Project Report: Length-agnostic document retrieval using sentence-level embeddings

Udacity Machine Learning Engineer Nanodegree

Agnes van Belle

March 27, 2020

In this project we examine an approach for retrieving documents based on semantic similarity using sentence embeddings, such that documents with relevant passages are considered similar to the query, regardless of the amount of information they contain about other topics.

The proposed approach uses the Longest Common Subsequence algorithm combined with a threshold on sentence embedding similarity to select the longest overlap in relevant sentences. We evaluate against several baselines: other sentence-embedding-based baselines, a Doc2vec-based baseline and BM25-based baselines.

Results indicate that the proposed solution is indeed working the best in alleviating the described problem, however at the same time it loses some general retrieval performance.

1 Background and motivation

1.1 Domain and terminology

The domain background of this project is: Information Retrieval (IR) and Natural Language Processing (NLP). The goal of the project is to use sentence embedding vectors for length-agnostic semantic document retrieval.

Akin to Manning et al. [29] we define IR here as finding textual documents of a possibly unstructured nature that satisfy a user's information need. The information need is also called a "topic" and usually expressed by a "query", a short sequence of words issued to the search engine. The documents are found from within a large collection. The collection of documents also called an "index".

NLP is defined more broadly as the set of methods for making human language accessible to computers [23]. The primary goal of NLP is automated understanding of the semi-structured language that humans use. It targets various applications such as speech recognition, semantic analysis, summarization, text classification etc. Many of

these applications overlap with IR, and one could see IR as a broad target task for NLP. Many modern IR systems use NLP-approaches to be able to search on semantic or conceptual basis. A few nice examples are [18, 25, 31, 37].

In the field of IR there are some well-established search engines like Lucene [6, 7] which by default support only term-based scoring functions like BM25 [1, 26]. By term-based, we mean the relevance of a document to a query is based on the terms (also called words or tokens) in the query and document.

When it comes high-dimensional dense vector representations to capture the semantics of specific words, sentences, or documents, the best application technique with respect to search engines is an ongoing field of research without great task-independent consensus or implementation in well-established search engines.

1.2 Problem statement

Here we will focus on a specific aspect of similarity scoring in information retrieval. This aspect is with not too much effort tunable using term-based scoring, but is not tunable when using embedding techniques for document retrieval: document length normalization.

Documents in an index often considerably differ in size. By default, the established scoring functions in IR systems like Lucene take into account the length of a document (technically, a field, but here, we will use the term document) and apply a length normalization technique when calculating the term-frequency-based score [3, 4, 35] such that when two documents match the same amount of query terms, the longer document will have a lower final score. This is a process we will henceforth call "length normalization".

Using length-normalization when scoring documents for a query is not always desirable. In many scenarios of an IR system it is beneficial to ignore the length of the document. The documents difference in size could not be meaningful to relevance due to the nature of the task. For example, when ranking motivation letters of job applicants, longer doesn't necessarily mean less relevant. Or document extraction and splitting could be rather granular, leading to an unavoidable amount of irrelevant boilerplate per document. For example, when the index consists of scanned articles from old newspapers on which OCR often fails to recognize the boundaries between sections.

The effect of the document length normalization on a term-based scoring function like BM25 is clear and the normalization can also be turned off [1].

But this doesn't apply to embedding-based scoring functions. If we want to search for example by using aggregated word embeddings [14] or document embeddings as defined by Le and Mikolov [27], we ideally also want a way to turn off any document length normalization. **We would like documents with good and relevant passages to be considered similar to the query, regardless if they also contain information about some other topics.** This project focuses on finding a method to achieve this.

[1] See for example: `http://stackoverflow.com/questions/35897641/disable-length-normalization-on-a-single-field`

1.3 Related work

An obvious choice for representing document vectors is aggregating word or sentence embeddings [19, 20] and then taking their distance in embedding space as similarity measure, where the distance is measured e.g. as Euclidean distance or is based on cosine similarity. Similar research has also been done on combining word embeddings to generate sentence embeddings. One established baseline for good sentence embeddings by using word embeddings is from Arora et al. [15] who represent the sentence by a weighted average of the word vectors, and then modify them a bit using PCA/SVD. But by using that method or taking the average or maximum of word embeddings, the length of the document will still influence the similarity.

The same applies to approaches that make a single embedding for the entire document such as the well-established Doc2vec approach from Le and Mikolov [27] .

On the level of using word embeddings, some research has been done on using word embeddings to generate length-agnostic document embeddings, for example by Gupta et al. [24] and Mekala et al. [30]. However, these approaches require us to model topics explicitly.

The current state-of-the-art model for generating sentence embeddings is SBERT [32]. To our knowledge, there hasn't been a model addressing the best way to combine sentence embeddings to generate document representations for the use case of length-agnostic retrieval.

Although our problem is similar to the document ranking task in TREC 2019 Deep Learning Track [2] as this task also is affected by document size relative to relevant-passage-size, it is not the same as we evaluate by modeling the document length explicitly in a controlled way. Here we will focus on that problem for the goal of document retrieval for documents with varying lengths.

2 Approach overview

2.1 Proposed solution

To rank documents, between each query and document we apply the Longest Common Subsequence algorithm [5]. But instead of using letters and a binary match indicator, we do the following: We replace the letters with sentence embeddings, and use a certain fixed similarity threshold to determine if two sentences embeddings "match" or not. If so, we average these sentence embeddings per document. Then we calculate the similarity between two documents based on the distance between these two aggregated vectors.

The rationale behind this is that it allows us to preserve order in the query and document; have complexity of $\mathcal{O}(m \cdot n)$ (where m is the number of sentences in the query and n the number of sentences in the document); and allows us to have "noise" or irrelevant contents in between relevant sentences.

We will calculate all sentence embeddings using SBERT [32].

