FALQAN: **Fa**ctoid Retrieva**l** **Q**uestion **A**nswering about CN**N**

James Bannon[1], Paul Fisher[2], Morten Pedersen[3], and Mathew Thomas[4]

[1]jjb509@nyu.edu
[2]pmf296@nyu.edu
[3]mvp288@nyu.edu
[4]mt3443@nyu.edu

Abstract

Question answering systems (QAS) have a long history in Natural Language Processing. Since computers were first able to effectively process text a major goal has been designing systems that could interact with humans and QAS are a specific instance of that kind of system. Modern QAS leverage many of the techniques we have seen this semester, such as syntactic parsing, named entity recognition, and bag-of-words model information retrieval (IR). Because a QAS pipeline provides a fairly panoptic view of the course — and because it seemed like an enjoyable challenge — for our term project we developed a question answer system called FALQAN.

FALQAN is an acronym for **Fa**ctoid Retrieva**l** **Q**uestion **A**nswering about CN**N**. In particular FALQAN is a corpus-based system that can answer 10 types of questions by referencing approximately 90,000 English-language CNN articles published between April 2010 and April 2015. In this report we describe the FALQAN architecture, give a detailed account of each element of the pipeline along with performance evaluations where possible, evaluate the whole system, primarily in terms of Mean Reciprocal Rank (MRR), describe the development process including stumbles, limitations, and outline potential future improvements.

Since the below report is detailed and fairly lengthy a description of the included files as well as a few "in action" screen-shots can be found in the appendices.

NB: The FALQAN zip file was too big for NYU classes and so it has been e-mailed to professor as a zip folder.

1 Introduction

While there are many kinds of questions and answers in every-day discourse the particular task of *factoid question answering* involves taking in a question such as "Who is the President of Brazil?" or "How much does a ticket to Chicago cost?" and returning a short sentence fragment like a noun or noun phrase, such as"Dilma Rousseff", or a number, such as"$400." This task is distinct from more complex, summarization style questions such as "What caused the war?" or "What is this book about?"

Factoid QAS represents a natural extension to typical bag-of-words document retrieval (often called information retrieval or IR) because the answer to a question is often not a full document but rather some small portion of a sentence within that document. This increases the difficulty of the task significantly because not only must appropriate documents be returned in response to a question, the correct piece of information must be extracted from the documents where it exists.

FALQAN is a factoid question answering system[1] that has access to a corpus of approximately 90 thousand documents published on CNN from April 2007 to April 2017 which, for reasons described below, were split into approximately 2.3 million sentences individual *sentences* for searching. From these documents FALQAN can answer 10 types of factoid questions: `Person`, `Location`, `Country`, `City`, `Date`, `Artifact`, `Money`, `Measure`, `Organization`, & `Disease`, which together form a subset of the question types from the TREC8 QA Track. Thus FALQAN is an open domain, closed question class QAS with access to a finite

[1]Strictly speaking, FALQAN is not a Question Answering System in that it does not return a single answer that is either right or wrong. It is a Question Answer Ranking System (QARS) that returns a ranked list of five answer candidates. Since in the literature often the 'R' is assumed and omitted we similarly drop it here and refer to FALQAN as a QAS.

corpus, which means it performs a task analogous to the main task in the early (2008-2004) TREC QA Track.

1.1 FALQAN Pipeline Overview

The FALQAN system is put together with four major components and the workflow/pipeline is illustrated in figure 1. The components consists of a Question Tagger, Information Retriever, Document Ranker and an Answer Extractor. The process, at a high level, is simple. We take in a question, classify it, and then retrieve related documents and rank them based on syntactic and semantic features related to the question type. Lastly, answers are extracted from each of the ranked document and the system returns a list of 10 ranked answers.

