Milestone 3 Report

Jeremy Doornbos, Anurag Kumar, Zexi Mao, Fei Xia
(alphabetical order)
{jdoornbo, alnu, zexim, feixia}@cs.cmu.edu
Language Technology Institute

1. Introduction

The goal of this project was to build a Biomedical Question Answering (BioQA) system. There are several important aspects of a BioQA system. The first one is domain specific knowledge. We need sources which can provide relevant information to answer the questions. This information is provided to us through different web services. Another important aspect of any BioQA system is what kind of questions it can answered with reliable accuracy. For example, the questions can be of Yes-No type, implying the answer to the question is either Yes or No. In other cases the question might require the system to give an exact answer consisting of a few words or a detailed paragraph as an answer to the question. Our focus in this project was on answering Yes-No type of questions.

The first step in answering the question would be to query it on available web services. The naive way to do this is to query the whole question. However, it is obvious that this is not a good approach. A better way is to actually build a query-text from the question and then use that for querying the web services. A query preprocessing and query builder step is thus needed. Once we have the information available from the web services the problem boils down to using this information to generate the exact answers. This is non-trivial not only because the web services can provide irrelevant information but also because mining out the exact answer itself is pretty hard. The information is available in forms of Concepts, Documents and RDF triples. Before we can use them to generate answers we need to make sure that we are retrieving the proper sets of concepts, documents and triples. This is achieved through various strategies as explained in further sections. Once we have good confidence on retrieval of these items we use them to generate the final answer.

2. System Design

The system is designed in a very flexible way, and some design patterns like singleton are also used. All the parameters (like model path, threshold) are not hard coded and can be easily modified in the descriptor file. In addition, we can take out or plug in many important components to compare the similarity and do some error analysis.

2.1 Overall Architecture

The overall architecture of the system is shown in Figure 1. This pipeline illustrates the flow from the input list of questions to the output results and evaluation. We use multiple UIMA components -- a CollectionReader, several Annotators, and a Consumer -- to implement this pipeline.

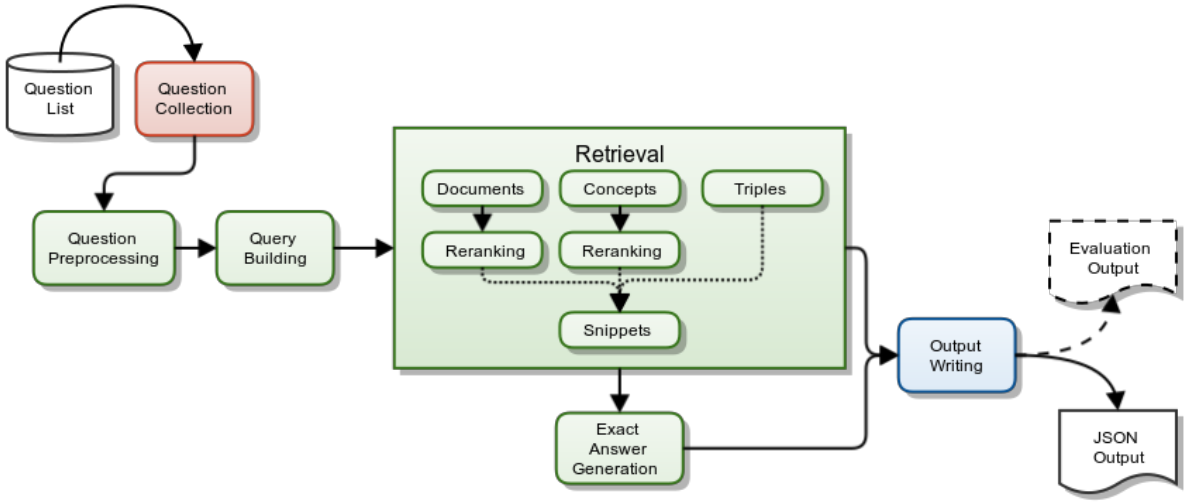

Figure 1. System Pipeline

2.1.1 Collection Reader

The collection reader performs question collection. This step is displayed in red in Figure 1. It simply reads in the list of questions, creating a CAS object for each one. These CAS objects will be updated by the annotators further down the pipeline.