[2] `https://microsoft.github.io/TREC-2019-Deep-Learning/`

2.2 Overview of this report

Section 3 contains more information about the data we used and how we processed it. We used as dataset a subset of MS MARCO [9, 10, 11] consisting of queries collected from the Bing search engine and their relevant and irrelevant passages from larger documents. We did not use the data as-is, but artificially mixed relevant with non-relevant content to generate relevant documents of different lengths.

Section 4 contains more information about SBERT and how we tuned it by transfer learning.

Section 5 contains more information about our baselines. Since the focus is a model based on sentence embeddings, we include several other SBERT-based baselines. We also include two well-established non-sentence-embedding baseline to see if sentence representations help at all, namely BM25 and Doc2vec. We also describe the training procedure for Doc2vec.

Section 6 contains the evaluation results.

Finally, section 6.4 concludes this report.

3 Data

As dataset, we use MS MARCO [9, 10, 11], a large scale dataset focused on machine reading comprehension, question answering, and passage ranking. The queries are collected from the Bing search engine and the documents and their relevant passages from real returned documents. More on the generation can be found in [12].

Here we use the so-called "Train Triples Small" dataset. This training set contains approximately 40 million tuples of a query, relevant and non-relevant passages. Queries are repeated such that every query has 1 relevant doc and multiple irrelevant ones.

The result documents and hence passages have been pre-retrieved using term-based scoring functions.

Note that queries are single "sentences" or rather keyword queries here.

3.1 Data preprocessing

The original data file, 'triples.train.small.tsv', consists of 39780811 lines and is 27.2 GiB in size. It consists of linea with three columns: the query, a relevant document and an irrelevant document. There is only one relevant document per query. The entries in the file are not ordered by query.

To generate a more manageable subset of the data, I read in an amount of rows that still fits in memory, select the unique queries and then issue a grep command that searches for all entries of the query in the original data and I pipe the output of that to a file with the name of the query. This resulted in 4942 documents. Note that a single document might still contain multiple queries due to the grepped query matching documents of other queries.

3.2 Artificial extension of data

After the data preprocessing, we can generate the artificially extended dataset by reading in all these "query-files" lazily in batches to generate the artificially extended train data.

We want to extend the data such by artificially mixing relevant with non-relevant content to generate relevant documents of different lengths.

This resulting csv file has as columns the unique query ID, a query, a document, the relevance of the document to the query (zero or one) and the "type" of the document. This document type refers to the generation procedure.

In short we have the following types of new relevant documents:

- Degree-2 documents: We pick a random relevant doc A and then another random doc B without replacement. B belongs to the same query but is irrelevant. We combine these in a random order The new combined document gets attached to the result list of that query and gets labeled as relevant.
- Degree-3 documents: Same as the degree-2 documents, but now we combine three documents instead of 2.
- Degree-4 documents and degree-8 documents: As above.
- Degree-4-split and degree-8-split documents: like the above, but we also split the relevant document up such that there is random irrelevant content in-between two relevant passages (sequences of sentences). There is at least a single sentence between two relevant passages. We only do this for the degree-4 and degree-8 documents, because in the other cases there would be relatively little content between two relevant passages.

Note that for the degree-k document types, the document is always relevant. For the original document type, the document can be either relevant or irrelevant.

I also experimented with generating the irrelevant documents from the other queries, instead of from the same query. Based on manual inspection and preliminary analysis however I suspected this task would be too easy, in the sense that it would not make for a realistic IR scenario.

The final counts per document type are, per query, as follows:

- Original documents

  Relevant: 1

  Irrelevant: 15
- Degree-2 documents: 4
- Degree-3 documents: 4
- Degree-4 documents: 4
- Degree-8 documents: 4

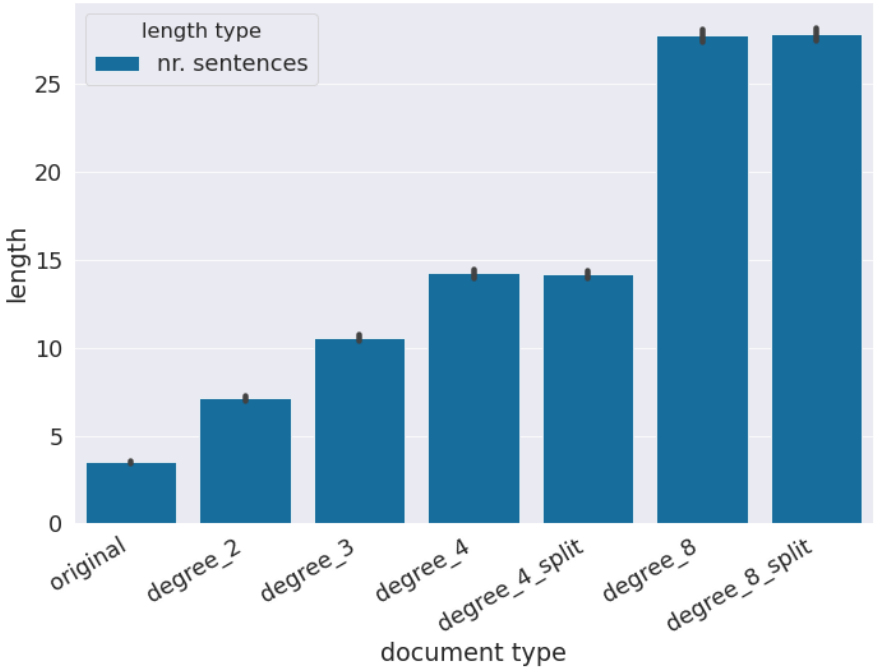

Figure 1: Average number of sentences per document type in the entire used dataset. The non-"original" types are artificially generated.

- Degree-4-split documents: 4
- Degree-8-split documents: 4

So in total we have 25 relevant documents and 15 irrelevant documents per query. Note that irrelevant documents are necessary to include for training the SBERT model.

The above procedure resulted in a csv file with 5152 queries and 206080 lines in total. This is a subset of 60% of all data in 'triples.train.small.tsv' which has 8852 queries.

