Second Term Project: TREC Collection

“The U.S. National Institute of Standards and Technology (NIST) has run a large IR test bed evaluation series since 1992. [...] TRECs 6–8 provide 150 information needs over about 528,000 newswire and Foreign Broadcast Information Service articles. [...] Because the test document collections are so large, there are no exhaustive relevance judgments.” [1, Section 8.2]

Your tasks, reviewed by your colleagues and the course instructors, are the following:

1. *Implement a supervised ranked retrieval system*, [1, Chapter 15] which will produce a list of documents from the TREC collection in a descending order of relevance to a query from the TREC collection. You SHOULD use training and validation relevance judgements from the TREC collection in your information retrieval system. Test judgements MUST only be used for the evaluation of your information retrieval system.
2. *Document your code* in accordance with PEP 257, ideally using the NumPy style guide as seen in the code from exercises.
   Stick to a consistent coding style in accordance with PEP 8.
3. *Reach at least 10% mean average precision* [1, Section 8.4] with your system on the Trec collection. You are encouraged to use techniques for tokenization, [1, Section 2.2] document representation [1, Section 6.4], tolerant retrieval [1, Chapter 3], relevance feedback, query expansion, [1, Chapter 9], learning to rank [1, Chapter 15], and others discussed in the course.
4. *Upload a link to your Google Colaboratory document to the homework vault in IS MU.* You MAY also include a brief description of your information retrieval system.

[1] Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge university press, 2008.

Loading the TREC collection

First, we will install our library and load the TREC collection. If you are interested, you can take a peek at how we preprocessed the raw TREC collection to the final dataset that we will be using.

The TREC collection is ca 1000× larger than the Cranfield collection that we used for the first term project! If the amount of RAM at Google Colab is insufficient for your information retrieval system, you can either:

1. download this notebook, install Jupyter on the `aura.fi.muni.cz` server, and use local port forwarding over SSH to access the running Jupyter notebook from your web browser, or

2. install Jupyter on the `aura.fi.muni.cz` server, use local port forwarding over SSH, and set up a local runtime for Google Colab.

Please, follow the rules of the `aura.fi.muni.cz` server and always run Jupyter with the lowest priority (`nice -n 19 ...`), so that you don't disrupt the interactive sessions of other students.

```
%%capture
! pip install git+https://gitlab.fi.muni.cz/xstefan3/pv211-utils.git
! pip install git+https://github.com/witiko/gensim.git@feature/bm25
```

Loading the documents

Next, we will define a class named `Document` that will represent a preprocessed document from the TREC collection. Tokenization and preprocessing of the `body` attribute of the individual documents is left to your imagination and craftsmanship.

```
from pv211_utils.trec.entities import TrecDocumentBase

class Document(TrecDocumentBase):
    """
    A preprocessed TREC collection document.

    Parameters
    ----------
    document_id : str
        A unique identifier of the document.
    body : str
        The abstract of the document.

    """
    def __init__(self, document_id: str, body: str):
        super().__init__(document_id, body)
```

