar>resources/parser/english
           /L2F.questionbank_modified.gr</parserGrammar>
    </analysis>
  </interpretation>
  ...
</config>
```

Passage Retrieval The engines in use, respective query formulators and maximum number of passages to be retrieved.

```
<config>
  ...
  <retrieval>
    <engines>
      <engine queryformulator="Question" passages="8">LocalCorpus</engine>
    </engines>
  </retrieval>
  ...
</config>
```

Answer Extraction The active strategies and the active steps in this component.

```
<config>
  ...
  <extraction>
    <strategies>
      <strategy active="true">lexicon</strategy>
      <strategy active="true">list</strategy>
      <strategy active="true">regex</strategy>
      <strategy active="true">statistical</strategy>
    </strategies>
    <normalize do="true"></normalize>
    <aggregate do="true"></aggregate>
    <cluster do="true"></cluster>
    <filter do="true"></filter>
  </extraction>
```

```
  ...
</config>
```

Appendix A.1. *Setting the Strategies to Passage Retrieval*

Setting a new strategy to passage retrieval in Just.Ask implies modifications in the XML configuration file (besides the inclusion of the class(es) that implement the strategy).

Adding a new engine

- An engine `engine_id` must be implemented in the package `l2f.retrieval.searcher` in the class <`engine_id`>`Seacher.java`.
- The class <`engine_id`>`Seacher.java` must extend the class `Searcher.java`[p] and override the method `call(Query quey, int resultsToRetrieve)`, which receives a query object and the number of results of be retrieved, and returns a set containing a set of search results.

Adding a new query formulator A query formulator `queryformulator_id` must be implemented in the package `l2f.retrieval.query.queryformulator` and implement the interface `QueryFormulator.java`[q]

The association between the query formulator and the engine is done in the XML configuration file. If the number of passages is not defined, a default value of 8 will be used. Regard that `engine_id` and `queryformulator_id` must be the ids of the engine and query formulator, as they appear in the name of the class that implement them (case sensitive).

```
<config>
  ...
  <retrieval>
    <engines>
      <engine queryformulator="queryformulator_id" passages="8">
              engine_id</engine>
    </engines>
  </retrieval>
  ...
</config>
```

If one desires to associate a search engine to a specific question category, this must be done programmatically in the class `SearcherFactory.java`[p]

[p] package `l2f.retrieval.searcher`
[q] package `l2f.retrieval.query`

Appendix A.2. *Setting the Strategies to Answer Extraction*

In Just.Ask, the object EntityRecognizer aggregates all the information related with the extraction of candidate answers from a sentence. The object that implements a specific strategy to answer extraction is a specific EntityRecognizer.

Setting a new strategy to answer extraction in Just.Ask implies modifications in three different files (besides the inclusion of the class(es) that implement the strategy):

- the XML configuration file, which contains reference to the answer extraction strategies active in the current session.
- `CandidateAnswerExtractor.java`[r], which keeps track of the list of available answer extraction strategies. When a new system session begins, it gives indication to the EntityRecognizer about which strategies are activated (as indicated in the configuration file).
- `EntityRecognizer.java`[s], which aggregates all the available recognizers in Just.Ask. Given an interpreted question and a sentence where the answers might be found, the Entity Recognizer is responsible for applying all the active recognizers to that sentence.

Adding (and activating) a new strategy In order to add a new strategy to answer extraction, the following steps are to be accomplished:

(1) Create the recognizer (in one or more Java classes) that implements the strategy;
(2) Indicate the existence of a new strategy by adding its identifier (`aestrategy_id`) to the enumeration `Strategy` in the class `CandidateAnswerExtractor.java`[r]
(3) Include the new recognizer in the class `EntityRecognizer.java`[s]. Associate the recognizer to the new strategy.
(4) In the XML configuration file, include and activate the strategy in the list of answer extraction strategies. Regard that `aestrategy_id` must be the same as that defined in the item 2 of this list.

