

Learning to Answer Biomedical Factoid & List Questions: OAQA at BioASQ 3B

Zi Yang, Niloy Gupta, Xiangyu Sun, Di Xu, Chi Zhang, and Eric Nyberg

Language Technologies Institute, School of Computer Science, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
{ziy, ehn}@cs.cmu.edu

Abstract. This paper describes the CMU OAQA system evaluated in the BioASQ 3B Question Answering track. We first present a three-layered architecture, and then describe the components integrated for exact answer generation and retrieval. Using over 400 factoid and list questions from past BioASQ 1B and 2B tasks as background knowledge, we focus on how to learn to answer questions using a gold standard dataset of question-answer pairs, using supervised models for answer type prediction and candidate answer scoring. On the three test sets where the system was evaluated (3, 4, and 5), the official evaluation results have shown that the system achieves an MRR of .1615, .5155, .2727 for factoid questions, and an F-measure of .0969, .3168, .1875 for list questions, respectively; five of these scores were the highest reported among all participating systems.

Keywords: biomedical question answering; learning to answer questions; type coercion; answer scoring; answer ranking

1 Introduction

A number of shared tasks have been organized to evaluate the performance of biomedical question answering (QA) systems. For example, TREC Genomics QA [4] focused on genomics questions for a subdomain of biomedical research, and made available a relatively small set of factoid and list questions (a total of 64 questions from two years). Recently, the CLEF QA4MRE task [11] organized a pilot track on multiple choice QA for questions related to Alzheimer’s disease. Compared to these shared tasks, the BioASQ challenge [13] covers a wider range of biomedical subdomains, and releases a larger topic set with much more detailed gold standard data, including relevant documents, snippets, concepts, triples and both exact and ideal answers.

This paper reports results for the last three batches of BioASQ phase 3B, and focuses on factoid and list QA for Phase B and Phase A. We describe both the system architecture and individual components. First, we adapted a leveraged a UIMA¹-based three-layered architecture that was previously developed for biomedical QA tasks (TREC Genomics-style questions [16] and CLEF QA4MRE-style questions [10]) for the BioASQ challenge; the architecture consists of a generic component layer (BaseQA), a biomedical component layer (BioQA) and a BioASQ-specific component

¹ <https://uima.apache.org/>

layer. Using the development set, we also investigated whether it is possible to design and train supervised models to answer factoid and list questions, without the use of manually-constructed rules or predefined templates. We utilized supervised models to merge answer scores obtained from various sources, a technique utilized by some systems in past years [14, 9], and to predict likely answer type(s) from among the 133 semantic types in the UMLS Semantic Network.

Our hypothesis, described in the OAQA technical report [2] and motivated by recent success in building and optimizing a TREC Genomics-style QA system, is that informatics challenges like BioASQ are best met through careful design of a flexible and extensible architecture, coupled with continuous, incremental experimentation and optimization over various combinations of existing state-of-the-art components, rather than relying on a single “magic” component or single component combination. We leveraged an existing framework [16], integrated commonly adopted components (e.g. MetaMap², ClearNLP³, etc.) and extracted features for statistical machine learning. Over the 70 days of intensive development between April 2 to Jun 10, our experiment database has recorded 717 experiments. Among 669 successful experiments, there were 167 executing the training pipeline (177.5 topics per run on average), 422 executing the testing pipeline (24.1 topics per run on average) and 80 “dummy” runs used to cache service results (284.5 topics per run on average). The official evaluation results indicate that the system achieves MRR scores of .1615, .5155, and .2727 for factoid questions, and F-measure score of .0969, .3168, and .1875 for list questions; five of these results are the highest scores reported among all participating systems. The architecture frameworks and most of the components are currently available as open-source downloads, and we are planning to release the remaining components that are used in the system as open source software in the near future.

The goal of this working note is to describe the overall architecture and the components that are necessary in order to rebuild the system and reproduce the results from the open-source software.

2 Architecture

The three-layered architecture uses the UIMA ECD/CSE framework⁴ [3, 16], which extends the UIMA framework with a YAML⁵-based language which supports formal, declarative descriptors for the space of system and component configurations to be explored during the optimization step. The CSE framework also provides evaluation APIs, experimental result persistence, and customized execution control flow to support automatic performance evaluation and optimization via scalable web services (UIMA-AS).

The first layer BaseQA⁶ is designed for domain-independent QA components, and includes the basic input/output definition of a QA pipeline, intermediate data objects (such as answer type, question type, relevant passages, relevant concepts,

² <http://metamap.nlm.nih.gov/>

³ <https://github.com/clir/clearnlp/>

⁴ <https://github.com/oaqa/cse-framework/>

⁵ <http://yaml.org/>

⁶ <https://github.com/oaqa/baseqa/>

etc.), QA evaluation components, and data processing components (e.g. LingPipe⁷ and Apache OpenNLP⁸ wrappers, Lucene⁹-based passage retrieval component, LibLinear¹⁰ wrapper, and models applicable to generic English questions). Although the BioASQ task focuses on the biomedical domain, it is the first shared task on QA to combine four types of questions and evaluate both exact and ideal answers along with other relevant elements (e.g. triples), so many aspects of the existing BaseQA framework were extended to accommodate BioASQ application development. We modified the intermediate object and input/output object definition (UIMA type system) according to the task requirements. For example, we added two new attributes `Begin/EndSection` to each `Passage` type, and changed the `Begin/EndPosition` attributes to `Begin/EndPositionInSection`. We also provided a BioASQ-compatible JSON format reader and writer at the BaseQA level, which we believe can be widely used in various QA tasks beyond BioASQ. We also implemented evaluation methods according to the specific BioASQ evaluation requirements.

