



University
of Glasgow



IR From Bag-of-words to BERT and Beyond through Practical Experiments

A CIKM 2021 tutorial with
PyTerrier and OpenNIR

Sean MacAvaney*
Craig Macdonald*
Nicola Tonello*

(*Alphabetical ordering)

A Tutorial Scheduled over 22 hours



UNIVERSITÀ DI PISA



University
of Glasgow

Run 1 – Live Lectures, Live Labs

Run 1: Aligned with Europe, Africa, and West/South/Central Asia, partially accessible to East Asia/Oceania

London (GMT)	Content	Live	Johannesburg (GMT+2)	Delhi (GMT+5:30)	Beijing (GMT+8)	Brisbane (GMT+10)
0900-1000	part 1 slides	✓	1100-1200	1430-1530	1700-1800	1900-2000
1000-1030	part 1 lab	✓	1200-1230	1530-1600	1800-1830	2000-2030
1030-1100	break	☕	1230-1300	1600-1630	1830-1900	2030-2100
1100-1200	part 2 slides	✓	1300-1400	1630-1730	1900-2000	2100-2200
1200-1230	part 2 lab	✓	1400-1430	1730-1800	2000-2030	2200-2230
1230-1330	break	☕	1430-1530	1800-1900		
1330-1430	part 3 slides	✓	1530-1630	1900-2000		
1430-1500	part 3 lab	✓	1630-1700	2000-2030		
1500-1530	break	☕	1700-1730	2030-2100		
1530-1630	part 4 slides	✓	1730-1830	2100-2200		
1630-1700	part 4 lab	✓	1830-1900	2200-2230		

Run 2 – Recorded Lectures, Live Labs

Run 2: Aligned with North/South America, partially accessible to East Asia/Oceania

LA (GMT-7)	Content	Live	New York (GMT-4)	São Paulo (GMT-3)	Beijing (GMT+8)	Brisbane (GMT+10)
1100-1200	part 1 slides	🎥	1400-1500	1500-1600		
1200-1230	part 1 lab	✓	1500-1530	1600-1630		
1230-1300	break	☕	1530-1600	1630-1700		
1300-1400	part 2 slides	🎥	1600-1700	1700-1800		
1400-1430	part 2 lab	✓	1700-1730	1800-1830		
1430-1530	break	☕	1730-1830	1830-1930		
1530-1630	part 3 slides	🎥	1830-1930	1930-2030	0630-0730 (6 Nov)	0830-0930 (6 Nov)
1630-1700	part 3 lab	✓	1930-2000	2030-2100	0730-0800 (6 Nov)	0930-1000 (6 Nov)
1700-1730	break	☕	2000-2030	2100-2130	0800-0830 (6 Nov)	1000-1030 (6 Nov)
1730-1830	part 4 slides	🎥	2030-2130	2130-2230	0830-0930 (6 Nov)	1030-1030 (6 Nov)
1830-1900	part 4 lab	✓	2130-2200	2230-2300	0930-1000 (6 Nov)	1030-1200 (6 Nov)

Breakout Lab Facilitators: Sean, Nic, Craig

Breakout Lab Facilitators: Eugene, Luca

Information Retrieval has massively benefitted from a long history excellent **test collections**

- This has allowed many retrieval models to be developed and shown to benefit, significantly, effectiveness
 - E.g. 1970s → 80s – Cranfield & Medlars: Salton, Rocchio et al. SMART & Relevance Feedback
 - E.g. 1990s – TREC 1-4 allowed City Univ. to develop and refine BM25
 - E.g. 2000s → 10s – TREC Web test collections, learning-to-rank

Hence, IR has been a dataset driven empirical science for 50 years!

IR Platforms

A key tenet of this empirical science is **reproducibility**. IR platforms or toolkits have helped our community, as a whole, to have a basis to work from

There are or have been many such well-used platforms over the years

- SMART
- INQUERY/Lemur/Indri
- MG4J
- Terrier
- Lucene/Anserini



The key principals of these IR platforms are well understood:

- Inverted indexes
- Best match retrieval, BM25
- Query expansion/rewriting/pseudo-relevance feedback
- Leveraging past queries
- Learning-to-rank

But IR is Experiencing a Renaissance



Deep neural
network models

Particularly contextual language models, which have allowed whole new areas of research to open up.

This drives us to ensure that we have the **correct tools** (i.e. IR platforms) for the next generation

This provides the 2 primary goals of this tutorial

Deep neural network models

Particularly contextual language models, which have allowed whole new areas of research to open up

This drives us to ensure that we have the **correct tools** (IR platforms) for the next generation

Review recent advances in neural text rankers & experience them in PyTerrier

Experience a new way of doing IR experiments using PyTerrier

But why is it that we need new tools for IR? Any why in Python?

Prevalence of Python



Most of the neural network implementations are accessed in Python, because:

- Easy to write
- Dynamic typing
- Accessibility of REPL tools, such as Jupyter notebooks
- Relatively easy to use native C code for speed when needed
 - E.g. numpy, Torch
- Expressiveness through operator overloading

Java `Tensor<Float> m = Tensor.create(new int[2][2]{{ {1,2},{3,4} }});
m = m.matmul(2)`