We will load documents into the `documents` ordered dictionary. Each document is an instance of the `Document` class that we have just defined.

```
from pv211_utils.trec.loader import load_documents

documents = load_documents(Document, cache_download='/var/tmp/pv211/trec_documents.json.gz')
```

```
Computing MD5: /home/xnovot32/.cache/gdown/https-COLON--SLASH--SLASH-drive.google.com-SI
MD5 matches: /home/xnovot32/.cache/gdown/https-COLON--SLASH--SLASH-drive.google.com-SLAS
```

```
print('\n'.join(repr(document) for document in list(documents.values())[:3]))
print('...')
print('\n'.join(repr(document) for document in list(documents.values())[-3:]))
```

```
<Document FR940110-1-00001 “Federal Register ␣/␣Vol. 59, No. 6␣/␣Monday, Janua ...”>
<Document FR940110-1-00002 “DEPARTMENT OF AGRICULTURE Agricultural Stabilizati ...”>
<Document FR940110-1-00003 “The ECP is authorized by the Agricultural Credit A ...”>
...
<Document LA123190-0132 “Actor Martin Sheen knows how to get his name in a  ...”>
<Document LA123190-0133 “IN THE SUMMER of 1989, this column wrote that the  ...”>
<Document LA123190-0134 “Tammy Wynette says a new generation of performers  ...”>
```

```
document = documents['FT911-3']
document
```

```
<Document FT911-3 “CONTIGAS, the German gas group 81 per cent owned b ...”>
```

```
print(document.body)
```

```
CONTIGAS, the German gas group 81 per cent owned by the utility Bayernwerk,
said yesterday that it intends to invest DM900m (Dollars 522m) in the next
four years to build a new gas distribution system in the east German state
of Thuringia.
Reporting on its results for 1989-1990 the company said that the dividend
would remain unchanged at DM8.
Sales rose 9.4 per cent to DM3.37bn, but post-tax profit fell slightly from
DM31.3m to DM30.7m.
In the first half of the current year sales rose 23 per cent.
Mr Jurgen Weber, currently vice-chairman of Lufthansa, the German airline,
is today expected to be named as the successor to the chairman Mr Heinz
Ruhnau who retires at the end of 1992.
Mr Weber is currently the technical director on the Lufthansa board.
```

Loading the queries

Next, we will define a class named `Query` that will represent a preprocessed query from the TREC collection. Tokenization and preprocessing of the `body` attribute of the individual queries as well as the creative use of the `title` and `narrative` attributes is left to your imagination and craftsmanship.

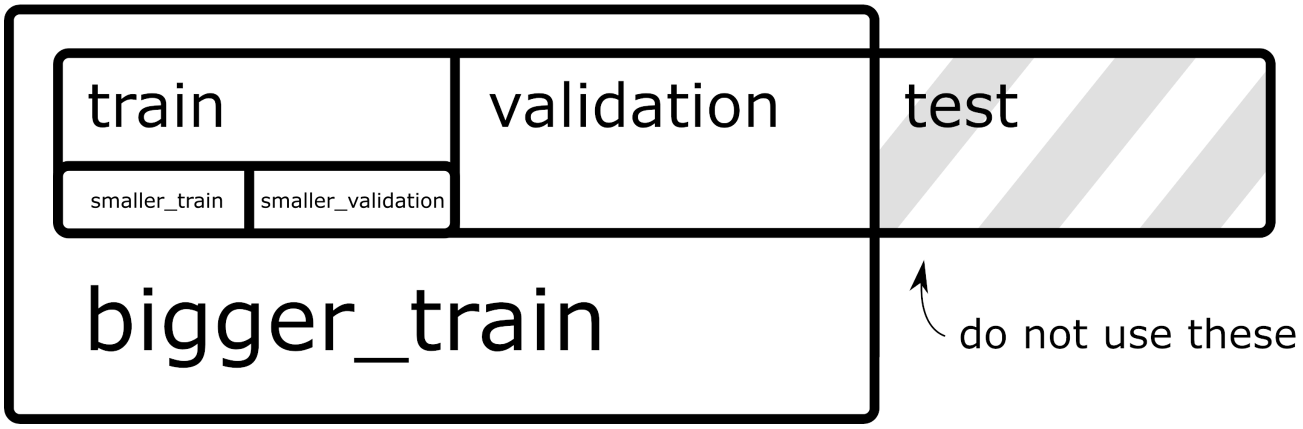
```
from pv211_utils.trec.entities import TrecQueryBase

class Query(TrecQueryBase):
    """
    A preprocessed TREC collection query.

    Parameters
    ----------
    query_id : int
        Up to three words that best describe the query.
    title : str
        Up to three words that best describe the query.
    body : str
        A one-sentence description of the topic area.
    narrative : str
        A concise description of what makes a document relevant.

    """
    def __init__(self, query_id: int, title: str, body: str, narrative: str):
        super().__init__(query_id, title, body, narrative)
```

We will load queries into the `train_queries` and `validation_queries` ordered dictionaries. Each query is an instance of the `Query` class that we have just defined. You should use `train_queries`, `validation_queries`, and *relevance judgements* (see the next section) for training your supervised information retrieval system.