2.1.2 Annotators

We use several annotators, in the preprocessing and query building steps, as well as in retrieval and answer generation. These steps are all displayed in green in Figure 1. First in question preprocessing, annotators extract terms using POS tag-based named entity recognition and biomedical tagging. Another annotator combines these terms in query building to prepare a basic query for use in retrieving the information needed for generating an answer. Then this query is used by multiple annotators to retrieve documents, concepts, and triples. The retrieved documents and concepts are reranked, and another annotator extracts snippets from the best documents. Finally, the retrieved items are used by a final annotator to generate an exact answer for the question.

2.1.3 Consumer

A CAS Consumer is used to output all of the retrieved items, as well as the exact answers. This step is displayed in blue in Figure 1. The output is written to a JSON file, and can optionally be used to evaluate the performance of the system based on gold standard data.

The Consumer can make use of an evaluation module, writing the results to a separate output file.

3. Algorithm Design

3.1 Question Preprocessing

Puting the raw questions directly into the search engine does not give us very good results. We have to do some question preprocessing. We tried text preprocessing like case normalization, stop word removal, punctuation removal, etc. Some advanced techniques like named entity recognition and parsing are used to find the different named entities and their relationships.

In addition, we use ABNER, a biomedical named entity recognizer, to figure out whether the question is related to genes or proteins, so that we can do better web service source selection later.

3.2 Query Builder

There are multiple operators we can use in given web services, the most popular ones are AND and OR. The Query Builder selects keywords from the preprocessed question and combines them with AND and OR operators. We tried different ways to do this, like only using AND, only using OR, using AND for any two keywords then doing an OR, etc. Some results are shown in "Results and Analysis" section.

3.3 Retrieval

3.3.1 Concept Retrieval

Multiple web services are available for concept retrieval. This requires us to come up with different ways to use all the sources and then rank the concepts retrieved from different sources. Different sources provide information about the query in different contexts. For example the disease ontology returns concepts with respect to diseases, the gene ontology returns concepts with respect to genes. The problem of combining concepts from different sources becomes hard because of this variability. Overall this means the returned concepts refer to different contexts and getting a single ranked list of concepts using all sources becomes hard. Overall the problem can be looked in terms of assigning weights to different services.

We came up with some simple yet effective methods to solve this problem. Our first method focuses on the reliability of the web service itself. For any query and a set of concepts retrieved from a web service we prune out results for which the scores returned by the server is below certain threshold. Once we have reliable sets of concepts corresponding to each web service we try to rescore the concepts from by assigning different weights to different web services. We obtain the mean score of all concepts (after pruning) for a web service. We use

this mean value as a measure of confidence for that web service. We normalize the means for different web services and then use that to reweigh the concepts for a web service. The final ranked list of concepts is obtained by ranking the concepts based on these reweighed scores.

The second method tries to utilize information present in the question itself. In the question preprocessing part we try to extract out information about the question itself. What this basically means is that we try to find out if the question actually refers to some specific ontology such as gene or protein. We use external tools such as ABNER and Lingpipe to do that. If such information can be extracted out it is stored and then used for assigning higher weights to the corresponding ontology web service. This is done by increasing the weight of the corresponding ontology by a certain amount. The weights are always normalized to sum to 1. The overall strategy is shown in Figure 2.

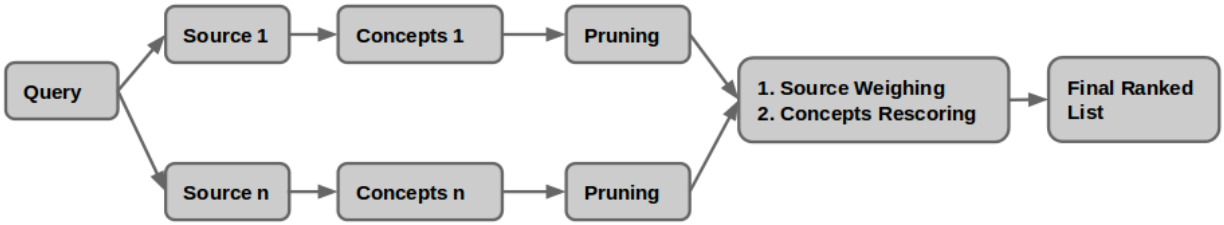

Figure 2. Flowchart for Concept Retrieval

3.3.2 Triple Retrieval

For triple retrieval, we take the query from the Query Builder and use the corresponding web services to do the retrieval. We found that the most of the returned triples are not very relevant.