In Figure 1 one can see the resulting average number of sentences per document type.

3.3 Creating train, development and test data

I split the data in a train set of 4602 queries, a development set of 150 queries and a final test set of 400 queries. The development set is used to tune the SBERT model (and the Doc2vec baseline model).

For the SBERT model, for the regression task, we need to give the data in a format where each row contains at least two sentences and their label (relevant or irrelevant, in our case).

For generating this data I used only the original documents, as they contain all the used sentences in the final evaluation task. I split the "query" and "document" content for each queries and documents in sentences. Then between the query sentences and document sentences I generated all possible combinations (the product). All sentences

from a relevant document were then labeled as positive and all sentences from negative doc were labeled as negative. To avoid bias during training, I also made sure that per query there were as many positive (relevant) as negative (irrelevant) labels, by duplicating the positive labels if necessary.

This resulted in a final train dataset for the SBERT regression model of 488047 instances, and a development dataset of 16259 instances.

4 The SBERT-based model

SBERT is an additional modification of a pretrained BERT network that uses siamese or triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity.

The main benefit of SBERT over BERT is that it generates meaningful sentence embeddings for single sentences, where the representation can be trained according to a specific task.

The main original motivation for creating such independent representations is to efficiently be able to find similar sentences. The intended way of using BERT is that one inputs two sentences and BERT gives a score as output. To find the most similar sentence in a set of N sentences to an input sentence, one would have to compare all N sentences with the input sentence. Whereas if one could use embeddings in the form of vectors as representations of the sentences, one could use a more efficient nearest-neighbours-based approach.

Another advantage of SBERT which is most relevant to this project, is that the embeddings that one could generate from BERT - this is either done by averaging the BERT output layer (averaging the tokens for a sentence) or, in case of a classification task, by using the output of the "special" first token, the so-called CLS-token - are performing rather poorly on downstream tasks (e.g. for regression or classification) [32]. SBERT generates better embeddings for these tasks by extending BERT such that the final network can be trained directly on a regression or classification task.

4.1 Choice of pretrained model

There are three types of objective functions implemented in SBERT which correspond to three different model types. There is a classification objective function, where it tries to predict a label based on the concatenations of two sentences. There is a regression objective function based on the Mean Squared Error of the cosine similarity difference between two sentences. This architecture is depicted in Figure 2. Finally, there is a triplet loss objective function that is based on three sentences and one checks if the positive example is closer to the anchor than the negative example.

In terms of the type of model, the authors of the SBERT paper have evaluated and provided different types of pretrained models. For classification or regression, there is the choice between an SBERT model trained on only a dataset that we will call "NLI" (a combination of the SNLI [16] and the Multi-Genre NLI [36]), which uses the classification objective function, and one thereafter also trained on the STS benchmark (STSb) [17]

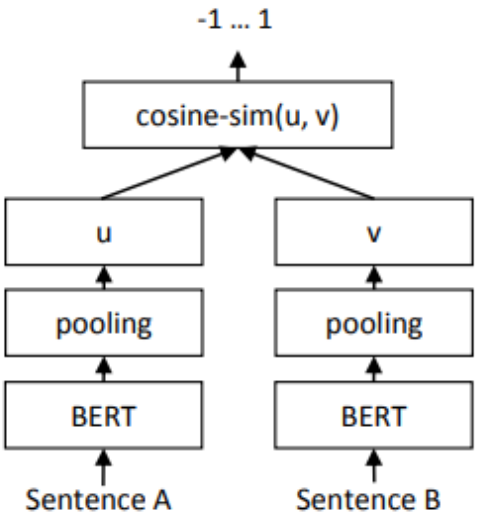

Figure 2: SBERT regression task learning and predicting architecture [32]

data, which uses the regression objective function. Then, there are differences regarding the original BERT models size ("base" or "large"), the type of "BERT-model used as basis (BERT [21], RoBERTa [28] or distilbert [33]) and the way the sentences are combined for the classification task on the NLI dataset. Finally, in case of the triplet type of task, they provide a model that is pretrained on the Wikipedia dataset of Ein Dor et al. [22].

For our project we want to choose the most similar document to a query, hence the regression-type of model that is pretrained on first the NLI and then the STSb dataset seemed the most natural choice. Of these, we chose the large RoBERTa-based one, as it achieved the best performance on STS benchmark tests according to the documentation in their GitHub repository [13], named 'roberta-large-nli-stsb-mean-tokens'. The dimension of the final embedding under this model is 1024.

I have also experimented with the single triplet loss model that is provided ('bert-base-wikipedia-sections-mean-tokens'), however I discontinued on that road as it did not produce good initial results.

4.2 Transfer learning approach

For training the model, I transformed the dataset to the appropriate format (see section 3). For the loss function I used the default recommended Mean Square Error of the Cosine Similarity.

I experimented manually a bit with the number of epochs, learning rate, the amount of data used for "learning rate warm-up (the amount of data used to linearly increase the learning rate in the beginning) and number of steps per epoch.

The learning rate proved to perform best a factor of 10 smaller than in the examples and paper, namely 2^{-06} (instead of 2^{-05} or 2^{-07}), and the best setting for the learning rate warm-up was 10% (instead of 5%). In the end I used 1000 steps (instances or rows) per epoch and a maximum of 100 epochs. This would lead to a maximum train data set usage of 100000 instances which would in its loop over about a quarter of the entire train data set (which contained 488047 instances, see section 3). I used shuffling of the train data before training.

Each epoch it is checked how the model performs on the development set, if it is the best-performing model so far, it is saved. For measuring the best performance several functions can be used, I chose the "cosine accuracy" which calculates the cosine distance between the pairs and checks if sentence pairs with a label of 1 are in the top 50% of the entire development set, and pairs with a label of 0 are in the bottom 50%. Due to the data format being provided for the regression task, one can not optimize per query. There are also other metrics provided such as the Pearson correlation between the ranking induced by the labels and the ranking induced by the cosine similarity scores ("cosine pearson"), however I found these to be correlated and also more subject to randomness (which seems natural as we have only binary labels, meaning there is not much granularity in ordering that we can capture with the labels).

