Project Oz:

A Question Answering Information Retrieval System

Anirudh Shrinivason 1004557
Jia Shuyi 1004576
Pheh Jing Jie 1004391

50.045 Information Retrieval Course Project @ SUTD
Project GitHub Repository

Abstract

Project Oz aims to implement a span extraction information retrieval system for question answering using the **S**tanford **Q**uestion **A**nswering **D**ataset (`SQuAD`). Our implementation of Okapi-BM25 outperformed other extractive models like vector space model and language model. Project Oz's end-to-end span extraction pipeline is capable of extracting exact answers for a given query with decent performance.

Contents

1 Introduction

Question answering is a sophisticated task in the fields of information retrieval (IR) and natural language processing (NLP). A question-answering system is concerned with developing systems that automatically respond to human-posed questions in natural language, and it is typically designed to retrieve answers from a *structured* database or an *unstructured* collection of natural language documents.

In this IR project, we built a span extraction IR system for question answering using the **S**tanford **Qu**estion **A**nswering **D**ataset (`SQuAD`) [1]. Specifically, we explored various extractive techniques to reduce the search space of relevant documents, and utilized pre-trained neural network-based models for span extraction.

1.1 `SQuAD` Overview

The **S**tanford **Qu**estion **A**nswering **D**ataset (`SQuAD`) [1] is a reading comprehension dataset distributed by Stanford University. In this project, we utilized `SQuAD2.0`, which consists of 100,000 answerable questions and over 50,000 unanswerable questions written adversarially by crowdworkers to look similar to answerable ones. To do well on `SQuAD2.0`, the system not only has to answer questions whenever possible, but also refrain itself from answering unanswerable questions.

The Hugging Face 🤗 version of `SQuAD2.0` has 130,319 examples in the training set and 11,873 examples in the validation set. Each example has the following 5 features:

No.	Feature	Example
1	id	56be85543aeaaa14008c9063
2	title	Beyoncé
3	text	Beyoncé Giselle Knowles-Carter [...] is an American singer, songwriter, [...]
4	question	When did Beyonce start becoming popular?
5	answer	in the late 1990s; [269]

Table 1: A selected example of the `SQuAD2.0` dataset. Each example has five features, as indicated in the table.

1.2 Overall Framework

Given the various IR techniques taught in the course[1], we decided to build a question answering system based on span extraction as shown in Fig. 1.

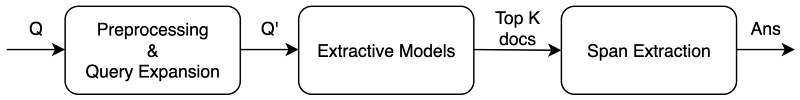

Figure 1: End-to-end view of our span extraction system for question answering.

[1]SUTD 50.045 Information Retrieval, see `https://istd.sutd.edu.sg/undergraduate/courses/50045-information-retrieval`

As depicted in Fig. 1, we obtain a new query Q' after pre-processing and expanding a query Q. This query Q' is fed into an extractive model, which returns the top K ranked documents. These K documents are then fed into a span extraction model to extract any plausible answers.

In the following sections, we will discuss the implementation details for individual components in Fig. 1.

2 Pre-processing

We implement pre-processing of both queries and documents in order to convert them to the same format, and remove unnecessary elements such as white spaces and certain punctuation marks. Our pre-processing implementation consists primarily of the following:

1. **Punctuation removal**: we remove all punctuation marks from the query and all passages.
2. **Case folding**: we make sure that the query and all passages have the same case in our documents.
3. **Remove white spaces**: we remove any trailing white spaces in the passages as well as query.
4. **Tokenization**: we convert the query and passages from strings to a list of tokens.

For example, the sentence

```
"What is the name of Harry Potter's owl?"
```

is pre-processed to become the following:

```
['what', 'is', 'the', 'name', 'of', 'harry', 'potters', 'owl']
```

2.1 Query Expansion

Query expansion (QE) is the process of reformulating a given query to improve retrieval performance. We implemented query expansion with the help of the `NLTK` library [2] by making use of their WordNet [3] implementation. By using WordNet, we only have to implement the tokenization step without the need for a separate stemming/lemmatization method as WordNet can take a sentence and give synsets irrespective of the word form i.e. having "dog" or "dogs" will give us same synsets. Our implementation of query expansion mainly consists of the following steps:

1. **Preprocessing**: the main steps in this preprocessing stage is the tokenization of the query using WordNet.
2. **Key term identification**: The top k terms with the highest idf value are selected from the query to perform expansion on.
3. **Part of speech tagging (POS)**: This step is mainly related to the synsets in WordNet. The POS tag for a word may depend upon the context which changes the meaning of the word, hence, there are different synsets for the same word depending upon the POS tag it has in the sentence.