3.3.3 Document Retrieval

Similarly, for document retrieval, we take the query and search using the PubMed web service. The retrieved documents are really important for components later, so we did extra reranking for them and it works very well.

3.3.4 Reranking

Here we mainly consider the reranking for retrieved documents, though there are also some reranking techniques being used in the Concept Retrieval part (related to Question Based Source Selection). Doing document reranking is necessary because we do not know whether the retrieval algorithms of the provided web services are reliable and how those services generate ranking scores. The reranking is mainly based on the abstract of a document.

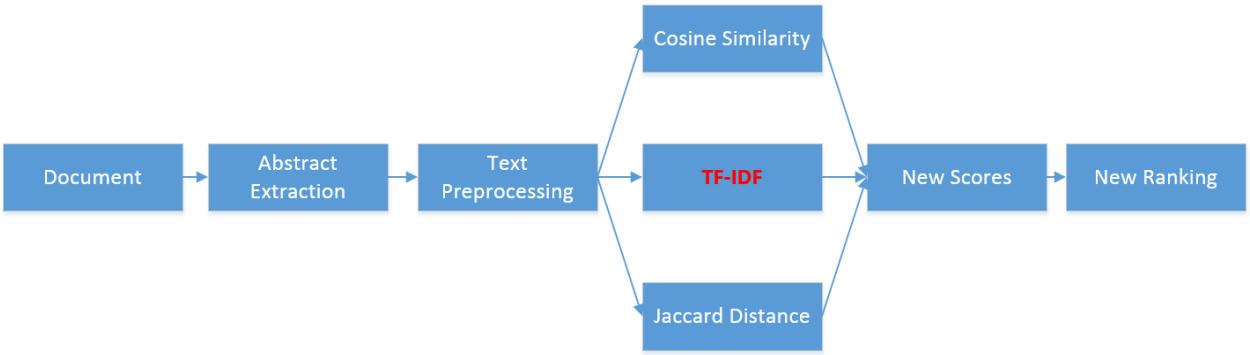

Figure 3. Reranking Algorithm

The reranking algorithm is shown in Figure 3. Firstly, the abstract is extracted. Secondly, we do some text preprocessing like tokenization, stop word removal and stemming. Then we use the processed question as the query to do retrieval for the extracted abstract. Different algorithms like cosine similarity, TF-IDF and Jaccard Distance were tried and we found the TF-IDF works best (please see the "Results and Analysis" section for comparison).

3.3.5 Snippet Retrieval

The process for Snippet Retrieval is similar to Document Reranking and is shown in Figure 4.

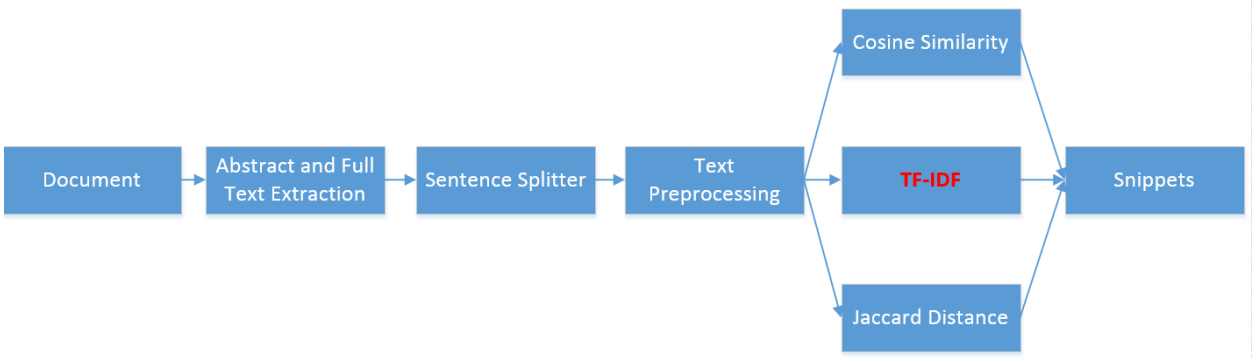

Figure 4. Snippet Retrieval Algorithm

Now, not only the abstract should be considered, but also the full text. Here we treat each sentence as one snippet so we need to a Sentence Splitter to determine the boundaries between sentences. Then this also boils down to a retrieval problem and we found the TF-IDF works the best. Note that later several small snippets can be merged to a larger snippet.