Python `m = tf.constant([[1.0, 2.0], [3.0, 4.0]])
m = m @ m`

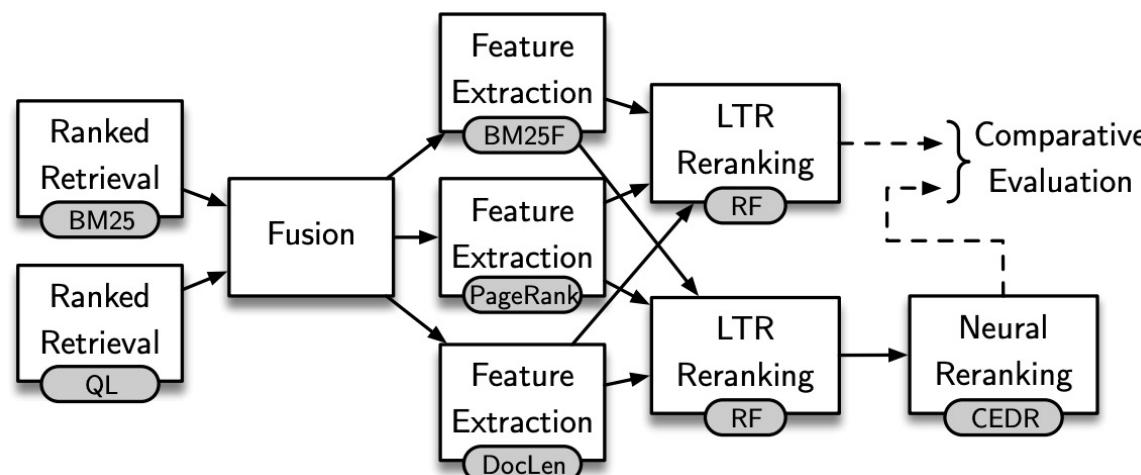
No “for loops” over
the dimensions of
the matrices!

So what is required for modern IR platforms

Python – for all the reasons stated in the previous slide

- Easy to write, and easy to integrate with NN frameworks

Moving beyond imperative “*for each query, for each document*” programming – we need an expressive language for constructing complex IR retrieval pipelines



Towards Declarative Information Retrieval



UNIVERSITÀ DI PISA



University
of Glasgow

We need a (Python) language for describing **what** our IR pipeline will do (not how)

- Linear combinations, re-rankers (incl. neural)
- This moves IR towards a **declarative** manner of writing code

Many IR tools still use `trec_eval` for evaluation – an inherently **file-based workflow**

- We show an alternative declarative workflow in PyTerrier, based on operations upon Python (Pandas) dataframes (relations)

Indeed, all common IR operations can be expressed within the PyTerrier workflow

0900-1030

Outline



University
of Glasgow

Part 1 – *Classical IR: Indexing, Retrieval and Evaluation*

1100-1300

Part 2 – *Modern Retrieval Architectures:*
PyTerrier data model and operators, towards re-rankers and learning-to-rank

Review & Experience

1415-1615

Part 3 – *Contemporary Retrieval Architectures:*
Neural re-rankers such as BERT, EPIC, ColBERT

1645-1815

Part 4 – *Recent Advances beyond the classical inverted index: doc2query, dense retrieval*

Principles

We **review classical, modern, contemporary and recent techniques**

You **experience** these technique within PyTerrier

In each part of the tutorial, we reserve 30 minutes for hands-on practical work



We provide Google Colab notebooks with examples based on the **TREC CORD19 (Covid) test collection**

Live tutoring help via Zoom Breakout Rooms

- Run 1: Craig, Sean, Nicola
- Run 2: Luca & Eugene

Links



PyTerrier repository

- <https://github.com/terrier-org/pyterrier>

PyTerrier documentation

- <https://pyterrier.readthedocs.io>

Tutorial repository

- Notebooks
- Slides
- <https://github.com/terrier-org/cikm2021tutorial>

Your Presenters Today



*Sean MacAvaney
University of Glasgow
(Recently of Georgetown Univ)*



*Craig Macdonald
University of Glasgow*



*Nicola Tonellotto
University of Pisa*

Intended Learning Outcomes (1/4)



Part 1 – *Classical* IR: Indexing, Retrieval and Evaluation

ILO 1A. Describe classical retrieval architecture components based on bag-of-words, including inverted index data structures and ranked retrieval.

ILO 1B. Manipulate index document collections, perform retrieval from indices, and access classical inverted index data structures within a Python notebook.

ILO 1C. Evaluate two retrieval systems using PyTerrier

Intended Learning Outcomes (2/4)



Part 2 – Modern Retrieval Architectures: PyTerrier data model and operators, towards re-rankers and learning-to-rank

ILO 2A. Understand modern retrieval architectures, such as re-rankers and ranking pipelines incorporating learning-to-rank strategies.

ILO 2B. Understand how different ranking and re-ranking operations can be *composed* to make more complex ranking models, in a declarative manner and leveraging a specific data model.

ILO 2C. Perform retrieval experiments within a Python notebook, including use of different features and learning-to-rank

ILO 2D. Understand the benefits of PyTerrier's declarative nature for teaching IR

Intended Learning Outcomes (3/4)



Part 3 – *Contemporary* Retrieval Architectures: Neural re-rankers such as BERT, EPIC, T5

ILO 3A. Understand contemporary retrieval architectures, such as using BERT, EPIC, ColBERT, and T5 as neural re-rankers.

ILO 3B. Understand how query time latency can be reduced by pre-computing representations or using neural models to augment an inverted index.

ILO 3C. Perform experiments with BERT, EPIC, T5, DeepCT, and doc2query in a Python notebook.

Intended Learning Outcomes (4/4)



