IR1 - Assignment 2

Nizar Hirzalla
10803165
n.a.hirzalla@student.vu.nl

Kimberley Boersma
11003464
k.l.m.boersma@student.vu.nl

Guido Ansem
12959448
guidoansem@gmail.com

ABSTRACT

KEYWORDS

datasets, neural networks, gaze detection, text tagging

ACM Reference Format:

Nizar Hirzalla, Kimberley Boersma, and Guido Ansem. 2020. IR1 - Assignment 2. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In this report we will conduct several experiments and we will consider different information retrieval models and accordingly evaluate them. We will use Word2Vec, Doc2Vec, LSI using BoW, LSI using TF-IDF and LDA and apply it on a a dataset containing AP articles from 1988 and 1989 to model topics from these how a given query will match these articles and what the model deems as a significant match.

2 WORD2VEC AND DOC2VEC MODELS

2.1 Word2Vec

.

Word2Vec is the name for a group of models which are used to get vector representations of specific words. One way of getting word embeddings is by using a skip gram. A skip gram is a shallow neural net with only one hidden layer, which tries to predict the context words given a certain word. Pairs of words are created where each pair contains the target (or center) word and a word within a predefined distance from the target word(context word). One hot encoding is used to represent each word in the vocabulary. The one hot encoded target word is than passed to the input layer of the neural net. This input layer is fully connected to the hidden layer which has the dimension $(numberofwordsinvocab * desiredvectorlength)$. Here the desired vector length is specified before hand, this equals the length the final word embedding will have. This hidden layer is then again fully connected to the output layer which has the same dimensions as the input layer. The output should ideally be the same as the one hot encoding for the context word. However, we are not actually interested in the output of the network, what we are interested in is the weights of the hidden layer to the output layer. Because similar words will most likely occur in similar context it is assumed that similar words will have similar weights. Therefore, the weights of a certain row in the hidden layer will be the vector that will eventually represent the word.

In this report a skip gram was designed using The AP Dataset, which consists of news articles published in Associated Press during 1988-1989. Scripts for pre-processing of the data were provided to perform actions like stop word removal, punctuation removal and stemming. These pre-processing steps were designed to fit data, it is import to consider the field in which the application will run when pre-processing data. (TQ 1.1) For example when the field would have been medical instead of news different steps might be considered. Among these steps might be different methods of stemming to take care of the many Latin words that are often used in the medical domain and a different set of stop words might be used for words that are common to occur in medical language. For the skip gram designed in this report a method called negative sampling was used. This method is used to speed up the training of the model. It does this by only updating a specific amount of weights for each iteration. In a normal skip gram for each pair of words that is passed only one word is the "correct" context word. Therefore, all the other words are "wrong" and the weights for all of these words needs to be updated. However, because the amount of unique words in a vocabulary often consists of thousands and sometimes millions of words, this becomes very computationally heavy very quickly. Negative sampling takes care of this problem by only updating a select amount of weights, often between 5 en 50. This decreases the time it takes to train the algorithm while still preserving good quality vector representations.

Because of the size of the original dataset (160.000 documents) and the limited capacity of the machine on which the algorithm was trained a subset of the data of a 1000 documents was created on which the algorithm was trained. The context window size was set to 2 to keep the amount of training pairs low, the amount of nodes that would be updated per iteration was set to 10 and the dimensions of the vector were set to 100. Furthermore, words that occurred less than 20 times in the training data were removed from the corpus. After the algorithm the 10 most similar words for 5 random words were gathered, these combinations are shown in the table below. From this table it can be observed that the performance of the skip gram is sub-optimal although some of the words do seem to be relevant, for example: birthday is linked to February and younger, danger is linked to fighter, jail and kick and paid is linked to card and job, most of the words seem irrelevant. This low performance is most likely due to the reduction in documents

Table 1: ***Top ten similar words versus five chosen words for word2vec***

Chosen words	Most similar words
1967	christian, command, jose, export, 7, inform, yorktown, mexican, met, iran-contra
Birthday	maintain, inflat, attend, card, declar, februari, younger, peac, elderli, win
Danger	five, fighter, gorbachev, space, educ, aim, media, cover, jail, kick
American	april, oscar, choos, first, commun, consist, count, 25, ahead, propos
Paid	decemb, sit, card, fill, failur, gen, job, novel, frigat

and the small window size being used. However due to technical limitations it was not feasible to increase these parameters within the time provided.