3.4 Generating Answer

We focus on Yes/No questions in our current system and we need to find the yes/no answer in the retrieved snippet. We use some effective and efficient ways to solve this problem, mainly by looking at the negative relations in a sentence. To make it clear, the sentence "Plants contain desmins" does not have any negative relations, however, "Plants do not

contain any desmins" has a negative relation because there is a "not" inside. Similarly, words such as "unclear", "undetermined" and "deny" also indicate negative relations.

By inspecting the given Yes/No questions, we found that there are no negative relations in the question text, so our general principle is: If most of the retrieved snippets are positive, the answer should be yes; otherwise, the answer should be no. This is illustrated in Figure 5.

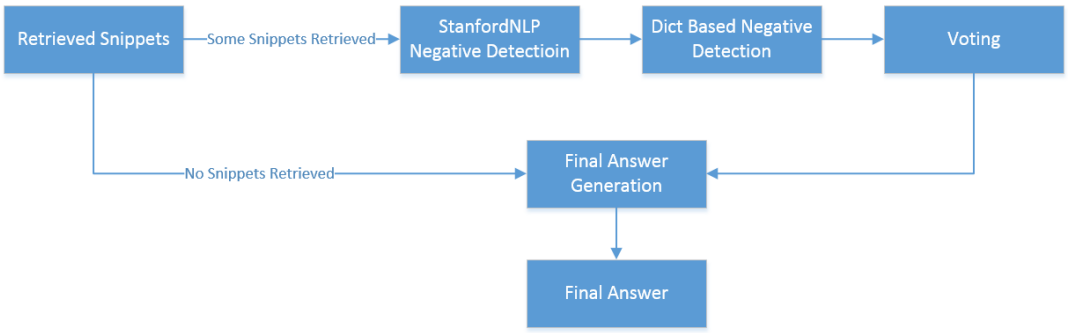

Figure 5. Answer Generation Algorithm

There are two main pathways:

(1) If there are no retrieved snippets, we directly give an answer of “No”. The intuition behind this is that if we cannot retrieve any snippets relevant enough to the question to indicate an affirmative answer, then it is likely that the answer is negative.

(2) If there are some snippets retrieved:

  (a) Do StanfordNLP parser based negative relation detection.

  (b) Do dictionary based negative relation detection. We built a negative word dictionary, so we can first do a lemmatization of retrieved snippets then compare those words with the words in our dictionary.

  (c) Voting. Every question may have multiple retrieved snippets, and different snippets may give different answers, so a voting process is needed.

4. Results and Analysis

4.1 Concepts

In the first step we used only a single web service to get concepts. Although this gives good results on the provided dataset, relying on just one source is not a good idea. The test set in the real world can be very diverse and it is important to use information available from other sources. As explained in Section 3.3.1 we used different methods to properly use all sources for concept retrievals. It can be seen from table of unordered measures that using MeSH only results in low recall. To improve precision for when using all sources pruning is very effective. It improves the precision while maintaining the recall.

Concept Perf.	Precision	Recall	F-Measure
Mesh Only	0.213	0.269	0.208
All - No pruning	0.096	0.439	0.129
All - Pruning	**0.142**	**0.405**	**0.178**

Table: Concepts on Unordered Measures

Having established that pruning can improve precision we show results for the methods described in Section 3.3.1. The ordered measures can be improved by reranking concepts retrieved from different sources. The first column shows results performance using only MeSH web service. After that we show results using all sources (namely MeSH, disease ontology, gene ontology and protein ontology). SW refers to source weighing. This basically refers to the method 1 in section 3.3.1. For each source its weight is defined by the mean of scores of pruned concept results. These weights are normalized to sum to 1. QS refers to query based weighing of sources. If query suggests information about some source being useful such as gene or protein then the weight for the corresponding source are increased by a certain amount. The weights of different sources are again normalized to sum to 1.

Concept Perf.	MAP	GMAP
Mesh Only	0.479	0.060
All - No pruning	0.344	0.106
All - Pruning	**0.435**	**0.119**
All - Prun. + SW	**0.445**	**0.130**
All - Prun. + QS	**0.426**	**0.142**
All-Prun.+QS+SW	0.435	0.120

Table 1. Concept Retrieval Results

As stated previously, using single source (MeSH) gives good result in itself. But from the perspective of a generic system design using just one source is not a good idea. Again a simple union of sources and ranking documents based on the scores provided by source does not give good results. Pruning the results becomes very important then.