To run the training I used an AWS instance of type ml.p2.16xlarge because of its relatively large GPU performance. However, it should be noted that the SBERT code sbe [13] does not use multiprocessing and hence the training could have been faster than it was. The batch size I kept as the same one as in the examples, 8, because a larger size led to CUDA memory limitations.

4.3 Results

After 31 epochs the SBERT model got its best performance of the primary development set metric, namely a "cosine accuracy" of .732, compared to .64 before training. In Figure 3 we show the plot of the "cosine accuracy" and "cosine pearson" scores (their definition is described above).

After 31 epochs, the model seemed to be overfitting, as its performance on the development set decreased. We used the model as achieved using 31 epochs as the final model.

This model had, by using 31 epochs, used 31000 instances of the dataset, of a total of 488047. Hence, we used only a random sample of 6% of the total amount of train data. It still took a considerable amount of time; about 12 hours.

5 Baselines

Below we describe our baselines. We use both other sentence embedding based approaches, as well as more common baselines like BM25 and Doc2vec. For the other sentence embedding based approaches, we used the same SBERT model as we used for our LCS approach.

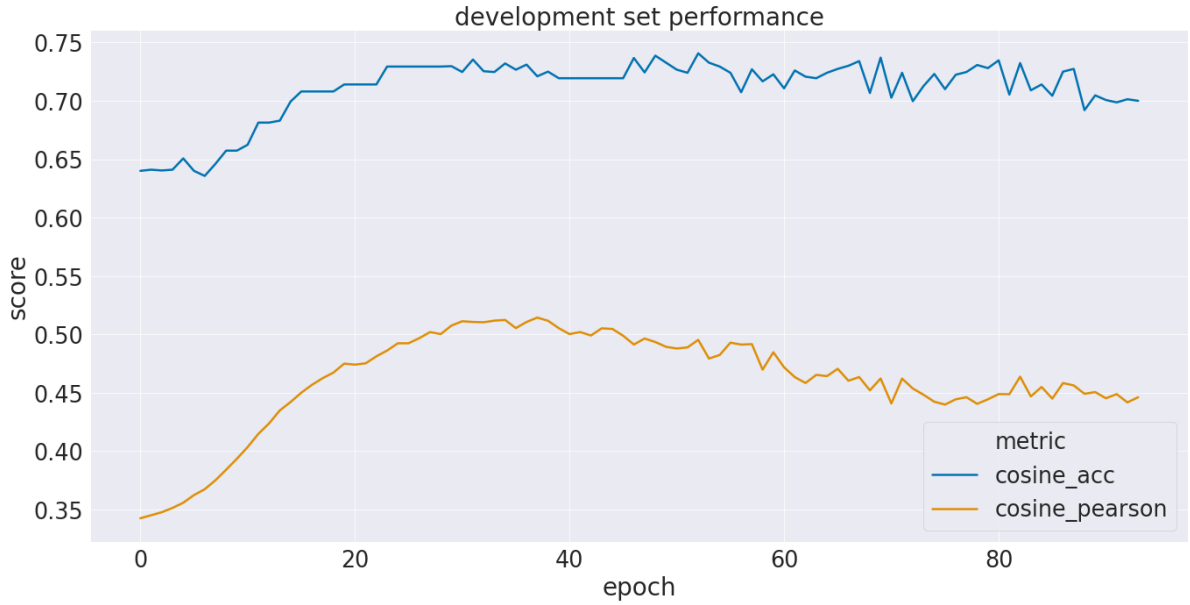

Figure 3: Performance of training the chosen pretrained SBERT model per epoch on the development set (150 queries)

For the all approaches outlined below, we also include a variant where we multiply the score between query and document based on the fraction of how much more sentences the document has compared to the query. For example, if the query is 4 sentences and the document 20, then we weight the similarity score by a factor of 5.

Originally, we also wanted to represent the entire query and document as a sentence in and calculate the similarity based on these two vectors' distances. However this proved to be not a good benchmark as the SBERT model can only accept up to 128 tokens as input, and most of our generated data has more tokens that that.

5.1 BM25

BM25 [1, 26] is not based on any type of semantic similarity at all and merely serves as a sanity check baseline.

We turn the length-normalization part in BM25 (parameter "b") off.

We also experiment with using stemming or not, and with we experiment with using the IDF or not. The IDF, in our case, is query-dependent and based on the documents that are supposed to be ranked for that query.

5.2 Doc2vec

Doc2vec [27] is not based on sentence embeddings but a well-established semantic similarity model.

We used the Gensim implementation [2] of Doc2vec.

When we train Doc2vec we use as "documents" all queries and documents in the train set. Note that Doc2vec expects the documents to be represented as already-preprocessed lists of words. We use preprocessing on all documents in the form of tokenizing, lowercasing, removing punctuation, and removing stop words.

We conducted a randomized grid parameter search to find the best parameters.

As loss function to evaluated the trained models during the parameter search, we use the Mean Squared Error of the cosine similarity between the pairs in the development set compared with the actual label (0 or 1). This is the same loss function as used by the regression-based SBERT model.

In total we tried forty different parameter combinations in two rounds. We first examined twenty sampled parameter combinations and then examined twenty other parameter combinations close to the best parameter combination of the previous round. The parameter combinations varied in the number of epochs, final embedding dimension, window size, minimum learning rate and model type ('distributed memory' or 'distributed bag of words').

The final model had a Mean Squared Error cosine similarity loss of 0.30725 on the development set. It used a distributed bag of words approach for 200 epochs, a window size of 3, and a minimum learning rate of 1^{-05}. Its embedding dimension is 1024 which is the same as the embedding dimensions of the SBERT model we use.

