Word Embeddings in Search Engines, Quality Evaluation

Bachelor Thesis

Eneko Pinzolas Murua

University of Freiburg
Faculty of Engineering
Department of Computer Science
31st of May, 2017 - 19th of September, 2017

Reviewer: Prof. Dr. Hannah Bast
Supervisor: Prof. Dr. Hannah Bast

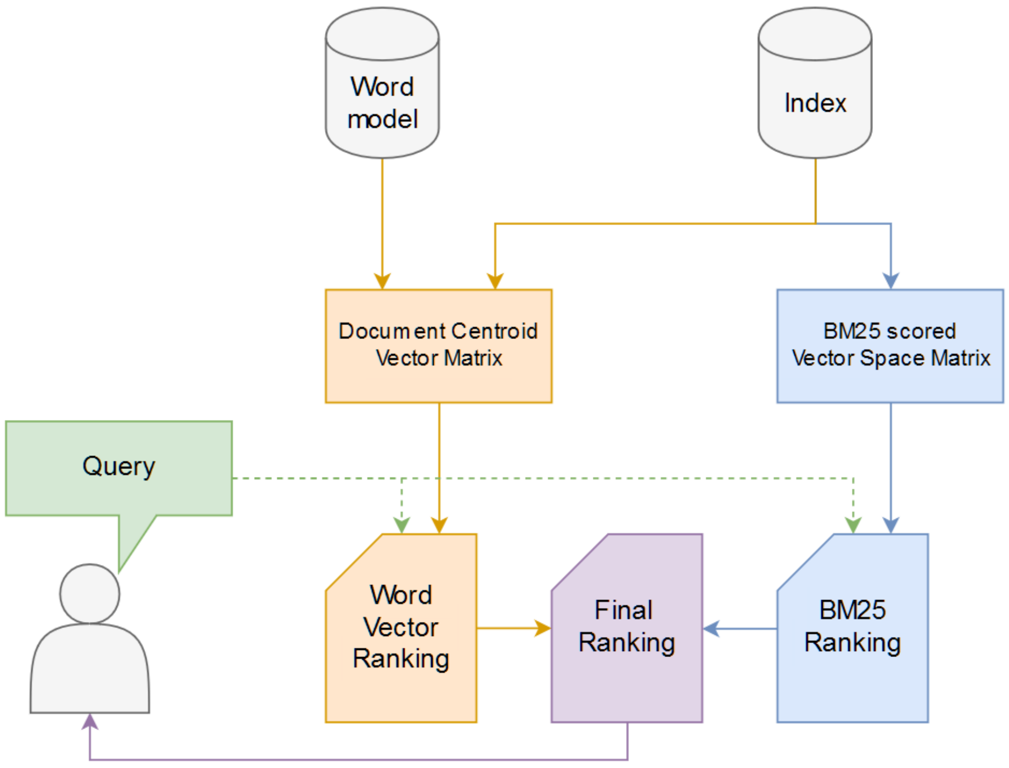
Declaration

Contents

Abstract

In the recent years, Neural networks have become very popular. They are being used in almost any field of computer science, and several of them are turning out to be successful. Information retrieval requires a fast method that compares two batches of text and gives a score. That score has to represent how similar both texts are. Most approaches rely on basic text comparisons, while trying to grab extra information from the users to cater the results more to their liking. In this bachelor thesis, we propose the addition of neural network techniques to the comparison process. More specifically, the usage of word vectors. Revolving word vectors, various techniques have been documented and tested, and while most of them did not have positive results, the one that did has been more thoroughly analysed. The approach that is presented here relies on representing queries and documents as clusters of words, which are points in space. Then, to quickly compute the score, the average linkage has been selected as a comparison method, that is, the distance between the means of the clusters. This approach has given positive results over the baseline in both test databases, and the results have been documented below. The results show a high potential in the use of word vectors in the IR area.

Figure 1: Complete flowchart of the ranking process



1 Introduction

Since the introduction of Neural Networks, their brilliant performance in some areas of Machine Learning has led plenty of researchers to try them out in every possible area, with varying success. Text comparison is one of those areas, and the results have been fairly successful. However, implementing those results into information retrieval impose some challenges that will be described in more detail in section 1.1.

