



University
of Glasgow



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

An ECIR 2021 tutorial with
PyTerrier and OpenNIR

Sean MacAvaney*
Craig Macdonald*
Nicola Tonellotto*

(*Alphabetical ordering)

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 has a whole 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 or 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 has allowed a 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

Things have moved on!

Deep neural network models

Particularly contextual language models has allowed a 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`

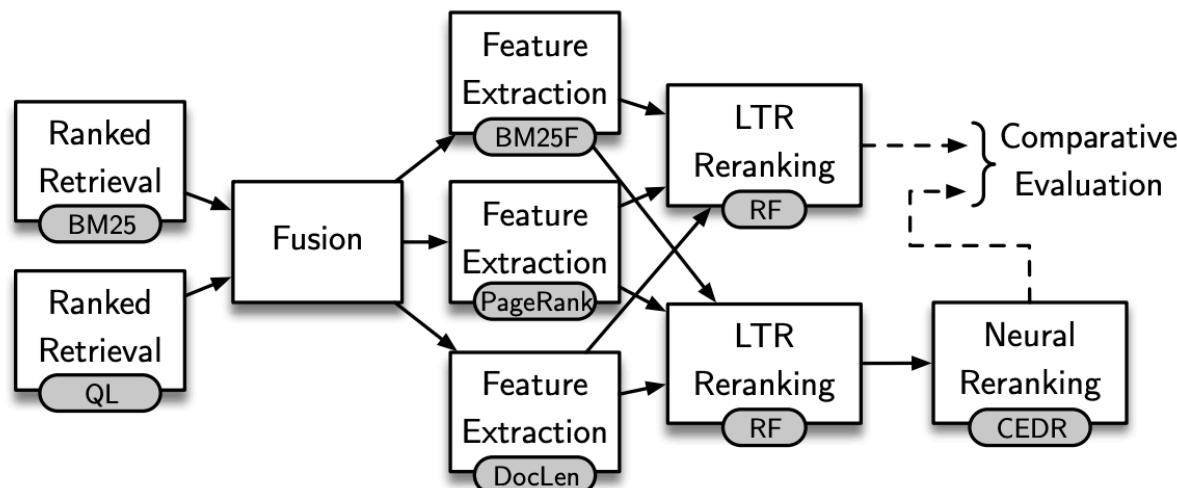
No “for loops” over
the dimensions of
the matrices!

So what is required for modern IR platforms

Python – for all the reasons stated in the previous slide

- Easy to write, 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



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

Outline



0900-1030

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

1100-1230

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

Review & Experience

1400-1530

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

1600-1730

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

There is time in each part for practical work



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

Live tutoring help via Zoom Breakout Rooms

Links

PyTerrier repository

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

PyTerrier documentation

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

Tutorial repository

- Notebooks
- Slides
- <https://github.com/terrier-org/ecir2021tutorial>

Intended Learning Outcomes (1/4)



UNIVERSITÀ DI PISA



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

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 architecture, such as re-rankers and ranking pipeline 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

Intended Learning Outcomes (3/4)



UNIVERSITÀ DI PISA



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

ILO 3A. Understand contemporary retrieval architectures, such as BERT re-rankers.

ILO 3B. perform BERT and T5 re-ranking experiments within a Python notebook.

Intended Learning Outcomes (4/4)



UNIVERSITÀ DI PISA



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

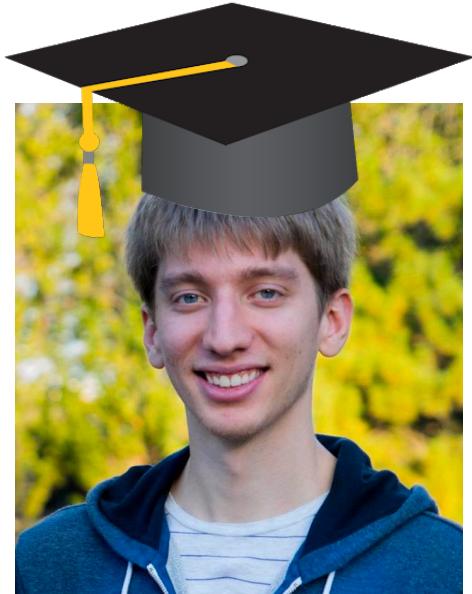
ILO 4A. understanding of the most recent effective retrieval architectures that augment traditional index structures such as doc2query and deepCT, as well as dense retrieval approaches

ILO 4B. experience doc2query and deepCT, as well as ANCE and ColBERT dense retrieval approaches

