Semantic Embeddings in IR

Text-Based Information Retrieval

Joren Verspeelt, Jochem Geussens, Vincent Tanghe

May 6, 2016

1 Introduction

In the field of Natural Language Processing (NLP) and Information Retrieval (IR), there is a large need for good presentations of text documents. Recently, a lot of researchers in the field are presenting dense word representations (also known as Word Embedding, Neural Embedding or Sementic Embedding). For the Course of Text-Based Information Retrieval, we are given the opportunity to work with state-of-the-art algorithms. The goal of this assignment is to gain practical experience with these algorithms by comparing them and using them in an application. First we will implement a small analogy solver that guesses related words given a simple analogy. Then, we will discuss our implementation for a search engine to find pictures based on their description.

Contents

2 Part I (Warm-up)

2.1 Task description

In this part, we discuss two analogy solvers that try to guess the correct word, given simple analogy questions. E.g.:

$$\begin{aligned} a : b = c :? \\ Bratislava : Slovakia = Bishkek :? \\ ate : eat = found :? \end{aligned}$$

Then, we will run several analogy solving models with several different representations on the benchmarking analogy dataset.

2.2 Setup

We compare the use of different word embeddings (GloVe and Word2Vec) towards different analogy arithmetic models[9]. To test these, we used the questions provided by Google[3].

We coded our solution in Python (version 2.7) using Jupyter Notebook. All experiments were run on a notebook with 8GB RAM memory, Intel(R) CORE$^{\text{TM}}$i7-3610QM CPU, SSD hard disk and a Windows 7 Operating System.

We use freely available pre-trained vector models for GloVe [5] and Word2Ve [4]. The GloVe models are almost the same format as the Word2Vec models, the only difference is that Word2Vec had a header with the dimensions. Once these headers are added, the GloVe model can also run with the default Gensim operations for Word2Vec.

To run the different analogy arithmetic models (see next paragraph), we introduce a separate function in our code for each model. These functions then use the Gensim library for Python [8] to calculate the different vector distances.

An analogy arithmetic function is used to calculate how close the input vector is to another vector in order to choose the most similar result. To calculate the angle between high dimensional vectors, Mikolov et al. [7] proposed using cosine similarity as explained in equation 1.

$$\begin{aligned} similarity(v1, v2) &= cos(\theta) \\ &= \frac{v1 \cdot v2}{||v1|| \cdot ||v2||} \\ &= \frac{\sum_{i=1}^{n} v1_i v2_i}{\sqrt{\sum_{i=1}^{n} v1_i^2}\sqrt{\sum_{i=1}^{n} v2_i^2}} \end{aligned} \tag{1}$$

We consider two analogy arithmetic models. The first is named the **Addition method** and is simply maximizing the cosine similarity between the result d_w and the vector equation $c - a + b$, where the variables $a, bandc$ have the same meaning as mentioned in the warm-up exercise. This method actually looks for words similar to c and b, and not to a. In equation 2 we introduce V as the whole vocabulary and d'_w as a chosen element from V.

$$\begin{aligned} v &= c - a + b \\ d_w &= argmax_{d'_w \in V}(similarity(d'_w, v)) \end{aligned} \tag{2}$$

The other method was referred to as the **Multiplication method** [6]. Because the Addition method risks being dominated by one large term, therefore the authors proposed that instead of adding the similarities, they could be multiplied as described in equation 3.

$$d_w = argmax_{d'_w \in V}(\frac{similarity(d'_w, c)similarity(d'_w, b)}{similarity(d'_w, a) + \epsilon}) \tag{3}$$

Two different recall numbers are generated if a word does not occur within the pre-trained vector model: One where the **missing word is considered a failure** for the analogy and one where the **missing word is just ignored**. We could also have done a third option, were we use the nearest word in the vector model instead of the missing word. However, we did not do this because this would take a lot more computing power and (as seen in the results below) we're not sure whether it would have made a difference.