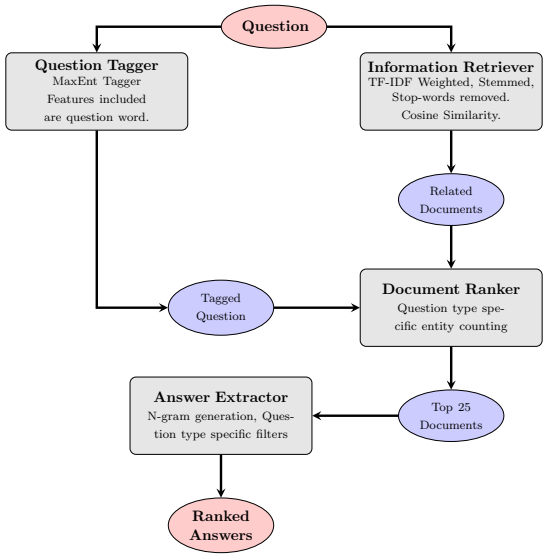

Figure 1: The FALQAN Pipeline. A question is tagged with one of ten questions types while the

FALQAN can answer 10 types of questions. We chose these to be varied, so that the system faced a challenge, but also small in number, so that we would have sufficient training data. Here are brief descriptions of what the question types mean:

1. `Person` a human individual.
2. `Location` states/provinces, mountains, other generic locations.
3. `Country` specifically countries and nations.
4. `City` specifically cities and municipalities
5. `Date` dates of any format; months, years, months and years, days of months, etc.
6. `Artifact` a broad class of miscellaneous nouns, such as inventions, books, movies, currency names, products, and substances.
7. `Money` prices and salaries
8. `Measure` these can be weights, volumes, temperatures, speeds, percentages, numbers, distances, or counts
9. `Organization` (a group or organization of human persons
10. `Disease` questions related to diseases and medicine.

2 Developing FALQAN

In this section we describe our initial attempt, design choices and stumbles in the development of the FALQAN system. After a short introduction, the major components of FALQAN are described in detail.

2.1 Initial Attempt & Pivot

The initial proposal for this project was to build a question answering system that focused on movie data. We ran into some stumbling blocks immediately, chief among them was that the data were were not stored in a linguistically "rich" format, meaning they were in an odd, non-standardized format that would have been difficult to search and nearly impossible to do typical answer extraction on. After spending a week or so on this we discovered the CNN corpus described in the next section and decided to pivot to that for a number of reasons. The main reason was that it would give us a body of text that would be open domain and would allow us to use a full battery of techniques as opposed to ad-hoc, domain-specific search.

2.1.1 The CNN Corpus

A natural question that arises when building a QA system is the choice of data source. In early TREC competitions the data was stored on disks and amounted to several hundred thousand documents. The FALQAN system uses a much more limited corpus of CNN articles compiled by [4]. The corpus consists of roughly 90.000 news articles from the CNN website posted between April 2007 and April 2015. In [4] the authors also compiled a second corpus of roughly 195,000 articles from The Daily Mail website posted during June 2010 and April 2015. For the sake of computation time and rapid prototyping of the FALQAN system. A design choice was made to use only the CNN corpus, Consequently, the FALQAN system is somewhat restricted in terms of topics, answerable questions and time-period. However, with CNN, a frequently visited and well-known news organization that is not domain specific, as a source it is believed that the training corpus will suffice for the testing of FALQAN pipeline prototype.

2.2 Question Ontology & Question Test Data

Again for the sake of rapid prototyping we chose the 10 question types listed above based on the the TREC8 question set and used questions from [6] to make sure we had enough data to train the tagger.

2.3 IR and Answer Generation

These were derived from the descriptions of these topics given in [5]. They involve techniques of ngram generation, cosine similarity, TFIDF weighting, as well as NER and POS tagging.

3 Detailed Description of Components

Here we describe in detail the components mentioned in Section 1.1.

3.1 Question Classification

To classify each question into one of our ten question types, we used `NLTK` to create a Maximum Entropy classifier. To train our classifier we took about 17000 questions from a corpus developed by Xin Li and Dan Roth [6]. This corpus was pre-tagged using their own taxonomy so we developed a mapping between their question tags and the subset of the Trec8 question types that we had decided to support. After screening out questions which did not map to our question types, we had a training corpus of 3200 questions and a test corpus of 300 questions. Before running our final tests, we trained the classifier on the union of the training and test questions.