In the second layer (BioQA), we implemented biomedical resources that can be used in any biomedical QA task (outside the context of BioASQ), including UMLS Terminology Services (UTS)¹¹-based synonym expansion component, a MetaMap annotation component, etc. For the components that are included in the BaseQA layer, we also created a descriptor for the component at the BioQA level by overriding the `model` value with a path to the specific model tailored for biomedical domain, where applicable. For example, the ClearNLP wrapper, which is provided at the BaseQA level with the default `general-en` model specified in the descriptor, has a new descriptor for the `bioinformatics-en` model, trained on the CRAFT treebank, defined at the BioQA level. Although the training and testing processes are performed on the BioASQ development set, the derived models can also be used for other biomedical questions, so we also place the models and training components in the BioQA layer.

A few BioASQ-specific components were integrated in the third design layer; for example, GoPubMed services are only hosted for the purpose of the BioASQ challenge. The introduction of this task-specific layer will facilitate easy replacement of proprietary and restricted components when we adapt the system to other biomedical QA tasks or deploy the system as a real-world application. The end-to-end training and testing pipelines are also defined in this layer. The test descriptor used for Batch 5 in Phase B is shown in Listings 1.1 to 1.3. Similar to the `resource-wrapper` providers which we introduced for the TREC Genomics QA task [16], we also created a caching layer, using Redis¹², for all outgoing GoPubMed service requests, along with a Java client for accessing either the official GoPubMed server or the caching server, specified by a `properties` file¹³, which helps to reduce the workload of the official server and reduce experiment run-time when multiple developers are evaluating their components.

⁷ <http://alias-i.com/lingpipe/index.html>

⁸ <https://opennlp.apache.org/>

⁹ <https://lucene.apache.org/>

¹⁰ <http://www.csie.ntu.edu.tw/~cjlin/liblinear/>

¹¹ <https://uts.nlm.nih.gov/home.html>

¹² <http://redis.io/>

¹³ <https://github.com/ziy/bioasq-gopubmed-client/>

3 Factoid & List Question Answering for Phase B

Factoid and list QA tasks have similar topic distributions and linguistic structures, and each exact answer also uses a similar language representation. Accordingly we designed two supervised models that are shared by both question types: *answer type prediction* (described in Sect. 3.1) and *candidate answer scoring* (described in Sect. 3.3), which allows us to best leverage the training data. In addition, we introduce approaches for *candidate answer generation* in Sect. 3.2. In comparison to factoid questions, list questions require the system to return a list of exact answers, of the same type, which requires an *answer pruning* component for list question answering only, which is described in Sect. 3.4. The overall pipeline diagram is illustrated in Fig. 1 in Appendix.

3.1 Question and Answer Type Prediction

Previous work has studied how to define rules to extract a *lexical answer type* (or LAT) from questions to predict the answer type, e.g. IBM’s Watson system [5]. Classification based approaches have been proposed to predict answer type from the question using syntactic and/or semantic features. Preparation of training data involves defining an answer type taxonomy manually or by leveraging existing ontologies (e.g. MUC), collecting training questions (e.g. TREC QA question set) and annotating gold standard answer type(s) [6, 8]. Weissenborn et al. [14] also define patterns for LAT extraction for BioASQ questions, and leverage the UMLS Semantic Network and map the LATs to the ontological hierarchy to obtain one of the UMLS semantic types as an “expected answer type”, which is used for type coercion checking. We took advantage of these ideas and further incorporated the Q/A pairs in the BioASQ data set for training a multi-class answer type classifier that predicts candidate answer type(s).

Answer Type Definition. We introduce two additional question types: CHOICE and QUANTITY in addition to the UMLS semantic types. CHOICE questions are those that have candidate answers expressed explicitly in the question, e.g. “Is Rheumatoid Arthritis more common in men or women?”. We treat CHOICE questions as a special case because the candidate answers can be directly extracted from the question, and no further answer type prediction is needed. Since there exist an unlimited number of quantitative values which cannot be all covered in the UMLS semantic network, we add the QUANTITY type to complement the existing q_{nco} (Quantitative Concept) type.

Answer Type Extraction. To identify the gold standard labels for the existing Q/A pairs used for training, we apply UTS to retrieve the semantic types for each gold standard exact answer, where we first use the `exact` search type, and if no results are returned, we further relax the search type to `words`. Since UTS may return more than one concept type for each input concept, and each training question may contain more than one gold standard answer variant (these may be synonyms or answer concepts for list questions), the gold standard answer type is assigned as the most frequent concept type. If multiple concept types have the same number of occurrences for all gold standard answer variants, we keep all of them as the gold standard labels for the question.

We identified 82 out of the 406 questions which do not have a single gold standard answer variant for which UTS can provide a semantic type. There are three major reasons for this phenomenon. First, some answer concepts are not included in the UMLS

Table 1. Answer Type Prediction Features

No.	Feature
1	the lemma form of each token
2	if the question begins with “do” or “be”
3	if the question contains a token “or”
4	if the question contains a quantity question phrase
5	the semantic type of each concept
6	a ⟨semantic type, dependency label⟩ pair, where we use the dependency label of the head token in the concept bearing phrase as the second element
7	also a ⟨semantic type, dependency label⟩ pair, where we use the dependency label of the head of the head token in the concept bearing phrase as the second element
8	the lemma form of the first child of the root in the parse tree that is a noun and has a dependency relation of <code>dep</code>

semantic network. For example, the question “Which histone marks are deposited by Set7?” has two answers: “H4K20 monomethylation” and “H3K4 monomethylation”, both of which cannot be mapped to a single semantic type. Second, some gold standard exact answers do not strictly follow the representation format. For example, the question “Which enzyme is deficient in Krabbe disease?” has a gold standard answer “Galactocerebrosidase is an enzyme that is deficient in . . .” In fact, “Galactocerebrosidase” alone should be the gold standard exact answer. Third, some questions (e.g. “Which is the most important prognosis sub-classification in Chronic Lymphocytic Leukemia?”, with a gold standard answer “The mutational status of the IGHV genes.”) have an answer which is not a simple biomedical entity, and thus cannot be mapped to a single concept type. Finally, we obtained gold standard labels for the 324 remaining questions.

