Query Expansion Using Word Embeddings for Ad Hoc Search

Angelos Sfakianos
Radboud University
angelos.sfakianos@ru.nl

Rajkumar Matariya
Radboud University
rajkumar.matariya@ru.nl

Yashika Sorathia
Radboud University
yashika.sorathia@ru.nl

1 ABSTRACT

In this paper we are going to use the Word2vec word embedding technique. Word2vec detects synonymous words for incomplete sentences. We will test whether or not it is possible to improve the effectiveness of ad hoc search. By using TREC Robust 2004 data set, we will check the above hypothesis for results, while at the same time we will be looking for limitations and drawbacks for the selected strategy. We present results on recall, precision, cumulative gain(normalized and non) ,mean average precision and mean reciprocal rank. Our practical goal was to use query expansion to see if we can improve the results of the queries, while also testing a variety of different ways to form the query.

2 KEYWORDS

Python, Word2Vec, Query Expansion, Information Retrieval

3 INTRODUCTION

Over the last few years, the search engines have become more vast and complex than ever. From searching for articles, to using images as the query subject, the world of information retrieval is exponentially growing. Our goal in this paper is to find a way to utilize certain smart features of the Word2vec technique, in order to enhance query search effectiveness.

We experiment on an existing dataset, and by applying the unified search tool created, we plan to test the important variables of the information retrieval workspace. Query expansions, by definition, are designed to find words that are semantically connected to the existing query ,and therefore improve the results of the search in various categories. Our approach is to utilize the Word2Vec technique to improve the query expansion on top of the default query results.In a more a direct way, we try to answer the following questions:

- Can we improve retrieval effectiveness by using Word2vec on the query expansion?
- Can we find limitations or conditions that improvement is not guaranteed?
- Can we find specific categories of queries where our technique works optimally?

Word2vec is a technique for natural language processing, like search queries for internet users. The idea lies on the network used by word2vec, which is used by the algorithm to learn and link words with synonyms, and then use that knowledge for queries, in order to expand the results while also making suggestions on incomplete sentences. While the methodology will be analyzed thoroughly later, the main idea, as the name word2vec indicates, is that every word is presented as a vector, which contains numbers, in order to convert punctual connections between words with similar meanings. The initial source of knowledge is a text corpus.

4 RELATED WORK

A lot of techniques have been developed for reformulating the queries to fulfill the queries of the users. These techniques have indeed provided efficient in dealing with the difficult queries. However, these techniques still impose certain limitations because of the classical approach of using term co-occurrence [6].

Majority of the query expansion approaches are use some contextual resources to extract the semantically proper expansion term. One such approach was using the Latent Semantic Analyses (LSA). Btihal El Ghali [2] and the team perfrom query expansion on short web queries using LSA. They used search engine query logs to extract the contextual terms. However, LSA model was computationally expensive and would affect the efficiency of the model with large size of data.

To overcome the limitations of the LSA, Tomas Mikolov [5] introduces two models that preserves the liner relationships in the embeddings. Today, this model is known as Word2Vec and the two models are CBOW (Continuous Bag Of Words) and SG (Skip-Grams). The models were computationally efficient irrespective the size of vocabulary used for training the model. As a result of this, qualitative embeddings were generated with more amount of information.

Kuzi et al [4] studied automatic query expansion by using feature extraction technique, which is based on the centroid method.This method calculates the average of the word embedddings of the words and consturcts a vector representation of the query. This representation is called AWE (Average Word Embeddings). However, there a limitation of this method that equal importance is given all the terms in the query.

5 METHODS

5.1 Ranking

Okapi BM25, is a ranking function used to generate the relevant score of the document to a given search query. BM25 is a bag of words retrieval functions than generates ranking for the documents based on the query terms appearing in the document [8].

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) = \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D)+1\cdot\left(1-b+b\cdot\frac{|D|}{angdl}\right)} \tag{1}$$

5.2 Word Embeddings

Word Embeddings is one of the most popular representation of text in NLP. In contrast to bag-of-words, word embeddings is capable of of capturing the both syntactic and semantic relations between the words. The embedding-based retrieval models could tackle the vocabulary mismatch problem by making use of the embedding's inherent similarity between distinct words, most of them struggle to compete with the prevalent strong baselines like BM25 and TF-IDF [3].

Word2Vec is a commonly used technique for grouping word with similar contextual meanings together and represent them in a vector space. There are two main learning algorithms proposed by Tomas Mikolov [5]: Skip-grams and CBOW (Continuous Bag Of Words), however our main focus is using CBOW.

In CBOW architecture, the current word is predicted using the distributed representation of context. While, in Skip-Grams architecture, the surrounding words are predicted using the current word.

5.3 Query Expansion

There query expansion has been proved very efficient in retrieving relevant documents for user search. Initially the queries are pre processed and tokenised in order to perform word embeddings. A similarity scores of the word vectors were generated by using Cosine Similarity equation 2.

$$similarity = \cos(\theta) = \frac{\bar{X} \cdot \bar{Y}}{\|\bar{X}\|\|\bar{Y}\|} \tag{2}$$

Based on these similarity score, we selected words with the highest score and embedded into the original query.