We constructed a feature set for each question based on the tokens present in the question and the results of running an NLTK part of speech tagger and noun chunker on the question. The most important features were based on the question word present in the question. The question types we hoped to answer required that we be able to answer, at minimum, who-, what-, where-, which-, how-much-, and how-many-questions. Our feature builder cycled through the tokens of the question and looked for words matching these question words. The identity of the question word and the token immediately following it were important features. These two features alone got us above 60 percent accuracy on our test questions. In order to improve our classifier, we needed information about the parts of speech of the words in the question. We tried many features which included POS tags, including features like the POS tag of the word following the question word, as well as combined features featuring tokens and parts of speech. These features did not improve performance. To make our classifier better we had to extract important words from the question. This was done in two ways. First, we simply created features for the presence of words, screening out words that were either too rare or too common. These cutoffs were set through trial and error at 10 and 100 occurrences in the training data. Words that occurred too rarely were assumed to excluded to avoid overfitting to a small amount of data. Words that occurred too often were screened out because they likely did not provide information about the type of the question. Verbs were thought to be particularly informative, so any word tagged by the POS tagger was given as the value of a feature "verbN", where N was the number of the verb in the question. These changes improved the performance on the test set to the mid 70's.

Two more approaches were able to extract more of the structure of the question and the relationship of the important nouns and verbs had to the question word. First, we added a variable to track the 'state' as we looped through the tokens of the question. This allowed us to keep track of what the question word was and whether we had encountered a noun or verb yet after reading the question word. This allowed us to add specific features for the next verb and noun following question words like "What," "Which," "How many," and "How much," which had accounted for most of our errors. Finally, we used the NLTK chunker to detect noun chunks and extract the final noun of each noun chunk. This final noun became a feature itself. These last additions improved the performance on the test set to over 85 percent.

3.2 Document Retrieval

The document retrieval component of FALQAN is based on a simple TF-IDF vector space model.Initially, the stories of the CNN corpus are split into individual sentences and highlight tags were stripped were stripped from each sentence. The individual sentences are loaded by the FALQAN system for further processing before the vector space model is computed. Using the library NLTK [2] for Python , the sentences are converted to lowercase, tokenized, stemmed and stop-words are removed. The preprocessing workflow is illustrated in figure 2.

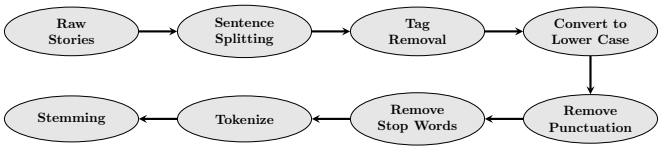

Figure 2: Document preprocessing pipeline required for document retrieval.

With the documents preprocessed, a smoothened TF-IDF weighted vector model is computed using the scikit-Learn library [8], again in Python . The vector model and related data, such as document indexing, are saved as a binary files for subsequent use. The FALQAN system favors precomputed models over recomputing the TF-IDF weights at every system load, and therefore replaces the preprocessing and modeling phase with a simple load function if the vector space model and related data exist. With the vector model ready, document similarity is measured using cosine similarity. The similarity is computed between the question and all individual sentences and the K most similar sentences are passed on to a document ranker, in their original form, i.e. with stop-words, punctuation, etc..

3.3 Document Ranking

Once we retrieve 60 documents from our IR component the documents are then ranked and culled. The ranking is done based on syntactic and semantic features that are related to the question type. For a given document (sentence) d and a question type q let $C_q(d)$ be the number of "correct entities" it contains, $r(d)$ it's original rank in the set of documents returned from the IR system, $K(d)$ the number of words it has in common with the query (stopwords removed) and $N(d)$ be the binary variable corresponding to the event that the document contains a word *not* in the original question. We give the document a new rank $\boldsymbol{R}(d)$ defined by

$$\boldsymbol{R}(d) := \frac{1}{r} + K(d) + N_d + C_q(d) \tag{1}$$

and return the top 25 highest ranking documents. This rank function was heuristically constructions and there are many other similar functions that could have been used, as well as more complex approaches. However, this heuristic approach provided decent results and can be seen as a "first pass" and future iterations of the software would try to improve upon this ranking function.