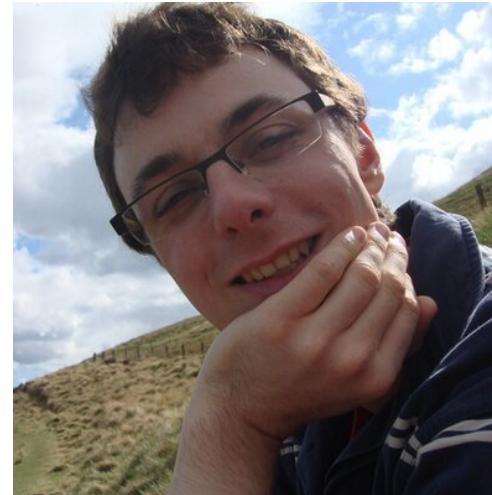
Your Presenters Today



University
of Glasgow



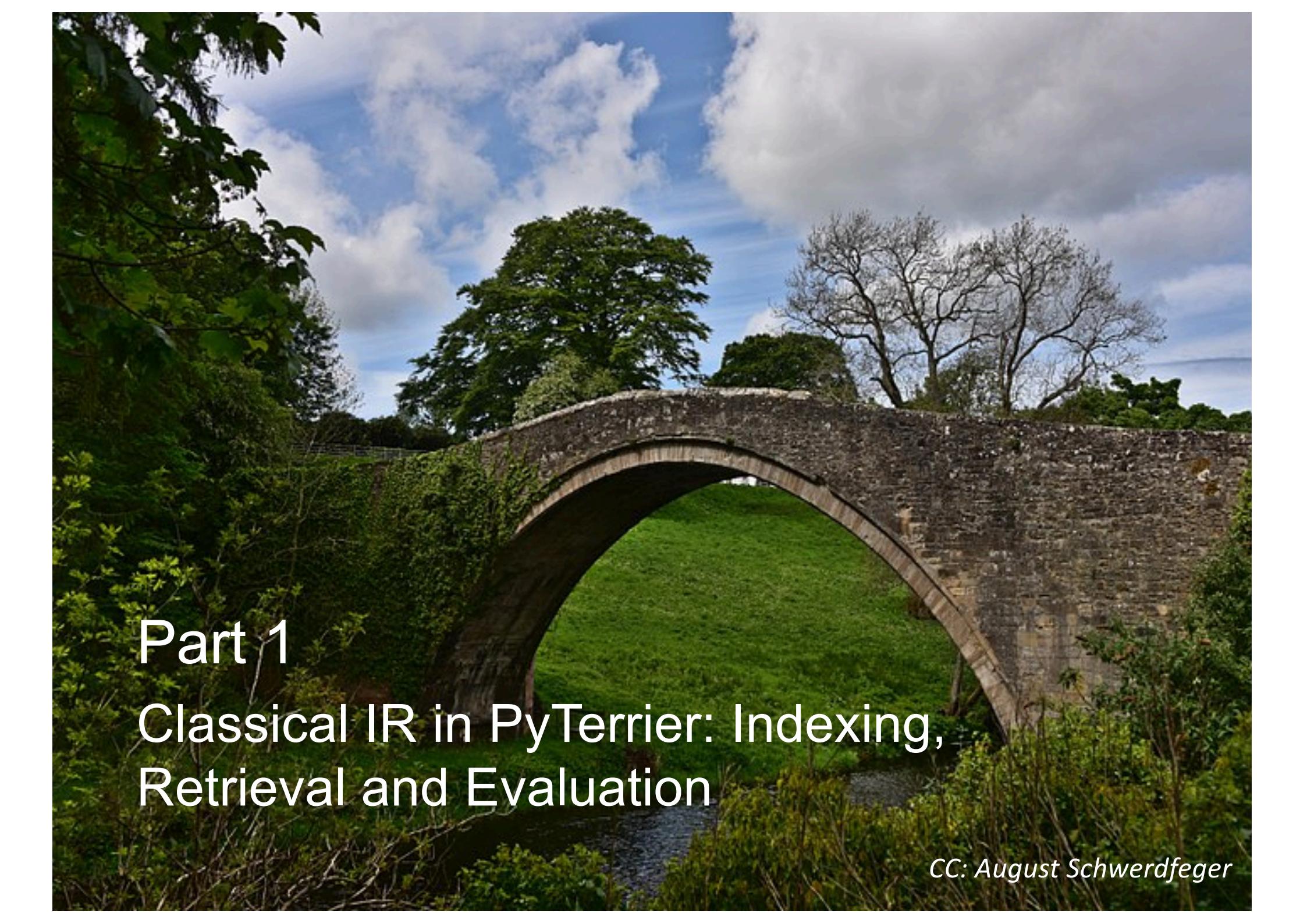
Sean MacAvaney
University of Glasgow
(Recently of Georgetown Univ)



Craig Macdonald
University of Glasgow



Nicola Tonellotto
University of Pisa

A photograph of a large, single-arched stone bridge spanning a grassy valley. The bridge is made of dark grey stone and has a thick, curved masonry arch. It sits atop a low stone wall. The surrounding landscape includes lush green trees and bushes on the left, and a mix of green and bare trees on the right. The sky above is a bright blue with scattered white and grey clouds.