2.2 Doc2Vec, AQ2.2

For Doc2Vec we were able to use the Python Gensim package which provides us a lot of utility. First we created a corpus for building and training the doc2vec using the doc2vec module in gensim. We build the vocabulary with the corpus and accordingly we trained the model with the parameters set as can be found in our code (in the 'doc2vec.py' file). Accordingly we made a ranking function that given a query will search in the doc2vec and using an inferred vector will rank the documents that match the most with the input query. In line with word2vec, for the purpose of question AQ2.2, we chose a piece of text and retrieved the most similar documents. These can be found in Table 2. We used the query 'Ronald Reagan diplomatic defense' and can see that the results we retrieved are in fact similar with the query, they all engulf a political theme/diplomacy/defense and upon further inspection in the AP documents themselves we can observe that Reagan and diplomatic defense have a central role or are mentioned in some or other way in relation to the news topic.

Table 2: ***Top ten similar documents for "Ronald reagan diplomatic defense"***

Top 10 documents
1. AP880314-0125: mask man fire four saudi diplomat...
2. AP880908-0224: vietnames diplomat allegedli...
3. AP890624-0097: chines diplomat base communist..
4. AP890103-0062: six week promis soviet withdraw...
5. AP890313-0118: central american offici week...
6. AP890323-0098: king bhumibol adulyadej queen..
7. AP880910-0077: struggl democraci burma threaten..
8. AP880808-0078: swiss author uncov 62 case spi..
9. AP890217-0108: unit state evacu personnel embassi..
10. AP890110-0027: presid reagan want congress..

3 LATENT SEMANTIC INDEXING AND LATENT DIRICHLET ALLOCATION

In this section we will elaborate on our implementation of the various LSI and LDA training and models using BoW or TF-IDF. We will also analyze the obtained results and reflect upon these.

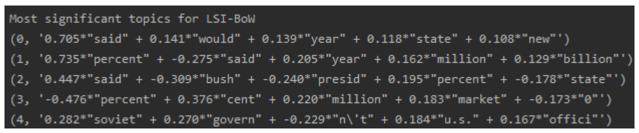
3.1 Implementation: LSI-BoW, LSI-TF-IDF, LDA-TF-IDF

We implemented training a LSI model through Bag-of-Words (BoW) and TF-IDF using the Gensim library in Python. For BoW we converted the corpus using the doc2bow for every word in the texts and accordingly created and trained the model with LSI using the BoW corpus. For the TF-IDF variation we initiliazed the model using the gensim TF-IDF implementation (known as TfidfModel). Accordingly we transformed the earlier created corpus by using the TF-IDF created object applicable for converting any vector from the old representation to the new representation. Afterwards we trained the LSI model by using this TF-IDF corpus as input corpus and serialized the creation of the lsi model. For LDA we used a LdaModel instead of a LsiModel, and accordingly fit in the parameters. We printed the topics for all to answer AQ3.1. In addition, we made a ranking function to rank a given query to set of documents. For LSI BoW and TF-IDF we calculated the cosine similarity score for this, while for LDA we used the Kullback Leibler Divergence score.

We experimented with the parameters used but ultimately mostly used the default parameters as suggested by the creator of Gensim. Mainly the corpus, num_topics (=500) and id2word were accordingly changed to the relevant parameters (see code comments for more information). For the topics printed to derive to subjects we set the num of words to 5.

3.2 AQ3.1, Analysis of the results

Overall considerable differences between the three implementations were found in the results. Starting off with LSI using BoW gives us lesser results as can be seen in Figure 1, we can see that the most significant topics do not strongly resemble subjects and meaningless 'words' get higher probabilities (even though we did some additional pre-processing). The topics contain words such as 'said' that are hard to classify without context. However, this is expected as BoW is in fact a simplistic model that mainly looks at frequency of terms without placing it in the greater context.

