On My Disk integration: Milestone 2, Deliverable 3 Training and optimization of PeARS model

The PeARS team members

This technical report describes the training and evaluation of the PeARS model that will be used by the On My Disk (OMD) system for the English language. The goal of this evaluation step is to quantify the performance of the search function. The associated code is available on the GitHub branch ngi-search-eval (`https://github.com/PeARSearch/PeARS-lite/tree/ngi-search-eval`).

1. Introduction

An accurate search engine has to fulfil two requirements: it should return *all* the documents relevant to a query q, and it should avoid returning *any* document irrelevant to q. The former requirement is known as the ‘recall’ of the system, while the latter is referred to as ‘precision’.

The recall and precision of a search system depend on the mathematical representations given to documents in the process of indexing. Such representations are the outcome of a complex pipeline involving data preprocessing, text tokenization and vectorization. The PeARS model itself relies heavily on the preprocessing of the data fed to the indexer, as well as the exact hyperparameters used during tokenization. Because of this, the model has to be optimized according to an evaluation pipeline.

This document describes the steps undertaken to ensure the best possible performance of the PeARS model in the context of a private cloud service such as On My Disk. It describes our evaluation pipeline and reports experimental results on artificial data, allowing us to draw conclusions about the desired properties of our model

2. Evaluation pipeline description

The PeARS evaluation pipeline is run independently from the OMD server, on a local machine. Its main purpose is to quantify the search model independently from user interfaces or APIs. It includes four steps: a) creation of a test user from a collection of documents; b) indexing of the document collection; c) automatic generation of search queries over the indexed data; d) quantitative evaluation of the system’s search results over the generated queries.

Testing search results in the context of a private cloud solution is challenging from a machine learning perspective, since we do not have access to the users’ privately stored data, and therefore cannot directly train and evaluate the PeARS model over the correct data distribution. To alleviate this problem, we deploy our development pipeline over artificially-generated data – a common practice in cases where data sparsity hinders the training process. We describe later how we collect such data.

2.1 The model

The PeARS model consists of an indexing and a search module. The indexer is responsible for processing raw text and transforming it into a fixed-size vector, amenable to linear algebra operations. Briefly, the indexing pipeline involves pre-processing the raw text, tokenizing it, and using the tokenized version to compute a document vector. The search module reuses the indexing process on-the-fly to compute a vector for a given user query, and ranks the relevance of indexed documents by computing cosine distance between the query representation and the previously indexed documents.

2.2 The data

We seek to generate 'extreme' user types, to assess the extent to which performance varies across very different data distributions. We decide to create four personae: an accountant, a HR manager, a film enthusiast, and a news reader. The kind of data stored by each persona follows its own specific distribution.

As shown in the Table, for replicability purposes, we mostly use a combination of freely available corpora[1] and Wikipedia data (preprocessed with the open-source WikiLoader).[2] We also create a small amount of data using a Large Language Model (LLM), which we are however not releasing due to the current debates on LLMs and copyright.

2.3 Evaluation metrics

Following standard practice, the model is evaluated using recall, precision and F-score. For a given query, recall is the percentage of relevant documents returned by the system, precision is the percentage of returned documents that are actually relevant, and F-score is the average of precision and recall.

In order to compute these metrics, we need a 'gold standard', i.e. a dataset of queries associated with the documents that match the query (the 'relevant' documents), for each individual persona. In order to create queries from our corpora, we randomly select words from our document collection. We generate examples containing 1, 2 and 3 words, occurring either in a sequence or in different parts of a single document. For each query, we also retrieve all documents in the collection that match that query. We provide scripts to automatically generate such gold standards, resulting in four evaluation datasets, one for each persona, containing each 5000 test queries.

Note that precision / recall is a very strict evaluation. For instance, if the system returns a document that is *partially* relevant to the query (for instance because it matches two out of three words), it will be counted as an error. So a precision of 80% with recall of 100% would mean that out of, say, 10 results, 8 documents contain exact matches for the query while 2 are either partial matches, or (in the worst case) fully irrelevant.