During inference, we use the "infer_vector" method of doc2vec to get the embedding of a query or document.

The final score between a query and a document is calculated using the cosine similarity between the aggregated embeddings.

5.3 Averaging sentence embeddings

We calculate the sentence embeddings for all sentences in the document and use their average on each dimension to represent the document. The final score between a query and a document is calculated using the cosine similarity between the aggregated embeddings.

5.4 Taking the maximum of sentence embeddings

Calculate sentence embeddings for all sentences in the document and use their maximum on each dimension to represent the document. The final score between a query and a document is calculated using the cosine similarity between the aggregated embeddings.

5.5 Examine all pairs of sentence embeddings

Calculate sentence similarity for all pairs between the query and document. The final score between a query and a document is calculated using the cosine similarity between the aggregated embeddings.

This is $\mathcal{O}(m^n)$ complexity in practice, and hence not practical. It merely serves as an interesting baseline as it should yield a higher performance than our proposed model.

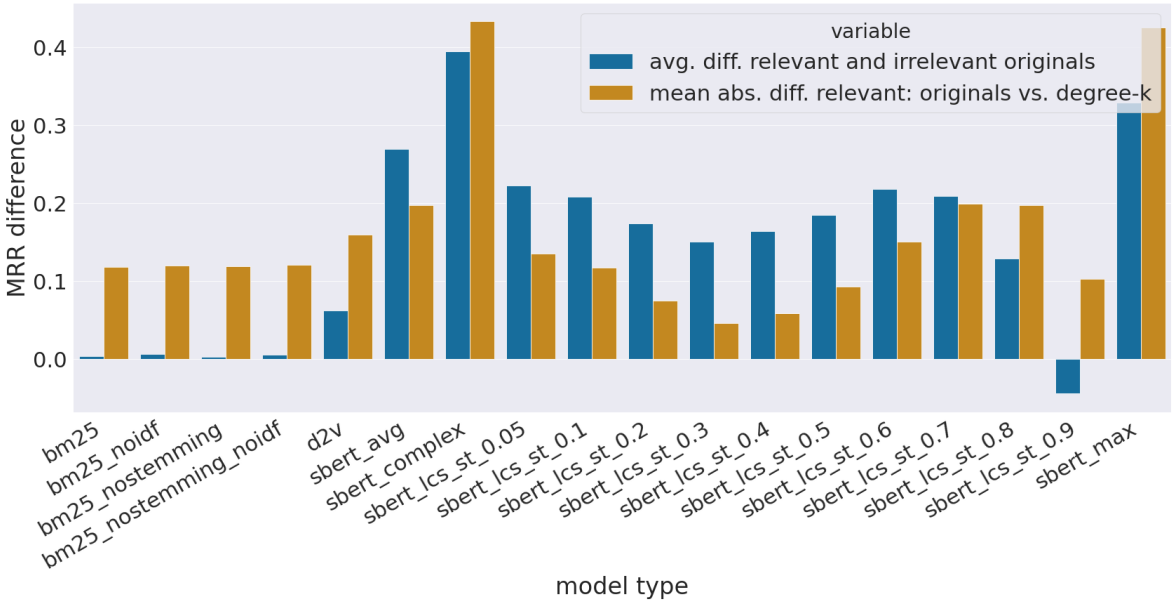

Figure 4: Performance of different proposed and baseline models on the test set. See text for details.

6 Evaluation

6.1 Metric

As metric we use the Mean Reciprocal Rank (MRR)[8], a well-established metric in the field of IR that is especially suitable for tasks that need a high precision in the top document ranks [34]. So it is suitable for our dataset where the goal is getting the first relevant result as high as possible in the ranked list.

The MRR can be described as measuring, per query, the inverse of the rank of the first correct answer. The MRR is hence defined to be 1 is the relevant document is at the first place, $\frac{1}{2}$ if it is at the second place, $\frac{1}{3}$ if it is at the third place, etc. Then, it takes the mean over these results for all queries.

6.2 Main result

What we want for our best model is twofold. First, we want it to ideally see no difference in MRR between relevant documents of varying length, which have an equal amount of relevant content.

Second, we want to keep the main task of an IR system in mind too: we should have both a good general MRR performance, meaning that relevant documents should have a higher score than irrelevant ones.

Figure 4 shows the performance of a subset of the tested models on these two metrics. Here, "complex" stands for the model that examines all pairs of sentence embeddings.

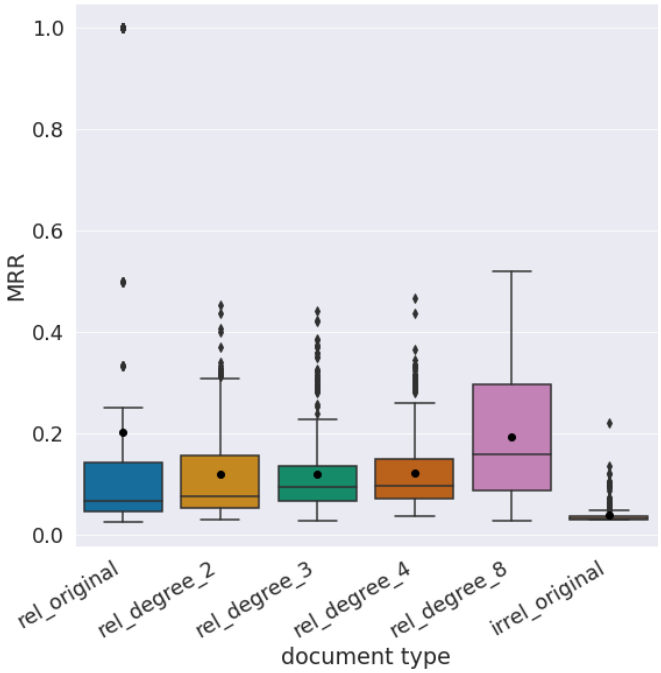

Figure 5: Performance of the SBERT LCS model with minimum similarity threshold of 0.4 on the original documents and the different artificially generated degree-k documents.