Part 1

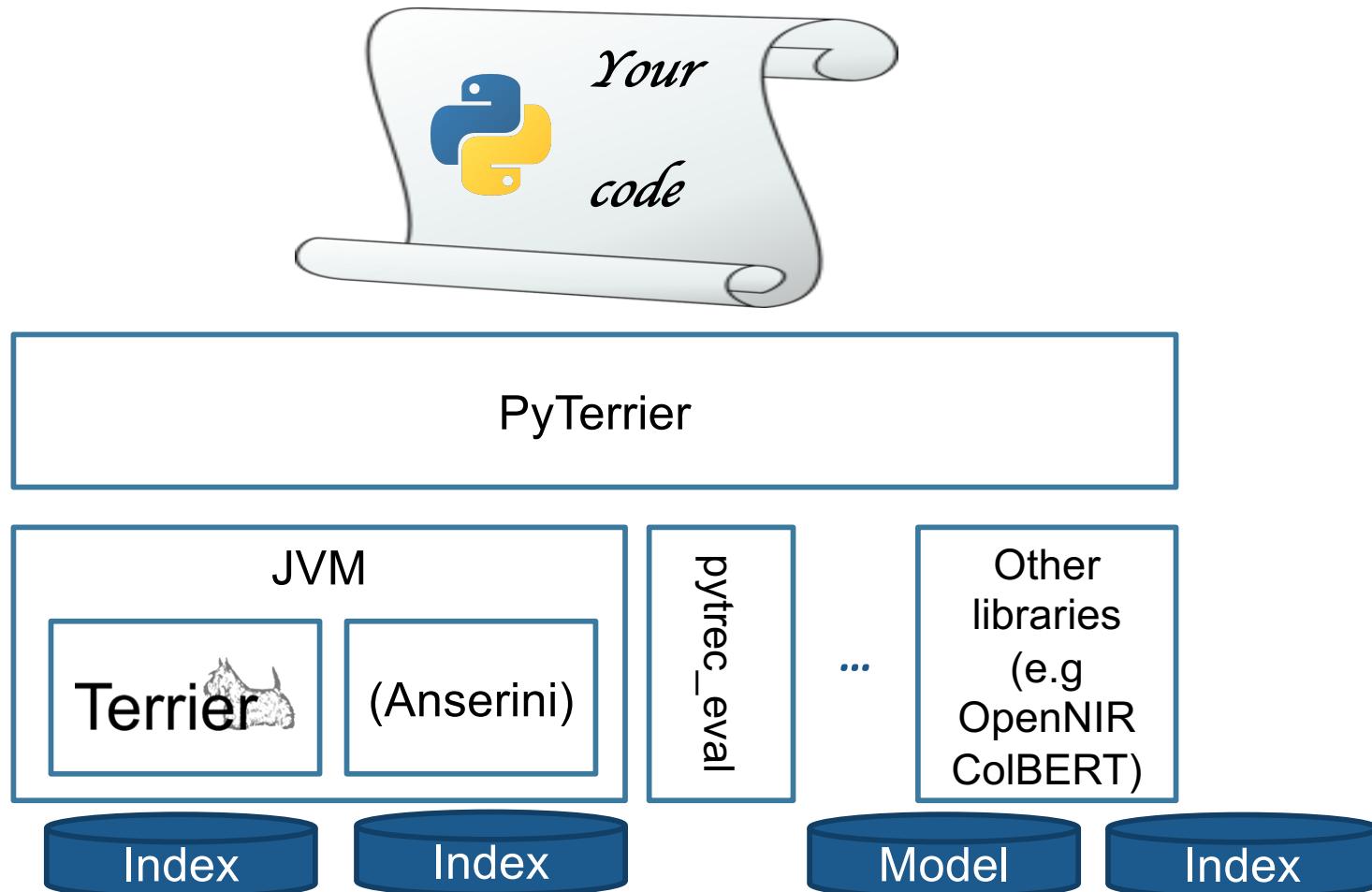
Classical IR in PyTerrier: Indexing, Retrieval and Evaluation

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)
- 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 run files
- Not just Terrier
- Avoiding writing new Java code

In Part 1



“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 an 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
- **MetaIndex:** Records document metadata

Document
“information”
Incl. raw text

Lexicon

term	df	cf	p
------	----	----	---

DocumentIndex

id	len
----	-----

MetaIndex

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

PostingIndex (inverted)