4.2 Documents

The results of our document retrieval is shown in Figure 6. We achieved significant improvement over our initial Milestone 1 results by incorporating both question preprocessing and document reranking. Preprocessing nearly doubled the MAP and adding document reranking increased it further by about 50%. The GMAP also increased accordingly.

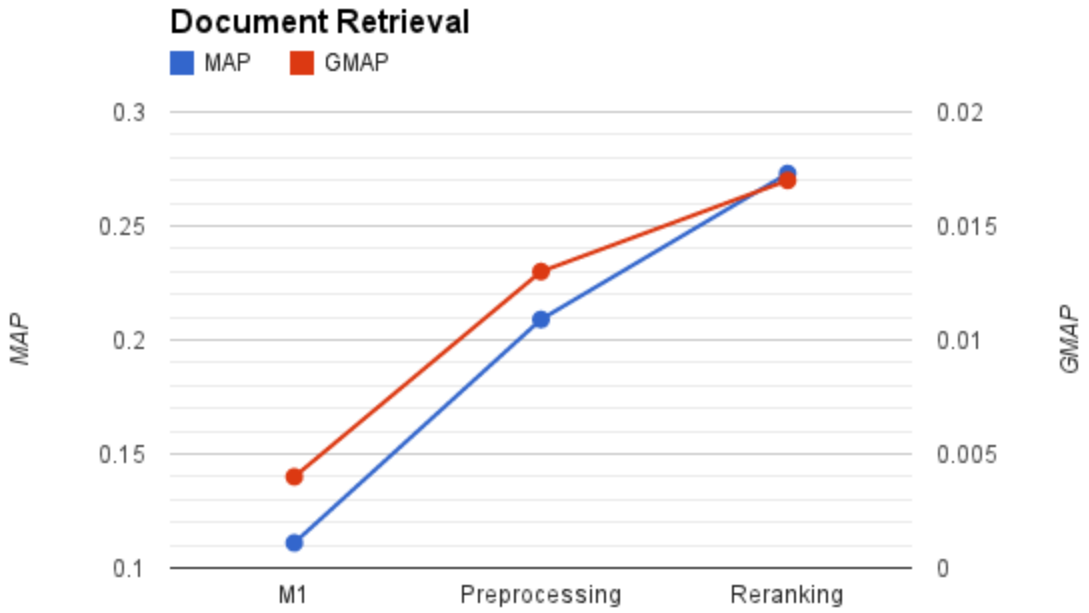

Figure 6. Document Retrieval Results

Query formulation affects the performance too. We tried multiple combinations of operations, and the results on document retrieval are shown in Table 2. Here, "AND" means we only use AND operator; "(AND) OR (AND)" means using AND for any two keywords then combining them with OR; "AND with root" means we not only use AND for detected named entities, but also add the relationship word to the query.

Query Formulation	Precision	Recall	F1 Score	MAP	GMAP
AND	**0.214**	**0.240**	**0.188**	**0.278**	**0.017**
(AND) OR (AND)	0.110	0.139	0.115	0.139	0.005
AND with root	0.115	0.161	0.100	0.163	0.008

Table 2. Query Formulation for Document Retrieval

We can easily see that the "AND" works best. Our mechanism for retrieving documents is to send documents ranked higher than the threshold to reranker, so the reasons that other two methods don't work well are: 1) "(AND) OR (AND)" is too loose, so it retrieves a lot of documents and irrelevant documents may be ranked high; Those irrelevant documents are sent to the reranker 2) "AND with root" gives too strict criterion so that limited number of documents are retrieved.

4.3 Triples

We found that triples are not very important in our retrieval pipeline. We did not spent too much time on improving snippet results.

4.4 Snippets

The results of our snippet retrieval is shown in Figure 7. We tried three algorithms for snippet reranking -- cosine similarity, TF-IDF, and Jaccard distance. As discussed earlier, TF-IDF performed the best of the three. We also experimented with changing the threshold on the number of snippets to return. Changing the threshold from 5 snippets to 10 snippets gives a slight improvement for two of the algorithms. However, performance decreases for TF-IDF, which is still overall the best one.