3. Optimization

Our evaluation strategy involves the identification of the training strategy that will result in optimal search performance. Three factors will influence the outcome of the system and must be investigated: a) the data preprocessing pipeline; b) the tokenization hyperparameters; c) the genericity of the trained model. We explain these three factors in more detail below.

[1] https://huggingface.co/datasets/cuad,
https://huggingface.co/datasets/Sachinkelenjaguri/Resume_dataset,
https://data.mendeley.com/datasets/tnj49gpmtz/2,
https://huggingface.co/datasets/singhsays/fake-w2-us-tax-form-dataset.

[2] https://github.com/possible-worlds-xyz/wikiloader

3.1 Preprocessing

Previous experience with the PeARS system has shown that some light preprocessing is beneficial to the indexer: standardly, PeARS applies word tokenization and lowercasing to the raw text data, so that punctuation is clearly separated from words and capitalization is ignored (note that users often search without respecting capitalization).

For the OMD use case, though, we are also considering more complex document types such as invoices and CVs, which may contain a much higher proportion of low-frequency 'words', such as numbers, dates and identifiers. This makes the work of preprocessing more complex and potentially adds requirements for more involved preprocessing steps. We therefore consider the addition of the following functions in the pipeline: punctuation removal, stopword removal and lemmatization (using the open-source *nltk*[3] toolkit).

3.2 Tokenization

The OMD system is required to run on entry-level hardware, making it vital for the integrated search system to be as lightweight as possible. Traditional search systems express documents using very large sparse vectors, which can reach sizes of several hundreds of thousands of dimensions. Such large representations would be problematic for our use case, which requires performing fast linear algebra over substantial document sets. We therefore need to keep the dimensionality of our representations as low as possible.

In order to solve this issue, we use a wordpiece tokenizer. This type of tokenizer is standardly integrated in deep learning system. It is a trained machine learning model which breaks words into statistically relevant subunits. The model can be trained for a predefined vocabulary size, which for us corresponds to the dimensionality of our document representations.

We use the open-source SentencePiece library[4] for wordpiece tokenization, coupled with the WikiLoader package. This combination allows us to automatically train a tokenizer at the desired dimensionality from a Wikipedia sample in any language.

Wordpiece tokenization involves two important hyperparameters which we seek to optimize. The first one is the type of algorithm used to identify subwords. We investigate two well-known options: *unigram* and *BPE* and find in preliminary experiments that *BPE* is better suited to our use case. The results reported in §4 are therefore based on *BPE*. The second hyperparameter is the vocabulary size. This corresponds to a trade-off between accuracy and performance: the large the vocabulary, the more precise the system is, but also the slower. We seek a good compromise between these two requirements.

3.3 Generic vs user-specific models

As stated before, the OMD use case is challenging because privacy concerns prevent us from studying the real distribution of data among users. At this stage of investigation, we decide to compare system performance across our four personae when using a generically-trained tokenizer (trained over a Wikipedia sample) and a user-specific tokenizer (trained over the stored data of each individual persona). This corresponds to a further trade-off between performance and system complexity: a user-specific model will result in higher accuracy, at the cost of an additional system component (scheduled retraining of the tokenizer on the user's system).

[3] https://www.nltk.org/
[4] https://github.com/google/sentencepiece

Precision	all preproc.	- lowercase	- punct. removal	- nltk tokeniz.	- nltk lemmatiz.	- stopwords removal
HR	84	83	81	84	81	83
Accountant	68	65	64	68	65	67
Recall	all preproc.	- lowercase	- punct. removal	- nltk tokeniz.	- nltk lemmatiz.	- stopwords removal
HR	100	99	96	100	95	100
Accountant	100	96	91	100	96	100
F1	all preproc.	- lowercase	- punct. removal	- nltk tokeniz.	- nltk lemmatiz.	- stopwords removal
HR	88	87	84	88	84	87
Accountant	73	70	68	73	70	73

Table 1
Precision, recall and F1 scores for preprocessing study. An ablation is conducted on the different features used in preprocessing.