If you are training just a single machine learning model without any early stopping or hyperparameter optimization, you can use `bigger_train_queries` as the input.

If you are training a single machine learning model with early stopping or hyperparameter optimization, you can use `train_queries` for training your model and `validation_queries` to stop early or to select the optimal hyperparameters for your model. You can then use `bigger_train_queries` to train the model with the best number of epochs or the best hyperparameters.

If you are training many machine learning models with early stopping or hyperparameter optimization, then you can split your train judgements to smaller training and validation sets. Then, you can use `smaller_train_queries` for training your models, `smaller_validation_queries` to stop early or to select the optimal hyperparameters for your models, and `validation_queries` to select the best model. You can then use `bigger_train_queries` to train the best model with the best number of epochs or the best hyperparameters.



```
from collections import OrderedDict
from itertools import chain

from pv211_utils.trec.loader import load_queries

train_queries = load_queries(Query, 'train')
validation_queries = load_queries(Query, 'validation')

bigger_train_queries = OrderedDict(chain(train_queries.items(), validation_queries.items()))

pivot = int(len(train_queries) * 0.8)
smaller_train_queries = OrderedDict(sorted(train_queries.items())[:pivot])
smaller_validation_queries = OrderedDict(sorted(train_queries.items())[pivot:])


print('\n'.join(repr(query) for query in list(train_queries.values())[:3]))
print('...')
print('\n'.join(repr(query) for query in list(train_queries.values())[-3:]))
```

```
<Query 301 “Identify organizations that participate in interna ...”>
<Query 302 “Is the disease of Poliomyelitis (polio) under cont ...”>
<Query 303 “Identify positive accomplishments of the Hubble te ...”>
...
<Query 378 “Identify documents that discuss opposition to the  ...”>
<Query 379 “Identify documents that discuss mainstreaming chil ...”>
<Query 380 “Identify documents that discuss medical treatment  ...”>
```

```
query = train_queries[301]
query
```

```
<Query 301 “Identify organizations that participate in interna ...”>
```

```
print(query.title)
```

```
    International Organized Crime
```

```
print(query.body)
```

```
    Identify organizations that participate in international criminal
    activity, the activity, and, if possible, collaborating organizations
    and the countries involved.
```

```
print(query.narrative)
```

```
    A relevant document must as a minimum identify the organization and the
    type of illegal activity (e.g., Columbian cartel exporting cocaine).
    Vague references to international drug trade without identification of
    the organization(s) involved would not be relevant.
```