1.1 Problem Description

Sadly, the positive results obtained in word comparison, and text comparison, cannot simply be translated into information retrieval. Most of the methods that compare two batches of words work with batches of similar size. Queries could be said that reside in a different space than documents, since they encode the information in a very different way that documents. This imposes a challenge when using text comparison techniques such as Document Vectors and Word Mover's Distance, explained at sections 3.2.1 and 3.3 respectively.

1.2 Motivation

Currently, typical IR techniques use simple processing of the words in the documents such as Bags of Words. It is a fast and robust method of text processing, that has had several years of optimizations, with the state of the art being the BM25 algorithm (section 3.1). This however, fails when trying to identify synonyms and discards a lot of information such as word ordering. Using advanced word comparison techniques has the potential to make use of both ordering of words, which helps to identify combined words, and synonyms. Ultimately, this all is used to improve the quality of the results that the engine provides.

2 Related Work

In this section we introduce some of the work that has been done in the field of IR with embeddings in the last year. They are mainly introductory works since this is a new field to work on. I will present two of the works in sections 2.1 and 2.2

2.1 Representing Documents and Queries as Sets of Word Embedded Vectors for Information Retrieval

In this [3] paper, the authors try to approach IR using word embeddings. They approach the problem in a different way. Their idea is that computing the mean for the document loses too much information, since in documents with more than a single thematic focus, the mean ends up being a point not so much related to the themes of the document.

Their approachis to show documents as clusters of words, and instead of computing the centroid, they apply agglomerative clustering to get a number of clusters that represent the topics of the document. Then, they compute the likelihood that the query would appear in the document.

With this method, they are able to approach documents with multiple focal points.

2.2 Bridging the Gap: a Semantic Similarity Measure between Queries and Documents

In this [4] paper however, the authors approach a more specific technique than the other paper. They try to use Word Mover's Distance (explained in section 3.3) as a base and try to optimize it.

They try two things here:

- One, to speed up the process.
- Two, to adapt queries to the semantic space of the documents.

Their solution to the first problem is basically to relax the constraint of WMD to be able to parallelize it. To solve the semantic space problem, they modify the weight of the words introducing IDF to it. These two changes both speed up and improve their results.

When trying to emulate their results however, I discovered that the speed up they propose is already implemented in the fastemd [7] implementation that Gensim uses, and it still is very slow.

3 Algorithms of the engine

3.1 Best Match 25

This is the baseline that has been picked for this thesis. BM25 is widely used as state of the art technique for functions based on TF-IDF (Term Frequency - Inverted Document Frequency), and it is simply a natural response to the problems that TF-IDF brings.

If we only use TF, longer documents will have preference over short documents. Furthermore, TF tends to go to infinity, and an upper bound is preferable in ranking systems.

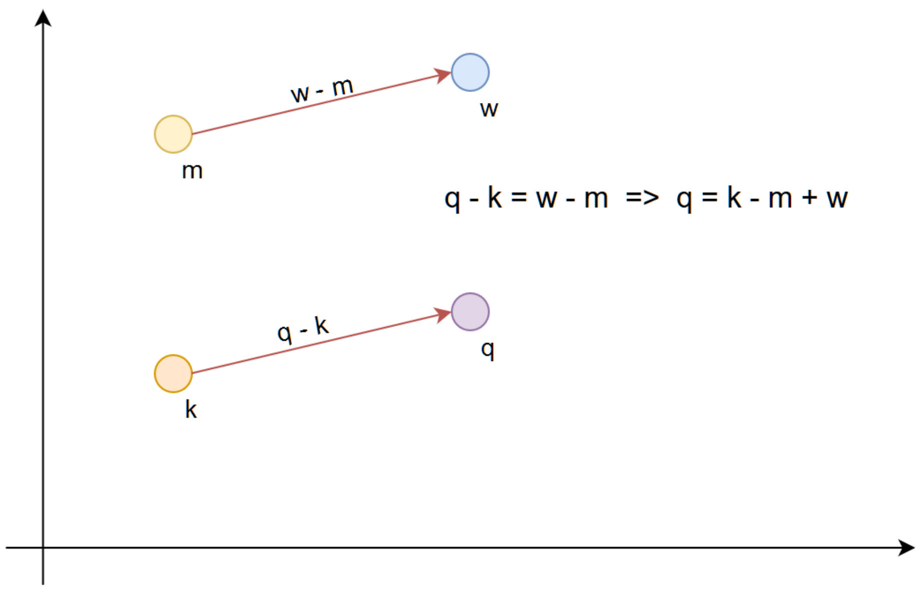
BM25 intends to solve that. First, TF is transformed to another value that has an upper bound, TF*. This is the simplest form to achieve that:

$$TF^* = TF(k+1)/(k+TF) \text{ where } k \in [0, \infty)$$

Even if the IDF of TF-IDF is not specially problematic, we are still interested in partially using the IDF. Again, we will divide TF* with α, which will be a value in between DF, and 1 (no normalization).

$$\alpha = (1-b) + b * DL/AVDL$$

Figure 2: A geometric illustration of the solution of the $King - Man + Woman = Queen$ problem



3.2 Word Vectors

Word2vec [1], uses a shallow (2 layered) Recurrent Neural Network to learn a more complex vector representation of words, given a corpus. The corpus corresponds to a large batch of text, where the larger, the more refined the vectors that are outputted. Each word will have a unique vector that represents it, and every different vector will theoretically represent a word. However, the algorithm will only recognize the vectors of the words that appeared in the training corpus.

The learned words can be used to compute semantic similarity by computing their cosine distance. This can be used to solve several problems, namely the solution to the sentence *If a man is a king, a woman X*. In a well trained model, the solution is found doing this simple algebraic operation:

$$vector(\text{"}king\text{"}) - vector(\text{"}man\text{"}) + vector(\text{"}woman\text{"})$$

And the result is of course $vector(\text{"}queen\text{"})$.

Word2vec embeddings are the basis of our engine. Every tried and tested engine of this thesis uses a technology that is based on these embeddings.

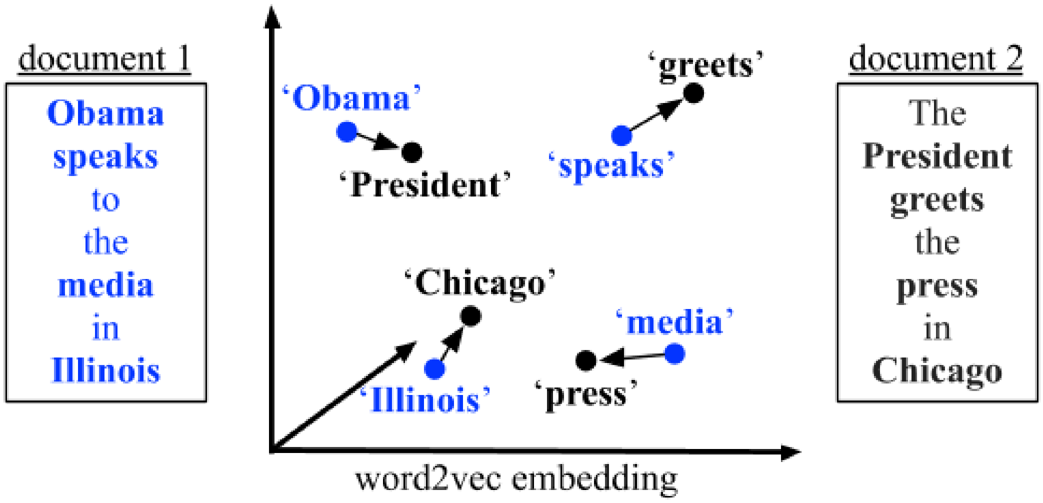
3.2.1 Document vectors

Document vectors, or paragraph vectors [2], are the natural evolution of word vectors. They try to capture a deeper meaning of a document's words, taking into account the ordering of the words.

The vector representation is computed by concatenating the document vector with several word vectors, and trying to compute the next word vector in the given context. The document vectors are unique per document, but the word vectors are shared. Therefore, similar documents will have similar vectors. Again, the learned vectors can be used to compute similarity between documents.