2.3 Results

In table 1 the results of the GloVe method combined with the multiplication method are shown. The results for the addition method are displayed in table 1b. The results of multiplication- and addition model of Word2Vec can respectively be found in table 3a and table 3b. A graph that visualizes the results next to each other is presented in figure 1.
No matter the dimension, GloVe always had the same amount of skipped words (all words) within a category. So we will not show the category results for these.

Category	Recall
Family	0.62846
Gram1 adjective to adverb	0.09980
Gram2 opposite	0.04926
Gram3 comparative	0.41366
Gram4 superlative	0.17112
Gram5 present participle	0.29451
Gram7 past tense	0.31153
Gram8 plural	0.46021
Gram9 plural verbs	0.25862
Total	0.14506
Total no skipped	0.29587

(a) GloVe50d multiplication model (3 min runtime)

Category	Recall
Family	0.77866
Gram1 adjective to adverb	0.22883
Gram2 opposite	0.15764
Gram3 comparative	0.74850
Gram4 superlative	0.50624
Gram5 present participle	0.65057
Gram7 past tense	0.52821
Gram8 plural	0.66667
Gram9 plural verbs	0.57356
Total	0.26668
Total no skipped	0.54394

(b) GloVe100d multiplication model (7 min runtime)

Category	Recall
Family	0.85178
Gram1 adjective to adverb	0.25302
Gram2 opposite	0.19581
Gram3 comparative	0.83784
Gram4 superlative	0.67162
Gram5 present participle	0.67045
Gram7 past tense	0.60321
Gram8 plural	0.76426
Gram9 plural verbs	0.65172
Total	0.03531
Total no skipped	0.62273

(c) GloVe200d multiplication model (10 min runtime)

Category	Recall
Total	0.32214
Total no skipped	0.65707

(d) GloVe300d multiplication model (20 min runtime)

Table 1: Results of GloVe multiplication models

Category	Recall
Family	0.85573
Gram1 adjective to adverb	0.25403
Gram2 opposite	0.22660
Gram3 comparative	0.86486
Gram4 superlative	0.69786
Gram5 present participle	0.68277
Gram7 past tense	0.60192
Gram8 plural	0.77402
Gram9 plural verbs	0.59770
Total	0.30778
Total no skipped	0.62774

(a) GloVe50d addition model (3 min runtime)

Category	Recall
Family	0.85573
Gram1 adjective to adverb	0.25403
Gram2 opposite	0.22660
Gram3 comparative	0.86486
Gram4 superlative	0.69786
Gram5 present participle	0.68277
Gram7 past tense	0.60192
Gram8 plural	0.77402
Gram9 plural verbs	0.59770
Total	0.30778
Total no skipped	0.62774

(b) GloVe200d addition model (10 min runtime)

Category	Recall
Total	0.31304
Total no skipped	0.63849

(c) GloVe300d addition model (20 min runtime)

Table 2: Results of GloVe addition models

Category	Recall
Capital common countries	0.85178
Capital World	0.80570
Currency	0.35450
City in state	0.71423
Family	0.84585
Gram1 adjective to adverb	0.31552
Gram2 opposite	0.42365
Gram3 comparative	0.90691
Gram4 superlative	0.91800
Gram5 present participle	0.80587
Gram6 nationality adjective	0.89368
Gram7 past tense	0.70641
Gram8 plural	0.89940
Gram9 plural verbs	0.73448
Total	0.75148

(a) Word2Vec multiplication model (3h25m runtime)

Category	Recall
Capital common countries	0.83202
Capital World	0.79134
Currency	0.35104
City in state	0.70896
Family	0.84585
Gram1 adjective to adverb	0.28528
Gram2 opposite	0.42734
Gram3 comparative	0.90841
Gram4 superlative	0.87344
Gram5 present participle	0.78125
Gram6 nationality adjective	0.89931
Gram7 past tense	0.65962
Gram8 plural	0.89865
Gram9 plural verbs	0.67931
Total	0.73588

