Information Retrieval 1 **Homework 2 (120 points)**
2019-2020 Due Date: 1 March, 2020

In this assignment, you will work with a standard TREC retrieval dataset. You will learn about different distributed representations for text and compare their performance for the task of retrieval.

The submission for this homework is a single zip file. The name of this zip file should be your student ids separated by an underscore. You are required to submit the following:

- Your report, written in LaTeX, in a single PDF file. You can use this template. The report should contain implementation details as well as answers to both theory questions and analysis questions. The page limit for the report is 6 pages (including images, excluding references). Answers to the theory questions should be put into a separate section in the report.
- Your python files and/or notebooks with instructions on how to execute them in a README.md file.
- The output of your retrieval systems (TREC style results), one for each of the following: word2vec, doc2vec, LSI-Bow and LSI-TF-IDF.
- Your final grade will be determined based on the report and the submitted result files.
- This assignment counts for 20% of your grade

This assignment requires knowledge of PyTorch. If you are unfamiliar with PyTorch, you can go over these series of tutorials. You also to need to have a system with minimum of 8 GB RAM and a Linux operating system (or a Linux VM) in your team.

1 Loading and processing the AP dataset

We will work with the AP dataset. This dataset contains news articles published in Associated Press in 1988-89. Each document in the dataset contains different fields. We will only use the text of the article, along with document id. The dataset was used in TREC Ad-hoc Retrieval Track, associated with a set of queries and judgements. You can read about the dataset on TREC website. You are provided a couple of scripts to go help you with the homework.

The unzipped folder contains the following files:

- `docs`: a directory containing the documents.
- `qrels.tsv`: a file containing the relevance judgements for the queries.
- `queries.tsv`: a file containing the queries.

Execute `download_ap.py` script. This downloads the files and unzips them. Explore `read_ap.py`. This file contains helper functions to read the input files and the qrels, as well as simple text pre-processing steps. You can see how these functions are used in `tf_idf.py`.

Theory Questions

TQ 1.1: We are using the *Associated Press* dataset, which is a dataset in the *News* domain. If we were using a dataset in the medical domain, what pre-processing steps would you include / exclude? Justify your choice.

2 Word2Vec / Doc2Vec Models (40 points)

Recommended Reading:

- Word2Vec [4, 5]
- Doc2Vec [3]

Word2vec is a shallow, two-layer neural network trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located close to one another in the space [4].

Paragraph Vector, or doc2vec, is an unsupervised algorithm that learns fixed-length feature representations from variable-length pieces of texts, such as sentences, paragraphs, and documents [5]. The algorithm is an adaptation of the word2vec model for documents.

In this part of assignment, you are required to train word2vec and doc2vec models on AP documents, and use your models to answer the corresponding analysis questions. More specifically, you should:

- Implement the Skip-gram word2vec model using Pytorch. You need to implement and train the model yourself. Off-the-shelf implementations (such as the one in gensim) are not permitted. Limit your vocabulary size by removing infrequent words (a word that occurs less than $\tilde{5}0$ times in the corpus can be considered infrequent). Your implementation needs to have at least two tunable parameteres, namely the context window size and the dimension of output vectors. Your model should contain an inference function that returns an embedding for a given word.To obtain a representation for a document, you need aggregate the word embeddings into a single document embedding. This aggregation function is a parameter. Consider using the `nn.Embedding` module and `SparseAdam`. Furthermore, since the vocabulary size can be quite large, approximations of the softmax using Negative Sampling is encouraged. [4, 5]
- Use the gensim library to build a doc2vec model on AP documents. This tutorial is useful for this part of assignment.

Analysis Questions
Using the cosine similarity metric,

AQ2.1: (5 points) Find the top ten similar words to five words of your choice, based on their word2vec representations. Analyze the results. Are the words similar in your opinion?

AQ2.2: (5 points) Find the top ten similar documents to an arbitrary piece of text, using the doc2vec representations. Analyze the results. Are the documents similar in your opinion? If not, what is your analysis for the error?

Theory Questions

TQ2.1: Word2Vec, and word embedding models in general, might might not be suitable for retrieval. Why do you think this is?

TQ2.2: Comment on the key differences between methods you encountered in the previous assignment (TF-IDF, BM25) and Doc2Vec / Word2Vec.

Rubric (Implementation)

- Word2Vec
  - (5 points) Training a Word2Vec model for the AP Dataset
  - (2 points) Tunable parameters: Context Window Size and Embedding Dimension
  - (2 points) Function for producing a single vector for a document (given an aggregation function like `mean`)
  - (5 points) Function that ranks documents given a query
  - (2 points) Negative Sampling
  - (2 points) Use of `nn.Embedding`
  - (2 points) Use of `SparseAdam`
- Doc2Vec
  - (5 points) Code to train Doc2Vec model using gensim
  - (5 points) Function that ranks documents given a query

3 Latent Semantic Indexing and Latent Dirichlet Allocation (25 points)

Latent Semantic Analysis (LSA), or latent semantic indexing (LSI) in IR literature, is a theory and method for extracting and representing the contextual-usage meaning of words by statistical computations applied to a large corpus of text. The underlying idea is that the totality of information about all the word contexts in which a given word does and does not appear provides a set of mutual constraints that largely determines the similarity of meaning of words and set of words to each other [2].

Latent Dirichlet Allocation (LDA) is a generative probabilistic model for collections of discrete data such as text corpora. In LDA, each item, i.e., each document, is modeled as a finite mixture over an underlying set of topics. Each topic is, in turn, modeled as an infinite mixture over an underlying set of topic probabilities. In the context of text modeling, the topic probabilities provide an explicit representation of a document [1].