For example, the sentence

```
"What is the name of Harry Potter's owl?"
```

undergoes expansion to become the following:

```
"what is the name of harry potters owl name language unit repute family important
person sanction defamation"
```

3 Extractive Models

3.1 Vector Space Models (VSM)

The representation of a set of documents as vectors in a common vector space is known as the **vector space model** (VSM) [4] and is fundamental to an array of information retrieval (IR) operations such as scoring documents on a query.

Let $\mathcal{D} := \{d_i\}_{i=1}^{N}$ be a collection of N documents and Ψ be our VSM model. Then we have:

$$\Psi\left(d_i\right) = \vec{d_i}, \qquad \vec{d_i} \in \mathbb{R}^m, \tag{1}$$

where Ψ maps an input document d_i to a m-dimensional vector. Similarly, for a given query q, we have

$$\Psi(q) = \vec{q}, \qquad \vec{q} \in \mathbb{R}^m. \tag{2}$$

Specifically, the VSM model Ψ maps all N documents to N m-dimensional vectors. For a vectorized query $\Psi(q) = \vec{q}$, we can then score all the documents by computing a similarity value $\phi(\vec{d_i}, \vec{q_i})$ through a similarity function ϕ.

In this project, we implemented two vectorization techniques:

1. TF-IDF (term frequency–inverse document frequency),
2. Doc2Vec,

and two similarity functions:

1. Cosine similarity,
2. Jaccard similarity.

3.1.1 TF-IDF

The TF-IDF is the product of two statistics, term frequency (TF) and inverse document frequency (IDF):

$$\mathrm{TF}(t, d_i) = \frac{f_{t,d_i}}{\sum_{t' \in d_i} f_{t',d_i}}, \tag{3}$$

$$\mathrm{IDF}(t, \mathcal{D}) = \log \frac{N}{|\{d_i \in \mathcal{D} : t \in d_i\}|}, \tag{4}$$

where t is a particular term in Document d_i, f_{t,d_i} is the frequency count of t in d_i, $\mathcal{D}$ is the collection of all documents and $N = |\mathcal{D}|$.

Implementation Details

We used `scikit-learn`'s `TfidfVectorizer` to map string documents to vectors. The default output vector $\vec{d_i}$ is a $|V|$-dimensional vector, where V is the vocabulary size of $\mathcal{D}$. Since this vector is sparse and too large to work with, we decided to select only the top 500 features. Furthermore, we also applied dimensionality reduction using truncated Singular Value Decomposition (`TruncatedSVD`). The final dimension of our vectorized output is $m = 40$.

3.1.2 Doc2Vec

Doc2Vec [5] provides a vectorized representation given a document d_i. It is a generalization from the Word2Vec model introduced in 2013.

Implementation Details

We used `gensim`'s `doc2vec` with a vector size of 40, which is trained on the entire document collection $\mathcal{D}$ for 30 epochs. We then used this trained model to generate vectorized representations for all N documents in $\mathcal{D}$.

3.1.3 Similarity Function ϕ

Cosine Similarity

To score the similarity between a vectorized document $\vec{d_i}$ and a query $\vec{q}$, we can calculate the angle θ of deviation between them. Specifically, the cosine similarity is defined as:

$$\phi_{\text{cosine}} := \cos\theta = \frac{\vec{d_i} \cdot \vec{q}}{\|\vec{d_i}\|\|\vec{q}\|} \tag{5}$$

Jaccard Similarity

The Jaccard similarity is calculated by dividing the number of observations in both sets by the number of observations in either set. Mathematically, we have

$$\phi_{\text{jaccard}} := J(\vec{d_i}, \vec{q}) = \frac{|\vec{d_i} \cap \vec{q}|}{|\vec{d_i} \cup \vec{q}|} \tag{6}$$

In practice, since the values of elements in a vector (e.g. $\vec{d_i}$) is very small, we scaled them up by a factor of 5 and rounded them up to the nearest integer, before using Eqn. 6.

3.2 Okapi-BM25

Okapi-BM25 [6] is one of the strongest scoring functions, and has proven a useful baseline for ranking documents. It predicts the relevance of documents to an input query and is based on probabilistic frameworks. The ranking function factors heavily in the term frequency of each document and the document length and it disregards the terms' proximity within the document. For a given document-query pair (d_i, q), we have

$$\text{score}(d_i, q) = \sum_{j=1}^{n} \text{IDF}\left(q_j\right) \cdot \left[\frac{f_{q_j,d_i} \cdot (k_1 + 1)}{f_{q_j,d_i} + k_1 \cdot \left(1 - b + b \cdot \frac{|d_i|}{l_{\text{avg}}}\right)} \right], \tag{7}$$

where q_j is the j-th term in q, f_{q_j,d_i} the frequency count of term q_j in d_i and l_{avg} the average document length. The hypper-parameters are k_1 and b, which are initialized to 1.5 and 0.5 respectively.