(b) Word2Vec addition model (1h15m runtime)

Table 3: Results of Word2Vec models

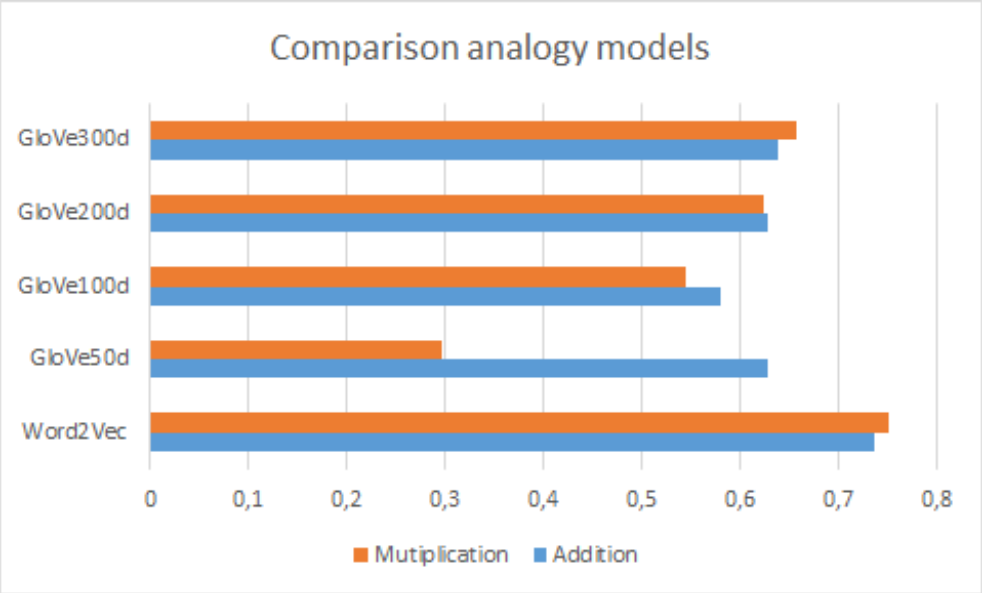

Fig. 1: A comparison between GloVe and Word2Vec results for the multiplication and addition method.

2.4 Discussion

We noticed that the GloVe model was not able to find all the words and this influences the results regarding execution times. The GloVe model seems to run a lot faster than the Word2Vec model because of our implementation strategy. If one word of the analogy is missing, the whole analogy is skipped resulting in smaller search space that needs to be covered. But that aside the GloVe model has the edge with faster calculations due to its lower dimensionality.

The complexity of the analogy models are O(3n) as 3 vectors need to be found in the space of n available vectors in our model. Worst case these vectors are at the end.

If we look at the dimensionality of the GloVe representations, we see that the overall accuracy increases as the number of dimensions increases. This is a trade-off compared to the time needed to run the model, but in general it seems worthwhile. If we then look at the analogy model compared to the dimensionality, we see that the multiplication model performs better on higher dimensions while the addition model performs better as the dimension decrease. The overall accuracy decrease with a lower dimension is most likely the reason for this.

While running the experiment, we frequently got missing words (using the GloVe representation). It is still impractical to have a model representing every possible word that exists, so a trade-off is made towards a reasonable model size and the amount of

represented words. We solved this problem by either counting missing words as failed analogies or by skipping the sentence. However, it seems that the same word categories are missing in every dimension and counting only makes the accuracy worse.

3 Part II (Retrieval task)

3.1 Task description

This part of the assignment is based on the annual Scalable Concept Image Annotation Challenge, organized by ImageCLEF [1]. The dataset (given by ImageCLEF), contains sentence descriptions aligned to actual images. In this part of the assignment, we bring our previous tests into a real-life retrieval scenario: retrieve images by their descriptions using textual queries. Each query has a corresponding image ID, which can be used to verify the correctness of the model. The goal is to match queries with image descriptions as good as possible, using precision and recall as evaluation measures.