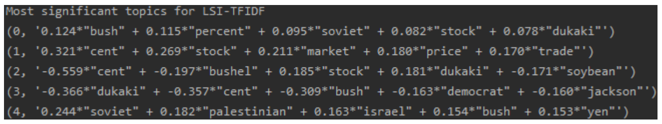


```
Most significant topics for LSI-BoW
(0, '0.705*"said" + 0.141*"would" + 0.139*"year" + 0.118*"state" + 0.108*"new"')
(1, '0.735*"percent" + -0.275*"said" + 0.205*"year" + 0.162*"million" + 0.129*"billion"')
(2, '0.447*"said" + -0.309*"bush" + -0.240*"presid" + 0.195*"percent" + -0.178*"state"')
(3, '-0.476*"percent" + 0.376*"cent" + 0.220*"million" + 0.183*"market" + -0.173*"0"')
(4, '0.282*"soviet" + 0.270*"govern" + -0.229*"n\'t" + 0.184*"u.s." + 0.167*"offici"')
```

Figure 1: Results for LSI using Bag of Words

When we go to the LSI implementation using TF-IDF we see much better results, all topics seem to resemble a subject. We derive that the most significant topics resemble 'hot' subjects in 1988 and 1989 such as the presidency campaign between Bush and Dukasis in

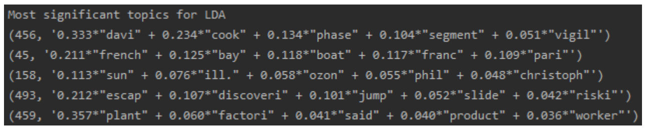
1988, the Palestinian and Israeli conflict and stock market crash (black monday) news. This can be seen in Figure 2.



```
Most significant topics for LSI-TFIDF
(0, '0.124*"bush" + 0.115*"percent" + 0.095*"soviet" + 0.082*"stock" + 0.078*"dukaki"')
(1, '0.321*"cent" + 0.269*"stock" + 0.211*"market" + 0.180*"price" + 0.170*"trade"')
(2, '-0.559*"cent" + -0.197*"bushel" + 0.185*"stock" + 0.181*"dukaki" + -0.171*"soybean"')
(3, '-0.366*"dukaki" + -0.357*"cent" + -0.309*"bush" + -0.163*"democrat" + -0.160*"jackson"')
(4, '0.244*"soviet" + 0.182*"palestinian" + 0.163*"israel" + 0.154*"bush" + 0.153*"yen"')
```

Figure 2: Results for LSI using TF-IDF

Lastly, LDA gives us different results as we can see in Figure 3. Since BoW is again used (as TF-IDF did not result in functioning code for 500 topics and this was allowed by the TA) we would expect lesser results. However we can see that the terms are more strongly linked than with the LSI results, and find higher probabilities. The topics seem to resemble subjects with france, the sun/ozon and a plant factory and workers. This observation is noteworthy, due to the fact that it was easier with LSI TF-IDF to trace the topics back to news topics in 1988 and 1989 and LSI TF-IDF seems to show more significant topics of that time (i.e. topics that everyone knows of). On the other hand, the set of terms do strongly resemble a subject here unlike with LSI BoW. The lesser traceability is likely the result of using 500 topics, which is handled differently by LDA versus LSI (see TQ3.1), and will in cases such as these result in these kind of 'oversaturated' distributions of topics (with a dataset of this size) that will make lesser significant topics more prominent.



```
Most significant topics for LDA
(456, '0.333*"davi" + 0.234*"cook" + 0.134*"phase" + 0.104*"segment" + 0.051*"vigil"')
(45, '0.211*"french" + 0.125*"bay" + 0.118*"boat" + 0.117*"franc" + 0.109*"pari"')
(158, '0.113*"sun" + 0.076*"ill." + 0.058*"ozon" + 0.055*"phil" + 0.048*"christoph"')
(493, '0.212*"escap" + 0.107*"discoveri" + 0.101*"jump" + 0.052*"slide" + 0.042*"riski"')
(459, '0.357*"plant" + 0.060*"factori" + 0.041*"said" + 0.040*"product" + 0.036*"worker"')
```

Figure 3: Results for LDA

The results show, overall, that there are staunch differences between the three. This can be attributed to using more simplistic models (BoW vs TF-IDF) as well as input parameters that get handled differently (num_topics for LDA and LSI). Overall with the specified settings for the purpose of the analysis we would say LSI TF-IDF worked best, while LDA gave the most interconnected terms.