Feature Extraction. We first apply the ClearNLP parser to annotate the tokens, part of speech tags, and dependency relations for the question (corresponding to Lines 21 – 26 of Listing 1.1 in the Appendix) . We use three approaches to identify the concept mentions in the question. We first use the MetaMap service to identify the concepts and use UTS to retrieve variant names for each concept (Lines 27 – 29). Only the first concept mapping with the confidence score returned from the service is used for each question. We also use a statistics-based LingPipe named entity recognizer (NER) (Lines 30 – 32), where the label of the named entity that is assigned by LingPipe NER is used as the semantic type of the concept. We then consider all noun phrases in the question as candidate concepts. Therefore, we employ the OpenNLP chunker to detect all noun phrases (NPs) and prepositional phrases (PPs) from each question, and extract all NPs and all NP-PP-NP occurrences (Lines 33 – 38). We then extract a number of linguistic and semantic features from the tokens and concepts, as detailed in Table 1.

Classification. We use Logistic Regression from the LibLinear tool [1] to train a multi-class classifier, and use 10-fold cross prediction to predict a list of up to five most likely semantic labels for each question in the training set, which is used in the downstream training process (Lines 42 – 44). The model can correctly identify answer types for most high-frequency sentence patterns, such as “which genes”, but it may fail for low-frequency question patterns, where UTS may not be able to resolve ambiguous cases (e.g. AUS is identified as a country name without the context).

3.2 Candidate Answer Generation

We first use the same set of token and concept identification tools used for the question (described in Sec. 3.1) to annotate all the relevant snippets provided as input for Phase B (corresponding to Lines 51 – 65 of Listing 1.1). We then integrate four components to generate candidate answers (corresponding to Lines 69 – 71, and the component level descriptor is presented in Listing 1.2).

Concepts as Candidate Answers. We create a candidate answer using each concept identified by one of three concept identification approaches described in Sect. 3.1 (corresponding to Line 6 of Listing 1.2). In Batch 3, we also filtered out any concept mention that is exactly a stopword, a token or phrase in the question, or a concept that is also annotated in the question. We used a stopword list that combines the most 5,000 frequent English words and the list of Entrez (PubMed) stopwords.

CHOICE Questions (Line 4). We first identify the “or” token in the question, and then identify its head token, which is most likely the *first option* in the list of candidate answers. Next, we find all the children of the first option token in the parse tree that have a dependency relation of `conj`, which are considered to be *alternative options*. We see this approach works well on most CHOICE questions, but still has problems in a few special cases. First, if two options have different prefixes but the same suffix, the suffix may be discarded in the first option, e.g. “Is the long non- coding RNA malat-1 up or downregulated in cancer?”. Another issue is that the head tokens can be semantically incomplete, such that a phrase which covers the head token should be used instead for the options; we expand the candidate answer using a minimal concept mention that covers the candidate answer occurrence (Line 7).

QUANTITY Questions. We identify all the tokens that have a POS tag of `CD` in all relevant snippets (Line 5). This approach can reliably produce a complete set of quantitative mentions. However, it does not give us a way to semantically interpret the extracted numbers. For example, it could correctly identify “20,687”, “24,500”, etc. as candidate numbers, but does not have the ability to “summarize” the numbers and produce a single answer, e.g. “Between 20,000 and 25,000” as required. Similar to CHOICE questions, another limitation is that this method can only identify a single token as a candidate answer (e.g. “3.0”) where semantically complete phrase (e.g. “3.0 mm”) is preferred. We apply the same approach used for CHOICE questions to include the `CD`-bearing phrase as the candidate answer (Line 7).

3.3 Candidate Answer Scoring

We predict a confidence score for each candidate answer (corresponding to Lines 75 – 77 of Listing 1.1, and the component level descriptor is presented in Listing 1.3). In Batch 3, we use a simple multiplication method to combine the type coercion score and the occurrence count.

In Batches 4 & 5, we define a feature space containing 11 groups of features, as shown in Table 2, which extend the approach used by Weissenborn et al. [14], and use Logistic Regression to learn the scoring function. We only use the questions with non-zero recall for training, where we assign “1” to each candidate answer variant if it is also contained in the gold standard answer set, and “0” otherwise. Since there are many