First we implemented a simple language model to retrieve the images, this was based on a unigram implementation [2]. Afterwards we extended it with word embedding, inspired by part I of this assignment. For the final method, we chose to implement Latent Dirichlet Allocation (LDA) with dirichlet priors. We have read that this performs well with twitter data, where the document size seems similar to our image descriptions. And this is a method that has been mentioned several times in class which sparked our interest.

3.2 Setup

We coded our solution in Python (version 3+) using Jupyter Notebook and regular python files. All experiments were run on a notebook with 8GB RAM memory, Intel(R) CORE^{TM}i7-3610QM CPU, SSD hard disk and a Windows 7 Operating System.

The data we received contains the following documents:

A. target_collection A table representing 17,784 images with as columns a unique numerical ID, an ID that corresponds to the actual image file and a description of the image. This data is used to train our model.

B. queries_val A similar table as the *target_collection* but it represents 1000 different images. This table will be used to test the accuracy of our trained model.

B. test queries_val A collection similar to *queries_val* but it only contains 200 queries and it does not tell which image id should be returned. This file is the real test data where we can't tune our method for optimal results. The similarities will be used to rate our results to other students.

Preprosessing Because comparison of words happen on the exact format of the words, we decided to do some preprocessing to make sure every word is in the same base form. Both the target collection as the queries were lemmatized. And, because we find that some words carry more meaning than other words, we also decided to remove the punctuation using regular expressions and the stop words (e.g. the, it, to, ...) using the freely available stop words list 2 (stopwordlist2 - http://www.lextek.com/manuals/onix/stopwords2.html).

Furthermore, we noticed that all queries and the target data are written in British English. Because we use the pre-trained Google News vector model, which only contains American English words, a lot of remained not found. So we did a hardcoded translation of the most frequent missing words - due to the lack of a good library - for the embedding results.

We will report the Recall and *Mean Average Precision* (MAP) with a cutoff of 1000 documents in our results.

3.3 Results

3.3.1 Unigram Language Model

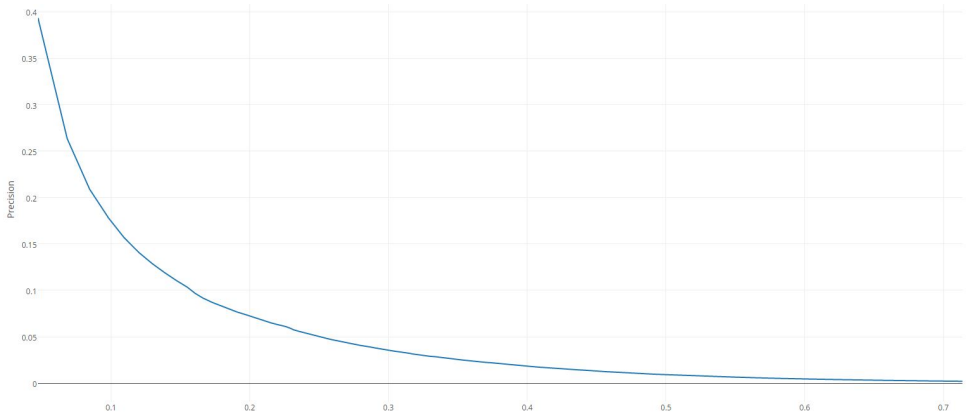

Fig. 2: The precision-Recall chart for Unigram Language Model with smoothing 0.