3.3.1 Calculating $C_q(d)$

The reason for including this term in the above equation was that we wanted to keep documents with as many possible candidate answers for answer extraction and not consider documents without potentially correct candidates.

For `Person`, `Organization`, `City`, `Location` and `Country` questions we ran the sentence through the Stanford CoreNLP NER Tagger (invoked via NLTK) and counted every token tagged as a `PERSON`, `ORGANIZATION`, or `LOCATION` as being an entity of the correct type. Ngrams of entities of the same, correct type were not searched for.

For `Artifact` questions we passed each sentence through the Stanford POS tagger and gave the document a score of +1 for each noun and additional +1 for each proper noun. Since artifacts could include currencies the words in each sentence were checked against a currency name gazetteer and if a match was found the score was increased by +1.25. These score increases were arbitrary and could definitely be improved/tuned in future iterations of FALQAN.

For `Money` and `Disease` questions the count was set to zero as we determined that there were sufficiently many ways to write monetary or medical expressions as to make it impossible to craft hand-written rules that did not introduce heavy bias.

For `Measure` questions the Stanford POS tagger was used to tag each token and each instance of a 'CD' was counted as a correct entity. Next the sentence was checked against a set of hand-written regular expression patterns where a match contributed an additional +1 to the correct entity count.

For `Date` questions we checked the sentence against a set of hand-written regular expression patterns where a match contributed an additional +1 to the correct entity count.

The other terms were included to give some bonus to cosine similarity as well as making sure we brought novel terms into consideration.

3.4 Answer Extraction

Answer extraction consists of a two-stage process: 1) N-gram generation and 2) question-type-specific filters.

3.4.1 N-gram generation

To extract answers we take the list of the top 25 ranked documents from 3.3 and generate all unigrams, bigrams, trigrams, and quadgrams. To each document of rank r in the list of 25 we assigned a score of

$s(d) = \frac{1}{r}$. For every $k-$gram G_k, $k = 1, 2, 3, 4$ in the candidate answers we assigned a score $S(G_k)$ via:

$$S(G_k) := \sum_{d} \sum_{G_i \in d; i \leq k} I(G_i \cap G_k \neq \emptyset) s(d)$$

where I is an indicator function. The intersection is taken word-wise, meaning the ngrams "the quick fox", "the", and "quick fox" all have non-empty intersection.

3.4.2 Question Type Specific Filters

The above step generates a lot of noise in the results, for example 'the' and a number of punctuation points will likely be at the top of the list. To remedy this we applied *post-hoc* filters to ensure correct answers. These filters are not exhaustively listed here but representative examples are given.

For all question types we heavily penalized N-grams containing stopwords or words from the question by giving them a score of -inf. For `Person`, `Organization`, `City`, and `Location` we checked shape features, such as initial capitalization, and set scores to -inf for those N-grams that did not pass these tests. Also, for `Person` questions N-grams of length 1 and 4 were penalized because most people are referred to as FIRST LAST or FIRST MIDDLE LAST.

For `Country` questions candidate answers were compared to a Gazetteer of countries and those that were not the names of countries were given a score of -inf. For `Location` questions if a country was in an N-gram its score was boosted by scaling by 1.5.

For `Artifact` and `Disease` we did no additional filtering because we did not see a way to do it without introducing bias.

For `Money`,`Measure`, and `Date` questions we checked the sentence against a set of hand-written regular expression patterns and for numbers. A match moved the N-gram to the top of the ranked list and no matches of any pattern in an N-gram penalized it.

4 Results & Analysis

To evaluate the performance of FALQAN we asked friends, family, and the class professor to provide us with triples of (question type, question, answer). We received in total 77 questions and 73 question,answer,type triples. After debugging the system we tested it on all 77 questions and the results are shown for the 73 QAT triples in the table in the next section.

As opposed to the TREC competitions we did not have a regex pattern set against which we checked ranked answers. We split the set of questions up to all members and hand checked them. We used two main metrics: mean reciprocal rank (MRR) and the mean partial reciprocal recall (MPRR). To calculate MRR for each question we adopted an exact match or "right or wrong approach" where if the exact correct answer as provided by the external people was ranked r, $r = 1, 2, \ldots, 10$ we scored the system with a reciprocal rank $\frac{1}{r}$. If the answer was not ranked in the top ten we gave it a reciprocal recall of 0. The performance of the system on a whole and within each category was calculated by computing the average of the reciprocal recall scores.