This algorithm also requires a corpus, albeit greater in size, since not only needs to compute word embeddings, it also has to compute documents embeddings. Given well trained embeddings, it is possible to infer the document embedding of a new document [6]. This is key in the field of IR, since it is highly improvable that all the documents and queries that are going to be introduced have been already trained.

Figure 3: An illustration of the $wordmover'sdistance$



3.3 World Mover's Distance

Instead of generating a single vector that represents the whole document WMD [5] uses every single word vector in the document to generate a distance from document to document that is sensible to synonims. This method tries to emulate the solution of the Earth Mover's Distance problem with word vectors.

The problem minimizes the cumulative distance that words from a document need to travel to become words from another document, in the context of some word vectors. That is, the sum of the vectors shown in this image [3]. This is achieved by several steps:

- The documents are represented as a normalized bag-of-words vectors. Meaning that if a word w_i appears f_i times in a document, the weight of that word will be $d_i = \frac{f_i}{\Sigma_{j=1}^{n} f_j}$ where n is the number of distinct words in the document, and f_j is the amount of times a word appears in it.

- Each word, in the context of word embeddings, has a place in space where it belongs. This is used to calculate euclidean distances between every word in both documents $c(i, j) = \|x_i - x_j\|$. This is not perfect, as theoretically similarity is measured with cosine distance, but it is the only distance that the algorithm accept.

- Finally we need to make sure that the flow of words that go from one document to the other is complete and exact. Let $\mathbf{T} \in R^{n \times n}$ be a sparse flow matrix, where $\mathbf{T}_{ij} \geq 0$ represents how much of word i travels to word j, and vice versa. To make the transformation right, the flow of the words must match the exact amount of the word's weight: $\Sigma_{j=1}^{n} \mathbf{T}_{ij} = d_i$.

- The resulting optimization function is as follows:

$$\begin{gathered} min_{\mathbf{T} \geq 0} \;\; \Sigma_{i,j=1}^{n} \mathbf{T}_{ij} c(i, j) \\ \text{subject to:} \\ \Sigma_{j=1}^{n} \mathbf{T}_{ij} = d_i \quad \forall i \in \{1, \cdots, n\} \\ \Sigma_{j=1}^{n} \mathbf{T}_{ij} = d'_j \quad \forall j \in \{1, \cdots, n\} \end{gathered}$$

- However, the constraint can be relaxed, limiting only the outward flow. This creates a lower bound for the WMD distance, which is tighter:

$$\begin{gathered} min_{\mathbf{T} \geq 0} \;\; \Sigma_{i,j=1}^{n} \mathbf{T}_{ij} c(i, j) \\ \text{subject to:} \\ \Sigma_{j=1}^{n} \mathbf{T}_{ij} = d_i \quad \forall i \in \{1, \cdots, n\} \end{gathered}$$

This leads to a method that is able to compare the similarity between two batches of text. To compute it in the most efficient way possible, fastemd [7] is the way to go, and luckily it is already implemented in Gensim.

3.4 Datasets

3.4.1 Training

To train the system, different datasets have been used. Ideally, a huge corpus with text on the topics of the topics that are going to be searched is wanted, but having that amount of text and the processing power is unlikely. Thus, a big corpus with text about anything will have to suffice.

For general training [8] has been used. The corpus was formed by crawling the WMT News in 2011. It is pre processed and already clean. It also comes separated beforehand in training and testing, and only the training part has been touched.

For word2vec, Google's pre-trained vectors have been used [9] instead of personally training new ones, since the size of the corpus they used to train them is very large.

3.4.2 Validation

Two datasets were used during validation. Sadly, as we did not have a benchmark for the beforehand mentioned corpus, we could not use their testing set for testing. Most benchmark datasets are not open to the public, or release the test sets, but not the data itself, so they require payment. Both datasets used for validation are from open sources, and there are links in the bibliography to their content for future use.

1. The first dataset and benchmark are obtained from here [10].

   It is a relatively small dataset, known for giving poor results on most IR techniques. However, our interest is not the final result, but to see if we can get some improvement over our baseline, so it suffices the requirements.

   Furthermore, it was mainly used as a second validation, after optimizing over the second one.

2. The second dataset and benchmark were obtained from [11].

   It is a significantly cleaned dataset, consisting on almost 140K documents that consist of movie title and their review. The benchmark was also provided with some queries and the relevant results that should show up.