4 RETRIEVAL AND EVALUATION

In this section we will compare the performance of the six models, for both default parameters as well as tuned the parameters and ultimately evaluate the results and seek the best performance.

Due to technical constraints we sampled the processed doc set and used 20% of randomly distributed documents for this purpose, this will reduce our scores overall as some of the queries will be strongly associated with some of the removed documents but due to lack of capable hardware we found this solution being the best for retrieving performance while obviously considering that we used a sampled set for conclusions.

4.1 AQ4.1

For this section we will run the retrieval methods (TF-IDF, word2vec, doc2vec, LSI-BoW, LSI-TF-IDF, LDA) for the default parameters. We will use MAP and nDCG values to evaluate the performance of these methods using the pytrec eval package in Python. The default parameters for this were vector dimension = 200, vocabulary size = 10000, window size = 5 for word2vec/doc2vec, and number of topics = 500 for LSI/LDA with the rest of the parameters as set default by gensim. The results can be found in Table 3

Table 3: ***Results AQ4.1 for all methods, with default parameters.***

	All		Query 76-100	
	MAP	nDCG	MAP	nDCG
TF-IDF 100%	0.2198	0.5819	0.1815	0.5414
TF-IDF 20%	0.0496	0.1712	0.0472	0.1618
word2vec*	0.00002	0.0008	0.00002	0.0007
doc2vec	0.0018	0.2462	0.00044	0.2297
LSI-BoW	0.0199	0.1191	0.0127	0.1201
LSI-TF-IDF	0.0295	0.1440	0.0281	0.1418
LDA	0.0111	0.0097	0.0102	0.0101

From Table 3 we can see differences between the different information retrieval methods, however some differences seem minimal. We can see in line with AQ3.1 that LSI-TF-IDF does in fact perform better than LSI-BoW, as well as LDA. We can see that doc2vec performs relatively well for nDCG but lacks for MAP, most likely as binary relevance is a too exclusive metric for doc2vec. Unfortunately we can't compare word2vec accurately with the rest*. To seek if it would indeed be better with the full dataset we ran TF-IDF for both our sampled down set of 1/5th of the documents as well as for the entire dataset. We can observe that TF-IDF does in fact perform better in the first case, and performs in line with the rest of our results for the same sample. We can derive from this that if we ran the full dataset for every method all scores would probably increase with a significant factor (with no changes in the implementation, but with better hardware or more time).

To seek if the differences are significant we will run the t-test as provided by pytrec_eval in AQ4.2.

*For word2vec the used sample was set at 1000 documents as the vector transformation for a word basis was too extensive for the computers, therefore the scores are much smaller than the rest.

4.2 AQ4.2

We ran the t-test functionality of pytrec_eval between each pair of

We set the nullhypothesis as 'There is no difference between model a and model b', we set the cut-off point for p at 0.05, meaning every value below this value can be rejected and thus indicate difference. From Table 4 we can observe that there is a significant difference between some model pairs, first off there is a significant difference between every pair where word2vec is involved in ($p<0.05$). This is most likely due to the fact of the small sample used for word2vec as this leads in great differences between the models. For doc2vec significant difference is found when comparing with LSI-BoW and LSI-TF-IDF, as it just under the 0.05. This is most likely due to differing approaches in calculating score for the documents. Lsi-BoW and LSI-TF-IDF do not give us significant differences, most