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

each entry (posting) represents a document

Could also contain other occurrence information: e.g. term positions, fields (title, URL)

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

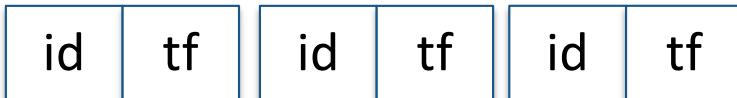
data.meta-0.fsomapfile
data.meta.idx
data.meta.zdata

data.properties

Index Configurations

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

PostingIndex (inverted)



No positions

PostingIndex (inverted)

Positions

(Up to 4x more space)



Document metadata can be stored in the MetalIndex

- 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

MetalIndex



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,
antique	True		[train, test]	[train, tes
vaswani	True	True	True	True
trec-deep-learning-docs	True		[train, dev, test, test-2020, leaderboard-2020]	[train, dev
trec-deep-learning-passages	True		[train, dev, eval, test-2019, test-2020]	[train, dev

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 plan to rename it TerrierRetrieve in the future

```
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

	qid	query
queryset =	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 ...

`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
	3 3990512	389678	882429_10	3	7.329420e+00	how can we get concentration onsomething
	4 3990512	237313	161518_0	4	7.310076e+00	how can we get concentration onsomething

	188628	1340574	291774	2489233_1	997	5.957036e+00
	188629	1340574	259693	3073253_4	998	5.955254e+00
	188630	1340574	265175	3610636_18	999	5.953573e+00

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

`br(queryset)`

Datasets offer the topics in the correct format

`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 field-based models such as BM25F

Other arguments:

- controls & properties – 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

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 the pipeline, let's say

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

- apply an evaluation tool, such as the ubiquitous trec_eval tool on RA and R, to obtain effectiveness measures such as MAP or NDCG
- We use pytrec_eval, which is a Python wrapper of trec_eval

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 pipelines 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

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 names

An example Experiment might look like

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

This outputs a dataframe as follows:

	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		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	NaN	NaN	False	NaN	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	0.299991	0.444409	46.0	45.0	0.237305	False	0.993016	20.0	40.0	0.408840	False	1.0			

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

Corrected p-value

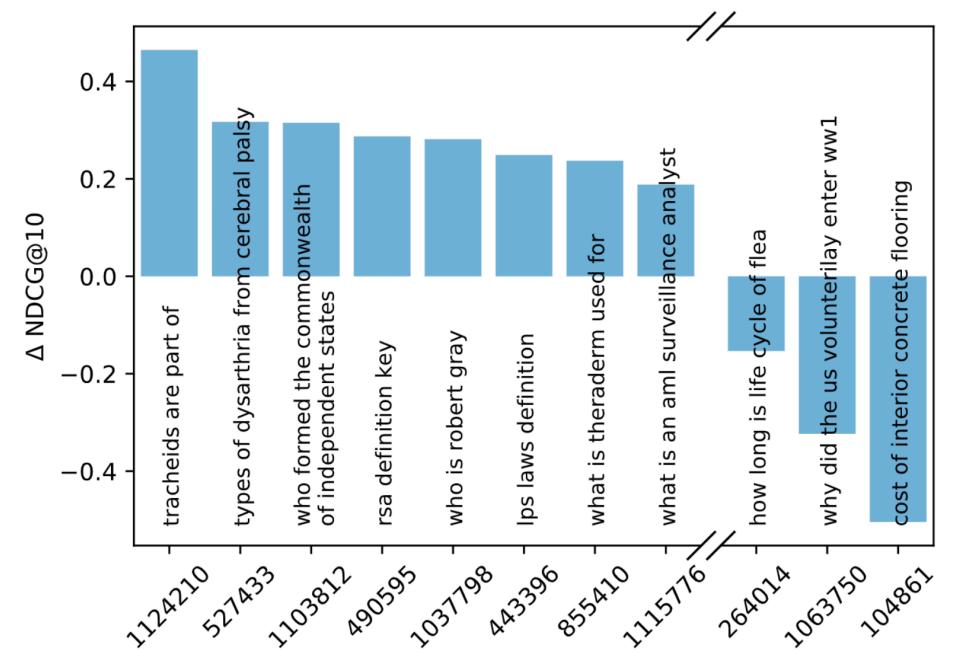
Experiment Variations (4)

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"])
```

Useful for large collections or complex retrieval mechanisms

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

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



UNIVERSITÀ DI PISA



QUESTIONS?

Practical Time



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

- <https://github.com/terrier-org/ecir2021tutorial>
- Press the  Open in Colab link for each notebook to start a Colab session

Timings:

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

If you leave:

- Please complete the feedback quiz (link at Github repo)