5.4 Evaluation

We used different rank based retrieval evaluation metrics to evaluate our model. The methods used are:

Precision@K

It determines the fraction of top K documents retrieved that are relevant to the user's need. Precision can be calculated using below equation:

$$\text{Precision@}K(q) = \frac{1}{K}\sum_{i=1}^{K} r(q,i) \tag{3}$$

Recall@K

It determines the fraction of top K documents that are relevant and are retrieved successfully.Recall can be calculated using below equation:

$$\text{Recall@}K(q) = \frac{\sum_{i=1}^{K} r(q,i)}{\sum_{j=1}^{D_q} r(q,j)} \tag{4}$$

F-score@K

It is a harmonic mean of Precision and Recall. It penalises the cases with high values of one of them. F-score can be calculated using below equation:

$$\text{F-Score@}K(q) = \frac{2 \cdot \text{Precision@}K \cdot \text{Recall@}K}{(\text{Precision@}K + \text{Recall@}K)} \tag{5}$$

Mean Reciprocal Rank

We are using this metrics since we want our model to return an item that is at highest rank and is the best relevant item for the search that has been performed. MRR can be calculated using below equation:

$$\text{Reciprocal Rank}(q) = \frac{1}{\min\{i : r(q,i) = 1\}} \tag{6}$$

6 EXPERIMENTAL SETUP

6.1 Dataset

Any Information retrieval dataset has 3 components to it. First, document corpus. It is nothing but collection of text. Second, query set. Query set contain information like query ID, title, description and narrative. Title is nothing but main query for search, description is an extensive title and narrative is detail about what should retrieved document contain. Last but not least, qrels set. Qrels set contains information about relevance of document with respect to query ID. We also have same structure for our dataset.

Robust04[7] represents the test collection from the TREC 2004 Robust Track. It comprises 250 queries, with relevance judgments on a collection of 528K documents (TREC Disks 4 and 5) with 17,45,40,872 terms out of which 9,23,436 are unique, whose average length is 2,800 characters or 460 words. This document set includes material from the Financial Times Limited (1991, 1992, 1993, 1994), the Congressional Record of the 103rd Congress (1993), the Federal Register (1994), the Foreign Broadcast Information Service (1996), and the Los Angeles Times (1989, 1990). We use the topic "titles" (short keyword phrases, much like the input to a search engine) as queries. Our Qrels set have 3,11,410 entry with fields like $query_id, doc_id, relevance$.

6.2 Data Pre-processing

The foremost step is cleaning and pre processing of raw data/text. Therefore, queries were converted into a consistent format, which is simpler to process. Other pre-processing steps that were used are as follow:

6.2.1 Tokenization. Tokenization is breaking the raw text into small chunks. Tokenization breaks the raw text into words, sentences called tokens. These tokens help in understanding the context or developing the model for the NLP. The tokenization helps in interpreting the meaning of the text by analyzing the sequence of the words.
For example, the text "Flash live in Central city" can be tokenized into 'Flash', 'live', 'in', 'Central', 'city'

6.2.2 Removing stop words. Stopwords are words in any language or corpus that appear frequently. Some NLP tasks do not provide additional or valuable information in the text they contain. Words such as a, she, who, is, an etc. are usually considered stop words.

6.2.3 Removing punctuation. One of the other text processing techniques is removing punctuations. there are total 32 main punctuations that need to be taken care of. As they can interfere with feature extraction so it is a good idea to remove them as well.

6.2.4 Making terms lower-cased. Lowercasing ALL your text data, although commonly overlooked, is one of the simplest and most effective form of text preprocessing. It significantly helps with consistency of expected output.

6.3 Baselines

We have used the baseline that is prepared by the Anserini Team. It is a simple search that uses a traditional query expansion model RM3. Documents are indexed and retrieved with the lambda value of 0.6 and changed the number of expansion terms to 10. Result for the baseline is as per Table 1.

precision@10	recall@10	F-score@10	MRR
0.420	0.86	0.517928	0.633548

Table 1: Baseline score

6.4 Model Training

We used the pre-processed data from the previous step to train a CBOW word2vec model. The training parameters were set as follows. We used a window size equal to 2, the vectors dimension is 100, $min_countis1$ and epochs to 100.

We came up with 4 different approaches for our experiment and trained the model accordingly. Detail for each is given below:

With only Title
We took text from only title column from our query dataset and trained model on it. We will refer this as T-model.

With Title and Description
We took text from title and desciption column from our query dataset and trained model on it. We will refer this as TD-model.

With all
We took text from title, description and narration column from our query dataset and trained model on it. We will refer this as A-model.

With Google News
For this approach we wanted to train model with Google News dataset which contain about 100 billion words. To train search a huge dataset we need highend hardware. As we didn't had access to a good hardware we used pre-trained model provided by google[1], which contains 300-dimensional vectors for 3 million words and phrases. We will refer this as G-model.

6.5 Query Expansion

For every word in the query we searched for similar word in the model dictionary using "$model.wv.most_similar$" method from gensim package. Refer section 5.3 for more detail. Then, we took the word with highest similarity score and added to the query which becomes new query.