   This dataset is the one that has been used to check if the methods used were good, and to optimize the hyper-parameters.

4 Tried and failed techniques

In this section all the techniques that have been tried are listed, and an assumption on why they failed to show positive results.

4.1 Document Vectors: Lack of power

Document vectors tend to need a corpus bigger in size than word vectors. To compute them, a lot of computer power is required, RAM specifically, since all the vectors have to be stored in there. Due to computer power limitations, it was not possible to use the whole corpus to compute document embeddings, and smaller batches had to be used. This heavily limits the potential of the vectors, and their quality is nowhere near the quality of Google's pre-trained word vectors.

Ideally, a separate corpus of queries would have been used to train the query vectors, and then a transformation applied to convert them to the document vector space.

4.2 WMD: Working space problem

A similar problem to document vectors arises. In this case the embeddings are good, and the results obtained at first glance made some sense. However, compared to the other techniques this one is painfully slow, with searches that reach the hour to be completed in a relatively small index of around 190K documents. The Gensim WMD function was copied to try to optimize the performance, but on average 1 second was spent per document in the index. Thus little tests were done with this one.

In all of those texts, if the results were personally checked, only some of the results in the list made sense, and most had either nothing to do with what was requested. This might have been due to two reasons, size and the space of existence.

- To solve the difference in size, the requested queries were duplicated to see if there was any improvement under the assumption that the words in the query had more weight to them than those in the documents. The results were different, but they didn't seem better.
- The difference in the semantic spaces of queries and documents cannot be easily overcome. Words in queries are not simply more important than in documents, they also tend to be more vague, and represent a broader space of meanings that in documents.

4.2.1 Top ranking reorder

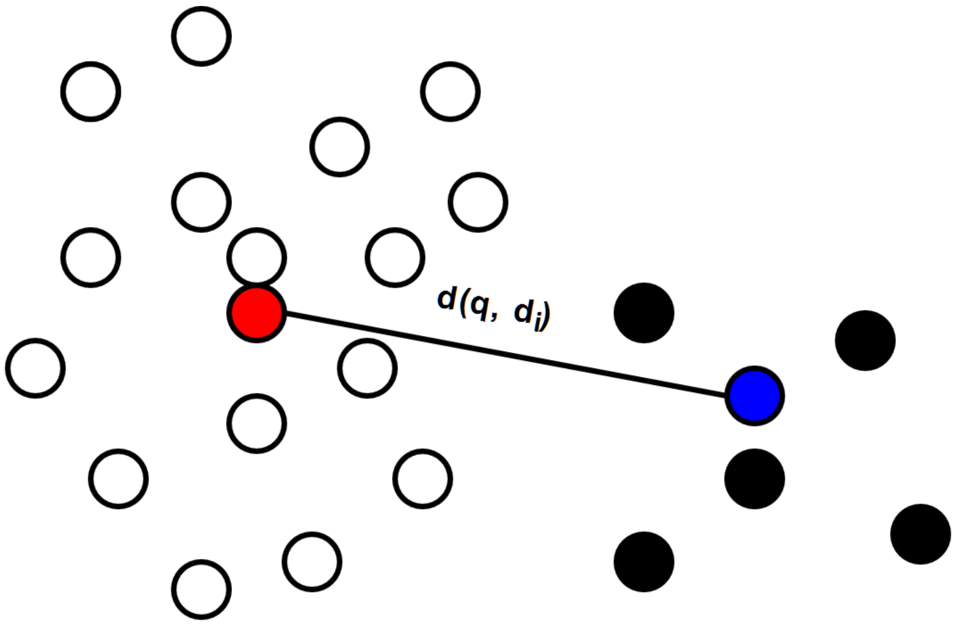
Due to the time limitations, a faster approach was tried too, which consisted on taking the top n elements of the ranking list (it was tested with n = 20) and reorder them according to their WMD distances to the query. However, as already mentioned, slowness was not the only problem with this approach, and this also lowered the results.

4.3 Generate sinonyms

With the premise that the queries are too short to obtain sufficient information, and duplicating them doesn't improve results, a more elaborated approach was tried, where synonyms are concatenated to the queries artificially extend their length in hopes of positive results. With the use of word vectors, we can try to find synonyms to a word. To obtain them, the words in the trained vocabulary are compared to the given word, and the closest words are picked as candidates.