"lcs_st_.." stands for a model based on our proposed Longest Common Subsequence approach, that uses a certain similarity threshold.

For each model, the first column shows the difference of the MRR between relevant and irrelevant original documents (higher is better). The second column shows the difference of the MRR between the original relevant documents and the degree-k relevant documents (lower is better).

We see that the models that perform the highest on ranking relevant original documents to the top are generally also performing poorly when it comes to minimizing the difference between the original relevant documents and the degree-k relevant documents.

If we agree that the average "difference between relevant and irrelevant original documents" should be clearly positive, let's say higher than 0.1, and that we want to maximize the difference between that metric and the "difference of the MRR between the original relevant documents and the degree-k relevant documents", then **we find that the LCS model with a similarity threshold of 0.4 performs best**.

6.3 Performance per document type

In Figure 5 we see the performance of the LCS model with a similarity threshold of 0.4, segmented per the different original documents and the different artificially generated degree-k documents. We see that the distributions of the different relevant documents

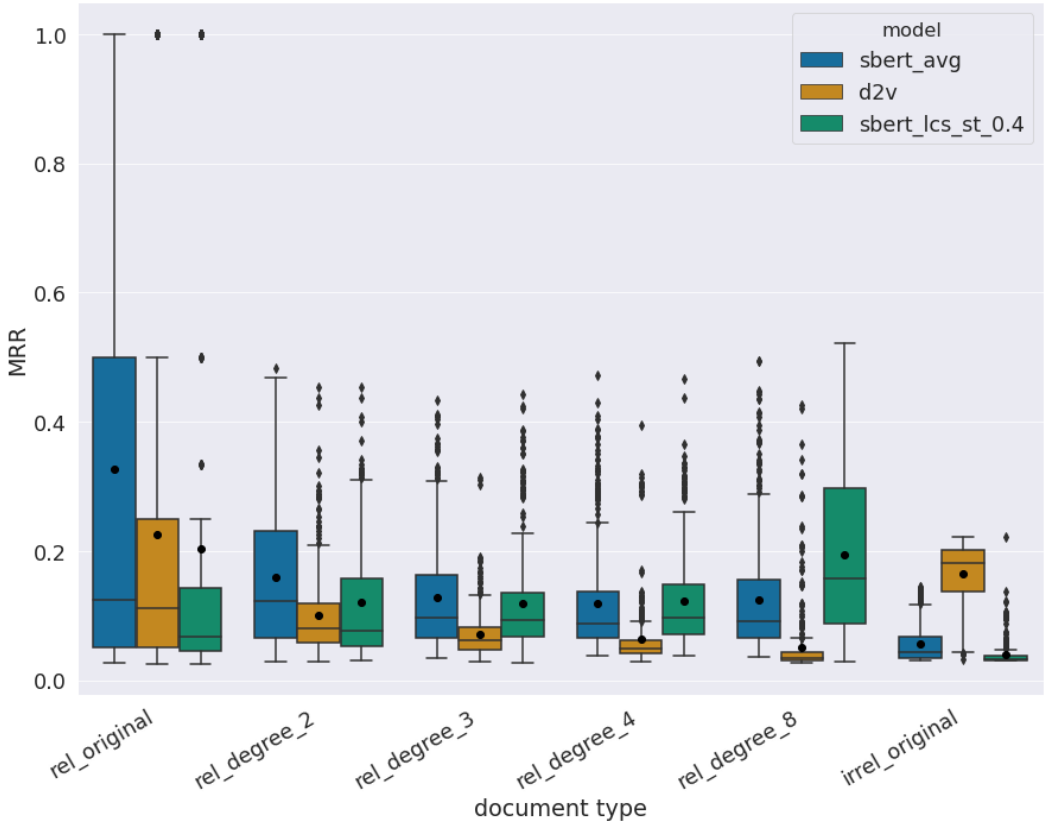

Figure 6: Performance of the SBERT-average; Doc2vec and SBERT-LCS model with minimum similarity threshold of 0.4, on the original documents and the different artificially generated degree-k documents.

indeed lie relatively close together. We also see that de degree-8 documents have strange enough a MRR distribution that is much more tailed towards higher values of the MRR, with a higher third quartile and maximum MRR values than the original documents.

In Figure 6 we compare this best model with the model that averaged the sentence embeddings, and the Doc2vec-based model. We see that the Doc2vec-based model has a bias for shorter documents, and additionally is not very good at ranking as its mean MRR for irrelevant documents is relatively high.

We also see that the model that averages the embeddings has a relatively large bias for shorter documents. However, it does not have the strange boost for degree-8 documents.

As can be see in Figure 7, this bias exists in multiple LCS-based models.

I also briefly looked at the difference between the degree-k documents and the degree-k-splitted documents. However I did not see any noticeable difference there. See Figure 8

In general, I also examined the variant pf each model where we multiply the score between query and document based on the fraction of how much more sentences the