3.2.1 Implementation

Our implementation of BM25 makes use of an index to reduce computation time, and make the extraction process more efficient. The BM25 index is represented in the `BM25Index` class.

The BM25 index structure is shown in Listing 1. It is implemented as a dictionary, with a term as the key, and the value being a dictionary, further keyed by the document id that the word appears in having the value as the term frequency of the word in the document.

```
{
    "term1": {
        "DocID1": "term_frequency",
      ...
  },
    "term2": {
        "DocID1": "term_frequency",
      ...
  },
      ...
}
```

Listing 1: Data structure for `BM25Index`

This index structure is extremely useful, and can be used effectively to compute term frequency, document frequency by simply calculating the length of the inner nested dictionary[2], and inverse document frequency which can in turn be computed from the document frequency.

The index is then fed into the main `BM25` class, which contains a method `score_docs` that calculates the BM25 score for all (d_i, q) pair, and returns the top K documents.

3.3 Language Model

Language modeling is the way of determining the probability of any sequence of words. Language models are very useful for a wide range of applications, e.g., speech recognition and machine translation. A language model can be defined as following [4]:

$$P(x_1, x_2, \dots, x_m) = \prod_{i=1}^{m} P(x_i | g(C_{i-1})), \tag{8}$$

[2]Document frequency can be calculated by `len(BM25Index[word])`.

where $g(C_{i-1})$ is the learned context C_{i-1} by function g.

Using Eqn. 8, we can calculate the probability of a query $q = x_1, x_2, \ldots, x_m$ appearing in a given language model built from a particular document d_i. Naturally, the document d_i that returns the highest probability of a query q is most relevant to q. In this project, we explored n-gram language models.

3.3.1 n-gram Language Models

For n-gram language model, we consider the previous $n-1$ words as the context:

$$g(C_{i-1}) = x_{i-n+1}, x_{i-n+2}, \ldots, x_{i-1}. \tag{9}$$

As an illustrative example, to compute a particular **tri-gram** probability of a word x_i given $f(C_{i-1}) = x_{i-2}, x_{i-1}$, we do

$$P(x_i|x_{i-2}, x_{i-1}) = \frac{\text{count}\,(x_{i-2}, x_{i-1}, x_i)}{\text{count}\,(x_{i-2}, x_{i-1})} \tag{10}$$

In this project, we implemented the unigram, bigram and trigram language models (LMs) for the extraction process.

3.3.2 Laplace Smoothing

Laplace smoothing is a smoothing technique that helps tackle the problem of zero probability in the n-gram language model. This k-smoothing technique is implemented by adding a scalar k to the numerator, and $k \times |V|$ to the denominator of the n-gram probability.

3.3.3 Interpolated n-gram Model

The interpolated n-gram model essentially consists of a linear combination of the unigram, bigram and trigram language models. Interpolation of the n-grams help to generalize better and produces more accurate results. The interpolation n-gram model is defined as following:

$$P(x_1, x_2, \ldots, x_m) = \prod_{i=1}^{m} \left(\lambda_1 \frac{\text{count}(x_i)}{N} + \lambda_2 \frac{\text{count}(x_{i-1}, x_i)}{\text{count}(x_{i-1})} + \lambda_3 \frac{\text{count}\,(x_{i-2}, x_{i-1}, x_i)}{\text{count}\,(x_{i-2}, x_{i-1})} \right), \tag{11}$$

where $\lambda_1 + \lambda_2 + \lambda 3 = 1$ are the weighting factors for each individual language model.

3.4 Evaluation

The information retrieval system evaluation revolves around the notion of relevant and non-relevant documents. We have implemented our own evaluation metric to measure and quantify the various extraction models.

3.4.1 Evaluation Techniques

Common evaluation techniques used for quantifying the performance of retrieval models include MAP or Precision@K. However, due to the structure of our dataset, we are unable to resort to these evaluation metrics. The `SQuAD` dataset consists of a many-to-one relation between the queries and the documents ie. many queries have the same relevant document. Since there is only one relevant document for each query, Precision@K will always be $1/k$ or 0 for all the extractive models. Under this context, Precision@K might not be the best metric for the purpose of evaluation. Therefore, we implemented average rank metric to quantify model performance:

$$\text{Rank} := \text{rank of } d_i \in \mathcal{M}(q_i), \tag{12}$$

$$\text{Rank}_{\text{avg}} := \frac{1}{n} \sum \text{Rank}\left(d_i, \mathcal{M}(q_i)\right), \tag{13}$$

x

where $\mathcal{M}$ is an extractive model that returns a list of ranked documents.

