现代信息检索
Modern Information Retrieval

第16讲 Using Tools for IR Experimentation
检索工具使用方法介绍

提纲

- 检索工具简介
- Sparse Retrieval
- Early Neural IR Models
- BERT Cross-encoder Re-ranker
- Late Interaction & Dense Retrieval

检索工具

- Anserini
  - 一个可重复信息检索工具包，通过Lucene构建
  - 主要支持稀疏检索模型，包括BM25等
  - https://github.com/castorini/anserini
- Pyserini
  - 一个Python工具包，同时支持稀疏和稠密检索
  - 与Anserini工具包集成提供稀疏检索，与Facebook的Faiss库集成提供稠密检索
  - https://github.com/castorini/pyserini
- PyGaggle
  - 提供了一组用于文本排序和问答的深度神经架构，与Pyserini紧密整合
  - 可直接用Hugging Face上的monoBERT和monoT5模型
  - https://github.com/castorini/pygaggle

检索工具

- Matchmaker
  - 支持对基于文本的神经重排序和检索模型进行快速训练，评估和分析
  - 支持多种neural IR models：
    - K-NRM, Conv-KNRM, MatchPyramid, PACRR, Co-PACRR, DUET, DRMM
    - BERT re-ranker, BERT dense retriever,
    - PARADE , TK, PreTTR, ColBERT
  - https://github.com/sebastian-hofstaetter/matchmaker
  - 教程： https://github.com/sebastian-hofstaetter/teaching

- MatchZoo
  - 通用的文本匹配工具包，同样支持DRMM, MatchPyramid, MV-LSTM, DUET, ARC-I, ARC-II, DSSM和CDSSM等深度匹配模型
  - https://github.com/NTMC-Community/MatchZoo

检索工具

- OpenNIR
  - An end-to-end neural ad-hoc ranking pipeline.
  - 支持DRMM, Duet, MatchPyramid, KNRM, PACRR, ConvKNRM, Vanilla BERT, CEDR models等神经排序模型
  - 支持TREC Robust 2004, MS-MARCO, ANTIQUE, TREC CAR等数据集
  - https://github.com/Georgetown-IR-Lab/OpenNIR

提纲

- 检索工具简介
- Sparse Retrieval
- Early Neural IR Models
- BERT Cross-encoder Re-ranker
- Late Interaction & Dense Retrieval

Sparse Retrieval

- 以BM25为例
  - Anserini
    - 安装： https://github.com/castorini/anserini#-getting-started

      git clone --recurse-submodules https://github.com/castorini/anserini.git

```
mvn clean package appassembler:assemble
```

    - 数据准备 (MS MARCO Passage, dev-subset)：
      - 下载

```
mkdir collections/msmarco-passage

wget https://msmarco.blob.core.windows.net/msmarcoranking/collectionandqueries.tar.gz -P collections/msmarco-passage

# Alternative mirror:
# wget https://rgw.cs.uwaterloo.ca/JIMMYLIN-bucket0/data/collectionandqueries.tar.gz -P collections/msmarco-passage

tar xvfz collections/msmarco-passage/collectionandqueries.tar.gz -C collections/msmarco-passage
```

      - 查询和标签： tools/topics-and-qrels/topics.msmarco-passage.dev-subset.txt
        tools/topics-and-qrels/qrels.msmarco-passage.dev-subset.txt

Sparse Retrieval

- 以BM25为例
  - Anserini
    - 建索引：

```
target/appassembler/bin/IndexCollection \
  -collection JsonCollection \
  -input /path/to/msmarco-passage \
  -index indexes/lucene-index.msmarco-passage/ \
  -generator DefaultLuceneDocumentGenerator \
  -threads 9 -storePositions -storeDocvectors -storeRaw \
  >& logs/log.msmarco-passage &
```

    - 检索：

```
target/appassembler/bin/SearchCollection \
  -index indexes/lucene-index.msmarco-passage/ \
  -topics tools/topics-and-qrels/topics.msmarco-passage.dev-subset.txt \
  -topicreader TsvInt \
  -output runs/run.msmarco-passage.bm25-default.topics.msmarco-passage.dev-subset.txt \
  -bm25 &   -rm3 -hits 1000
```

      - -bm25: ranking model BM25 (default: false)
      - -bm25.k1: BM25: k1 parameter (default: 0.9)
      - -bm25.b: BM25's b parameter (default: 0.4)
      - -rm3: use RM3 query expansion model (default: false)
      - -hits: max number of hits to return (default: 1000)

Sparse Retrieval

- 以BM25为例
  - Anserini

- The setting "default" refers the default BM25 settings of `k1=0.9`, `b=0.4`.
- The setting "tuned" refers to `k1=0.82`, `b=0.68`.