For MPRR we considered the rank of a *partial match* to be the highest rank of an answer with a non-empty word-wise intersection with the exact correct answer. Again, if this was in the top 10 the reciprocal partial rank was $\frac{1}{r}$ no partial answer was there then the reciprocal partial rank was given a value of 0.

The reason for including the MPRR is that the MRR "right or wrong " approach is a harsh metric. This is due to linguistic ambiguity. Two good examples of pathological kinds ambiguity from this point of view are hypernymy and hyponymy. These apply mostly to `Location`, `Measure`, and `Date` questions. As an example, if the question is "Where is NYU Shanghai?" and the test data says the answer should be "Shanghai", if the system returns "China" or "Zhejiang Province" as the top 1 and 2 answers, and the other answers are all not "Shanghai" this will get scored as a zero. However, both those answers are correct because the city of Shanghai stands in a "IS-IN" relation to both those entities. Similarly, "vertebrate" is correct answer to the question "What kind of animal is a cat?" but if the test-answer is "mammal" the system will get a 0 score on this question.

Thus MPRR is a way of being more "generous" to the system while also accounting for the fact that for some questions it is possible for their to be more than one correct answer.

We also tracked the number of questions of each type as well as the number of times, in each type, the correct answer was in the top ten. When a pair (a, b) is recorded a represents the number of questions that had the exact answer in the top ten and b represents the number of questions that had a partial answer in the top 10.

In the table below we also break down the results by question type. In general the system tends to before better in classes where the filtering rules are precise. These classes are `Person`, `Location`, `Money`, `City` and `Organization`. The latter two will be revisited below. We are able to filter out all wrong country answers via a gazetteeer and all wrong person answers via shape and word filters[2]. The fact that these do not have mean reciprocal recall of 1 can be attributed to the fact that the system doesn't show any preference to valid candidate answers with the same shape, and, in the case of persons, the same length. This is because we set valid answers to the maximum non-negative-infinite value meaning they are all "equally good" so the correct answer may be returned second or third but be tied for first in terms of score.

The system performed its worst in the most vague of our question types: `Artifact`, `Disease`, and `Measure`. Since we did not find a way to formulate high-precision rules for these categories answers varied wildly and were incorrect and often nonsensical. Here we could benefit from subtyping the questions into specific kinds of artifacts and measures, which would make more precise rules possible. However, we would need more data to make our classifier more accurate with a finer-grained taxonomy. This points to a broader issue: questions with more heterogeneity in terms of acceptable answers are prone to compounded amounts of ambiguity which makes answer extraction difficult.

Interestingly, `Date` also performed poorly in MRR despite the question being tagged mostly correctly and the reason for this can be due to the fact that we treated all date regex patterns equally and matched sequentially, so "November" and "November 28th, 1985" would be considered equally good matches. Since these have a lot of overlap for exact answers MRR will be low but the MPRR is the highest among all the question types. A solution to this might be to penalize shorter answers to date questions.

The `Organization`and `City` categories are also interesting in-betweens which have high MPRR but low MRR. A possible reason for this is we cannot strip out obvious non-organization names like we can with persons. For example "Jackson Used Books" can be assumed not to be a person but "Fannie Mae" cannot be assumed to not be an organization. Similar to `Date` questions city questions may suffer in MRR by ranking "New" above "New York" even though they tie in score. In the case of "New" over "New York" this is a fair judgment of the system because "New" is not an informative answer about a location.

We must mention the preponderance of `Person` questions in the test set, which since it is one of our better performing categories increases the overall performance of the system. For a more balanced test set it is possible our MRR and MPRR would be much worse.

Also, we kept track of how well the question classifier did. It tagged 53 out of 73 questions correctly, meaning it had an accuracy of approximately %72, which suggests we need more and perhaps more precise training data.