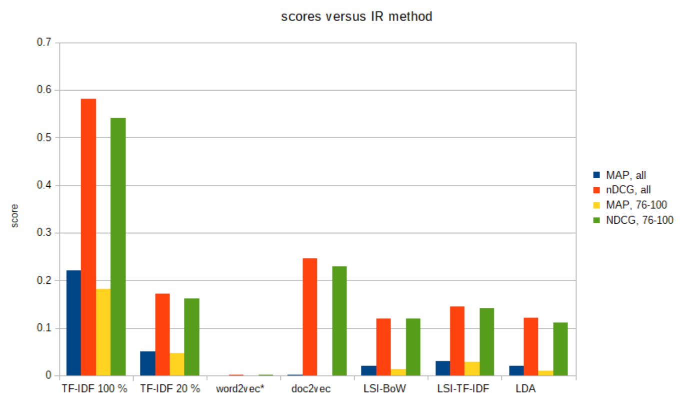


Figure 4: Results for AQ4.1, the different performances for IR

Table 4: *Results for the T-test for default parameters*

model-pair	p-value
word2vec* -> doc2vec	0.000038
word2vec* -> LSI-BoW	0.00000127
word2vec* -> LSI-TF-IDF	0.000000091
word2vec* -> LDA	0.000000132
doc2vec -> LSI-BoW	0.043
doc2vec -> LSI-TF-IDF	0.0408
doc2vec -> LDA	0.091
LSI-BoW -> LSI-TF-IDF	0.129
LSI-BoW -> LDA	0.2120

likely because both methods (albeit using different models) are based on LSI and thus are transformed via the LSI space. LDA (using bow) also does not significantly differ from LSI BoW, most likely for the same reason, while it does differ however for LSI-TF-IDF. Overall LSI-TF-IDF also outperforms LDA, just as we could visually inspect for AQ3.1.

4.3 AQ4.3

Table 5: *Results AQ4.3 for all methods*

	All		Query 76-100	
	MAP	nDCG	MAP	nDCG
word2vec	0.00028	0.0018	0.00032	0.0017
doc2vec	0.0021	0.2782	0.0081	0.2591
LSI-BoW	0.0222	0.1291	0.0169	0.1403
LSI-TF-IDF	0.0301	0.1701	0.0301	0.1572

We can see that tuning the parameters does in fact increase the scores for different methods. However, the increase is still relatively small. This is most likely due to the fact that we used a sampled dataset. If we used the full 160.000 documents we would see larger increase, as we demonstrated in AQ4.1.

4.4 AQ4.4

In line with AQ4.2 we conduct another t-test, this time between the model with the default parameters and the model with the updated parameters.

Table 6: *Results for the T-test for between default and tuned models*

model	p-value
word2vec	0.320
doc2vec	0.292
LSI-BoW	0.391
LSI-TF-IDF	0.298

The p-values we obtain are all above 0.05, indicating that the differences between the tuned and normal variation are not significant. This is most likely for the same reason as mentioned in AQ4.4, as we used 20% of the dataset making it

4.5 AQ4.5

We also observed the values for different parameters, avg MAP values for all queries and the test queries. In Table 7 we can observe the influence of varying the different parameter, with the vector dimension, window size and vocab size applicable for Doc2vec and Word2vec. The varying number of topics are applicable for LSI BoW, LSI TF-IDF and LDA. We visualized the results in 5 for comparison purposes.

Table 7: *Results AQ4.5 for different parameters*

	avg. all MAP	avg. test MAP
Vector dim: 200	0.00183	0.00210
Vector dim: 300	0.00413	0.00423
Vector dim: 400	0.00527	0.00519
Vector dim: 500	0.00357	0.00321
Window size: 5	0.00155	0.00150
Window size: 10	0.00112	0.00090
Window size: 15	0.00086	0.00070
Window size: 20	0.00068	0.00060
Vocab size: 10k	0.00231	0.00240
Vocab size: 25k	0.00244	0.00245
Vocab size: 50k	0.00378	0.004
Vocab size: 100k	0.0048	0.0047
Vocab size: 200k	0.0031	0.0029
Number of topics: 10	0.0006	0.0007
Number of topics: 50	0.0041	0.0060
Number of topics: 100	0.0051	0.0073
Number of topics: 500	0.0143	0.0129
Number of topics: 1000	0.0156	0.0160
Number of topics: 2000	0.0175	0.0180

Evaluating Figure 5 we can see that there are some key notions. The most influencing parameters seems to be the number of topics