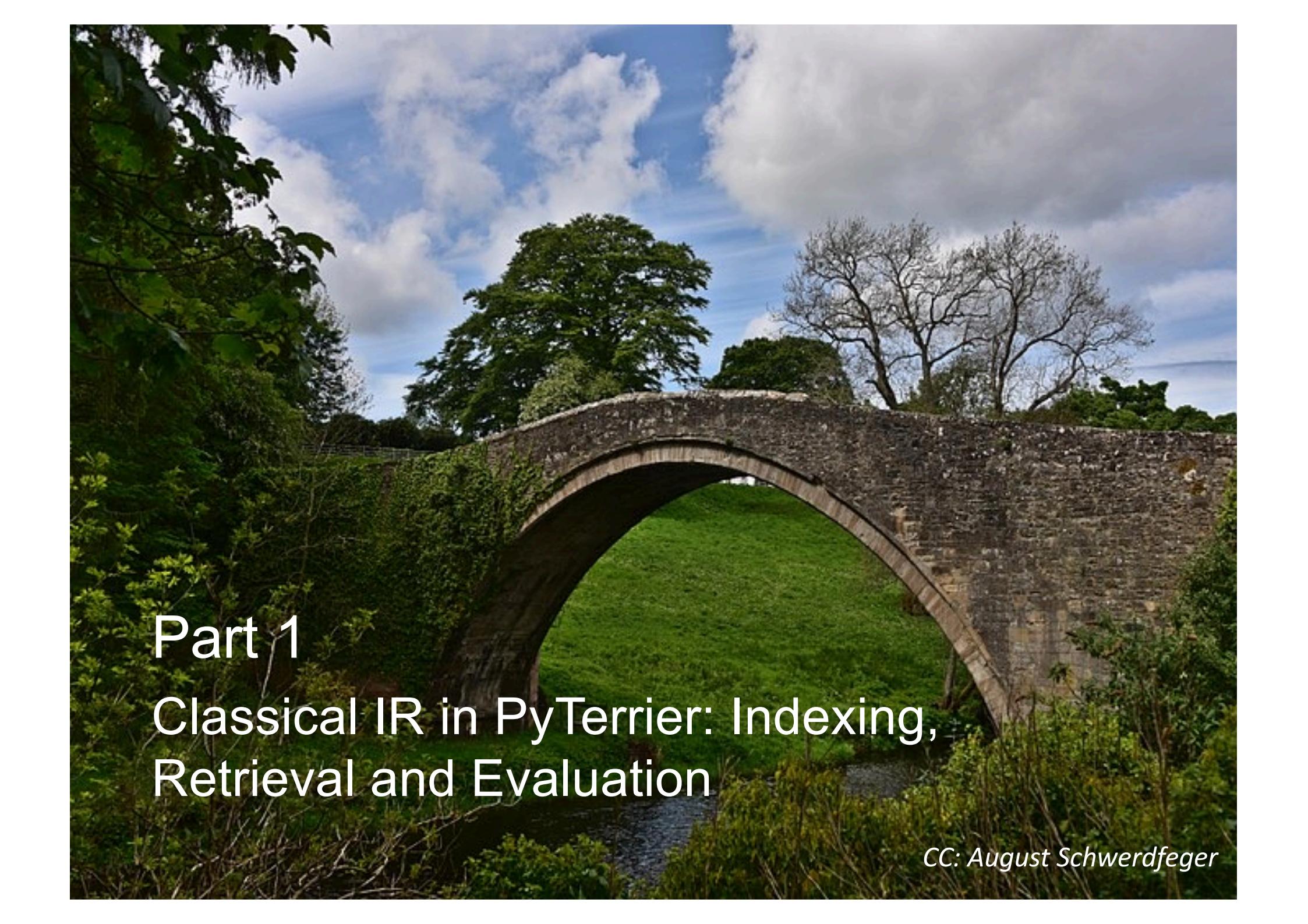
UNIVERSITÀ DI PISA



Part 4 – Recent Advances beyond the classical inverted index: learned sparse retrieval & dense retrieval

ILO 4A. Understand the most recent effective retrieval architectures that learn representations of documents, both in leveraging traditional index structures (sparse retrieval), as well as new similarity search systems (dense retrieval)

ILO 4B. Experience DeeplImpact sparse retrieval approaches, as well as ANCE and ColBERT dense retrieval approaches



Part 1

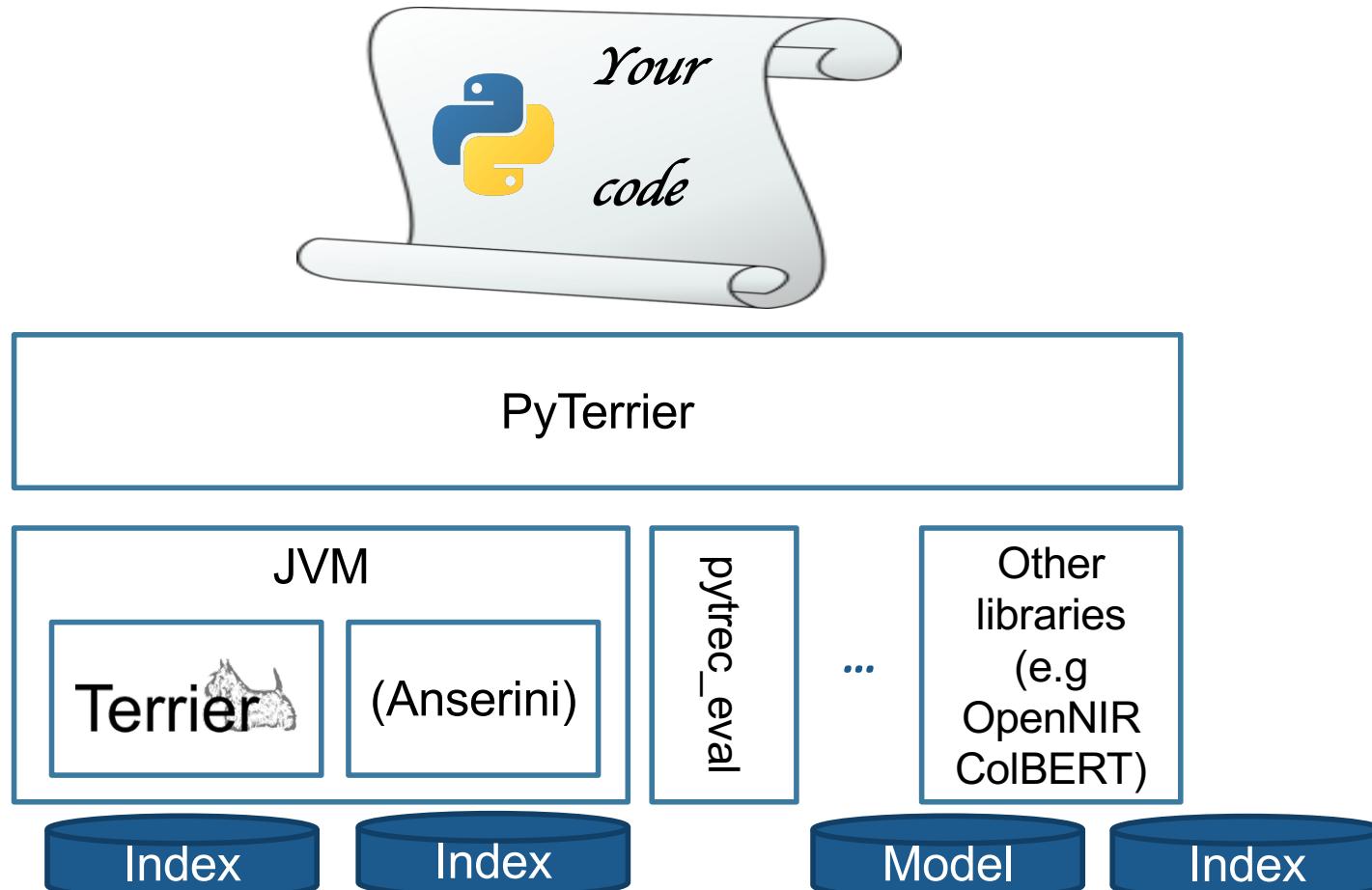
Classical IR in PyTerrier: Indexing, Retrieval and Evaluation