3.4.2 Evaluation Results

Models			Average Rank
VSM	TF-IDF	Cosine Similarity	149
	Doc2Vec	Cosine Similarity	214
	TF-IDF	Jaccard Similarity	383
	Doc2Vec	Jaccard Similarity	592
BM25	With Query Expansion		10.1
	Without Query Expansion		**9.22**
N-gram LM	Interpolated Language Model		604

Table 2: Evaluation results for all extractive models using average rank (the lower the better).

As can be seen from Table 2, BM25 performed the best out of all the other extractive models over a significant margin. The interpolated n-gram model was the least efficient for document ranking. Therefore, BM25 is the selected choice of extractive model to use in our end-to-end IR pipeline, as depicted in Fig. 1.

4 Span Extraction

In this part of the project, we have fine-tuned a pre-trained Transformer model from Hugging Face 🤗 [7] to a question answering task. Specifically, the model extracts answers (sub-strings) from an array of input documents $\mathcal{D}' = \{d_i\}_{i=1}^{K}$, which is generated by our extractive models such as Okapi-BM25.

4.1 Pre-processing

We first tokenized the documents using `transformer.AutoTokenizer` with pre-trained `distilbert-base-uncased`. Since the document length can be relatively long, each document is divided into chunks of length of 384 tokens. To account for the possibility that an answer spans across two chunks of texts, a third chunk which overlaps both chunks is created.

4.2 Fine-tuning

We instantiated a `AutoModelForQuestionAnswering` model from the `transformers` library using pre-trained weights. To fine-tune the model on our tasks, we used `transformers.Trainer` to perform the learning. The trainer ran 5 epochs with a learning rate of `2e-5`. It is worth noting that the model is trained on the entire `SQuAD2.0` dataset.

4.3 Evaluation Results

We used `load_metric` from the `datasets` package to evaluate the performance of our fine-tuned Transformer model. The results are shown in Table 3.

Metric	Score
Exact	63.3
F1	66.9
Exact (has ans)	66.0
F1 (has ans)	73.1
Exact (no ans)	60.6
F1 (no ans)	63.6

Table 3: Performance of the fine-tuned Transformer model for question answering.

4.4 End-to-end Demonstration

Here is a sample output of our question answering IR system.

- **Question**: `Who determined that the air quality was unhealthy on those 4 days?`
- **Ground truth**: `THE EPA`

To conduct span extraction, we duplicated the query and pair it with each document from the top K documents returned by the extractive models such as Okapi-BM25. The results are shown in Table 4.

No.	BM25		VSM		LM	
	DocID	Answer	DocID	Answer	DocID	Answer
1	2986	the epa	18185	-	2461	-
2	8777	-	9700	mahzer mahmood	2615	-
3	8735	-	992	-	3770	-
4	10969	-	1777	-	2450	-
5	5498	-	2791	-	2908	-
6	3387	nanjing	16494	mitscher	4938	-
7	13784	-	12832	-	2924	catalan
8	17129	-	10736	-	4346	-
9	15396	the luftwaffe	11506	-	1382	-
10	13020	-	2313	elisha gray	2197	-

Table 4: Results returned by the span extraction model. BM25, VSM (TF-IDF with cosine similarity) and LM (interpolation) are used to extract the top 10 documents. We see that the BM25 model is able to return the correct document ID as the top document retrieved and our Transformer model is also able to extract the ground truth answer `the epa`.

5 Challenges

The `SQuAD2.0` dataset consists of a many-to-one relation between the queries and the documents. This is particularly inconvenient as there is only one relevant document for a query that can be obtained from the dataset. This structure makes it difficult to implement Precision@K evaluation metric, as it is not feasible to calculate relevance of the extracted k documents from the models. Due to this dataset structure, we were also unable to implement a pseudo-relevance feedback mechanism for the extractive models. This limitation was overcome by using average rank as an evaluation metric, and implementing various extractive models without the pseudo-relevance feedback instead.

6 Conclusion

In conclusion, this project demonstrated the implementation of a question answering system that consists of various extractive models such as the vector space model (VSM), Okapi-BM25, language model (LM), and neural network-based span extraction. The results shown in Table 2 showed that BM25 performed significantly better than VSM and LM with the Average Rank of ≈ 10. BM25 is thus selected to be used in the end-to-end pipeline with Transformer span extraction.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[2] Edward Loper and Steven Bird. Nltk: The natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[3] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[4] Christopher D Manning. *Introduction to information retrieval.* Syngress Publishing,, 2008.

[5] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[6] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.

[7] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.