    - 评价：trec_eval
      - tools/eval/trec_eval.9.0.4/trec_eval -c -M 10 -m recip_rank tools/topics-and-qrels/qrels.msmarco-passage.dev-subset.txt runs/run.msmarco-passage.bm25-default.topics.msmarco-passage.dev-subset.txt

MRR@10

AP@1000	BM25 (default)	BM25 (tuned)
MS MARCO Passage: Dev	0.1926	0.1958
RR@10	**BM25 (default)**	**BM25 (tuned)**
MS MARCO Passage: Dev	0.1840	0.1875
R@100	**BM25 (default)**	**BM25 (tuned)**
MS MARCO Passage: Dev	0.6578	0.6701
R@1000	**BM25 (default)**	**BM25 (tuned)**
MS MARCO Passage: Dev	0.8526	0.8573

    - 参考：https://github.com/castorini/anserini/blob/master/docs/regressions-msmarco-passage.md

Sparse Retrieval

- 以BM25为例
  - Pyserini
    - 建索引：

```
python -m pyserini.index.lucene \
  --collection JsonCollection \
  --input tests/resources/sample_collection_jsonl \
  --index indexes/sample_collection_jsonl \
  --generator DefaultLuceneDocumentGenerator \
  --threads 1 \
  --storePositions --storeDocvectors --storeRaw
```

    - 单query检索：
      - 可直接用Anserini建的索引

```python
from pyserini.search.lucene import LuceneSearcher

searcher = LuceneSearcher('indexes/sample_collection_jsonl')
hits = searcher.search('document')

for i in range(len(hits)):
    print(f'{i+1:2} {hits[i].docid:4} {hits[i].score:.5f}')
```

      - 可用已有索引

```python
searcher = LuceneSearcher.from_prebuilt_index('msmarco-v1-passage')
```

Sparse Retrieval

- 以BM25为例
  - Pyserini
    - 输出结果：

```
1 doc2 0.25620
2 doc3 0.23140
```

    - 批次检索：

```
python -m pyserini.search.lucene \
  --index indexes/sample_collection_jsonl \
  --topics tests/resources/sample_queries.tsv \
  --output run.sample.txt \
  --bm25
```

```
$ cat run.sample.txt
1 Q0 doc2 1 0.256200 Anserini
1 Q0 doc3 2 0.231400 Anserini
2 Q0 doc1 1 0.534600 Anserini
3 Q0 doc1 1 0.256200 Anserini
3 Q0 doc2 2 0.256199 Anserini
4 Q0 doc3 1 0.483000 Anserini
```

  - 参考： https://github.com/castorini/pyserini/blob/master/docs/usage-index.md#building-a-bm25-index-direct-java-implementation

提纲

- 检索工具简介
- Sparse Retrieval
- Early Neural IR Models
- BERT Cross-encoder Re-ranker
- Late Interaction & Dense Retrieval

Early Neural IR Models

- MatchZoo (以DSSM为例)
  - 从Pypi安装：

```
pip install matchzoo
```

  - 从Github源安装：

```
git clone https://github.com/NTMC-Community/MatchZoo.git
cd MatchZoo
python setup.py install
```

  - 中文文档：https://matchzoo.readthedocs.io/zh/latest/
  - 参考： https://github.com/NTMC-Community/MatchZoo#get-started-in-60-seconds

Early Neural IR Models

- MatchZoo (以DSSM为例)
  - 导入matchzoo并准备输入数据：

```
import matchzoo as mz

train_pack = mz.datasets.wiki_qa.load_data('train', task='ranking')
valid_pack = mz.datasets.wiki_qa.load_data('dev', task='ranking')
```