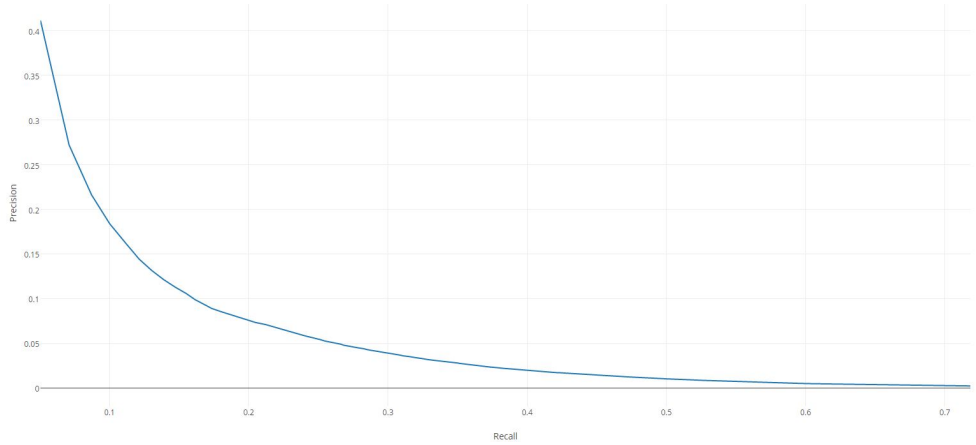

Fig. 3: The precision-Recall chart for Unigram Language Model with smoothing 10.

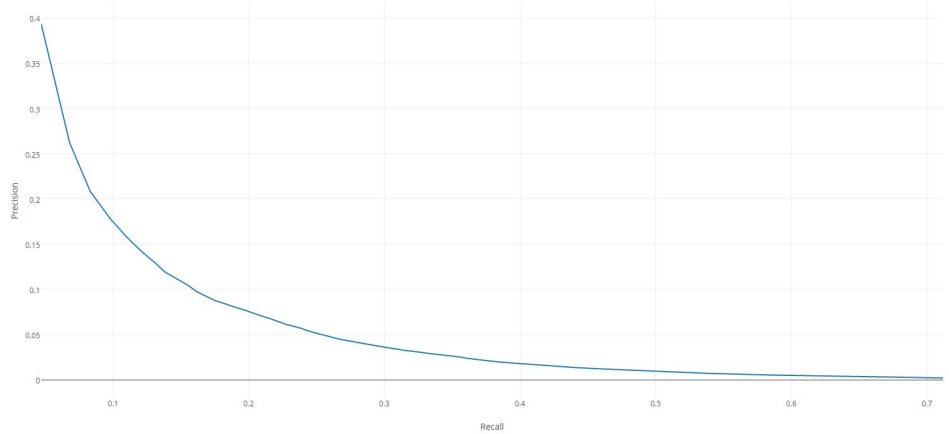

Fig. 4: The precision-Recall chart for Unigram Language Model with cutoff 50.

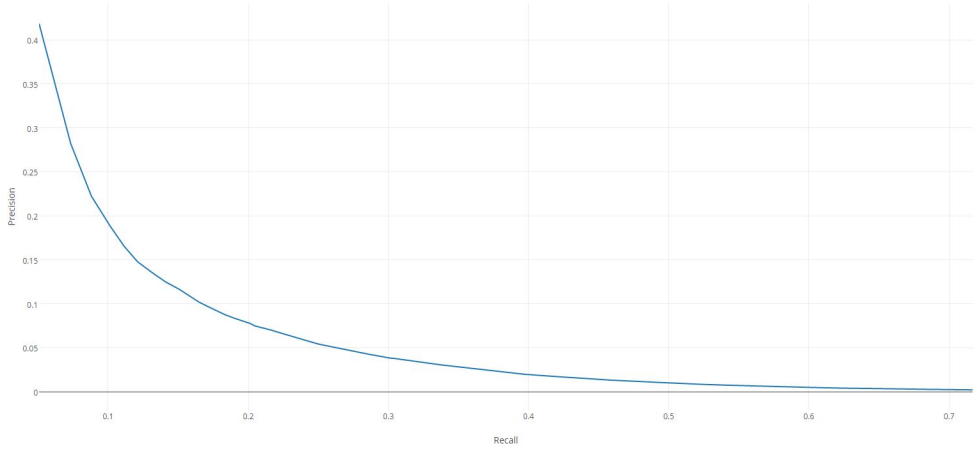

Fig. 5: The precision-Recall chart for Unigram Language Model with cutoff 100.