However, despite the good quality of the embeddings, words have more than one meaning, and embeddings are not able to distinguish to which meaning of those a specific word in a query is referring to. This leads to valid synonyms that don't have much to do with the query, and do more damage than help. It also introduces words that do not exist in our index sometimes, and have to be eliminated, making the effort moot. This is a preprocess method that can be applied to any search technique in hand, and it was tested with all the methods without results.

Figure 4: Illustration of the distance between the query (black) and a document (white)



5 Combining average word vectors with BM25

While the previous attempts were more complex, they could not beat this simple combination. Due to the poor quality of the document embeddings generated, this workaround is proposed. For each document, all the individual word vectors are computed, and their mean calculated. With this, we lose information such as word ordering, but the mean still provides us with a good representation of the topic of the document.

5.1 Computational Complexity

The computation of the average is linear with the total amount of words in the document, per document. This however is precomputed once, so at search time the only thing that needs to be computed is the average of the words in the query, which should be small.

With:

- W: the total amount of words in the longest document.
- D: the total amount of documents in the index.
- w: the total amount of words in the query.
- L: the length of the word vectors.
- V: the amount of distinct words in the vocabulary.

Preprocessing complexity:

- Embedding preprocessing time: $O(L \times W \times D)$. Per word per document, the word's vector has to be added to the average. This does not account the time that it takes to load the embeddings.
- BM25 preprocessing time: $O(W \times D)$. Every word in every document has to be parsed once, and then to put the BM25 scores again. A third time to pass it into the vector space model. Note that even if the complexity is lower, this has a bigger constant and is slower than the embedding preprocessing time.

Retrieval processing complexity:

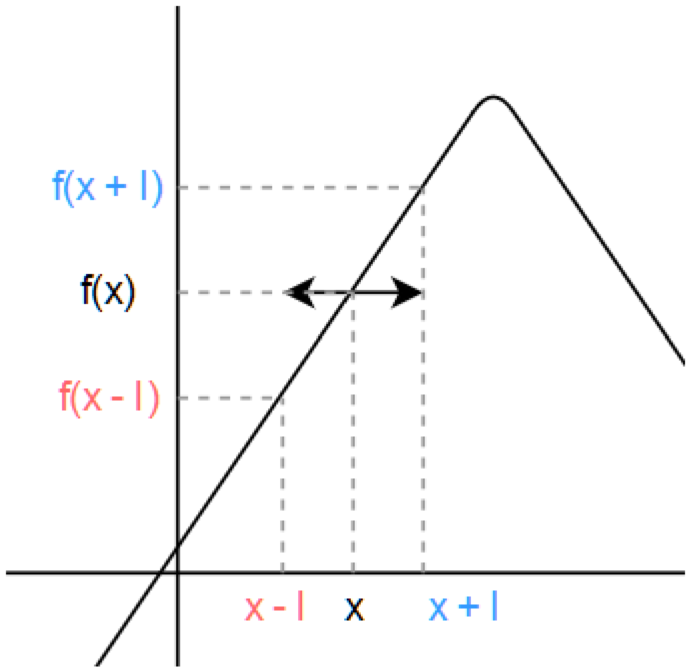
- Query processing time: $O(L \times w)$. This can be assumed as a constant because $w <<< D$.
- Embedding processing time: $O(L \times D)$. The distance computation takes L time, per document.
- BM25 processing time: $O(V \times D)$. A dot product is required for this with the vocabulary matrix.

5.2 Average Word Vectors

Documents can be represented as clusters of words (fig[4]), their *topic* being the centroid. By computing the mean, we get the *topic* of the document. Queries can be represented in a similar way. This way, the problem of words in queries being more elusive, with broader meaning, is circumvented since that would mean that the variance of the cluster that the query would form would be bigger, while the centroids would remain in similar values.

This means that we aren't missing that much information when comparing query means with document means. Ideally, we would use cosine distance to compute the similarities, but to make use of the vector space model, euclidean distance has been used.