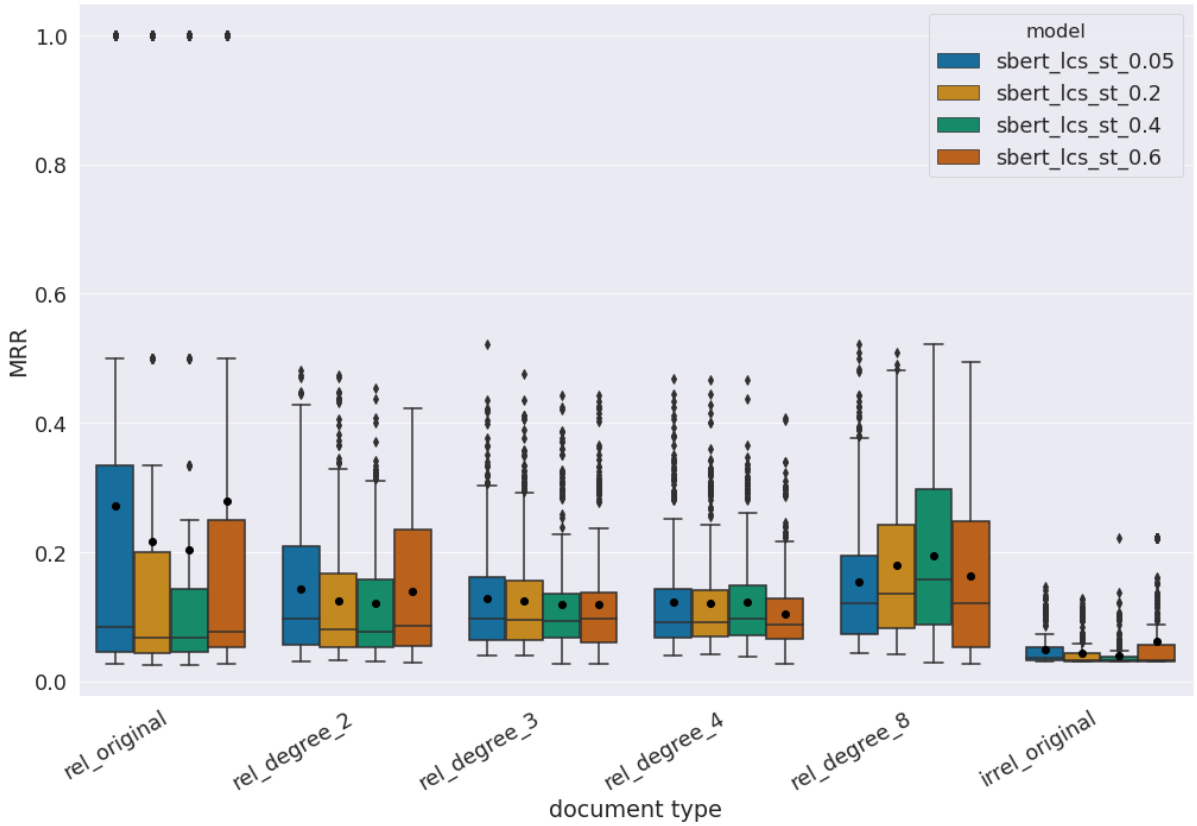

Figure 7: Performance of the SBERT-LCS model with various minimum similarity thresholds, on the original documents and the different artificially generated degree-k documents.

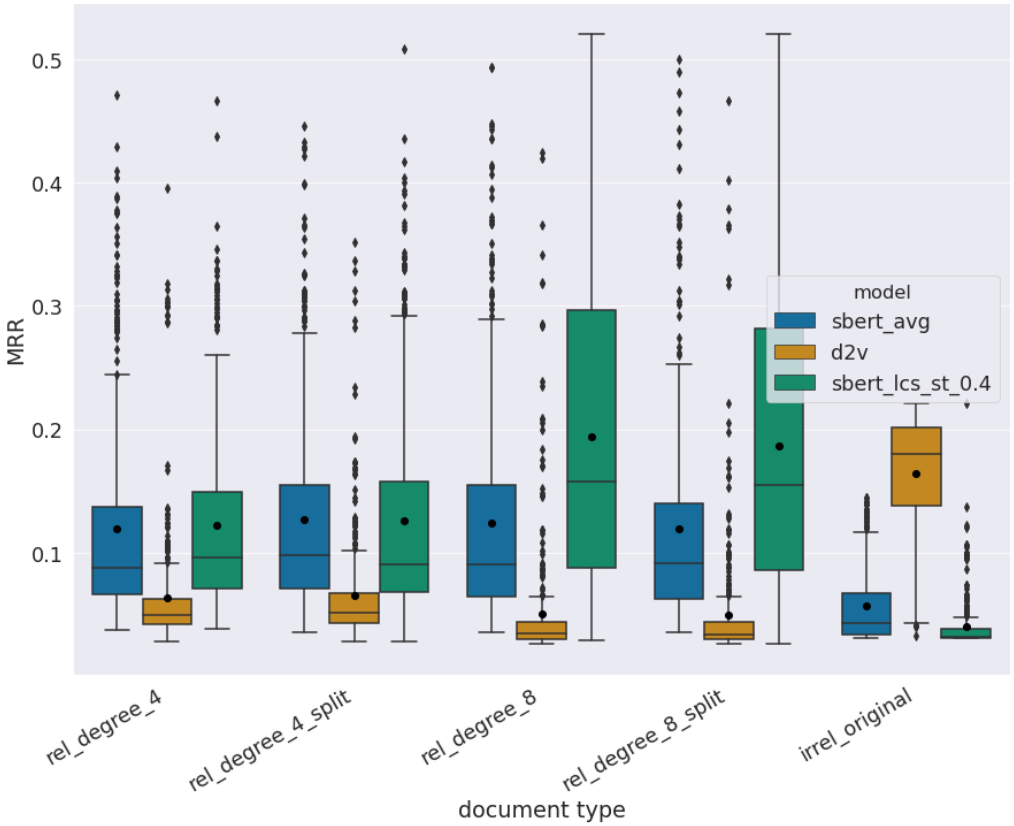

Figure 8: Performance of the SBERT-average; Doc2vec and SBERT-LCS model on the degree-k documents versus the degree-k-splitted documents and irrelevant documents

document has compared to the query. However, these performed all very clearly very poorly and I hence did not further include them in my analysis.

Also note that, on first sight it may seem odd that BM25 performs so poorly here, but note that, as described in section 3, the data we used was already containing per query solely documents pre-retrieved using similar term-based scoring functions.

6.4 Conclusion

It seems possible to reduce the bias for shorter documents, but it comes at the cost of the ranking quality in terms of ranking relevant documents above irrelevant documents. Hence this procedure should be chosen based on a trade-off of these two aspects, whose weights should be determined by the practical task (search engine use cases, document collection) at hand. The proposed Longest Common Subsequence-based approach, with a similarity threshold of 0.4, seems in general the best approach when wanting to minimize the length bias while at the same time ensuring at least an average difference in Mean Reciprocal Rank of 0.1 between relevant and irrelevant documents.