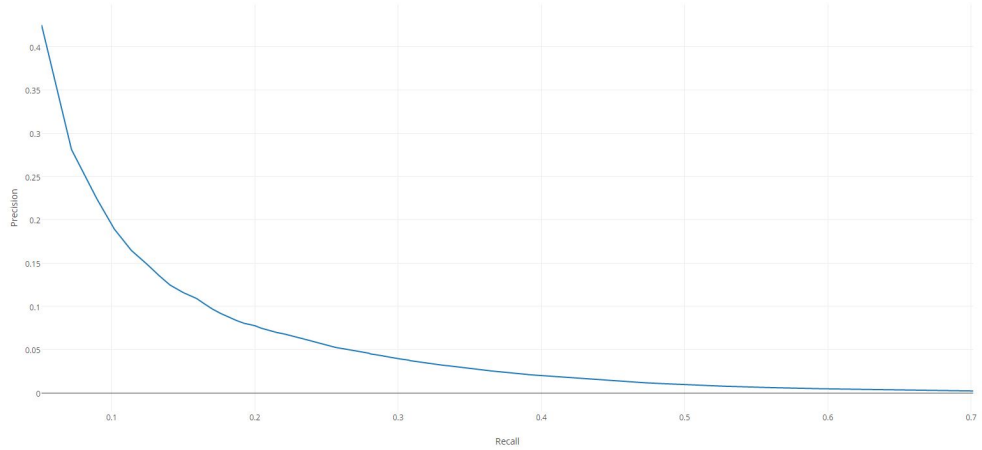

Fig. 6: The precision-Recall chart for Unigram Language Model with cutoff 250.

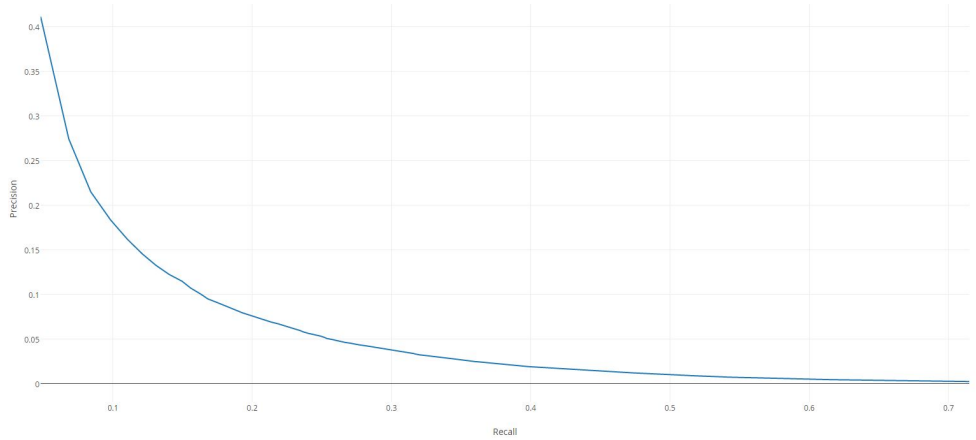

Fig. 7: The precision-Recall chart for Unigram Language Model with cutoff 500.