Figure 5: Illustration on how a local search optimization with a learning rate of l is performed on a given function



5.3 Optimization

The initial values that were used for BM25 were 1.75 for k and 0.7 for b. This was decided as the baseline, and an improvement was found when combining the normalized score of it with the normalized score of average word vectors.

From there, a local search was applied to improve the results even a bit, as shown in figure [5]. To automatize it, a learning rate was manually defined per variable, which is dependant on the space of the values. The result was monitored to change the learning rate in accordance of the progress.

Two separate optimizations were done. The first one maximized the MAP of the second dataset and the second one the overall score (best of three of MAP, MP@3 and MP@R) of both datasets, taking turns per run. Maximizing for a single dataset is bound to overfit, and that is the reason the second optimization was performed, but it is still interesting to see the optimized hyper-parameters of each dataset.

- Results of the MAP optimization on DB2: k=1.4 b=0.6822 alpha=0.6695
- Results of the overall optimization on both DBs: k=6.987 b=0.3199 alpha=0.4396

6 Evaluation

A thorough evaluation was conducted in the system. For the doc2vec approach, a language model was trained over $\frac{1}{100}$ of the full corpus. Doc2vec is very punishing to the usage in ram. Training the model with this slice of the full corpus took 1 hour and filled 16 GB of ram, alongside with 8 GB of swap. It clearly isn't feasible doing it with the full corpus. As an alternative method, memory mapping was attempted.

With memory mapping, the system uses a file on disk as if it were part of the ram. This way, when the processor needs part of it, it will fetch it from the disk. However, this also means that the I/O process will be way slower and it will take much longer to finish the same process. The same slice was trained using memory mapping, and it took almost a full day of computation. Thus, using the full corpus to train the model was labelled as infeasible, since the building of the model is worse than linear for the size of the corpus, and in the best case, it would take 100 times longer to complete the computation.

Besides that, all the other processing was done using Google's pre-trained word vectors.

6.1 Results

The metric that was mainly used for evaluation and optimization was MAP, although MP@3 and MP@R were looked too. This was computed for both evaluation databases. Here are the results for all the methods that were not slow enough to benchmark, that is, all but WMD. The boosting method of adding synonims is not in the results either, as it was clear that it lowered all of the three scores. The chosen hybrid had also its hyper-parameters optimized for the second database, and both databases at the same time. The rounding has been done at 10^{-4} for visibility.

Table 1: 1st database

Method	Hyper-parameters	MP@3	MP@R	MAP
Base mixed & mean vectors (base)	k=1.75 b=0.7 alpha=0.67	0.4265	0.2210	0.1850
Base BM25	k=1.75 b=0.7	0.4194	0.2375	0.2095
Base mixed & mean vectors (mixed-opt)	k=6.987 b=0.3199 alpha=0.4396	0.4121	0.2326	0.1904
Doc2vec mixed & base	k=1.75 b=0.7 alpha=0.67	0.4194	0.2375	0.2095
BM25 with WMD reorder	k=1.75 b=0.7	0.2115	0.1940	0.1543

Table 2: 2nd database

Method	Hyper-parameters	MP@3	MP@R	MAP
Base mixed & mean vectors (db2-opt)	k=1.4 b=0.6822 alpha=0.6695	0.5667	0.3639	0.3641
Base mixed & mean vectors (base)	k=1.75 b=0.7 alpha=0.67	0.5	0.2881	0.3162
Base BM25	k=1.75 b=0.7	0.4333	0.2869	0.3092
Base mixed & mean vectors (mixed-opt)	k=6.987 b=0.3199 alpha=0.5	0.5	0.3105	0.2964
Doc2vec mixed & base	k=1.75 b=0.7 alpha=0.67	0.4333	0.2869	0.3072
BM25 with WMD reorder	k=1.75 b=0.7	0.3333	0.2423	0.2914

As we can see in table 2, there is an improvement of 2.48% in the MAP from the BM25 to the unoptimized hybrid. Furthermore, another 4.79% of increase has been had with the optimized version. The combined optimization shows that the base that we picked was already overfitting for the first database, as the MAP is lower than with the unoptimized hyper-parameters.