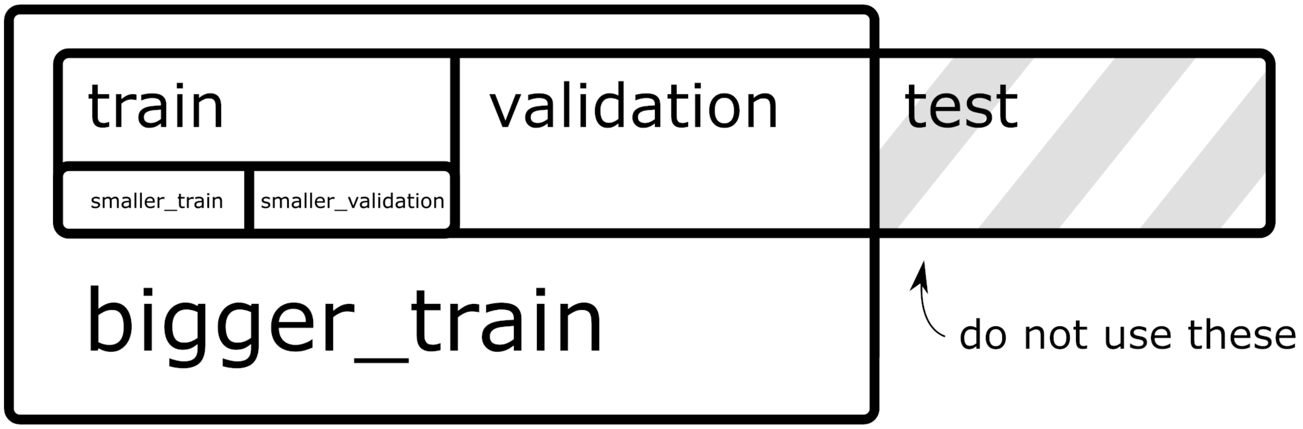
Loading the relevance judgements

Next, we will load train and validation relevance judgements into the `train_judgements` and `validation_judgement` sets. Relevance judgements specify, which documents are relevant to which queries. You should use relevance judgements for training your supervised information retrieval system.

If you are training just a single machine learning model without any early stopping or hyperparameter optimization, you can use `bigger_train_judgements` as the input.

If you are training a single machine learning model with early stopping or hyperparameter optimization, you can use `train_judgements` for training your model and `validation_judgements` to stop early or to select the optimal hyperparameters for your model. You can then use `bigger_train_judgements` to train the model with the best number of epochs or the best hyperparameters.

If you are training many machine learning models with early stopping or hyperparameter optimization, then you can split your train judgements to smaller training and validation sets. Then, you can use `smaller_train_judgements` for training your models, `smaller_validation_judgements` to stop early or to select the optimal hyperparameters for your models, and `validation_judgements` to select the best model. You can then use `bigger_train_judgements` to train the best model with the best number of epochs or the best hyperparameters.



```
from pv211_utils.trec.loader import load_judgements

train_judgements = load_judgements(train_queries, documents, 'train')
validation_judgements = load_judgements(validation_queries, documents, 'validation')

bigger_train_judgements = train_judgements | validation_judgements

pivot = int(len(train_judgements) * 0.8)
smaller_train_judgements = set(sorted(train_judgements)[:pivot])
smaller_validation_judgements = set(sorted(train_judgements)[pivot:])
```

```
query = train_queries[301]
relevant_document = documents['FBIS3-10937']
irrelevant_document = documents['FBIS3-10634']
```

```
query
```

```
     <Query 301 “Identify organizations that participate in interna ...”>
```

```
relevant_document
```

```
     <Document FBIS3-10937 “Language: Spanish Article Type:CSO [Text] Amid the ...”>
```

```
irrelevant_document
```

```
     <Document FBIS3-10634 “Language: Spanish Article Type:BFN [Report by Miri ...”>
```

```
(query, relevant_document) in train_judgements
```

```
     True
```

```
(query, irrelevant_document) in train_judgements

     False
```