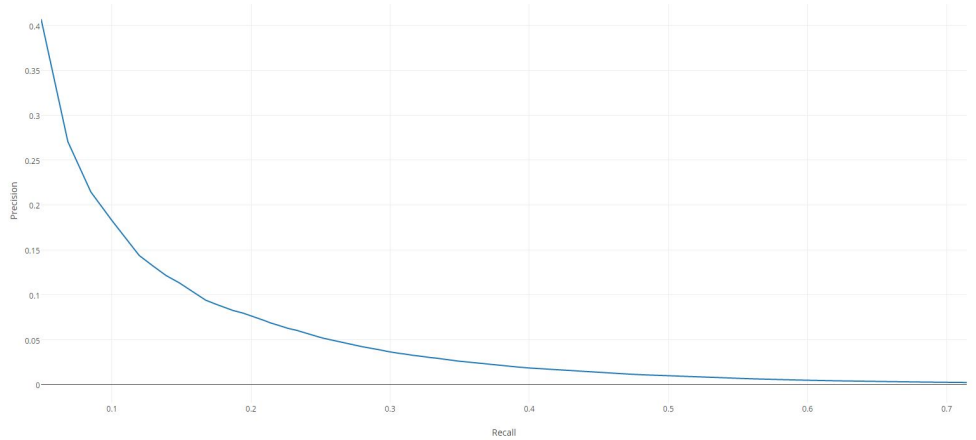

Fig. 8: The precision-Recall chart for Unigram Language Model with cutoff 1000.

3.3.2 Wordembedding

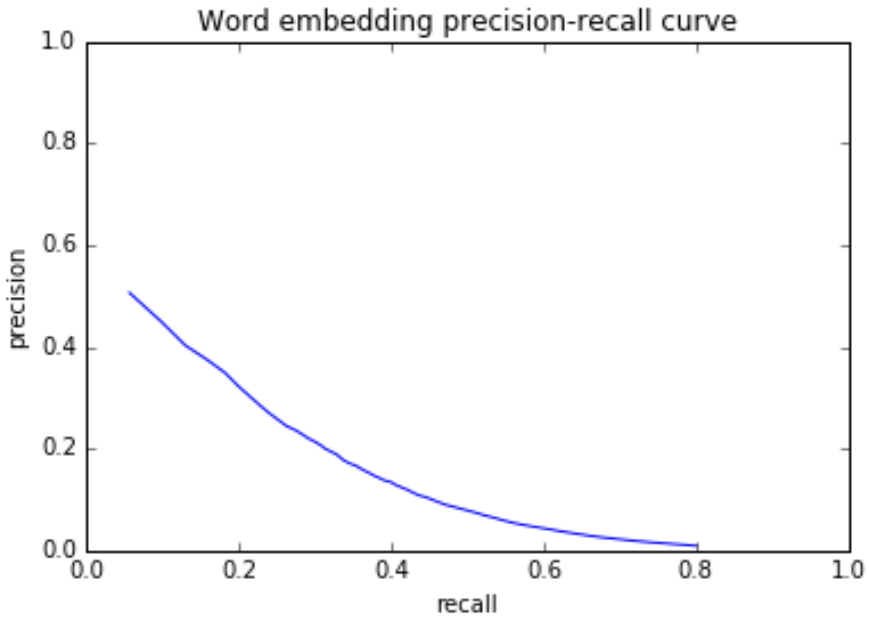

Fig. 9: The precision-Recall chart for word embedding.

MAP: 0.258010537928

#top queries	Recall	Precision
10	0,2529983	0,2565
100	0,5341049	0,06583
250	0,6483118	0,032468
500	0,7281287	0,018326
1000	0,8003459	0,010071

3.3.3 LSI

3.4 Discussion

First of all, we can see that the recall increases at we increase the number of top N queries. This is logical because the recall gives ratio between found and to-be-found. So if we would cut off at the maximum amount of documents, the recall would be 100 Secondly we see that the precision decreases as the amount of documents increases.

3.4.1 Unigram Language Model

The simple Unigram Model is the simpelest of all methods that we attempted. It is also the fastest of all methods that we attempted. However, this speed comes at an accuracy cost. The Simple Language Model does not account for words with a similar meaning, something the word embedding does account for and we can see that the word embedding is superior in accuracy. Although this simple unigram model manages to score fairly good on recall, we see the precision drop very rapidly. Even with a very small cutoff (¡ 10), a lot of mistakes are made. The general concensus for this unigram model is that most of the documents that should be suggested are retreived, but their rating is not high enough, making this a weak predictive model.