CC: August Schwerdfeger

What is PyTerrier

A Python layer above Terrier (and other IR platforms)

A Python framework for expressing and evaluating IR experiments



Goals and Anti-goals



Goals

Work exclusively in Python

Minimal installation

Indexing of existing collections (e.g. from TREC)

Available standard retrieval techniques

Easy evaluation

Easy integration for learning-to-rank [part 2]

Easy integration for neural re-rankers [part 3]

Anti-goals

No extra configuration files

No results files

Not just Terrier

Avoiding writing new Java code

Avoiding writing command-line scripts

In Part 1



UNIVERSITÀ DI PISA



University
of Glasgow

“From bag-of-words”

Classical IR in PyTerrier: Indexing, Retrieval and Evaluation

1. Getting started
2. Indexing
3. Retrieval
4. Evaluation

Lets get started...



Install PyTerrier

```
pip install python-terrier
```

We do semi-regular releases

Get things going:

```
import pyterrier as pt  
pt.init()
```

```
terrier-assemblies 5.4  jar-with-dependencies not found, downloading to /root/.pyterrier...  
Done  
terrier-python-helper 0.0.5  jar not found, downloading to /root/.pyterrier...  
Done  
PyTerrier 0.4.0 has loaded Terrier 5.4 (built by craigm on 2021-01-16 14:17)
```

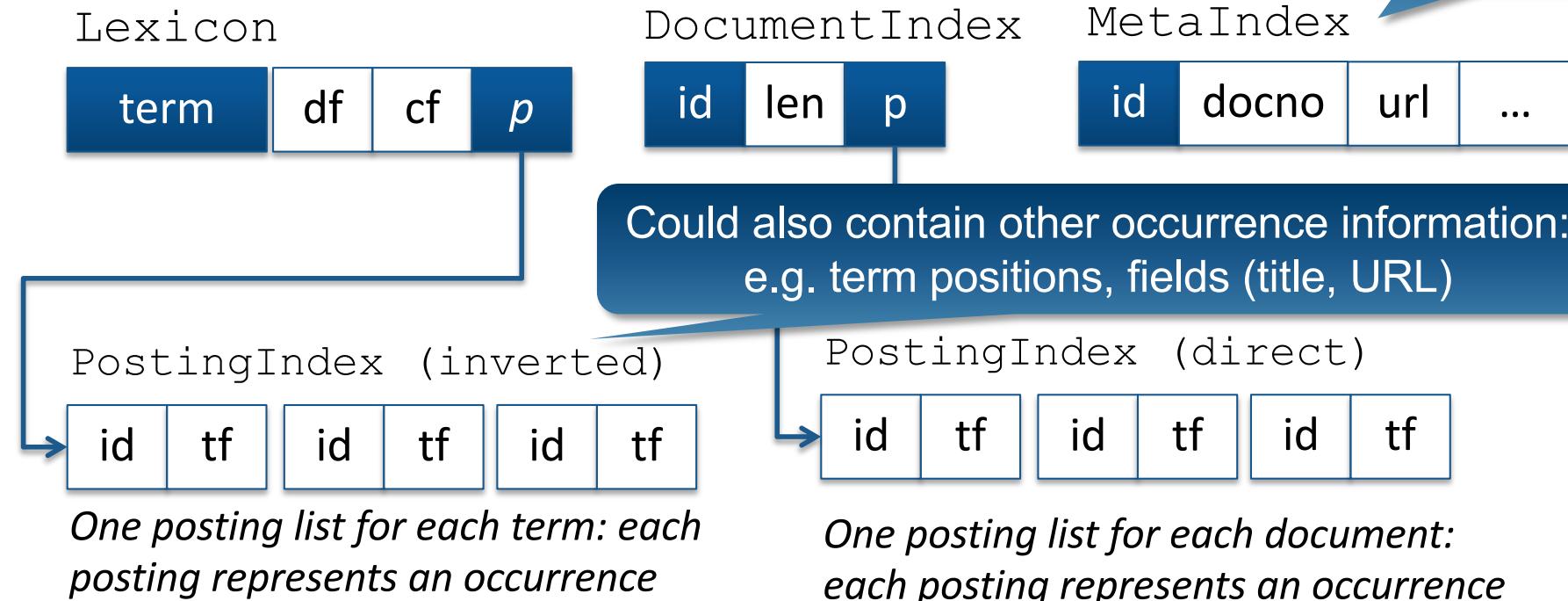
JAR files for Terrier etc are downloaded automatically

The Format of a (“Sparse”) Index

An index normally contains several data structures

- **Lexicon:** Records the list of all unique terms and their statistics
- **Document Index:** Records the statistics of all documents
- **Inverted Index:** Records the mapping between terms and documents. Contains many posting *lists*.
- **MetaIndex:** Records document metadata
- **Direct Index:** Records terms for each document