There is a small correlation of approximately 0.44 between the MRR and the percentage of questions classified correctly. While we do not have enough data to test the hypothesis statistically, this makes sense because misclassifying questions changes downstream behavior. The fact that the correlation is not 1 can be attributed to the fact that similar shape features can be used for different kinds of questions and so a misclassification may still yield the correct answer.

Lastly, **the only component of our system that was individually tested during development was the question classifier**. As such we cannot, at this time, attribute errors to particular stages of the pipeline aside from question classification.

[2] i.e: we don't consider "Award" a valid name

4.1 Table of MRR by question type

Question Type	MRR	MPRR	Number of Questions	Top 10 Count	Correctly Tagged
All	0.229	0.333	73	(35,38)	53
Artifact	0.000	0	3	0	1
City	0.167	0.333	3	1	1
Country	0.285	0.367	6	5	2
Date	0.172	0.481	12	(7,9)	11
Disease	0	0.143	7	(0,1)	1
Location	0.449	0.449	7	4	7
Measure	0.022	0.022	5	1	5
Money	0.333	0.333	3	1	2
Organization	0.139	0.371	6	4	4
Person	0.349	0.356	21	12	19

5 Discussion

We consider this project a success; we have put together a working system them performs all of the specified tasks and has them arranged such that their can communicate with each other and generate answers to user questions. We are extremely pleased with the result, and here we discuss its relationship to earlier work and ways in which we think it fell short and could be improved.

5.1 Connections To Earlier Work

Our system is very similar to two early QA systems the AskMSR system built at Microsoft [3] and Aranea[7]. Rather than performing analyses such as POS tagging, Named Entity Extraction and Parsing, etc., for answer extraction these systems relied on the redundancy of information using the world wide web as their knowledge base. The AskMSR system used a combination of components that all dramatically improve the performance compared to using only a subset. The components consisted of Query Reformulation, N-gram filtering and N-gram Tiling. FALQAN only performs one of these steps, N-gram filtering, and in some cases we can see how N-gram tiling would have improved our performance.

The two systems perform a number of operations that FALQAN does not, and some of these are due to the fact that these are web-based search engines. What both AskMSR, Aranea, and FALQAN have in common is the process of filtering the N-grams and re-weighting them according to a measure of how well they matched the expected answer-type. In all three systems the filtering scheme applied was dependent on the question-type assigned to the original query. Having chosen a filter, each N-gram was analyzed for features that were relevant to the filter and scored accordingly. The filters and their relevant features were once again, based on human domain specific knowledge and included features such as capitalization, presence of digits and regular expression patterns.

The final N-gram tiling component, which FALQAN does *not* do, both constructed new more complex answers and merge answers that were similar. The tiling was implemented using a greedy approach. The highest rated candidate answer is, if possible, merged with all subsequent answers. When all combinations have been checked, if one of the newly tiled candidates rates higher than the pre-tiled candidate, then it replaces the pre-tiled candidate as the highest rated and the process is repeated. This continues until no n-grams can be tiled.

5.2 Shortcomings

As with most systems, the first iteration of development rarely yields a perfect product by any measure. In this section we discuss ways in which our system falters and potential methods that can improve the system.

5.2.1 Indexing

As described in Section 2 the FALQAN systems relies on a corpus of CNN articles. In order for the system to consider the full corpus when asked a question, we preprocessed the corpus and saved the TF-IDF weights for later use. Loading the preprocessed weights is heavily favorable over computing the vector space model at system load time. Computing the vector model takes several minutes on a personal laptop and includes processing roughly 95,000 articles split into 2.3 million sentences. Since a large proportion of the documents likely do not contain information related to the query, it may have been smarter to use document indexing *first* and then do sentence splitting because then we are comparing and ranking only 90k documents as opposed to 2.3 million.

5.2.2 Question Tagging

One limitation that we faced is one faced by any NLP system based on a pipeline of components, that errors in various parts of the pipeline compound. We noticed that at times words were not tagged correctly by the `NLTK` POS tagger. This would inevitably affect our performance, since some of our features relied on correctly identifying nouns and verbs in the question. Another limitation may have been the amount of training data we had. More training data may have been helpful because we would have seen more occurrences of various nouns and verbs following each question word. In the future, our classifier could also benefit from using a full parser to dig deeper into the structure of the question. For our purposes, we decided that the possible benefit was not worth the cost in running time and programmer time.