Table 2. Answer Scorers

Line	Feature
5	<i>Type coercion.</i> For each candidate answer occurrence (CAO), the percentage of semantic types that are also among the top- k ($k = 1, 3,$ and 5) predicted answer types. To accumulate the scores from multiple CAOs, we use “average”, “maximum”, “minimum”, “non-zero ratio”, “one ratio”, and “boolean or”.
6	<i>CAO count.</i> We use the number of CAOs for each answer variant and we also count the total number of tokens in all occurrences.
7	<i>Name count.</i> The number of distinct candidate answer names, which differs from CAO count; if two CAOs have the same text string, only one will count.
8	<i>Avg. covered token count.</i> Averaged number of tokens in each CAO.
9	<i>Stopword count.</i> For each CAO, we calculate the stop word percentage. We use the same stoplist as described in Section 3.2. We accumulate the scores from multiple CAOs using “average”, “minimum”, “one ratio”, and “boolean or”.
10	<i>Token overlap count.</i> For each CAO, we calculate the percentage of tokens that overlap with the question. We accumulate the scores from multiple CAOs using “average”, “non-zero ratio”, and “boolean or”.
11	<i>Concept overlap count.</i> For each CAO, we calculate the percentage of covered concept mentions that overlap with the question. We accumulate the scores from multiple CAOs using “average”, “non-zero ratio”, and “boolean or”.
12	<i>Token proximity.</i> For each CAO, we calculate the averaged distance to the nearest occurrence of each question word in the relevant snippet. We set a window size of 10, and if any question word falls out of the window, we use a fixed distance of 20. We also transform the distance to its negation and inverse, and accumulate the scores from multiple CAOs using “average”, “maximum”, “minimum”, and “non-zero ratio”.
13	<i>Concept proximity.</i> Similar to token proximity, we calculate the distance from each CAO to each question concept mention in the relevant snippet.
14	<i>LAT count.</i> For each CAO, we calculate the percentage of tokens that overlap with a LAT token (i.e. the 8th feature in Table 1). We accumulate the scores from multiple CAOs using “average” and “non-zero ratio”.
15	<i>Parse proximity.</i> Similar to token proximity, we use the distance in the parse tree, which is important for list questions, as answer bearing sentences may be in the form of “includes A, B, C, ...”.

more negative instances than positive instances, we assign to each negative instance a weight of $\frac{\# \text{positive instances}}{\# \text{negative instances}}$.

3.4 Answer Pruning

In Batch 3, we used the factoid QA pipeline to produce answers for list questions without any pruning. In Batch 4, we used an absolute threshold to select only the answers that have a confidence score, predicted by the candidate answer scoring model, above threshold. Starting from Batch 5, instead of an absolute threshold for all questions, we use a relative threshold to filter the answers that have a confidence score above a percentage of the highest predicted score for the question (corresponding to Line 78 – 80 of Listing 1.1). We tune the threshold on the development set.

4 Retrieval Approaches for Phase A

In this section, we describe the approaches that are used for retrieval tasks in Phase A. The pipeline diagram for Phase A is illustrated in Fig. 2 in Appendix.

4.1 Document Retrieval

Our approach is similar to what we have proposed in the TREC 2014 Web Track [15], with some modifications made for better performance and efficiency.

Offline Indexing of Medline Baseline Corpus. We used Lucene to index a Medline baseline corpus using title, abstract and keywords fields, if available. We used the standard Lucene tokenizer combined with the Krovetz Stemmer, which is less aggressive compared to the Porter Stemmer. This is an important step, because many biomedical terms (in particular gene names) are not recognizable by stemmers, and the Porter stemmer is likely to truncate many of the words, causing increased confusion between the stemmed biomedical terms and common terms during search time. We also kept the stopwords in the index. The motivation is that since we only have the abstract text for the document, removing stopwords may result in less accurate field length statistics, thus affecting the performance of many language model based retrieval models.

Hierarchical Retrieval Architecture. The fact is that given a query, we have more retrievable documents than we can perform a deeper analysis for. However, to ensure better retrieval performance, in-depth analysis of the documents is necessary. Therefore, a hierarchical retrieval architecture is introduced here to find a good balance between performance and efficiency. In summary, each search task is processed by three stages:

1. *Obtaining an affordable high recall candidate set.* During the query time, we have removed all stopwords from the query, as they provide no useful information and will likely cause serious efficiency issues. We use the Dirichlet smoothing retrieval model implemented in Lucene to conduct this search. In our implementation, we consider only the top 10,000 ranked documents.
2. *Precision oriented reranking.* We incorporate the Negative Query Generation (NQG) model [7], which utilizes a negative language model by assuming that all documents in the corpus are non-relevant, thus making more accurate adjustments to query term weights and relevance calculations. After re-ranking with NQG, we can now further cut down the candidate set by considering only the top 100 documents in the ranked list.
3. *Deep document feature extraction and learning to rank (LETOR).* We use ranker scores (e.g. BM25, Jelinek-Mercer smoothing, Dirichlet smoothing, Indri two-stage smoothing, NQG, etc), similarity scores (e.g. Jaccard coefficient and Dice coefficient, etc.), raw features (e.g. document length, vocabulary size, etc.), and customized features (e.g. harmonic means of the ranker scores across all fields, the distribution of the query terms across the documents, etc.). We simply score the K documents with a pre-trained LETOR model which was optimized for Precision@10. Here, we are using Random Forest, an ensemble method known for robustness against overfitting.

The details of the proposed document retrieval approach can be found in our previous work [15].

Table 3. Snippet Retrieval Features

No.	Feature
1	BM25: We index all the candidate snippets using Lucene, and then use a query that contains not only words but also phrases and confidence scores of all the different query concepts returned by the MetaMap service.
2	Skip-bigram: Based on the dependency relations generated from the dependency parser for each question, we count the number of matched pairs and calculate the F-1 based skip-bigram score.
3	Textual alignment: Surface similarity of a snippet and a question. We also consider the relative order of the different words.
4	Some other question independent features, such as the length of the snippet.

4.2 Snippet Retrieval

The snippet retrieval module analyzes the 10 most relevant documents returned from the upstream document retrieval component. We first identify the extent of a snippet and then apply a LETOR approach for snippet retrieval.

Candidate Snippets Generation. The definition of “snippet” is the original piece of text extracted from the document. In our initial study, we found that the distribution of snippet length in the gold standard answers is similar to that of sentence length. Therefore, we apply a sentence segmenter to split the snippets and define each sentence as a snippet candidate.

Feature Extraction and LETOR. We define four types of features for LETOR in Table 3, and also apply the logistic regression classifier for scoring.