  - 预处理输入数据：

```
preprocessor = mz.preprocessors.DSSMPreprocessor()
train_processed = preprocessor.fit_transform(train_pack)
valid_processed = preprocessor.transform(valid_pack)
```

  - 设置损失函数和评估指标：

```
ranking_task = mz.tasks.Ranking(loss=mz.losses.RankCrossEntropyLoss(num_neg=4))
ranking_task.metrics = [
    mz.metrics.NormalizedDiscountedCumulativeGain(k=3),
    mz.metrics.MeanAveragePrecision()
]
```

Early Neural IR Models

- MatchZoo (以DSSM为例)
  - 初始化模型，微调超参数：

```
model = mz.models.DSSM()
model.params['input_shapes'] = preprocessor.context['input_shapes']
model.params['task'] = ranking_task
model.guess_and_fill_missing_params()
model.build()
model.compile()
```

  - 实时生成成对训练数据，使用验证数据评估模型性能：

```
train_generator = mz.PairDataGenerator(train_processed, num_dup=1, num_neg=4, batch_size=64, shuffle=True)
valid_x, valid_y = valid_processed.unpack()
evaluate = mz.callbacks.EvaluateAllMetrics(model, x=valid_x, y=valid_y, batch_size=len(valid_x))
history = model.fit_generator(train_generator, epochs=20, callbacks=[evaluate], workers=5, use_multiprocessing=False)
```

Early Neural IR Models

- OpenNIR (以ConvKNRM为例)
  - git clone https://github.com/Georgetown-IR-Lab/OpenNIR.git
  - 安装依赖：
    ```
    pip install -r requirements.txt
    ```
  - 训练并验证模型(在ANTIQUE上的ConvKNRM)
    ```
    scripts/pipeline.sh config/conv_knrm config/antique
    ```
  - 参考：https://github.com/Georgetown-IR-Lab/OpenNIR#quick-start

Early Neural IR Models

- OpenNIR (以ConvKNRM为例)
  - pipeline.sh

```
python -m onir.bin.pipeline "$@"
```

  - onir/bin/pipeline.py

```
import onir


def main():
    context = onir.injector.load({
        'vocab': onir.vocab,
        'train_ds': onir.datasets,
        'ranker': onir.rankers,
        'trainer': onir.trainers,
        'valid_ds': onir.datasets,
        'valid_pred': onir.predictors,
        'test_ds': onir.datasets,
        'test_pred': onir.predictors,
        'pipeline': onir.pipelines,
    }, pretty=True)

    context['pipeline'].run()


if __name__ == '__main__':
    main()
```

config/conv_knrm

```
ranker=conv_knrm

ranker.pretrained_kernels=True
vocab=wordvec_hash
vocab.source=convknrm
vocab.variant=convknrm-bing
```

提纲

- 检索工具简介
- Sparse Retrieval
- Early Neural IR Models
- BERT Cross-encoder Re-ranker
- Late Interaction & Dense Retrieval

BERT Cross-encoder Re-ranker

- PyGaggle
  - 可用现成的模型做重排序，支持BERT和T5 re-ranker
    - MonoBERT, MonoT5, DuoT5等
  - 安装：

    0. Clone the repo with `git clone --recursive https://github.com/castorini/pygaggle.git`
    1. Make you sure you have an installation of Python 3.8+. All `python` commands below refer to this.
    2. For pip, do `pip install -r requirements.txt`
        - If you prefer Anaconda, use `conda env create -f environment.yml && conda activate pygaggle`.

  - 参考：https://github.com/castorini/pygaggle#a-simple-reranking-example

BERT Cross-encoder Re-ranker

- PyGaggle
  - 初始化BERT re-ranker：

```
from pygaggle.rerank.base import Query, Text
from pygaggle.rerank.transformer import MonoBERT

reranker =  MonoBERT()
```

  - 初始化T5 re-ranker：

```
from pygaggle.rerank.base import Query, Text
from pygaggle.rerank.transformer import MonoT5

reranker =  MonoT5()
```

  - 可用模型：https://huggingface.co/castorini

```
from transformers import T5ForConditionalGeneration
model = T5ForConditionalGeneration.from_pretrained('castorini/monot5-base-msmarco-10k')
reranker = MonoT5(model=model)
```