The two methods that are shown besides the mean hybrid, are the doc2vec hybrid and the WMD reordering. The doc2vec hybrid had really bad results, and mixing it with the base query yields either no change, or a minimal decrease in the final scores. The reordering method did more harm that help, as it seems that although it didn't modify the MAP heavily, the MP@3 and MP@R were lowered. It seems to lower the positions of the relevant documents in the ranking.

6.2 Error in the normalization that mixes the methods

Regarding the results in the first database, albeit lower, it helped in uncovering an error in the system. The mixture method had lower MAP than the base query, and after letting it optimize for several rounds, it did not reach the results of the base query (0.17%). This makes no sense, since if alpha was 1, the results should be identical to those of the base method. After some investigation, it turned out that the normalization done when combining the rankings of both methods, modify the ordering of the ranks.

Sadly, due to time limitations, I was not able to pinpoint the exact problem. It might be a floating point precision error but it requires further investigation. However, after testing alpha=1 in the second database, I discovered that the yielded MAP was 0.2386%, which is much lower than the one that the actual base method yields. I suspect that if this is solved, the general improvement will increase.

7 Conclusion

Word embeddings are a very powerful mechanism that can be used for mostly all word comparison problems. Although Query-Document comparison brings problems they can be circumvented to get positive results. This is a novel way to boost search results that does not depend on extra information outside of the query itself, and has potential to be improved further in the future.

7.1 Future Work

The work with doc2vec stands to be completed. They theoretically have the potential to work, but the computational power required to generate some quality ones for the held requirements is their biggest handicap. Would that be overcomed, the retrieval processing effort is as low as the other techniques proposed, so it could be used in practice to boost results.

Using simply the average is also not a very powerful technique, there is obviously more work to be done in the field, and new techniques to be formed from the concept. The first paper presented in the related work shows a interesting approach to treat documents with more than a single topic that can be further expanded, though time limitations didn't allow to do so in here.

8 Acknowledgements

I would like to thank Prof. Dr. Hannah Bast for allowing me to work on this novel topic. I also want to thank the University of Freiburg for offering me the opportunity to finish my bachelor and write this work in there, it was really valuable.

I want to thank my home university too, Euskal Herriko Unibertsitatea, for helping me in the course of my studies. All the teachers whose lectures I had the pleasure to attend were a great model to follow in my studies.

Finally, I want to thank my family for supporting my study decisions. I know that it was not simple nor easy and their support and encouragement has been invaluable for me.

9 References

[1] Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean *Efficient Estimation of Word Representations in Vector Space.* (2013) `https://arxiv.org/pdf/1301.3781.pdf`

[2] Quoc Le and Tomas Mikolov *Distributed Representations of Sentences and Documents.* (2014) `https://arxiv.org/pdf/1405.4053v2.pdf`

[3] Dwaipayan Roy, Debasis Ganguly, Mandar Mitra and Gareth J.F. Jones *Representing Documents and Queries as Sets of Word Embedded Vectors for Information Retrieval.* (2016) `https://arxiv.org/pdf/1606.07869.pdf`

[4] Sun Kim, W. John Wilbur and Zhiyong Lu *Bridging the Gap: a Semantic Similarity Measure between Queries and Documents.* (2016) `https://arxiv.org/pdf/1608.01972.pdf`

[5] Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, Kilian Q. Weinberger *From Word Embeddings To Document Distances.* (2015) `http://proceedings.mlr.press/v37/kusnerb15.pdf`

[6] *Gensim ML library, doc2vec module* `https://radimrehurek.com/gensim/models/doc2vec.html#gensim.models.doc2vec.Doc2Vec.infer_vector`

[7] Ofir Pele, Michael Werman *Fast and Robust Earth Movers Distances.* `http://www.ariel.ac.il/sites/ofirpele/FastEMD/code/`

[8] *Dataset from "One Billion Word Language Modeling Benchmark".* http://www.statmt.org/lm-benchmark/1-billion-word-language-modeling...

[9] *Google's pre computed word2vec vectors* `https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?usp=sharing`

[10] *Link to the first dataset: NPL* `ir.dcs.gla.ac.uk/resources/test_collections`

[11] *Link to the second dataset and the benchmark* `https://drive.google.com/drive/folders/0B-K9ndb0VB1WSWVEWGlDbDR3LU0?usp=sharing`