4.3 Concept Retrieval

We first identify the text spans from each question and search these texts from various GoPubMed concept services. Since only a single list of concepts is returned, we also propose to merge and rank the concept lists returned from multiple sources.

Candidate Queryable Concept Generation. We use MetaMap to identify the UMLS concepts from the question, and our results indicate a significant improvement in recall. However, one of the major drawbacks of MetaMap is that it is poor at identifying gene and protein names. To overcome this issue, we use LingPipe NER with the model trained on the GeneTag corpus to recognize gene names to enrich the retrieved metathesaurus concepts. We then use the combination of tokens retrieved from the MetaMap service and the LingPipe NER to query various biomedical ontologies.

Concept Ranking and Merging. We create a ranking model that can rank the search results from different ontologies. We use the federated search approach [12], which trains a relevance mapping logistic function that maps the relevance scores of each result from each ontology to a global relevance scale.

4.4 Triple Retrieval

Similar to concept retrieval, we rely on the BioASQ provided service to retrieve relevant triples. Therefore, our goal is to construct an effective query string. Beyond the baseline

method that simply concatenates all the keywords from the concept retrieval result, we made three improvements:

- Append “[obj]” and “[sub]” identifiers to each keyword in the query string.
- Enumerate all letter case possibilities for keywords: lower case, upper case, and capitalized word.
- Add all words in the original question to the keyword set while excluding the stop words and SQL reserved keywords.

The first improvement is to help the triple query server understand that most of our keywords are used as objects or subjects. This finding is intuitive through observation; since most of the words are nouns or adjectives, which are unlikely used as predicates in triples. The second improvement is based on an observation from examination of gold standard answers, where triple results indicate case-sensitivity during triple matching. Therefore, we need to include all casing variants to ensure that keywords are matched during triple retrieval. The third improvement ensures that we do not omit keywords from the original question, to make the query more robust.

5 Results & Analysis

We summarize the official evaluation results of document and snippet retrieval in Phase A and factoid and list QA in Phase B in Batches 3, 4, and 5 from the official evaluation portal in Table 4.

Among all the systems that participated in Phase A evaluation, the performance of our document retrieval pipeline is scored at the bottom of the first tier. The absolute performance gaps between our pipeline and the system that is scored one place behind ours in Batches 3, 4, and 5 are measured as .0915, .0225, and .0869 respectively in terms of MAP, which are larger than those between our pipeline and the best performing system (.0435, .0204, and .0466 respectively).

Due to a relatively steep learning curve for the developers who have not had much experience with the system and the task, Phase A system used a different question analysis pipeline from the Phase B system, which had no concept retrieval module integrated and tested, which should expand each concept with synonyms. Therefore, we believe document and snippet retrieval evaluated in Phase A can be further improved by considering synonyms expanded using UTS during query formulation. Moreover, the snippets extracted by the latter snippet retrieval stage can be fed back to the search engine as an expanded query to harvest more relevant information; reinforcement learning can thus be utilized in this scenario.

For Phase B, we see that our system achieved five of six highest performance scores among all participating systems for factoid and list question answering in Batches 3, 4, and 5. We notice that the performance in Batch 4 is higher than in other batches, which we believe is because Batch 4 set contains more questions seeking for the types of answers that have occurred more frequently in the training set, e.g. gene, disease, etc.

To further understand what causes the error and how we may improve the system, we manually answer each factoid question in Batches 3, 4, and 5 using the gold standard snippets provided for the input of Phase B, and compare with the output of our system

Table 4. Partial official evaluation result. Ranks among systems (as of the manuscript completion) are shown in the parentheses.

Phase A: Document						
Batch	Precision	Recall	F-measure	MAP	GMAP	
3rd	.2310 (15)	.3242 (15)	.2311 (15)	.1654 (15)	.0136 (15)	
4th	.2144 (15)	.3320 (15)	.2263 (15)	.1524 (15)	.0081 (14)	
5th	.2130 (15)	.4474 (15)	.2605 (15)	.1569 (15)	.0267 (8)	
Phase A: Snippet						
Batch	Precision	Recall	F-measure	MAP	GMAP	
3rd	.1133 (3)	.1044 (5)	.0891 (3)	.0892 (1)	.0013 (5)	
4th	.1418 (5)	.1264 (10)	.1153 (8)	.0957 (5)	.0027 (6)	
5th	.1472 (9)	.1756 (9)	.1391 (9)	.1027 (9)	.0040 (5)	
Phase B: Exact Answers						
Batch	Factoid			List		
	Strict Acc.	Lenient Acc.	MRR	Precision	Recall	F-measure
3rd	.1154 (1)	.2308 (1)	.1615 (1)	.0539 (8)	.6933 (1)	.0969 (7)
4th	.4483 (1)	.6207 (1)	.5155 (1)	.3836 (1)	.3480 (1)	.3168 (1)
5th	.2273 (1)	.3182 (1)	.2727 (1)	.1704 (1)	.2573 (5)	.1875 (1)

to label the error types (multiple types allowed) for each incorrectly answered question. We list the error categories and give definition and examples to each category in Table 5, where we also show the occurrence of each error category in each test batch.

Based on the analysis, we believe a better concept identification model and concept type prediction model will make the hugest impact to the overall performance improvement. Moreover, we plan to conduct a thorough ablation study to estimate how much each component or feature contributes to the overall performance, as soon as we have the gold-standard outputs for the 3B dataset.

6 Conclusion

This paper describes the CMU OAQA system evaluated in the BioASQ 3B Question Answering track. We first present a three-layered architecture, and then describe the components that have been integrated into the participating system for exact answer generation and retrieval. We also investigate how to learn to answer questions from such a large gold standard biomedical QA dataset, using an answer type prediction model and an answer scoring model. The official evaluation results show the effectiveness of the proposed approach in factoid and list QA.