```
<config>
  ...
  <extraction>
    ...
    <strategies>
      ...
      <strategy active="true">aestrategy_id</strategy>
    </strategies>
  </extraction>
```

[r]package `l2f.extraction.candidateanswerextraction`
[s]package `l2f.nlp.entityrecognizer`

```
</config>
```

Deactivating an existing strategy Disabling an existing strategy for answer extraction in Just.Ask boils down to changing the variable `active` to `false` in the XML configuration file.

References

1. J. Silva, L. Coheur, A. Mendes, and A. Wichert. From symbolic to sub-symbolic information in question classification. *Artificial Intelligence Review*, 2011.
2. Bernardo Magnini, Simone Romagnoli, Alessandro Vallin, Jesus Herrera, Anselmo Penas, Victor Peinado, Felisa Verdejo, Maarten de Rijke, and Ro Vallin. The multiple language question answering track at clef 2003. In *CLEF 2003. CLEF 2003 Workshop*. Springer-Verlag, 2003.
3. R. F. Simmons. Answering english questions by computer: a survey. *Commun. ACM*, 8(1):53–70, 1965.
4. Jr. Bert F. Green, Alice K. Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question-answerer. In *IRE-AIEE-ACM '61 (Western): Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, pages 219–224, New York, NY, USA, 1961. ACM.
5. W.A.Woods, R.M. Kaplan, and B.N. Webber. The lunar sciences natural language information system: Final report. Technical report, Bolt Beranek and Newman Inc., Cambridge, Massachussets, 1972.
6. L. Hirschman and R. Gaizauskas. Natural language question answering: the view from here. *Nat. Lang. Eng.*, 7(4):275–300, 2001.
7. I. Androutsopoulos, G.D. Ritchie, and P. Thanisch. Natural language interfaces to databases–an introduction. *Journal of Language Engineering*, 1(1):29–81, 1995.
8. Boris Katz. Using english for indexing and retrieving. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1988.
9. Boris Katz. Annotating the world wide web using natural language. In *Proceedings of the 5th RIAO Conference on Computer Assisted Information Searching on the Internet (RIAO '97)*, 1997.
10. Ellen M. Voorhees. The trec-8 question answering track report. In *In Proceedings of TREC-8*, pages 77–82, 1999.
11. Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition) (Prentice Hall Series in Artificial Intelligence)*. Prentice Hall, 2 edition, May 2008.
12. Xin Li and Dan Roth. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics*, pages 1–7, Morristown, NJ, USA, 2002. Association for Computational Linguistics.
13. Yan Pan, Yong Tang, Luxin Lin, and Yemin Luo. Question classification with semantic tree kernel. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 837–838, New York, NY, USA, 2008. ACM.
14. Phil Blunsom, Krystle Kocik, and James R. Curran. Question classification with log-linear models. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 615–616, New York, NY, USA, 2006. ACM.
15. Zhiheng Huang, Marcus Thint, and Zengchang Qin. Question classification using head words and their hypernyms. In *EMNLP*, pages 927–936, 2008.
16. B. Grau, A. Ligozat, I. Robba, A. Vilnat, and L. Laura Monceaux. FRASQUES:

A Question Answering system in the EQueR evaluation campaign. In *Language Resources and Evaluation Conference*, 2006.
17. Charles L. A. Clarke and Egidio L. Terra. Passage retrieval vs. document retrieval for factoid question answering. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 427–428, New York, NY, USA, 2003. ACM.
18. Christof Monz. *From Document Retrieval to Question Answering*. PhD thesis, University of Amsterdam, 2003.
19. Carlos Amaral, Adán Cassan, Helena Figueira, André Martins, Afonso Mendes, Pedro Mendes, Cláudia Pinto, and Daniel Vidal. Priberam's question answering system in qa@clef 2007. In *Advances in Multilingual and Multimodal Information Retrieval: 8th Workshop of the Cross-Language Evaluation Forum, CLEF 2007, Budapest, Hungary, September 19-21, 2007, Revised Selected Papers*, pages 364–371, Berlin, Heidelberg, 2008. Springer-Verlag.
20. Ana Mendes, Luisa Coheur, Nuno J. Mamede, Ricardo Daniel Ribeiro, David Martins de Matos, and Fernando Batista. Qa@l2f, first steps at qa@clef. In *Advances in Multilingual and Multimodal Information Retrieval: 8th Workshop of the Cross-Language Evaluation Forum, CLEF 2007, Budapest, Hungary, September 19-21, 2007, Revised Selected Papers*, volume 5152 of *Lecture Notes in Computer Science*. Springer-Verlag, September 2008.