Document
“information”
Incl. raw text



Creating an Index

Lets consider a few sample documents

```
df = pd.DataFrame({  
    'docno':  
        ['1', '2', '3'],  
    'url':  
        ['url1', 'url2', 'url3'],  
    'text':  
        ['He ran out of money, so he had to stop playing',  
         'The waves were crashing on the shore; it was a',  
         'The body may perhaps compensates for the loss']  
})
```



docno	url	text
1	url1	He ran out of money, so he had to stop playing
2	url2	The waves were crashing on the shore; it was a
3	url3	The body may perhaps compensates for the loss

```
pd_indexer = pt.DFIndexer("./pd_index")
```

```
# Add metadata fields as Pandas.Series objects, with the  
# name of the Series object becoming the name of the meta field.  
indexref = pd_indexer.index(df["text"], df["docno"])
```

This creates a directory called ‘pd_index’ with files for the various index data structures

data.direct.bf
data.document.fsarrayfile
data.inverted.bf

data.lexicon.fsomapfile
data.lexicon.fsomaphash
data.lexicon.fsomapid

data.meta-0.fsomapfile
data.meta.idx
data.meta.zdata

data.properties

Index Configurations

**Record positions to allow proximity/phrase search by adding
blocks=True to the constructor**

PostingIndex (inverted)

id	tf	id	tf	id	tf
----	----	----	----	----	----

No positions

Positions
(more space)

PostingIndex (inverted)

id	tf	p	p	p	id	tf	p	id	tf	p	p
----	----	---	---	---	----	----	---	----	----	---	---

Document metadata can be stored in the MetaIndex

- E.g. saving the raw document text can be useful for neural re-rankers
- Minimum/default: “docno” a unique string identifier for each document
- Recording more metadata takes more space and can slow down retrieval
- How these are sourced depends on the particular indexer

MetaIndex

id	docno	url	text	...
----	-------	-----	------	-----

Indexers and IndexingTypes



Indexers are utility classes in PyTerrier defining how we pass documents to be indexed by Terrier:

- **pt.DFIndexer** – index a dataframe
- **pt.TRECCollectionIndexer** – index files formatted in TREC or WARC (ClueWeb) format
- **pt.FilesIndexer** – index files of HTML, Word, PDF TXT etc.
- **pt.IterDictIndexer** – index iterable dictionaries (very similar to *streams* of dataframe rows)

PyTerrier will focus on this indexer in the future:

(1) multi-threading

(2) able to build into indexing pipelines [part 2]

(3) generic interface that other indexers (e.g. dense) can use [part 4]

IndexingTypes

- Classical – creates a direct index first, then inverts it. Default, but slower
- SinglePass – creates slices of inverted index in memory, then merges
- Memory – creates indices in memory

Overall aim – make it easy to make an index...

Datasets



There are now a plethora of test collections

- Some document datasets require a license/agreement
 - E.g. Some TREC corpora such as Disks 4 & 5; GOV2; ClueWebxx
- Topics/qrels are distributed over the web
 - They can be burdensome to find

Moreover, often we are using experimental environments with ephemeral storage – e.g. Google Colab

PyTerrier's datasets API provides downloading of queries, relevance assessments (aka qrels) & corpus documents, e.g.

```
dataset = pt.get_dataset('irds:cord19/trec-covid')
pt.IterDictIndexer('./index').index(dataset.get_corpus_iter())
```

Datasets (2)



PyTerrier has 128 datasets

- Full listing at



<https://pyterrier.readthedocs.io/en/latest/datasets.html>

This includes 109 datasets from the `ir_datasets` package, developed by Sean and colleagues at AllenAI Inst.

- Prefixed by `irds:` in

PyTerrier

- <https://ir-datasets.com/>

Available Datasets

The table below lists the provided datasets, detailing the attributes available for each dataset. In each column, True designates the presence of a single artefact of that type, while a list denotes the available variants. Datasets with the `irds:` prefix are from the `ir_datasets` package; further documentation on these datasets can be found [here](#).

dataset	corpus	index	topics	qrels
50pct		['ex1', 'ex2']	[training, validation]	[training, validation]
antique	True		[train, test]	[train, test]
vaswani	True	True	True	True
trec-deep-learning-docs	True		[train, dev, test, test-2020, leaderboard-2020]	[train, dev, test, test-2020, leaderboard-2020]
trec-deep-learning-passages	True		[train, dev, eval, test-2019, test-2020]	[train, dev, eval, test-2019, test-2020]

Dataset Index

✓ : Data available as automatic download

⚠ : Data available from a third party

Dataset	docs	queries	qrels	scoreddocs	docpairs
antique	✓				
antique/test	✓	✓	✓		
antique/test/non-offensive	✓	✓	✓		
antique/train	✓	✓	✓		
antique/train/split200-train	✓	✓	✓		
antique/train/split200-valid	✓	✓	✓		
 aquaint			⚠		
aquaint/trec-robust-2005	⚠	✓	✓		

Accessing Terrier Index Data Structures from Python

Lexicon:

```
index.getLexicon()["chemic"].getDocumentFrequency()
```

How many documents does
'chemical' appear in?

DocumentIndex

```
index.getDocumentIndex().getDocumentLength(22)
```

What is the length of the 23rd
document?

Inverted Index

```
meta = index.getMetaIndex()
inv = index.getInvertedIndex()

le = lex.getLexiconEntry("chemic")
# the lexicon entry is also our pointer to access the inverted index posting list
for posting in inv.getPostings( le ):
    docno = meta.getItem("docno", posting.getId())
    print("%s with frequency %d" % (docno, posting.getFrequency()))
```

What documents contain
'chemical'?



