Information Processing and Retrieval

Instituto Superior Técnico 2024

Lab 2: IR Models and Evaluation

This lab extends our IR system for evaluation ends and continues the study of IR models.

1 Evaluating IR systems

1.1. Index the pri_cfc.txt[1] collection (one document per line). You can use whoosh facilities.

1.2. Implement *precision*, *recall* and *F1* measures. These functions take two simple inputs: i) the set of retrieved document IDs; and ii) the set of relevant document IDs.

1.3. The file pri_queries.txt[1] contains a set of queries and, for each query, the set of relevant documents. For each of the first 10 queries, execute the corresponding search using two different scoring criteria: i) TF-IDF, and ii) BM25.

1.4. Using the reference and produced results in 1.3, compare the performance of TF-IDF and BM25 scoring criteria for the given collection with regards to precision, recall and F1. Print the average score and standard deviation for each measure.

1.5. Extend the evaluation criteria to further return the values that define the precision-recall curve, as well as the subsequent Mean Average Precision (MAP) statistic. You can either assess precision for each relevant document or directly use sklearn.metrics.precision_recall_curve.

2 LLM Embeddings using BERT

2.1. Before learning large language model embeddings, let us first look to the TF-IDF vectors for the pri_cfc.txt collection. Using sklearn.feature_extraction.text.TfidfVectorizer(), compute the TF-IDF vectors for the documents in the collection.

2.2. Let us visualize the proximity of documents based on their TF-IDF vectors by reducing the dimensionality of the vector space to 2 dimensions. To this end, we can use PCA, tSNE or other dimensionality reduction technique. Consider applying uMAP[2] using the following code.

```
import umap, umap.plot
mapper = umap.UMAP(metric='cosine', random_state=0).fit(data.toarray())
umap.plot.points(mapper, ax=axs[i])
```

[1] https://fenix.tecnico.ulisboa.pt/disciplinas/RGI/2022-2023/1-semestre/labs
[2] https://arxiv.org/abs/1802.03426

2.3. BERT is a widely applied Large Language Model for extracting features from text, including word and document embeddings, for multiple information retrieval tasks (e.g. document retrieval, entity recognition, question answering). BERT produces word representations that are informed by the surrounding words, as well as representations for the enclosed sentences.

Let us consider the base BERT model, composed of twelve transfomers and a final pooling layer. Using the transformers[3] package, download this pretrained LLM, visualize its architecture, and count the number of parameters.

```
from transformers import BertModel, BertTokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', truncation=True, do_lower_case=True)
bert_model = BertModel.from_pretrained('bert-base-uncased', output_hidden_states=True)
print('params', sum(p.numel() for p in bert_model.parameters()))
print(bert_model.eval())
```

2.4. Each of the 12 transformers produce an embedding (either capturing lower or higher-level features according to their position) that are arguably useful for downstream information retrieval tasks. As such, different strategies have been considered to produce a unified embedding:

- the embedding from the last transformer or from the pooling layer
- an aggregation (e.g. averaging, concatenation) of the embeddings from either all transformers or the last 4 transformers

For each strategy, identify the embeddings for the documents of the given collection.

```
def get_bert_output(tokenizer, model, sentence, mode='cls', optype='sumsum'):
    tokenized_text = tokenizer.tokenize(sentence)
    tokens_tensor = torch.tensor([tokenizer.convert_tokens_to_ids(tokenized_text)])
    segments_tensors = torch.tensor([[1] * len(tokenized_text)])
    outputs = model(tokens_tensor, segments_tensors)
    if mode == 'cls': embedding = outputs["last_hidden_state"].squeeze()[0]
    elif mode == 'pooled': embedding = outputs["pooler_output"].squeeze()
    else: #'hidden'
        layers = torch.stack(outputs['hidden_states'][-4:])
        if optype == "sumsum": embedding = torch.sum(layers.sum(0).squeeze(), dim=0)
        elif optype == "summean": embedding = torch.sum(layers.mean(0).squeeze(), dim=0)
        elif optype == "meanmean": embedding = torch.mean(layers.mean(0).squeeze(), dim=0)
        else: embedding = torch.mean(layers.sum(0).squeeze(), dim=0)
    return embedding.detach().numpy()
```

Complementary notes:

- BERT is originally prepared to produce embeddings for sentences. Hence, embeddings produced for documents with an arbitrary number of sentences are suboptimal;
- maximum input document length is 512 tokens. BERT will ignore additional tokens;
- markers are generally placed to distinguish between embeddings from a single text piece ([CLS] <text> [SEP]) or paired sentences ([CLS] <text> [SEP] <text> [SEP]);
- alternative implementations are available in tensorflow[4] or keras[5];

[3] https://github.com/huggingface/transformers
[4] https://github.com/google-research/bert
[5] https://github.com/Separius/BERT-keras

- you can fine tune BERT for your collection (up to 3 epochs), yet this is generally computationally heavy (few hours) as no layers are frozen during fine tuning.

2.5. Visualize the proximity of the BERT embeddings for the given documents using uMAP.

3 Pen and paper exercises

Consider the following collection of 4 text documents.

ID	text
1	shipment of gold damaged in fire
2	delivery of silver arrived in silver truck
3	shipment of silver arrived in truck
4	truck damaged in fire

3.1. Calculate the vector representations for the documents in the collection using TF-IDF. The TF-IDF score of a term t in a document d from a collection D can be given by

$$\text{TF-IDF}(t,d) = \text{IDF}(t) \times \text{TF}_{t,d} \quad \text{where} \quad \text{IDF}(t) = \log\left(\frac{|D|}{|\{d' \in D : t \in d'\}|}\right)$$

Note: for simplification, the TF-IDF formula above does not normalize the TF component.

3.2. Using the cosine similarity, which document is the most relevant to the query silver truck?

3.3. Consider the Okapa BM25 model with $b = 0.75$ and $k_1 = 1.2$,

$$\text{BM25}(q,d) = \sum_{t \in q} \text{IDF}(t) \times \frac{TF_{t,d} \cdot (k_1+1)}{TF_{t,d} + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{avgdl}\right)},$$

Using BM25, find the most relevant document to the query: silver truck.

3.4. Compare your findings against the two following models:

(a) a variant of the binary independence model given by:

$$\text{BIM}(q,d) = \sum_{t \in q} \text{IDF}(t) \times \tau(t,d),$$

where $\tau(t,d)$ is a binary indicator for the presence of term t in document d.

(b) simple unigram language model without parameter smoothing

$$\text{LM}(q,d) = \text{P}(\{t_1, \ldots, t_{|q|}\} \mid d) = \prod_{1 \le i \le |q|} P(t_i \mid d) = \prod_{1 \le i \le |q|} \frac{TF_{t_i,d}}{|d|}$$

3.5. Consider a larger collection of 20 documents. A query q yielded as answer the ordered set $R = \{d_1, d_2 \cdots d_{20}\}$. In R, documents d_i where i is odd are judged as *relevant*, whereas documents d_i where i is even are judged as *not relevant*. Compute:

(a) precision@5

(b) R-precision

(c) the interpolated precision and recall curve

(d) 11-point precision

(e) MAP