3.4.2 Wordembedding

Because the location of the word in the model gives semantic similarity, words with similar meanings are given similar vectors. This results in a more accurate result than the Unigram Language Model, as we can see in the results. Because of the limited size of the documents, a lot of similar words are still found. If the descriptions would be longer, a more accurate result could be found and the overall accuracy would increase. But with the current length of the queries and the target collection, we think we got acceptable results.

To calculate similarity, we sum the different vectors. The main problem with this approach is that it introduces a lot of noise. Every word contains a similar value. We have already attempted to reduce this problem by removing the stopwords (which increased our results), but this is not a total solution. A more intelligent method would be using weighted sum based on other information. (e.g. using Term Frequency Inverse Document Frequency (TF-IDF) to weigh terms based on their frequency) However, we have not attempted to add TF-IDF for a weighted sum (as it would increase the computational need) and can therefore only assume that it would further increase the accuracy.

3.4.3 LSI

Latent Dirichlet Allocation is one of the more sophistated methods for building word representations. Using priors from a dirichlet distribution, per-document-topic and per-topic-word distributions can be updated. LDA did not give the results that one would expect, scoring even lower than the simple unigram model. These results can be explained by the small length of the documents. After removing the stop words and lemmatizing the remaining words, most documents contain only five to ten words. This implies that, if we purely look at the length of things, we are actually matching queries with other queries. Per-document-topic distribution will therefor not give good results. In general, topics will not have a high chance of occuring in documents. Similarities are calculated by vectorizing and calculating the cosine similarity between document and query representations.

4 Conclusion

In retrospect to the methods, we have learned that the older models, such as the Simple Language Models performs very well while taking way less computational power. Our favorite method was Word2Vec, but it takes a lot of RAM memory. In fact, the used models took up all our RAM which resulted in extra long running times because some things had to be written on the hard disk. And finally, hyper-parameters make a big difference, something that we noticed most using LDA.

In retrospect of the assignment, we may need to start sooner and dwell less on fine-tuning less important parts of the assignment. And when using methods like Word2Vec, a cloud computer or online server can save a lot of time.

Overall, it was an interesting to work with the algorithms and see empirical results instead of just the theoretical results. We have experienced at first-hand what difficulties occur during the experiments (e.g. how important it is that all words are in a similar form).

5 References

[1] http://imageclef.org/2016/annotation. http://imageclef.org/2016/annotation.

[2] *Language models for information retrieval*, 2009. `http://nlp.stanford.edu/IR-book/pdf/12lmodel.pdf` [Accessed: 6-5-2016].

[3] *Question words provided by Google Code*, 2013. `http://word2vec.googlecode.com/svn/trunk/questions-words.txt` [Accessed: 3-5-2016].

[4] *Word2vec*, 2013. `https://code.google.com/archive/p/word2vec/` [Accessed: 3-5-2016].

[5] Christopher D. Manning Jeffrey Pennington, Richard Socher. *GloVe: Global Vectors for Word Representation*, 2014. `http://nlp.stanford.edu/projects/glove/` [Accessed: 3-5-2016].

[6] Goldberg Y. Levy, O. *Linguistic Regularities in Sparse and Explicit Word Representations. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (pp. 171–180).*

[7] Yih W. Zweig G. Mikolov, T. *Linguistic Regularities in Continuous Space Word Representations, 746–751.*

[8] Radim Rehurek. *gensim: topic modelling for humans*, 2016. `https://radimrehurek.com/gensim/` [Accessed: 3-5-2016].

[9] Marek Rei. *Linguistic regularities in word representations*, 2014. http://www.marekrei.com/blog/linguistic-regularities-word-representations [Accessed: 3-5-2016].