<https://pyterrier.readthedocs.io/en/latest/terrier-index-api.html>



Retrieval

Now we have an index, lets perform retrieval on it. We use a class called BatchRetrieve

- Don't let the name mislead you! It can also be used for single queries. We are renaming it to TerrierRetrieve...

```
br = pt.BatchRetrieve(index, wmodel="BM25")
```

We can search it

```
br.search("chemical reactions")
```

	qid	docid	docno	rank	score	query
0	1	251931	449027_8	0	11.954261	chemical reactions
1	1	251934	449027_11	1	11.091389	chemical reactions
2	1	251926	449027_3	2	11.087365	chemical reactions
3	1	36690	2769813_5	3	10.902714	chemical reactions

NB: There is also a AnseriniBatchRetrieve for retrieving from an Anserini index

Retrieving more than one query



For experiments, often we operate on more than one query. Lets say we have a dataframe of queries, queryset

```
queryset = pt.io.read_topics("./trec.topics")  
          qid           query  
0    3990512  how can we get concentration onsomething  
1    714612   why doesn t the water fall off earth if it s r...  
2    2528767  how do i determine the charge of the iron ion ...  
  
queryset =  
  
res = br.transform(queryset)  
  
          qid  docid  docno  rank      score           query  
0    3990512  173781  4366141_0  0  7.705589e+00  how can we get concentration onsomething  
1    3990512  381364  3378079_2  1  7.485244e+00  how can we get concentration onsomething  
2    3990512  269289  3270641_2  2  7.351503e+00  how can we get concentration onsomething  
...  
188628  1340574  291774  2489233_1  997  5.957036e+00  why do some people only go to church on easter...  
188629  1340574  259693  3073253_4  998  5.955254e+00  why do some people only go to church on easter...  
188630  1340574  265175  3610636_18  999  5.953573e+00  why do some people only go to church on easter...
```

In fact, the transform() method is used so often, it can be omitted

```
br(queryset)
```

Datasets also offer the topics as dataframes, ready to use

```
br(dataset.get_topics())
```

BatchRetrieve configurations



We expose most useful configuration through constructor arguments

```
pt.BatchRetrieve(index, wmodel="BM25")
```

Terrier has many such weighting models

- <http://terrier.org/docs/current/javadoc/org/terrier/matching/models/package-summary.html>
- Including TF_IDF, Divergence from Randomness models such as PL2, DPH, Dirichlet language model etc
- Also field-based models such as BM25F, PL2F

Other arguments:

- controls – other internal Terrier configuration
- num_results – how many results to retrieve
- verbose – display a progress bar
- metadata – what document metadata to export (default ["docno"])

Do we even need an index?

Imagine our dataframe contains query and doc texts

```
pt.BatchRetrieve(index, metadata=[“docno”, “text”])
```

	qid	query	docno	text
0	q1	chemical reactions	d1	professor proton poured the chemicals
1	q1	chemical reactions	d2	chemical brothers turned up the beats

We can score that directly using a weighting model

```
Tfs = pt.text.scorer(wmodel="Tf")  
Tfs.transform(query_docs)
```

	qid	docno	rank	score	query
0	q1	d1	0	1.0	chemical reactions
1	q1	d2	1	1.0	chemical reactions

```
BM25s = pt.text.scorer(wmodel="BM25", background_index=index)
```

```
BM25s.transform(query_docs)
```

	qid	docno	rank	score	query
0	q1	d1	0	13.823546	chemical reactions
1	q1	d2	1	13.823546	chemical reactions



These are like re-ranking operations – more on those in Part 2!

Evaluating Pipelines



A basic experiment typically has 3 procedural steps:

- obtain the queries Q and the corresponding relevance assessments (qrels) RA
- transform those queries into results using a BatchRetrieve instance, let's say

$$R = \text{retrieve}(Q)$$

- apply an evaluation tool, such as the ubiquitous trec_eval tool on RA and R (saved as as files?), to obtain effectiveness measures such as MAP or NDCG

This would need to be repeated for each retrieval instance

- And for loops are bad!

Also complex to compute things like number of queries improved, significance testing, etc.

An Experiment Abstraction



We define an **Experiment abstraction**, which performs an evaluation of multiple systems on queries Q and labels RA

```
pt.Experiment([br1, br2], Q, RA, ["map", "ndcg"])
```

Returns a dataframe with measure values for each system

This is **declarative** in nature – we say what we want, not how to get it

- No for loops! ☺
- No writing files and then evaluating
- No longer dealing with results at all. Experiment does this for you – we have abstracted away from the results entirely

Internally, **pt.Experiment** uses **pytrec_eval**, which is a Python wrapper of **trec_eval**

Experiment Example

An example Experiment might look like

```
pt.Experiment(  
    [tfidf, bm25],  
    dataset.get_topics(),  
    dataset.get_qrels(),  
    eval_metrics=["map", "ndcg_cut_10"])
```

What are we evaluating?
What topics?
What qrels?
What measures?
(trec_eval names)

This outputs a dataframe as follows:

	name	map	ndcg_cut_10
0	BR(TF_IDF)	0.290905	0.444411
1	BR(BM25)	0.296517	0.446609

Experiment Example, with more measures

If you are a trec_eval expert, “ndcg_cut_10” might make sense. For mere mortals...

```
from pyterrier.measures import *
pt.Experiment(
    [tfidf, bm25],
    dataset.get_topics(),
    dataset.get_qrels(),
    eval_metrics=[MAP, NDCG@10]
)
```

Import the available measures

Name the measure, @ cutoff

This outputs a dataframe as follows:

	name	AP	nDCG@10
0	BR(TF_IDF)	0.290905	0.444411
1	BR(BM25)	0.296517	0.446609

This is due to ir_measures – a more elegant evaluation tool by Sean. It wraps (py)trec_eval, ndeval, cwl_eval, etc.. ir_measures: <https://ir-measur.es/> cwl_eval: Azzopardi et al, SIGIR 2019

Experiment Example, with names

We can put names in the output dataframe

```
pt.Experiment(  
    [tfidf, bm25],  
    dataset.get_topics(),  
    dataset.get_qrels(),  
    names=["TF.IDF", "BM25"], ← Names of our approaches  
    eval_metrics=["map", "ndcg_cut_10"])
```