BERT Cross-encoder Re-ranker

- PyGaggle

```
# Here's our query:
query = Query('who proposed the geocentric theory')

# Option 1: fetch some passages to rerank from MS MARCO with Pyserini
from pyserini.search import LuceneSearcher
searcher = LuceneSearcher.from_prebuilt_index('msmarco-passage')
hits = searcher.search(query.text)

from pygaggle.rerank.base import hits_to_texts
texts = hits_to_texts(hits)

# Option 2: here's what Pyserini would have retrieved, hard-coded
passages = [['7744105', 'For Earth-centered it was  Geocentric Theory proposed by greeks under the guidance of Ptolemy and Sun-

texts = [ Text(p[1], {'docid': p[0]}, 0) for p in passages] # Note, pyserini scores don't matter since T5 will ignore them.

# Either option, let's print out the passages prior to reranking:
for i in range(0, 10):
    print(f'{i+1:2} {texts[i].metadata["docid"]:15} {texts[i].score:.5f} {texts[i].text}')

# Finally, rerank:
reranked = reranker.rerank(query, texts)

# Print out reranked results:
for i in range(0, 10):
    print(f'{i+1:2} {reranked[i].metadata["docid"]:15} {reranked[i].score:.5f} {reranked[i].text}')
```

BERT Cross-encoder Re-ranker

- Matchmaker
  - 支持BERT re-ranker的训练
  - git clone https://github.com/sebastian-hofstaetter/matchmaker.git
  - 安装环境

We recommend using a fresh conda environment with Python 3.8 *(can't use 3.9 atm, because of faiss)*

```
conda create -n matchmaker python=3.8
conda activate matchmaker
```

Then cd to the root folder of this repo, activate the conda environment, and install faiss & pytorch via conda. *We have to install faiss separately, because it does not have official pypi packages*

```
conda install --file conda-requirements.txt -c conda-forge -c pytorch
```

Then install the rest of the dependencies (allennlp, huggingface, ...) via pip install of the pip-requirements.txt

```
pip install -r pip-requirements.txt
```

BERT Cross-encoder Re-ranker

- Matchmaker
  - 训练数据：
    ```
    query-text<tab>pos-text<tab>neg-text
    ```
  - 测试数据：
    ```
    query-id<tab>doc-id<tab>query-text<tab>doc-text
    ```
  - 参考：https://github.com/sebastian-hofstaetter/matchmaker/blob/master/documentation/data_format.md
  - dataset.yaml

```yaml
#
# training path
#
# format: query-text<tab>pos-text<tab>neg-text
# format (if train_pairwise_distillation:True): score-pos<tab>score-neg<tab>query-text<tab>pos-text<tab>neg-text
train_tsv: "/path/to/train/triples.train.tsv"

#
# continuous validation path
#
validation_cont:
  # format: query-id<tab>doc-id<tab>query-text<tab>doc-text
  tsv: "/path/to/validation/bm25_plain_top100.tsv"
  qrels: "/path/to/qrels/qrels.dev.tsv"
  binarization_point: 1 # qrely label >= for MRR,MAP,Recall -> 1 others 0
  save_only_best: True
```

BERT Cross-encoder Re-ranker

- Matchmaker
  - 模型训练：

    python matchmaker/train.py

    --config-file config/train/defaults.yaml config/data/<your dataset here>.yaml config/train/models/bert_cat.yaml

    --run-name bert_cat_default

  - config文件部分设置：

```
#
# Models
# ------------------------------------------------------
model: "bert_cat"

bert_pretrained_model: "distilbert-base-uncased"
bert_trainable: True
```

```
#
# optimization
#

loss: "ranknet"
validation_metric: "nDCG@10"

optimizer: "adam"

# default group (all params are in here if not otherwise specified
param_group0_learning_rate: 0.000007
param_group0_weight_decay: 0

param_group1_names: ["top_k_scoring"] # "position_importance_layer"
param_group1_learning_rate: 0.0007
param_group1_weight_decay: 0

embedding_optimizer: "adam"
embedding_optimizer_learning_rate: 0.000007
embedding_optimizer_momentum: 0.8 # only when using sgd
```

提纲

- 检索工具简介
- Sparse Retrieval
- Early Neural IR Models
- BERT Cross-encoder Re-ranker
- Late Interaction & Dense Retrieval