21. D. Laurent, S. Nègre, and P. Sgula. Qristal, le qr à l'épreuve du public. *Traitement Automatique des Langues*, 46:1–32, 2005.
22. M. M. Soubbotin. Patterns of potential answer expressions as clues to the right answers. In *In Proceedings of the Tenth Text REtrieval Conference (TREC*, pages 293–302, 2001.
23. Jochen L. Leidner Michael Wiegand and Dietrich Klakow. Cost-sensitive learning in answer extraction. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, may 2008. European Language Resources Association (ELRA).
24. Jimmy Lin. An exploration of the principles underlying redundancy-based factoid question answering. *ACM Trans. Inf. Syst.*, 25(2):6, 2007.
25. Alessandro Moschitti and Silvia Quarteroni. Kernels on linguistic structures for answer extraction. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, HLT-Short '08, pages 113–116, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.
26. Huanyun Zong, Zhengtao Yu, Cunli Mao, Junjie Zou, and Jianyi Guo. Parameter learning for multi-factors of entity answer extracting. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on*, volume 5, pages 2478 –2482, aug. 2010.
27. Ana Cristina Mendes and Lusa Coheur. An approach for answer selection in question answering based on semantic relations. In *Twenty-second International Joint Conference on Artificial Intelligence*, Proceedings of the Twenty-Second International Joint Conference of Artificial Intelligence, pages 1852–1857. AAAI Press/International Joint Conferences on Artificial Int, July 2011.
28. Nico Schlaefer, Jeongwoo Ko, Justin Betteridge, Manas A. Pathak, Eric Nyberg, and Guido Sautter. Semantic extensions of the ephyra qa system for trec 2007. In *TREC*, 2007.
29. Slav Petrov and Dan Klein. Improved inference for unlexicalized parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the*

Association for Computational Linguistics; Proceedings of the Main Conference, pages 404–411, Rochester, New York, April 2007. Association for Computational Linguistics.
30. John Judge, Aoife Cahill, and Josef van Genabith. Questionbank: creating a corpus of parse-annotated questions. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 497–504, Morristown, NJ, USA, 2006. Association for Computational Linguistics.
31. Michael John Collins. *Head-driven statistical models for natural language parsing*. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, 1999. Supervisor-Marcus, Mitchell P.
32. C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.
33. Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *In 6th Int'l Semantic Web Conference, Busan, Korea*, pages 11–15. Springer, 2007.
34. Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370, Morristown, NJ, USA, 2005. Association for Computational Linguistics.
35. Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.
36. Lucian Vlad Lita, Warren A. Hunt, and Eric Nyberg. Resource analysis for question answering. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 18, Morristown, NJ, USA, 2004. Association for Computational Linguistics.
37. V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707–710, February 1966.
38. Ellen M. Voorhees. Overview of trec 2003. In *TREC*, pages 1–13, 2003.
39. Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.
40. Dell Zhang and Wee Sun Lee. Question classification using support vector machines. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 26–32, New York, NY, USA, 2003. ACM.
41. Vijay Krishnan, Sujatha Das, and Soumen Chakrabarti. Enhanced answer type inference from questions using sequential models. In *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 315–322, Morristown, NJ, USA, 2005. Association for Computational Linguistics.
42. Alberto Téllez-Valero, Manuel Montes y Gómez, Luis Villaseñor-Pineda, and Anselmo Peñas Padilla. Learning to select the correct answer in multi-stream question answering. *Information Processing & Management*, In Press, Corrected Proof:–, 2010.
43. Gracinda Carvalho, David de Matos, and Vitor Rocio. Improving idsay: A characterization of strengths and weaknesses in question answering systems for portuguese. In Thiago Pardo, Antnio Branco, Aldebaro Klautau, Renata Vieira, and Vera de Lima, editors, *Computational Processing of the Portuguese Language*, volume 6001 of *Lecture Notes in Computer Science*, pages 1–10. Springer Berlin / Heidelberg, 2010.