Resulting output:

	name	map	ndcg_cut_10
0	TF.IDF	0.290905	0.444411
1	BM25	0.296517	0.446609

Experiments Variations (1)

Scientifically sound conclusions needs a baseline

- And significance testing

Declaring a baseline – specify the system you want to compare to

```
pt.Experiment(  
    [tfidf, bm25],  
    dataset.get_topics(),  
    dataset.get_qrels(),  
    names=["TF.IDF", "BM25"],  
    baseline=0, ← Index of the baseline system  
    eval_metrics=["map", "ndcg_cut_10"] )
```

	name	map	ndcg_cut_10	map +	map -	map p-value	ndcg_cut_10 +	ndcg_cut_10 -	ndcg_cut_10 p-value
0	TF.IDF	0.290905	0.444411	NaN	NaN	NaN	NaN	NaN	NaN
1	BM25	0.296517	0.446609	46.0	45.0	0.237317	16.0	23.0	0.63001

of queries improved
and degraded

P-value of two-tailed
paired t-test

Repeated for each
measure

Experiment Variations (3)

Lets add another model, say DPH, and compare with plain TF.IDF and BM25 again, with significance testing.

- This leads to the problem that the probabilities of the type I errors of the tests add up
- Increases likelihood of rejecting a true null hypothesis, i.e. declaring a significant difference when there actually is none

```
pt.Experiment(  
    [tfidf, bm25, dph],  
    dataset.get_topics(),  
    dataset.get_qrels(),  
    names=["TF.IDF", "BM25", "DPH"],  
    baseline=0,  
    correction='bonferroni', # or just 'b'  
    eval_metrics=["map", "ndcg_cut_10"])
```

What correction method

name	map	ndcg_cut_10	map		map p-value	map reject	map p-value corrected	ndcg_cut_10	ndcg_cut_10		ndcg_cut_10 p-value	ndcg_cut_10 reject	ndcg_cut_10 p-value corrected
			+	-					+	-			
0	TF.IDF	0.290905	0.444411	NaN	NaN	NaN	False	NaN	NaN	NaN	NaN	False	NaN
1	BM25	0.296517	0.446609	46.0	45.0	0.237317	False	0.711951	16.0	23.0	0.630010	False	1.0
2	DPH	0.296517	0.446609	46.0	45.0	0.237317	False	0.993016	20.0	10.0	0.403840	False	1.0

Can null hypothesis be rejected
($p < 0.05$) after correction

Corrected p-value

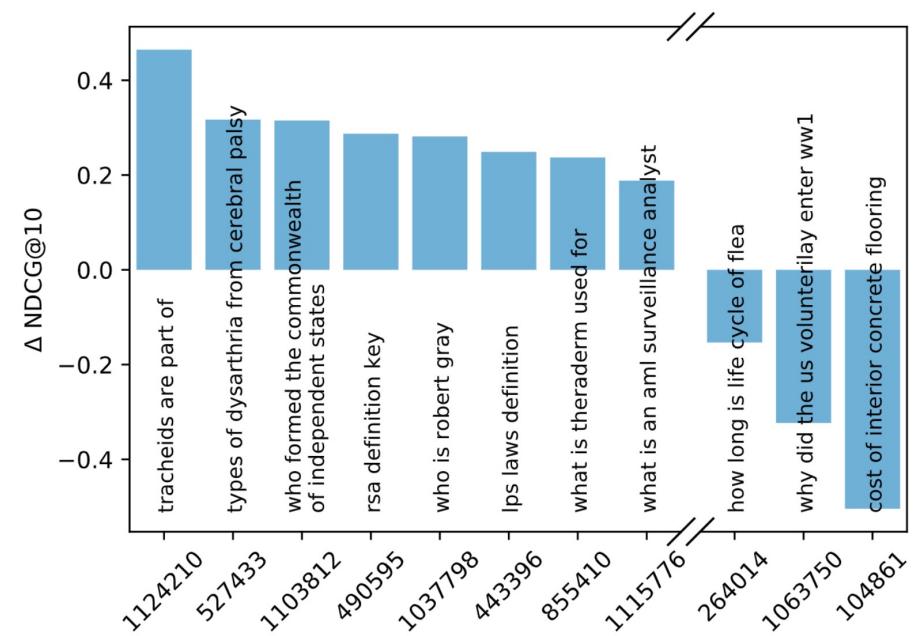
Experiment Variations (4)

Which queries were affected

```
pt.Experiment(  
    [tfidf, bm25],  
    dataset.get_topics(),  
    dataset.get_qrels(),  
    names=["TF.IDF", "BM25"],  
    perquery=True,  
    eval_metrics=["map", "ndcg_cut_10"] )
```

← Breakdown per query

	name	qid	measure	value
0	TF.IDF	1	map	0.268860
1	TF.IDF	1	ndcg_cut_10	0.573690
2	TF.IDF	2	map	0.056448
3	TF.IDF	2	ndcg_cut_10	0.094788



Experiment Variations (5)

Comparing to saved results

- Use `pt.io.write_results()` and `pt.io.read_results()`

```
pt.io.write_results(tfidf(dataset.get_topics()), "tfidf.res")
```

```
baselineDF = pt.io.read_results("tfidf.res")
```

```
pt.Experiment(  
    [baselineDF, bm25],  
    dataset.get_topics(),  
    dataset.get_qrels(),  
    names=["baseline DF", "BM25"],  
    eval_metrics=["map", "ndcg_cut_10"])
```



A results dataframe can be directly passed to `pt.Experiment`

Useful for large collections or complex retrieval mechanisms



<https://pyterrier.readthedocs.io/en/latest/experiments.html>

Standard Indices – the Terrier Data Repository



For common corpora (e.g. TREC-Covid, MSMARCO), we provide standard indices, which can be downloaded and instantiated in a single-line:

```
bm25_terrier_stemmed = pt.BatchRetrieve.from_dataset( use .from_dataset())
    'msmarco_passage', 'terrier_stemmed',  Name the dataset and index variant
    wmodel='BM25' )  Any more args for BatchRetrieve?
```

The Terrier Data Repository (<http://data.terrier.org/>) contains listings of indices. E.g. for MSMARCO passage corpus, we provide:

- 6 indices: terrier_stemmed, terrier_stemmed_deepct, terrier_stemmed_docT5query, terrier_stemmed_text, terrier_unstemmed, terrier_unstemmed_text, ance
- Snippets of code to use each index, including with BERT-based re-rankers
- Notebook reproducing baseline results, including pt.Experiment() configurations

Terrier Data Repository

PyTerrier Data Repository



This is the index repository for Terrier/PyTerrier.