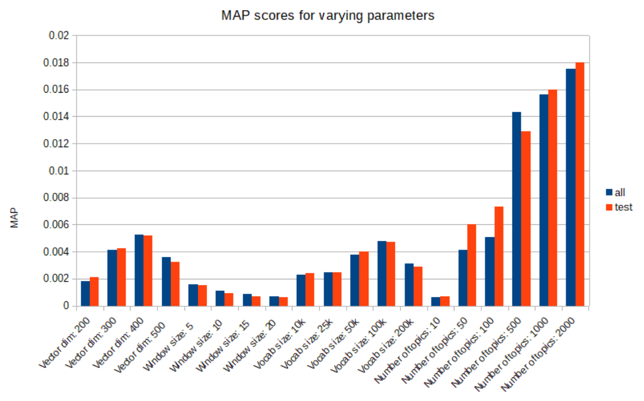


Figure 5: Results for AQ4.5, influence of varying parameters

used for LSI, as an increase in number of topics results in better performance up until 2000. From a theoretical perspective this makes sense as 160.000 news article will cover most likely a broad range of topics and thus a relatively larger number of topics will cover more space in the LSI distribution, increasing matching between queries and news topics. However bigger numbers of topics result in lesser results, (5000 and 10000) also from a computational perspective. This also makes sense as you can't oversaturate the dataset with topics, otherwise picking the highest possible number would result in the highest accuracy. Furthermore we can observe that for a vocabulary size of 100k the best results are gained and for vector dimension this is set at 400. we can see again the same pattern as with the number of topics: a gradual increase in performance but with a cap (as 100k performs better than 200k, and 400 performs better than 500), which will be due to the aforementioned reason for number of topics.

The window size seems to have the smallest difference between varying numbers and seems to not impact performance too much. Per gensim documentation it is said that window size is actually the 'maximum window size' and the actual size for any training example is configured between 1 and the set window size to improve the resulting vector performance on most posted metrics, implying that the set window size will most of the time not be used but rather be used as a limit which possibly explains the marginal differences.

4.6 AQ4.6

In this section of the report we will discuss the performance of the different retrieval methods on different queries and look and some of the queries the methods performed best and worst on.

LSI-BoW For LSI-BoW highest overall performance was found when the number of topics was set to 2000. In this setting the highest MAP for the queries in the range of 76-100 was found at query 78 with a MAP value of 0.099. The corresponding query to this index is "*Greenpeace*". The lowest performance of this algorithm found was at query query 87 with a MAP value of 0.0001. The corresponding query to this index is *Criminal Actions Against Officers of Failed Financial Institutions*. Possible reasons for the high performance of the query greenpeace could be that with the LSI models, unlike 2vec models, short queries can result in a higher amount of words directly related to the query whereas longer words might reduce the amount the relevance score for documents to the query.

LSI-TFIDF With the LSI-TFIDF algorithm the highest MAP score was found at 0.0775 for the query at index 78 being *Greenpeace*. Lowest MAP score was registered for the query at index 91 being *U.S. Army Acquisition of Advanced Weapons Systems* Just like in the previous model this shows that the LSI algorithms tend to perform well on short queries consisting only of a couple of words. Furthermore, the fact that the dataset is retrieved from the news might be a reason for the U.S. army being over represented in the data, which especially with TFIDF algorithms will result in lower score.

4.6.1 *Doc2Vec.*

For doc2vec the highest MAP score found was 0.013 for the query *"Downstream" Investments by OPEC Member States*. The lowest MAP score detected was for query at index 96 *Computer-Aided Medical Diagnosis* with a MAP score of 4.26e-5. The low performance of this query could possible be dedicated to the short length of the query. Furthermore, the difference in topics used in the query could lead to lower performance compared to the top query.