Further work is necessary to improve the retrieval components in Phase A. We are also interested in investigating how to learn to answer yes/no questions and summary questions from existing Q/A pairs. We plan to integrate the system into the BioQUADS (biomedical decision support system) [17] to process biomedical complex decision processes represented in natural language.

Table 5. Error categories and occurrences for factoid questions in test batches 3, 4, and 5.

Error category	Batch		
	3rd	4th	5th
Concept type identification/answer type prediction The highest ranked answer has a different concept type from the answer type that question asks for, which may be caused by a wrongly predicted answer type, an incorrect score combination equation from the score prediction model, or the concept identification module.	9	8	8
Concept identification Some answer variants are not identified as concepts or we can find little evidence from the relevant snippets for the concept. For example, for the question “Neurostimulation of which nucleus is used for treatment of dystonia?”, none of the components is able to identify “Bilateral globus pallidus internus (GPi)” as a concept and further candidate answer variant.	4	4	2
Complex answer The ideal answer is a complex phrase or sentence, rather than a single-entity concept, usually in response to the questions containing “effect”, “role”, “function”, etc. For example, “executors/mediators of apoptosis” should be extracted to answer the question “What is the function of caspases?”, but we only see “apoptosis” in the candidate answer list.	2	2	5
Mistakenly use question phrase as answer Although we design a scorer in the ranking module to identify whether each candidate answer co-occurs in the original question, which should lower the rank of those candidate answers, we still see some question phrase variants are chosen as the top answer. For example, the question “What is the effect of enamel matrix derivative on pulp regeneration” mentions a concept “enamel matrix derivative”, but the system ranks its acronym “EMD” at the top.	3	2	2
Tokenization Tokenization module may fail if the concept contains punctuation marks, e.g. parentheses, colon, semicolon, etc, and/or numbers, as in the example “t(11;22)(q24;q12)”.	2	4	0
Definition question The asker knows the terminology but asks for the definition, e.g. “What is Piebaldism?”, or knows the properties and asks for terminology, e.g. “How are ultraconserved elements called when they form clusters?”. We believe we need to introduce special question types and modules.	2	0	1
Question type Identification of QUANTITY and CHOICE questions may fail in some cases. For example, “Alpha-spectrin and beta-spectrin subunits form parallel or antiparallel heterodimers?” does not use “Do” at the beginning. Another example is that “risk” is a QUANTITY indicator in the question “What is the risk of developing acute myelogenous leukemia in Fanconi anemia?”	1	0	1
Snippets that have no information Some snippets do not contain any answer variant. For example, “What is the main role of Ctf4 in dna replication?” has a gold standard snippet “Ctf4 remains a central player in DNA replication”.	0	0	2
Relation concept identification A relation concept refers to a verb or verbal adjective, e.g. “responsible” or “leading” that distinguishes the expected answer from other candidates that have the same concept type.	0	1	1
Syntactic function The key to answer the question is embedded in the syntactic structures of the relevant snippets. For example, in the snippet “Medicarpin, the major phytoalexin in alfalfa, ...”, no explicit relation word is used between “Medicarpin” and “the major phytoalexin”, but the syntactic structure clearly implies that the latter explains the former.	0	1	1

Acknowledgments. We thank Qianru Zhu and Avner Maiberg for their involvement in the early stages. We also thank Ying Li, Xing Yang, Venus So, James Cai and the other team members at Roche Innovation Center New York for their continued support of OAQA and biomedical question answering research and development.

References

1. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: Liblinear: A library for large linear classification. *J. Mach. Learn. Res.* 9(Aug), 1871–1874 (2008)
2. Ferrucci, D., Nyberg, E., Allan, J., Barker, K., Brown, E., Chu-Carroll, J., Ciccolo, A., Duboue, P., Fan, J., Gondek, D., et al.: Towards the open advancement of question answering systems. Tech. Rep. RC24789 (W0904-093), IBM Research Division (2009)
3. Garduno, E., Yang, Z., Maiberg, A., McCormack, C., Fang, Y., Nyberg, E.: Cse framework: A uima-based distributed system for configuration space exploration. In: *UIMA@GSCL'2013*. pp. 14–17 (2013)
4. Hersh, W., Voorhees, E.: Trec genomics special issue overview. *Inf. Retr.* 12(1), 1–15 (2009)
5. Lally, A., Prager, J.M., McCord, M.C., Boguraev, B., Patwardhan, S., Fan, J., Fodor, P., Chu-Carroll, J.: Question analysis: How watson reads a clue. *IBM J. Res. Dev.* 56(3.4), 2–1 (2012)
6. Li, X., Roth, D.: Learning question classifiers. In: *ACL'2002*. pp. 1–7 (2002)
7. Lv, Y., Zhai, C.: Query likelihood with negative query generation. In: *CIKM'2012*. pp. 1799–1803 (2012)
8. Nyberg, E., Mitamura, T., Callan, J., Carbonell, J., Frederking, R., Collins-Thompson, K., Hiyakumoto, L., Huang, Y., Huttenhower, C., Judy, S., et al.: The javelin question-answering system at trec 2003: A multi-strategy approach with dynamic planning. In: *TREC'2003* (2003)
9. Papanikolaou, Y., Dimitriadis, D., Tsoumakas, G., Laliotis, M., Markantonatos, N., Vlahavas, I.: Ensemble approaches for large-scale multi-label classification and question answering in biomedicine. In: *CLEF'2014* (Working Notes). pp. 1348–1360 (2014)
10. Patel, A., Yang, Z., Nyberg, E., Mitamura, T.: Building an optimal question answering system automatically using configuration space exploration (cse) for qa4mre 2013 tasks. In: *CLEF'2013* (Working Notes) (2013)
11. Peñas, A., Hovy, E.H., Forner, P., Rodrigo, Á., Sutcliffe, R.F.E., Sporleder, C., Forascu, C., Benajiba, Y., Osenova, P.: Overview of qa4mre at clef 2012: Question answering for machine reading evaluation. In: *CLEF'2012* (Working Note) (2012)
12. Si, L., Callan, J.: Modeling search engine effectiveness for federated search. In: *SIGIR'2005*. pp. 83–90 (2005)
13. Tsatsaronis, G., Balikas, G., Malakasiotis, P., Partalas, I., Zschunke, M., Alvers, M.R., Weissenborn, D., Krithara, A., Petridis, S., Polychronopoulos, D., et al.: An overview of the bioasq large-scale biomedical semantic indexing and question answering competition. *BMC Bioinformatics* 16(1), 138 (2015)
14. Weissenborn, D., Tsatsaronis, G., Schroeder, M.: Answering factoid questions in the biomedical domain. In: *BioASQ'2013* (2013)
15. Xu, D., Callan, J.: Towards a simple and efficient web search framework (2014)
16. Yang, Z., Garduno, E., Fang, Y., Maiberg, A., McCormack, C., Nyberg, E.: Building optimal information systems automatically: Configuration space exploration for biomedical information systems. In: *CIKM'2013*. pp. 1421–1430 (2013)
17. Yang, Z., Li, Y., Cai, J., Nyberg, E.: Quads: Question answering for decision support. In: *SIGIR'2014*. pp. 375–384 (2014)