Nevertheless, it would be impossible to build a perfect classifier and there will always be questions whose forms or vocabulary are not represented in the training data. Questions of types outside the taxonomy of the training corpus can give funny results that make sense given what the system is doing. Our favorite result was when, during development, our system classified the question "What is the meaning of life?" and "Disease."

5.3 Potential Future Improvements

In this section we discuss ways we would attempt to improve upon or expand upon our current system given more time, and, in some cases, more data. These improvements are discussed separately but of course combinations are possible. We don't mention it below but the best thing for us to do would be to improve every part of the pipeline as much as possible as errors in one stage can cause large degradations in performance downstream. A good example of this is the fact that a misclassified question can lead to a set of nonsensical answers. For all machine learning techniques mentioned below there is likely a trade-off between runtime performance and accuracy, meaning that while these techniques may improve *accuracy* acquiring the features — NER tags, noun phrase chunking, POS tagging, syntactic or dependency parsing — can be time consuming.

5.3.1 Question Taxonomy/Ontology

Due to time limitations our question type ontology is rather limited. A natural first step to improving FALQAN would be to change our question taxonomy to make it both more flexible by making the taxonomy more fine grained. The obvious candidate for such a taxonomy that in [6] which is already related to our current set, as described in Section 2, and expands on it. This does not have to be the stopping point as there are many kinds of questions not included in the COGCOMP list. Expanding beyond factoid questions, however, would require a considerable amount of work.

5.3.2 Query Reformulation

Query sentences are formulated "as is", meaning we look at the most important parts of words in a sentence and turn those into a vector of unique query tokens. What this fails to do is capture ambiguity inherent in natural language which means we ignore other ways of asking the same thing that might, if

used, enrich and improve the documents returned. A set of simple rewrite rules, like those described in [3], could be used to implement this.

5.3.3 Expanding The Corpus or Adding Web Interface

Searching through 90k documents, when contrasted to searching through the entire Internet, is a trivial task. While our corpus has the virtue of being "open domain" it *of fixed size and membership* meaning that for some questions the answer may *simply not be there*. A first solution is to add documents but as discussed in Section 5.3.4 this could degrade performance. Thus a natural extension for us would be to replace the CNN corpus with the CNN API called NewsGraph[3] or even *any* web search engine.

5.3.4 Improving Runtime Performance

In its current incarnation FALQAN is *SLOW*. There are several things we could do to improve this. As mentioned above in 5.2 document indexing was done on a sentence-by-sentence level which means we performed *exhaustive search* for every query. This is inefficient and only gets worse if we add documents to the current architecture. Thus if we were to keep the current structure and add documents, we would need to find ways to speed up the system.

As a first approach, we could leverage the fact that the FALQAN system is highly parallelizable. After document retrieval the ranker and answer extractor components could easily be run in parallel on splits of the input data, i.e. the TF-IDF weighted vector model to several computational nodes. The partial results the answer extractor yields could then be compared using a final ranking module. The use of a web search engine, as described in the preceding section, could also improve runtime performance.

5.3.5 Machine Learning With Syntactic & Semantic Structure

Our approaches to both question classification and answer extraction employ "low level" or "local" syntactic and semantic features in that they look at the part of speech, NER status, and shape of the tokens in a sentence and, in the case of question classification, syntactic and semantic features of a sliding window of tokens.

We could probably achieve better results by looking at “richer” syntactic that capture the full syntactic structure of a sentence that give us long range dependencies. Recent literature has looked at full parse trees for question classification using tree kernels[9] and support vector machines[10]. By looking at full parse trees we capture question-class-specific features in long-distance dependencies in sentences. Thus, a first step for improving the question classifier could be to implement such a system for our question ontology.

Other work has looked at answer extraction as a *sequence labeling* task[1]. This is an intriguing approach that mitigates the fact that our ngram approach generates a lot of noise, e.g. the trigram "and , the" which is (almost) certainly not the answer to any factoid question. An interesting problem is reacting to scenarios where all the tokens in a sentence are given a tag of `O-ANS`. In [1] the authors use the median absolute deviation (MAD) to count outliers as candidates. A thing we think would be exciting is to replace or augment the MAD approach with high-precision, hand-written “failsafe” regex rules for extracting answers. To our knowledge this latter hybrid approach has not been tried before.