4.6.2 *Word2Vec.*

For the word2vec algorithm the highest noted score in the queries numbered 76 till 100 was 0.038 for query 90. The corresponding query to this index is *Data on Proven Reserves of Oil Natural Gas Producers*. The lowest score found for this algorithm was 9.2e-6 for query 80, which is the query *1988 Presidential Candidates Platforms* The low value of the later one could possibly be contributed to the short length of the query combined with the use of numbers and words that very often occur together like presidential candidates. Furthermore, it should be noted that a lot of queries for the word2vec ranking system had a MAP value of 0.0. This is because a subsample of the data had to be used to train the algorithm on due to technical limitations. This resulted in a lot of words present in queries not occurring in the vocabulary. When a word dit not occur in the vocabulary a vector of all zeros was used to represent the word. This resulted in some queries ending up as a vector of zeros or very small values.

4.7 AQ4.7

In this section we will look at the queries with the biggest deviation over the different systems. One of the most obvious differences was the difference in performance of the two main categories of algorithms on short queries. For example the query *Greenpeace* led to the best performance of both the LSI-BOW and the LSI-TFIDF algorithm while it showed very low MAP scores for the 2vec algorithms. The query *Poaching* showed similar behavior in that both the LSI algorithms found high MAP scores for this query (0.05 for LSI-BOW) contradictory to the 2vec algorithms (0.0004 for doc2vec), as well as *Iran-Contra Affair* (0.121 for LSI-TFIDF and 0.0005 for doc2vec). One of the longer queries of the corpus *Criminal Actions Against Officers of Failed Financial Institutions* also showed to be the lowest performing query for LSI-BOW.

Another query that showed major deviation in the performance when using different algorithms is the query *"Downstream" Investments by OPEC Member States*. This query, unlike the previous ones, scored higher when using the 2vec algorithms instead of the LSI algorithms. Although this is not one of the longest queries in the test set it does use some infrequent terms like OPEC. This could be a possible reason for the difference in performance, since the use of infrequent words can result in a high relevance of the top documents due to the word not being over represented in the training data when using 2vec models like skip grams.

5 THEORY QUESTIONS

(1) **TQ 1.1** We would make the medical data of the patients during the pre-processing step anonymous for research purposes. We would also analyze the missing values of patients and impute them or interpolate if needed. Otherwise exclude patients with too much missing data or have another reason to exclude them. Furthermore, we would analyze wrong values and data types and look for unusual outliers, and remove as a correction step.

(2) **TQ 2.1** Word embeddings are not always suitable for information retrieval due to the large size of many documents. When the dataset contains very large documents a metrix of the word embeddings would become very large very quickly. A common way to deal with this problem is to get a single vector representation for a document with a size similar to a single word representation by taking the mean over all the vectors that represents the words in the document. Again, however, when dealing with large documents all these means will become the same due to the enormous amount of words in the document even though the content of the documents might be very different

(3) **TQ 2.2** Word2Vec is the name for a group of models which are used to get vector representations of specific words. The ability to represent words as vectors makes them easier to interpret for a computer and makes it possible to perform calculations on words like we would normally do with numbers. One way of vector representations of words is by word embeddings. In word embeddings the vector is based on the similarity of the context of words and, therefore, caries information about the meaning of a word. For example, with word embeddings words like cat and dog will be closer together than cat and car, since these words often occur in similar context. (TQ 2.2) This ability to capture information about the meaning of words makes word embeddings different from other methods commonly used for information retrieval like TF-IDF and BM25, which only look at the words themselves.

(4) **TQ 3.1** The main difference between LSI and LDA is that LSI uses matrix decomposition on the associated term-document matrix to learn latent topics and therefore can be classified as a principal component analysis (PCA) which is applied to text data. In contrast, LDA is a generative probabilistic model that assumes a Dirichlet prior over the latent topics. There is also a difference in how the number of topics chosen impacts both, For LDA the number of topics is enforced before LDA is performed whilst for LSI the number of topics is chosen to get a hold of a significant amount of variation in the texts.

Similarity between the two is that both will return a (vector) representation of the original texts (pre-processed documents) and therefore have a similar 'approach' in how they model the topics from a given set of texts.

In our opinion LDA seems a better fit for information retrieval as the proven effective nature of the technique for large corpora (and the possibility to shard and map-reduce), which is necessary for information retrieval as usually relevant information will have to be extracted from large datasets. Accordingly, this will result in obtaining better accuracy from using LDA over LSI.

(5)

REFERENCES