4. Experimental results

4.1 Preprocessing

Table 4.1 shows the results of an ablation study for the preprocessing step. We consider here the two personae with ‘non-standard’ text distributions (accountants and HR managers) and evaluate search using a generic Wikipedia-based wordpiece tokenizer, trained with 10,000 dimensions. The table shows precision, recall and F1 scores when all preprocessing is applied (including lowercasing, basic tokenization, punctuation and stopword removal, lemmatization). It also demonstrates how the ablation of a single preprocessing step affects performance, allowing us to assess the importance of each step in the pipeline.

The first takeaway is that recall remains near perfect (above 96%) regardless of the preprocessing steps. The only exception is the recall for the accountant dataset when punctuation removal is ablated (resulting in a recall of 91%). For precision, the picture is more variagated. Punctuation removal and lemmatization seem to be the most important steps, followed by lowercasing.

The issue of lemmatization overlaps with vocabulary size (which we inspect in the next sections). To take a simple example, a tokenizer that generates the wordpieces *cat* and *cats* is likely to perform better than one that only generates *cat* and the plural marker *s*. But it also increases vocabulary size, which in turn affects efficiency.

4.2 Tokenization

Our next area of investigation is the vocabulary size of the tokenizer. We experiment with sizes ranging from 8k to 24k, as well as the maximum possible vocabulary (i.e. the case where each word is allowed to fill one token, as in the traditional search paradigm). The maximum setting gives us an upper bound for the system’s precision, but is in most cases impractical: for our personae, it goes up to $361,422$ tokens, which is a much too high dimensionality for our use case.

As expected, results show an increase in precision when the vocabulary size increases (Table 4.2). They also demonstrate that our four datasets behave very differently. While the HR

tokenizer / eval dataset	8K	12K	16K	20K	24K	max
enwiki-bpe ->hr	79	86	90	93	94	99
enwiki-bpe ->accountant	64	68	70	73	75	91
enwiki-bpe ->news	70	78	82	85	87	w
enwiki-bpe ->moviesummary	53	64	71	76	80	97

Table 2
Effect of vocabulary size on precision, using a generically trained BPE wordpiece tokenizer.

tokenizer / eval dataset	8K	12K	16K	20K	24K	max
hr-bpe ->hr	94	96	97	-	-	98
accountant-bpe ->accountant	90	92	93	-	-	93
news-bpe ->news	76	82	86	88	90	97
moviesummary-bpe ->moviesummary	64	74	79	83	85	96

Table 3
Effect of training a BPE wordpiece tokenizer on user-specific data, for different vocabulary sizes. Performance reported as precision.

dataset is close to 80% precision for 8k already, the news dataset only reaches that threshold with 16k and the movie dataset with 24k. The accountant dataset is still below at 24k.

5. User-specific tokenization

In the light of Table 4.2's results, we also train the tokenizer in a user-specific way. System performance is shown in Table 5. We can see that the datasets with the most unusual word distributions (HR and accountant) benefit hugely and already reach over 90% precision with a vocabulary size of 8k. The news and movie datasets also see a substantial increase in performance, especially at lower dimensionalities. They however require larger vocabularies to reach 90%. This is not unexpected, since they treat a much wider variety of topics than the more specific HR and accountant datasets. It remains to be seen how much variety there is in actual user data.

6. Interim conclusions

Our experiments have highlighted an expected trade-off between system precision and computing efficiency. Higher vocabulary sizes result in better precision, and so do models trained on user-specific data. Across our experiments, though, recall remains close to 100%, showing that the system adequately finds all documents relevant for a query. For now, we are releasing a standard Wikipedia-based model in 20k dimensions, which gives us a good compromise between system efficiency and precision. We will however further discuss the issue of potentially adding on-device, user-specific training to the OMD system, as it provides unrivalled precision.