[Find out more about information](#)

[PyTerrier Home](#)

[About The Site](#)

About This Site

This site is a repository of indices for PyTerrier and Terrier.



UNIVERSITÀ DI PISA



University
of Glasgow

Vaswani

Last Update 2021-09-28 • 5 index variants

The Vaswani NPL corpus is a small test collection of 11,000 abstracts from the Glasgow IR group for many years (created 1990). Due to its small size, it is often used as a test case in both Terrier and PyTerrier.

[More details →](#)

MSMARCO Document Ranking

Last Update 2021-09-27 • 5 index variants

A document ranking corpus containing 3.2 million documents. Also includes a Learning track.

[More details →](#)

*Prebuilt indices for seven common datasets
For each dataset, various indices are provided*

Terrier Data Repository



terrier_stemmed_text

Last Update 2021-09-27 • 9.6GB

Terrier's default Porter stemming, and stopwords removed. Text is also saved in the MetalIndex to facilitate BERT-based reranking.

[Browse index](#)

Use this for retrieval in PyTerrier:

```
#!/usr/bin/python3
# This file is part of the OpenNIR project.
# Copyright (c) 2021 Georgetown University Library
# See LICENSE for details.

import onir_pt

# Lets use a Vanilla BERT ranker from OpenNIR. We'll use the Capreolus model available from Huggingface
vanilla_bert = onir_pt.reranker('hgf4_joint', text_field='body', ranker_config={'model': 'Capreolus/bert-base-uncased'}, reranker_config={})

bm25_bert_terrier_stemmed_text = (
    pt.BatchRetrieve.from_dataset('msmarco_document', 'terrier_stemmed_text', wmodel='BM25', metadata=['docname'])
    >> pt.text.sliding(length=128, stride=64, prepend_attr='title')
    >> vanilla_bert
    >> pt.text.max_passage()
```

Code snippets to instantiate retrieval pipelines using various index variants

Terrier Data Repository



Evaluation on trec-2019 topics and qrels

43 topics used in the TREC Deep Learning track Passage Ranking task, with deep judgements

```
pt.Experiment(  
    [bm25_terrier_stemmed, dph_terrier_stemmed, bm25_terrier_stemmed_text, bm25_bert_terrier_stemmed_text, bm25_terrie  
r_stemmed_docT5query, bm25_terrier_stemmed_deepct],  
    pt.get_dataset('msmarco_passage').get_topics('test-2019'),  
    pt.get_dataset('msmarco_passage').get_qrels('test-2019'),  
    batch_size=200,  
    filter_by_qrels=True,  
    eval_metrics=[RR(rel=2), nDCG@10, nDCG@100, AP(rel=2)],  
    names=['bm25_terrier_stemmed', 'dph_terrier_stemmed', 'bm25_terrier_stemmed_text', 'bm25_bert_terrier_stemmed_tex  
t', 'bm25_terrier_stemmed_docT5query', 'bm25_terrier_stemmed_deepct'])
```

```
11:17:26.746 [main] WARN org.terrier.applications.batchquerying.TRECQuery - trec.encoding is not set; resorting to pl  
atform default (ISO-8859-1). Retrieval may be platform dependent. Recommend trec.encoding=UTF-8  
config file not found: config  
[2021-09-23 11:19:03,772][onir_pt][DEBUG] using GPU (deterministic)  
[2021-09-23 11:19:06,355][onir_pt][DEBUG] [starting] batches
```

```
[2021-09-23 11:22:36,997][onir_pt][DEBUG] [finished] batches: [03:31] [10502it] [49.86it/s]
```

	name	RR(rel=2)	nDCG@10	nDCG@100	AP(rel=2)
0	bm25_terrier_stemmed	0.641565	0.479540	0.487416	0.286448
1	dph_terrier_stemmed	0.667307	0.502513	0.485995	0.308977
2	bm25_terrier_stemmed_text	0.641565	0.479540	0.487416	0.286448
3	bm25_bert_terrier_stemmed_text	0.829457	0.685453	0.635066	0.444111
4	bm25_terrier_stemmed_docT5query	0.761370	0.630835	0.592220	0.404429
5	bm25_terrier_stemmed_deepct	0.689009	0.534393	0.521540	0.323135

Example notebooks to demonstrate reproducing results

In today's practical labs, we'll often be using the pre-built indices for TREC-Covid

Round Up



We have **reviewed** the classical IR infrastructure, as implemented by PyTerrier, namely:

- Indexing, including datasets, accessing an index
- Retrieval & Evaluation

Next, you can use the provided Part 1 notebook to **experience** this infrastructure using the TREC Covid19 test collection

Coming Next: You have seen two retrieval objects: BatchRetrieve and pt.text.scorer(). These are two **transformers**

- They *transform* a dataframe into another dataframe
- In part 2, we'll generalise what we have seen into different types of transformers, and show how to combine them



University
of Glasgow



CIKM
2021
1-5 NOVEMBER

QUESTIONS?

Practical Time



The tutorial Github repo has links to the notebook for
Part 1

- <https://github.com/terrier-org/cikm2021tutorial>
- Press the  Open in Colab link for each notebook to start a Colab session

Timings:

- Practical – in breakout rooms – until 1030 GMT
- Coffee break 1030-1100 GMT
- Part 2 resumes at 1100 GMT

Run 1 times

If you leave, please complete our feedback quiz

<https://forms.office.com/r/RiYSAxAKhk> [link also on repo]