It is notable that using our proposed Longest Common Subsequence-based models, the length bias is eliminated for the documents up to degree-4 but an opposite bias seems to emerge for the longest degree-k documents. Perhaps the larger size of the documents allows for more false positive "sentence matches". This is something that can be investigated further.

References

[1] BM25 Wikipedia article, 2020. URL `https://en.wikipedia.org/wiki/Okapi_BM25`.

[2] Doc2vec homepage, 2020. URL `https://radimrehurek.com/gensim/models/doc2vec.html`.

[3] Lucene's TF-IDF Similarity class documentation, 2020. URL `https://lucene.apache.org/core/4_5_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html`.

[4] Lucene's Similarity class documentation, 2020. URL `https://lucene.apache.org/core/8_4_1/core/org/apache/lucene/search/similarities/Similarity.html`.

[5] Longest common subsequence Wikipedia article, 2020. URL `https://en.wikipedia.org/wiki/Longest_common_subsequence_problem`.

[6] Lucene Core homepage, 2020. URL `https://lucene.apache.org/core/`.

[7] Lucene Wikipedia article, 2020. URL `https://en.wikipedia.org/wiki/Apache_Lucene`.

[8] Mean Reciprocal Rank Wikipedia article, 2020. URL https://en.wikipedia.org/wiki/Mean_reciprocal_rank.

[9] TREC 2019 Deep Learning Track Guidelines, 2020. URL https://microsoft.github.io/TREC-2019-Deep-Learning/.

[10] MS MARCO homepage, 2020. URL https://microsoft.github.io/msmarco/.

[11] MS MARCO Passage Reranking GitHub repository, 2020. URL https://github.com/microsoft/MSMARCO-Passage-Ranking.

[12] MS MARCO Question Answering GitHub repository, 2020. URL https://github.com/microsoft/MSMARCO-Question-Answering.

[13] SBERT GitHub repository, 2020. URL https://github.com/UKPLab/sentence-transformers.

[14] Felipe Almeida and Geraldo Xexo. Word embeddings: A survey, 01 2019.

[15] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. 1 2019. 5th International Conference on Learning Representations, ICLR 2017 ; Conference date: 24-04-2017 Through 26-04-2017.

[16] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1075. URL https://www.aclweb.org/anthology/D15-1075.

[17] Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/S17-2001. URL https://www.aclweb.org/anthology/S17-2001.

[18] Ilias Chalkidis and Dimitrios Kampas. Deep learning in law: early adaptation and legal word embeddings trained on large corpora. *Artificial Intelligence and Law*, 27: 1–28, 12 2018. doi: 10.1007/s10506-018-9238-9.

[19] Md Faisal Mahbub Chowdhury, Vijil Chenthamarakshan, Rishav Chakravarti, and Alfio M. Gliozzo. A study on passage re-ranking in embedding based unsupervised semantic search, 2018.

[20] Cedric De Boom, Steven Van Canneyt, Thomas Demeester, and Bart Dhoedt. Representation learning for very short texts using weighted word embedding aggregation. *Pattern Recogn. Lett.*, 80(C):150156, September 2016. ISSN 0167-8655. doi:

10.1016/j.patrec.2016.06.012. URL https://doi.org/10.1016/j.patrec.2016.06.012.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[22] Liat Ein Dor, Alon Halfon, Yoav Kantor, Ran Levy, Yosi Mass, Ruty Rinott, Eyal Shnarch, and Noam Slonim. Semantic relatedness of Wikipedia concepts – benchmark data and a working solution. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL https://www.aclweb.org/anthology/L18-1408.

[23] J. Eisenstein. *Introduction to Natural Language Processing*. Adaptive Computation and Machine Learning series. MIT Press, 2019. ISBN 9780262042840. URL https://books.google.de/books?id=72yuDwAAQBAJ.

[24] Vivek Gupta, Ankit Kumar Saw, Partha Pratim Talukdar, and Praneeth Netrapalli. Unsupervised document representation using partition word-vectors averaging, 2019. URL https://openreview.net/forum?id=HyNbtiR9YX.

[25] Isa Inuwa-Dutse, Mark Liptrott, and Ioannis Korkontzelos. A deep semantic search method for random tweets. *Online Social Networks and Media*, 13:100046, 2019. ISSN 2468-6964. doi: https://doi.org/10.1016/j.osnem.2019.07.002. URL http://www.sciencedirect.com/science/article/pii/S2468696419300382.

[26] K Jones, Shelia Walker, and Stephen Robertson. A probabilistic model of information retrieval: development and comparative experiments: Part 2. *Information Processing & Management*, 36:809–840, 11 2000.

[27] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents, 2014.

[28] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

[29] C.D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN 9781139472104. URL https://books.google.de/books?id=t1PoSh4uwVcC.

[30] Dheeraj Mekala, Vivek Gupta, Bhargavi Paranjape, and Harish Karnick. SCDV : Sparse composite document vectors using soft clustering over distributional representations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 659–669, September 2017.

[31] Manish Patel. TinySearch – Semantics based Search Engine using Bert Embeddings, 08 2019.

[32] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks, 08 2019.

[33] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2019.

[34] Chirag Shah and W. Bruce Croft. Evaluating high accuracy retrieval techniques, 2004.

[35] Doug Turnbull. BM25 The Next Generation of Lucene Relevance, 2020. URL `https://opensourceconnections.com/blog/2015/10/16/bm25-the-next-generation-of-lucene-relevation/`.

[36] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1101. URL `https://www.aclweb.org/anthology/N18-1101`.

[37] Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu, and Chang Liu. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE 16, page 404415, New York, NY, USA, 2016. Association for Computing Machinery.