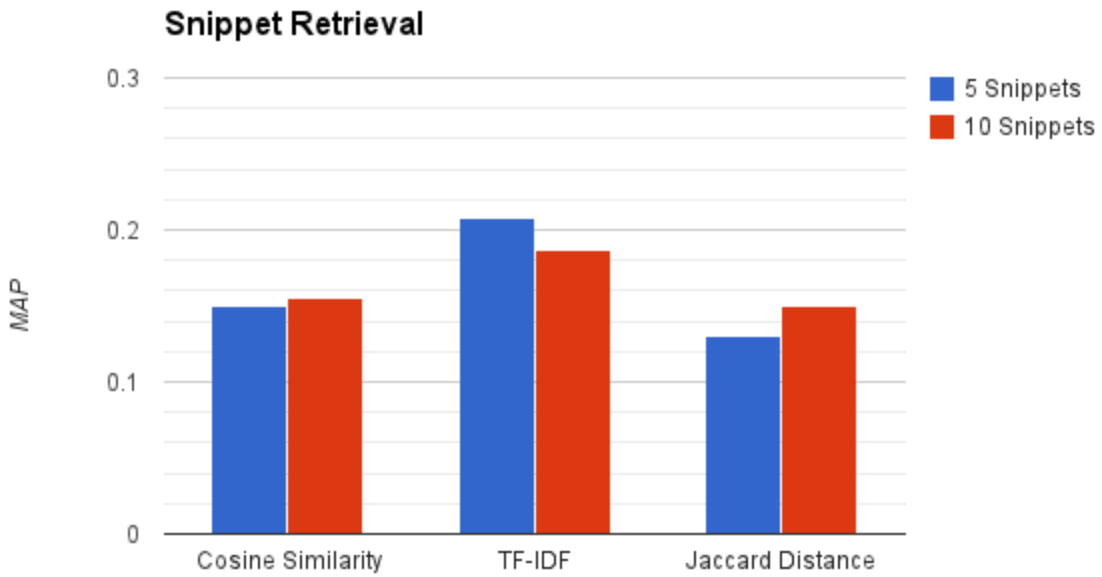

Figure 7. Snippet Retrieval Results

4.4 Exact Answer Generation

With the training data, there were only 8 Yes/No questions. Of these, 6 had an answer of "Yes" and 2 had "No". Our system managed to correctly generate all 8 answers. However, it is

difficult to determine how much of this is actually due to our algorithm for generating answers. For the two questions which should have an answer of “No”, we did not retrieve any snippets -- this, according to our algorithm defaults to an answer of “No”. However, the gold standard does have retrieved snippets for these questions, so it is possible our answer would be different had we retrieved those snippets.

The intuition behind our algorithm is that the snippets we retrieve are in theory relevant to all parts of the question, and therefore if a snippet does not contain all the entities mentioned in the question, it likely cannot indicate an affirmative answer to that question. In looking at the snippets retrieved in the gold standard for the two “No” questions mentioned above (snippets we did not retrieve), we see that they do not seem to contain all the entities in the question. For example, for the question “Are there any desmins present in plants?”, the gold standard snippets contain the entity “desmin”, but none contain the entity “plant”. They mention animals and muscles -- presumably one could infer that desmins are specific to animals rather than plants.

So while these snippets may be relevant to the question, for yes/no questions, we really only care about snippets which are relevant to an affirmative answer for the question. If we cannot find any such snippets, we conclude the answer must be negative.

Once again, however, it must be stated that this is based on only 8 test questions. A more comprehensive test is required to better evaluate our system.

4.5 Final Results for All

Using the best strategies for each step in our process yields the results in Table 3.

Retrieved Items	Precision	Recall	F-measure	MAP	GMAP
Concepts	0.142	0.405	0.178	0.445	0.130
Documents	0.211	0.248	0.186	0.278	0.016
Triples*	0.000	0.000	0.000	0.000	0.001
Snippets	0.093	0.043	0.052	0.208	0.010

Table 3. Final Retrieval Results
*See Section 4.3

Exact Answer Generation : We successfully answered all questions correctly.
Accuracy = 100%

5. Technologies and Third Party Tools

In addition to the web services provided in the assignment, we also used a few other tools:

- StanfordNLP
- ABNER
- LingPipe

Support

The program can work smoothly in our own computers. If you have problems in running the code, please contact cmu11791-01@googlegroups.com.