Appendix

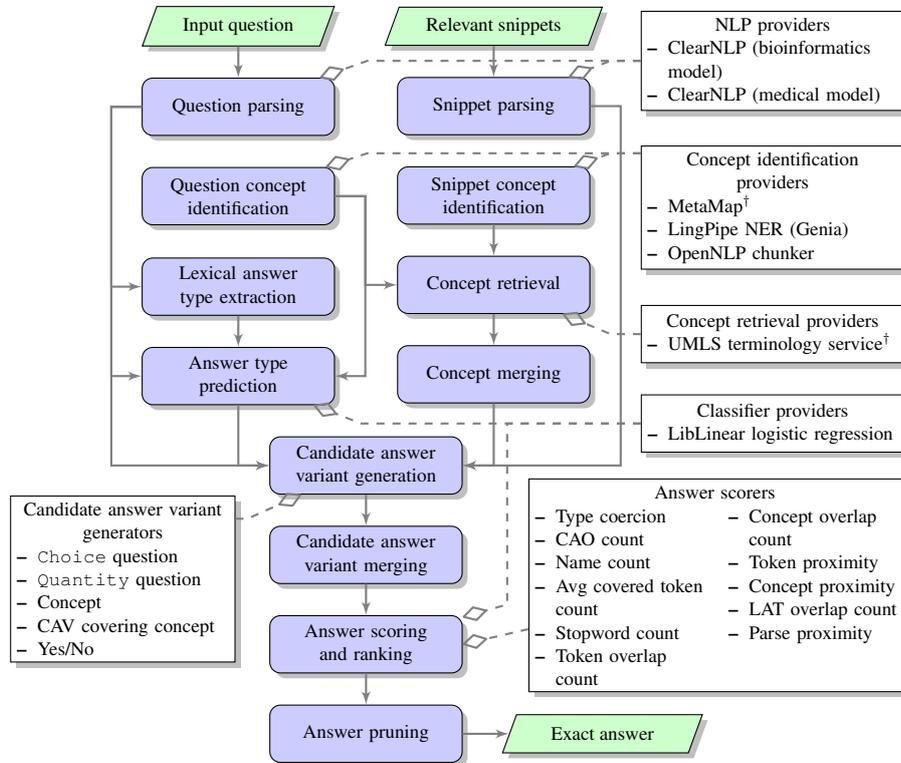


Fig. 1. Phase B pipeline diagram. † represents a provider that requires accessing external Web services.

Listing 1.1. ECD main descriptor for test batch 5 in Phase B

```

1 # execute
2 # mvn exec:exec -Dconfig=bioasq.
  test
3 # to test the pipeline
4
5 configuration:
6   name: test
7   author: ziy
8
9 persistence-provider:
10  inherit: baseqa.persistence.
11     local-sqlite-persistence-provider
12
13 collection-reader:
14  inherit: baseqa.collection.json.
15     json-collection-reader
16  dataset: BIOASQ-QA
17  file:
18    - /input/3b-5-b.json
19  type: [factoid, list, yesno, summary]
20  decorators: |
21    - inherit: bioasq.gs.
22      bioasq-qa-gs-decorator
23  persistence-provider: |
24    inherit: baseqa.persistence.
25      local-sqlite-persistence-provider
26
27 pipeline:
28 - inherit: ecd.phase
29   options: |
30     - inherit: bioqa.quesanal.
31       parse-clearnlp-bioinformatics