Late Interaction (ColBERT)

- Matchmaker
  - 与训练BERT re-ranker类似，更改model config文件即可
  - 模型训练：

```
python matchmaker/train.py --config-file config/train/defaults.yaml
config/data/<your dataset here>.yaml config/train/models/colbert.yaml
--run-name colbert_default
```

  - model config文件：

```
model: ColBERT

colbert_compression_dim: 768
query_augment_mask_number: -1
```

  - 官方实现：https://github.com/stanford-futuredata/ColBERT

Dense Retrieval

- Pyserini
  - 构建dense索引（通过已有模型）
  - 语料格式:

```
{
  "id": "CACM-2636",
  "contents": "Generation of Random Correlated Normal ... \n"
}
```

  - 编码:

```
python -m pyserini.encode \
  input   --corpus tests/resources/simple_cacm_corpus.json \
          --fields text \  # fields in collection contents
          --delimiter "\n" \
          --shard-id 0 \   # The id of current shard. Default is 0
          --shard-num 1 \  # The total number of shards. Default is 1
  output  --embeddings path/to/output/dir \
          --to-faiss \
  encoder --encoder castorini/tct_colbert-v2-hnp-msmarco \
          --fields text \  # fields to encode, they must appear in the input.fields
          --batch 32 \
          --fp16  # if inference with autocast()
```

Dense Retrieval

- Pyserini
  - 可直接构建Faiss Flat索引： `output --to-faiss`

  - 可先编码为向量（保存为json文件），再通过pyserini.index.faiss构建各种类型的Faiss索引：
    - 向量文件：

```
{
  "id": "CACM-2636",
  "contents": "Generation of Random Correlated Normal ... \n"},
  "vector": [0.126, ..., -0.004]
}
```

    - Flat索引：

```
python -m pyserini.index.faiss \
  --input path/to/encoded/corpus \  # in jsonl format
  --output path/to/output/index \
```

    - HNSW索引：

```
python -m pyserini.index.faiss \
  --input path/to/encoded/corpus \  # either in the Faiss or the jsonl format
  --output path/to/output/index \
  --hnsw
```

Dense Retrieval

- Pyserini
  - 检索：

```
from pyserini.search import FaissSearcher

searcher = FaissSearcher(
    'indexes/dindex-sample-dpr-multi',
    'facebook/dpr-question_encoder-multiset-base'
)
hits = searcher.search('what is a lobster roll')

for i in range(0, 10):
    print(f'{i+1:2} {hits[i].docid:7} {hits[i].score:.5f}')
```

  - 可用pre-build索引：

```
from pyserini.search.faiss import FaissSearcher, TctColBertQueryEncoder

encoder = TctColBertQueryEncoder('castorini/tct_colbert-msmarco')
searcher = FaissSearcher.from_prebuilt_index(
    'msmarco-passage-tct_colbert-hnsw',
    encoder
)
hits = searcher.search('what is a lobster roll')

for i in range(0, 10):
    print(f'{i+1:2} {hits[i].docid:7} {hits[i].score:.5f}')
```

  - 参考： https://github.com/castorini/pyserini/blob/master/docs/usage-index.md#building-a-dense-vector-index

Dense Retrieval

- Matchmaker
  - 可训练dense retrieval模型
  - 与训练BERT re-ranker类似，更改数据格式和config文件即可
  - 待编码数据格式：

The collection file:

```
doc-id<tab>doc-text
```

The query file:

```
query-id<tab>query-text
```

  - 参考：

https://github.com/Sebastian-hofstaetter/matchmaker/blob/master/documentation/dense_retrieval_train.md

Dense Retrieval

- Matchmaker
  - 训练模型：

CUDA_VISIBLE_DEVICES=0,1,2,3 python matchmaker/train.py

--config-file config/train/defaults.yaml config/train/data/<your dataset here>.yaml config/train/models/bert_dot.yaml

--run-name your_experiment_name

  - model config文件：

```
model: bert_dot # for shared bert model weights (q & d = the same)

#bert_dot_compress_dim: 128 # or -1 for no compression

run_dense_retrieval_eval: True
```

  - 编码、建索引、检索： https://github.com/sebastian-hofstaetter/matchmaker/blob/master/documentation/dense_retrieval_evaluate.md