Now we have 4 models T-model, TD-model, A-Model and G-model, we repeat same process to generate new queries based on respective model. So at this point we have total 5 type of queries.

(1) Original Query
(2) Expanded Query using T-model
(3) Expanded Query using TD-model
(4) Expanded Query using A-model
(5) Expanded Query using G-model

6.6 Documents Ranking

We used Pyserini Information Retrieval platform. Pyserini is primarily designed to provide effective, reproducible, and easy-to-use first-stage retrieval in a multi-stage ranking architecture. Our toolkit is self-contained as a standard Python package and comes with queries, relevance judgments, pre-built indexes, and evaluation scripts for many commonly used IR test collections[1]. We used pre-existing index provided by pyserini for robust04 datatset[2]. We opted to use the well-established BM25 probabilistic weighting scheme with default parameters. For each run, the top 10 documents retrieved for each expanded topic were submitted, in descending order by score.

6.7 Evaluation of ranking

As described in section 5.4 we will use precision, recall, F-score@10 and MRR as matrices for evaluation but to use any matrix we need ground truth. For ground truth we used qrels from TREC[3]. Our simplicity we changed relevance category from 0,1,2, where 0 being no relevance, 1 relevant and 2 highly relevant, to 0,1 we converted 2 to 1.

7 RESULTS AND ANALYSIS

We can see that while precision dropped when we added just the title to the query expansion, when we added more information to it, such as name and description or description and narration, the precision did increase, but it was still lower than the original one. Similarly, recall did decrease on the query expansion with title, but then goes back up and even suprases the original query recall by a bit.

Detailed result for our experiment is mentioned in the table2.

Model	P@10	R@10	F@10	MRR
RM2	0.420	0.86	0.517928	0.633548
T-model	0.254	0.68	0.343845	0.410770
TD-model	0.372	**0.90**	0.487726	0.651333
A-model	0.334	0.86	0.447017	0.608000
G-model	0.388	0.86	0.49487	0.639

Table 2: Score

As we can see TD-model has best recall of 0.86. It is the model which is trained on query data like title and description. Each data point has strong relation between each other as description is nothing but extension of title. Due to this relation, embedded word used for query expansion was more likely to be present in the search documents as well.

[1]See the https://code.google.com/archive/p/word2vec

[2]See the https://github.com/castorini/pyserini/blob/master/docs/prebuilt-indexes.md

[3]See https://trec.nist.gov/data/robust/qrels.robust2004.txt

We expected G-model to out perform all the model but interestingly it didn't even though it was trained on news data. One of the reason for this behaviour might be data on which model was train. Data used was from google news which was collected from million of articles which are not part of document which needed to be search due to which there is possibility that top similar word might or might not be part of our search corpus which is not case in our best performing model i.e TD-model with recall of 0.9. G-model can be a really good model if our task was to predict next word in query or any text suggestions application.

Based on all the observations we can say effectiveness of the retrieval model can be increased by expanding query using Word2Vec trained on documents on which search needs to be performed in terms MRR. As main function of any search retrieval system is to match query's words and provide relevant document, so by embedding extra word which are similar to the words in the query helps to improve model.

Futhermore, we did discovered few limitation in our approach. The model does does not perform better with other dataset. We used pre-trained word2vec model of Google which was trained on Google News dataset and as described earlier it didn't out performed the TD-model even though it had very huge dataset.

REFERENCES

[1] castorini. [n.d.]. GitHub - castorini/pyserini: Pyserini is a Python toolkit for reproducible information retrieval research with sparse and dense representations. https://github.com/castorini/pyserini

[2] Btihal El Ghali, Abderrahim El Qadi, Mohamed Ouadou, and Driss Aboutajdine. 2015. Context-Based Query Expansion Method for Short Queries Using Latent Semantic Analyses. In *Networked Systems*, Ahmed Bouajjani and Hugues Fauconnier (Eds.). Springer International Publishing, Cham, 468–473.

[3] Lukas Galke, Ahmed Saleh, and Ansgar Scherp. 2017. Word Embeddings for Practical Information Retrieval. In *INFORMATIK 2017*, Maximilian Eibl and Martin Gaedke (Eds.). Gesellschaft für Informatik, Bonn, 2155–2167. https://doi.org/10.18420/in2017_215

[4] Saar Kuzi, Anna Shtok, and Oren Kurland. 2016. Query expansion using word embeddings. In *Proceedings of the 25th ACM international on conference on information and knowledge management.* 1929–1932.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[6] Helen J Peat and Peter Willett. 1991. The limitations of term co-occurrence data for query expansion in document retrieval systems. *Journal of the american society for information science* 42, 5 (1991), 378–383.

[7] Ellen M Voorhees et al. 2004. Overview of TREC 2004.. In *Proceedings of the Thirteenth Text REtrieval Conference.* 52–69.

[8] Wikipedia contributors. 2021. Okapi BM25 — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Okapi_BM25&oldid=1008742667 [Online; accessed 24-December-2021].