```

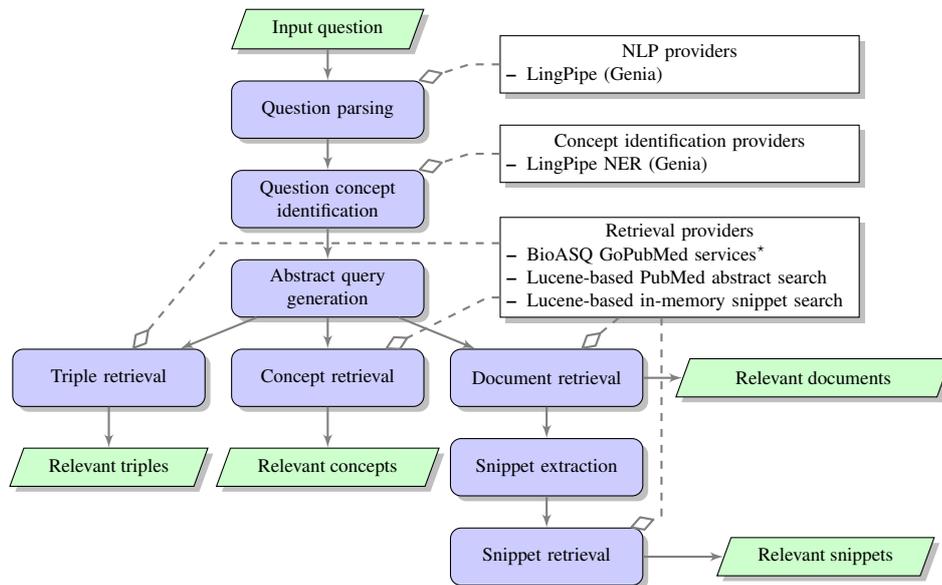


Fig. 2. Phase A pipeline diagram. ★ represents a provider that requires accessing BioASQ Web services.

```

27 - inherit: ecd.phase
28 options: |
29   - inherit: bioqa.quesanal.
30     concept-metamap
31 - inherit: ecd.phase
32 options: |
33   - inherit: bioqa.quesanal.
34     concept-lingpipe-genia
35 - inherit: ecd.phase
36 options: |
37   - inherit: baseqa.quesanal.
38     concept-opennlp-np
39 - inherit: ecd.phase
40 options: |
41   - inherit: baseqa.quesanal.
42     lexical-answer-type
43 - inherit: ecd.phase
44 options: |
45   - inherit: bioqa.quesanal.at.
46     predict-liblinear
47 - inherit: ecd.phase
48 options: |
49   - inherit: baseqa.retrieval.
50     passage-to-view
51 - inherit: ecd.phase
52 options: |
53   - inherit: bioqa.retrieval.
54     passage-parse-cleanlp-bioinformatics
55 - inherit: ecd.phase
56 options: |
57   - inherit: bioqa.retrieval.
58     passage-concept-metamap
59 - inherit: ecd.phase
60 options: |
61   - inherit: bioqa.retrieval.
62     passage-concept-lingpipe-genia
63 - inherit: ecd.phase
64 options: |
65   - inherit: baseqa.retrieval.
66     passage-concept-opennlp-np
67 - inherit: ecd.phase
68 options: |
69   - inherit: baseqa.retrieval.
70     passage-concept-opennlp-nppnp
71 - inherit: ecd.phase
72 options: |
73   - inherit: bioqa.retrieval.
74     concept-search-uts
75 - inherit: ecd.phase
76 options: |
77   - inherit: baseqa.retrieval.
78     concept-merge
79 - inherit: ecd.phase
80 options: |
81   - inherit: bioqa.answer.generate
82 - inherit: ecd.phase
83 options: |
  
```

```

74     - inherit: baseqa.answer.modify
75 - inherit: ecd.phase
76   options: |
77     - inherit: bioqa.answer.
78       score-predict-liblinear
79 - inherit: ecd.phase
80   options: |
81     - inherit: baseqa.answer.pruner
82 post-process:
83   # answer evaluation
84 - inherit: baseqa.eval.base
85   calculator: |
86     inherit: bioasq.eval.calculator.
87     answer-eval-calculator
88   evaluatee-provider: |
89     inherit: baseqa.eval.evaluatee.
90     answer-evaluatee-provider
91   persistence-provider: |
92     inherit: baseqa.eval.persistence.
93     jdbc-eval-persistence-provider
94   # report
95 - inherit: report.csv-report-generator
96   builders: |
97     - inherit: baseqa.report.
98       accumulated-measurements-report-component
99   # submission
100 - inherit: bioasq.collection.json.
101     json-cas-consumer

```

Listing 1.2. ECD component descriptor of

```

bioqa.answer.generate
1 class: edu.cmu.lti.oaqa.baseqa.answer.
2   CavGenerationManager
3 generators: |
4 - inherit: baseqa.answer.generators.
5   choice

```

```

5 - inherit: baseqa.answer.generators.
6   quantity
7 - inherit: bioqa.answer.generators.
8   concept
9 - inherit: baseqa.answer.generators.
10  cav-covering-concept
11 - inherit: baseqa.answer.generators.
12  yesno

```

Listing 1.3. ECD component descriptor of

```

bioqa.answer.score-predict-liblinear
1 inherit: baseqa.answer.score-predict
2
3 classifier: 'inherit: bioqa.answer.
4   score-classifier-liblinear'
5 scorers: |
6 - inherit: baseqa.answer.scorers.
7   type-coercion
8 - inherit: baseqa.answer.scorers.
9   cao-count
10 - inherit: baseqa.answer.scorers.
11  name-count
12 - inherit: baseqa.answer.scorers.
13  avg-covered-token-count
14 - inherit: bioqa.answer.scorers.
15  stopword-count
16 - inherit: baseqa.answer.scorers.
17  token-overlap-count
18 - inherit: baseqa.answer.scorers.
19  concept-overlap-count
20 - inherit: baseqa.answer.scorers.
21  token-proximity
22 - inherit: baseqa.answer.scorers.
23  concept-proximity
24 - inherit: baseqa.answer.scorers.
25  lat-overlap-count
26 - inherit: baseqa.answer.scorers.
27  parse-proximity

```