Recommended Reading:

- for LSI: Chapter 18 from "Introduction to Information Retrieval" book.
- for LDA: [1]

In this part of assignment, you are required to build a LSI model over AP documents using the LSI module in gensim. Set the number of topics to 500, and build two models: (1) Using a (binary) 'Bag of Words' term-document matrix and (2) a TF-IDF term-document matrix. Next, using the same number of topics, build a LDA model on AP documents with gensim. This tutorial is useful.

Analysis Questions

AQ3.1: (5 points) For each of the LSI models you built over AP, and for LDA, select the five top significant topics from your model. Check the top terms in each topic. Which topics actually represent a particular subject? Analyse the results. Do you observe a difference?

Theory Questions

TQ 3.1: Explain the differences and similarities of LDA and LSA. Which one is more suitable for IR in your opinion?

Rubric (Implementation)

- (5 points) Training an LSI Model using BoW
- (5 points) Training an LSI Model using TFIDF
- (5 points) Training an LDA Model using TFIDF
- (5 points) Function that ranks documents given a query

4 Retrieval and Evaluation (55 points)

In this part of assignment, we are going to use different models that are built in previous sections for retrieval, and compare their performance. For retrieval evaluation, we will use MAP and nDCG as metrics.
The aim is to compare the retrieval performance of five models:

1. TF-IDF: This is your benchmark. You are provided with a script (`tf_idf.py`) to run the benchmark and evaluate it on the AP dataset.
2. word2vec: for each query and document, the representation is obtained by averaging the vectors of their terms. For each query document pair, the relevancy score is the same as the cosine similarity score between their vectors.
3. doc2vec: for each query and document, the representation is obtained by inferring their vector from the model. The relevancy score is the same as the cosine similarity score between the their vectors.
4. LSI-BoW: LSI with bag of words weighing in term-document matrix. Each query and document is represented by a distribution over topics in a vector. The relevance score is the same as the cosine similarity score between the their vectors. This tutorial is useful for this part of assignment.
5. LSI-TF-IDF: Same as above, with TF-IDF weighting in term-document matrix.

To evaluate the retrieval performance, we will use pytrec_eval library in python [6]. For your reference, an example using this package is included in `tf_idf.py` script. If you are unable to use pytrec_eval, install trec_eval and use the API provided to you in the scripts.

Analysis Questions

AQ4.1: (5 points) Report the retrieval performance in terms of MAP and nDCG for all of the methods, on a) all queries, and b) queries 76-100. To be precise, you need to report 20 numbers in a table.

AQ4.2: (5 points) Significance testing is a way of showing that if the difference observed between the performance of two models is due to chance or not. Use `pytrec_eval` to perform a t-test between every pair of models, on the complete query set. You need to report the results of six tests.[a]

[a] take a look at this example.

To obtain the best retrieval performance, parameters of the retrieval models need to be tuned. For doc2vec, we will tune `window size`, `vocabulary size`, and `vector dimension` model. The optimal parameter set then is used for word2vec as well. For LSI, the `number of topics` is tuned. Use queries 76-100 as the validation set, and the rest of the queries as the test set. Select the parameters that produce the best retrieval performance in terms of MAP on the validation set.

- Search over $\{200, 300, 400, 500\}$, $\{5, 10, 15, 20\}$, $\{10k, 25k, 50k, 100k, 200k\}$ for `vector dimension`, `window size`, and `vocabulary size`, respectively. When fine-tuning each parameter, set the other two to the default value of your choice.
- For `number of topics`, search over $\{10, 50, 100, 500, 1000, 2000, 5000, 10000\}$.

Analysis Questions

AQ4.3: (5 points) Report the retrieval performance in terms of MAP and nDCG for all of the methods, on a) all queries, and b) test queries. Use the parameters leading to the best performance on validation set. Attach the result files corresponding to each run to your report.

AQ4.4: (5 points) Perform a t-test between the result of each method using the best parameters and the default ones used in AQ4.1. Describe your observations.

AQ4.5: (5 points)For each parameter, plot the retrieval performance in terms of MAP with respect to the value of the parameter, for a) all queries, and b) test queries. Describe your findings.

AQ4.6: (5 points) Doing a query-level analysis provides insights on the weaknesses and strengths of the retrieval models. For each of the four retrieval methods implemented above, find success and failure cases: queries for which the MAP was highest or lowest. Analyse the results, possibly with checking the qrels file. Discuss why you think each case happened.

AQ4.7: (5 points) Find the top-5 queries that have the highest variance in terms of MAP between different retrieval models. Provide an analysis why the performance of the models differs a lot on these queries compared to the rest.

The Result Files

The results file contains a ranking of documents for each query generated by the retrieval model. Each line in this file has the following format:

```
query-id Q0 document-id rank score STANDARD
```

The field query-id is the id of the query. The second field, with "Q0" value, is currently ignored by trec_eval, just put it in the file. The field document-id is the id of the document. The field rank is an integer value which represents the document position in the ranking. Make sure it is sorted, i.e., starts from rank 1. The field score can be an integer or float value to indicate the similarity degree between document and query, so the most relevant docs will have higher scores. The last field, with "STANDARD" value, is used only to identify each run (this name is also showed in the output), you can use any alphanumeric sequence.

For each retrieval model, attach the result file for test queries, corresponding to the parameters with the best performance on validation queries.

Rubric (Implementation)
Each TREC style result file carries 5 points.

References

[1] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[2] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

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

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

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[6] Christophe Van Gysel and Maarten de Rijke. Pytrec_eval: An extremely fast python interface to trec_eval. In *SIGIR*. ACM, 2018.