Implementation of your information retrieval system

Next, we will define a class named `IRSystem` that will represent your information retrieval system. Your class must define a method name `search` that takes a query and returns documents in the descending order of relevance to the query. The example implementation returns documents in random order. Replace it with your own implementation.

```
from typing import Union, List

from gensim.utils import simple_preprocess
from nltk.stem import PorterStemmer


stemmer = PorterStemmer()


def preprocess(document: Union[Query, Document]) -> List[str]:
    tokens = simple_preprocess(document.body)
    tokens = list(map(stemmer.stem, tokens))
    return tokens
```

```
from multiprocessing import Pool

from tqdm.notebook import tqdm


with Pool(None) as pool:
    document_bodies = [
        document_body
        for document_body
        in pool.imap(preprocess, tqdm(documents.values()))
    ]
```

```
index_to_document = dict(enumerate(documents.values()))
```

```
from gensim.corpora import Dictionary
```

```
dictionary = Dictionary(tqdm(document_bodies))
```

```
from gensim.models import TfidfModel, OkapiBM25Model


tfidf_model = TfidfModel(dictionary=dictionary, smartirs='bnn')
bm25_model = OkapiBM25Model(dictionary=dictionary)


from typing import Tuple


def _doc2bow_worker(document_body: List[str]) -> List[Tuple[int, int]]:
    return dictionary.doc2bow(document_body)


def _tfidf_worker(document_body: List[str]) -> List[Tuple[int, float]]:
    vector = _doc2bow_worker(document_body)
    return tfidf_model[vector]


def _bm25_worker(document_body: List[str]) -> List[Tuple[int, float]]:
    vector = _doc2bow_worker(document_body)
    return bm25_model[vector]


from gensim.matutils import corpus2csc
from gensim.similarities import SparseMatrixSimilarity


with Pool(None) as pool:
    document_vectors = [
        document_vector
        for document_vector
        in pool.imap(_bm25_worker, tqdm(document_bodies))
    ]

index = SparseMatrixSimilarity(None)
index.index = corpus2csc(
    document_vectors,
    num_docs=len(documents),
    num_terms=len(dictionary),
    dtype=float,
).T
index.normalize = False





def document_to_vector(document: Union[Query, Document]) -> List[Tuple[int, float]]:
    tokens = preprocess(document)
    if isinstance(document, Document):
        vector = _bm25_worker(tokens)
    else:
        vector = _tfidf_worker(tokens)
        # vector = _doc2bow_worker(tokens)
    return vector


from typing import Iterable

from pv211_utils.trec.irsystem import TrecIRSystemBase

from gensim.matutils import cossim, sparse2full, dense2vec
import numpy as np


class IRSystem(TrecIRSystemBase):
    """
    A system that returns documents in TF-IDF order.

    """
    ROCCHIO = False
    ROCCHIO_ALPHA = 1.0
    ROCCHIO_BETA = 0.75
    ROCCHIO_K = 50

    def search(self, query: Query) -> Iterable[Document]:
        """The ranked retrieval results for a query.

        Parameters
        ----------
        query : Query
            A query.

        Returns
        -------
        iterable of Document
            The ranked retrieval results for a query.

        """
        results = []
        query_vector = document_to_vector(query)
        if self.ROCCHIO:
            relevant_vectors = [
                index.index[document_number].todense()
                for document_number, _
                in zip(self._search(query_vector), range(self.ROCCHIO_K))
            ]
            updated_query_vector = dense2vec(
                self.ROCCHIO_ALPHA * sparse2full(query_vector, len(dictionary)) +
                self.ROCCHIO_BETA * np.mean(relevant_vectors)
            )
        else:
            updated_query_vector = query_vector
        for document_number in self._search(updated_query_vector):
            document = index_to_document[document_number]
            yield document

    def _search(self, query_vector: List[Tuple[int, float]]) -> Iterable[int]:
        similarities = enumerate(index[query_vector])
        for document_id, _ in sorted(similarities, key=lambda item: (-item[1], item[0])):
            yield document_id
```

Evaluation

Finally, we will evaluate your information retrieval system using the Mean Average Precision (MAP) evaluation measure.

```
from pv211_utils.trec.leaderboard import TrecLeaderboard
from pv211_utils.trec.eval import TrecEvaluation

submit_result = False
author_name = "Novotný, Vít"

system = IRSystem()
test_queries = load_queries(Query, 'test')
test_judgements = load_judgements(test_queries, documents, 'test')
evaluation = TrecEvaluation(system, test_judgements, TrecLeaderboard(), author_name)
evaluation.evaluate(tqdm(test_queries.values(), desc="Querying the system"), submit_result)
```

✓ 0s completed at 7:26 PM ● ✕