Machine learning for document ranking is also possible. In particular in [5] they discuss that this step is often performed by a supervised machine learning algorithm. Since we could not find, and did not have time to construct, a sufficiently large dataset we could not have done this in the first iteration.

We also could include “off-the-shelf” semantic features — such as WordNet synsets — or automated feature selection algorithms — such as forward selection — into our current system.

[3] https://developer.cnn.com/page

6 Conclusion

For our term project we developed a QARS/QAS called FALQAN that can answer 10 types of questions about approximately 90k CNN articles. FALQAN consists of four major components:

1. **Question Tagger**
2. **Information Retrieval**
3. **Document Ranker**
4. **Answer Extraction and Generation**

Using an externally generated set of test questions we achieved an overall mean reciprocal recall of $\approx 22\%$ using a harsh binary "right or wrong" metric on answers. Relaxing to a "partial match" score we achieved an overall mean reciprocal recall of $\approx 33\%$. We managed to stitch together a functioning and cohesive system that uses many different tools and techniques that we learned about over the semester. Moreover, the results, while not extraordinary, were not abysmal and suggest that FALQAN is a good first prototype of a QAS.

References

[1] Answer extraction as sequence tagging with tree edit distance.

[2] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[3] Eric Brill, Susan Dumais, and Michele Banko. An analysis of the askmsr question-answering system. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 257–264. Association for Computational Linguistics, 2002.

[4] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701, 2015.

[5] Daniel Jurafsky and James H. Martin. *Speech And Language Processing, 2nd. Ed.* "Pearson/Prentice Hall", 2009.

[6] Xin Li and Dan Roth. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics, 2002.

[7] Jimmy Lin. An exploration of the principles underlying redundancy-based factoid question answering.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[9] Jun Sun, Min Zhang, and Chew Lim Tan. Tree sequence kernel for natural language.

[10] Dell Zhang and Wee Sun Lee. Question classification using support vector machines.

Gazetteer and stopword sources:

`https://gist.github.com/kalinchernev/486393efcca01623b18d`
`https://gist.github.com/sebleier/554280`

A Included Files

Here are the files included in the archived `FALQAN.zip` file:

- `FALQAN/`:=Master Folder of FALQAN

- bin/
  * FALQAN.py:= file containing python script to run FALQAN.
  * __pycache__/:=files from loading other modules at execution time.
- docs/
  * README:= File containing instructions for running FALQAN.
  * dependencies:= file that can be used to install required dependencies
  * FALQAN.mov := A movie that shows an execution of FALQAN that is approximately 30 minutes long and answers 5 questions.
  * TestQuestions.csv:= A CSV file with all of our test questions. Unused questions are unnumbered.
  * Results.csv := Raw data from our scoring.
- loads/ := folder containing necessary binary files loaded by FALQAN.
  * FALQAN_DocsDict.pk:= Binary file of the sentences.
  * FALQAN_DocsInd.pk := Binary file of index of documents
  * FALQAN_Model.pk := The vector space model IR system.
  * FALQAN_TFIDF.pk := The TF-IDF weightings of documents.
  * FALQAN_QuestionTagger.pk:= The MaxEnt Question Classifier.
- src/:= Python packages that are used by the FALQAN system.
  * aeparams.py := Hard-coded values loaded & used the by the answer extractor.
  * buildFeatureSet.py:= Code that creates a feature-enriched version of a question sentence.
  * display.py := Contains FALQAN logo.
  * docrank.py := Code that ranks documents returned from the IR module & returns the k highest-ranking ones.
  * drparams.py := Hard-coded values loaded & used the by the document ranker.
  * ExtractAnswers.py := Code to extract answers from a list of ranked documents.
  * genVectorModel.py := Code that implements the IR module.
  * QC_*.txt := Files used to train the question tagger.
  * qtag.py := Code that classifies question sentences.
  * stanford/ := folder containing used Stanford NLP tools used by the document ranker.

B Screenshots