



University  
of Glasgow



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

A Search Solutions 2022  
tutorial with PyTerrier

Sean MacAvaney\*  
Craig Macdonald\*  
Nicola Tonello\*

(\*Alphabetical ordering)

# **Tutorial Schedule**



|             |  |
|-------------|--|
| 10:00-11:30 | Part 1 Presentation (a): Indexing, Retrieval, Evaluation |
| 11:00-11:15 | Morning break  |
| 11:15-11:45 | Part 1 Presentation (b): Learning-to-rank                |
| 11:45-12:15 | Part 1 Lab   |
| 12:15-13:00 | Lunch break  |
| 13:00-14:00 | Part 2 Presentation: Neural re-ranking                   |
| 14:00-14:30 | Part 2 Lab   |
| 14:30-14:45 | Afternoon break  |
| 14:45-15:45 | Part 3 Presentation: Learned sparse, Dense retrieval     |
| 15:45-16:15 | Part 3 Lab   |

# *IR is Experiencing a Renaissance*



Deep neural  
network models

For instance, contextual language models 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*

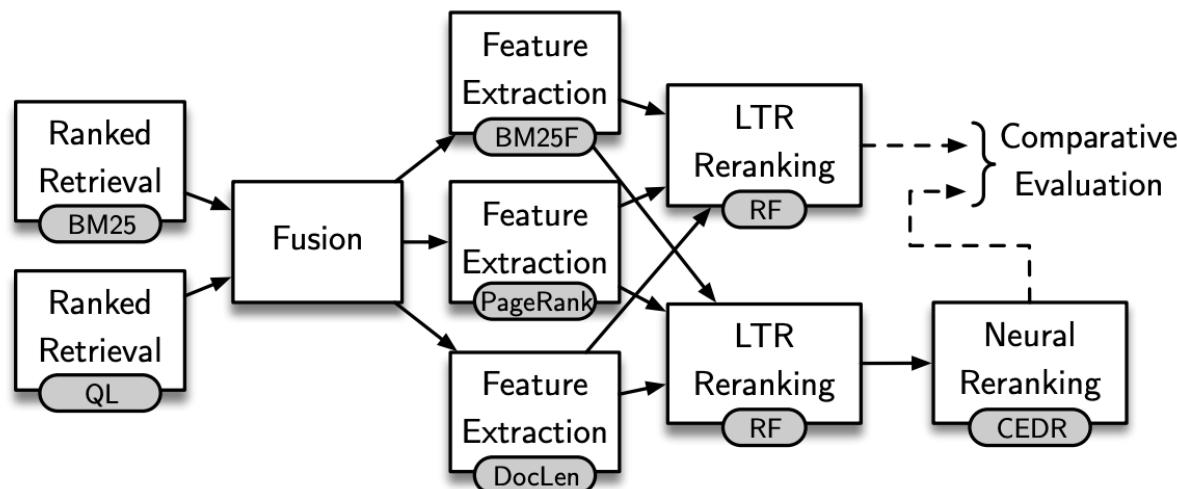
*But why is it that we need new tools for IR? Any why in Python?*

# *So what is required for modern IR platforms*

## Python!

- 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*



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(a) – *Classical IR: Indexing, Retrieval and Evaluation***

– *Modern Retrieval Architectures:*

**PyTerrier data model and operators**

1100-1300

**Part 1(b) Exemplar using learning-to-rank**

*Review & Experience*

1415-1615

**Part 2 – *Contemporary Retrieval Architectures: Neural re-rankers such as BERT, monoT5, ColBERT***

1645-1815

**Part 3 – *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

# *Links*



## **PyTerrier repository**

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

## **PyTerrier documentation**

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

## **Tutorial repository**

- Notebooks
- Slides
- <https://github.com/terrier-org/searchsolutions2022-tutorial>

# *Your Presenters Today*



University  
of Glasgow



*Sean MacAvaney*  
*University of Glasgow*  
*(Previously 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 1 (cont.) – Modern Retrieval Architectures: PyTerrier data model and operators, towards re-rankers and learning-to-rank**

ILO 1D. Understand modern retrieval architectures, such as re-rankers and ranking pipelines incorporating learning-to-rank strategies.

ILO 1E. 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 1F. 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 2 – Contemporary Retrieval Architectures: Neural re-rankers such as BERT, monoT5, ColBERT**

ILO 2A. Understand contemporary retrieval architectures, such as using BERT, EPIC, ColBERT, and T5 as neural re-rankers.

ILO 2B. Understand how query time latency can be reduced by pre-computing representations or using neural models to augment an inverted index.

ILO 2C. Perform experiments with BERT, EPIC, T5, DeepCT, and doc2query in a Python notebook.

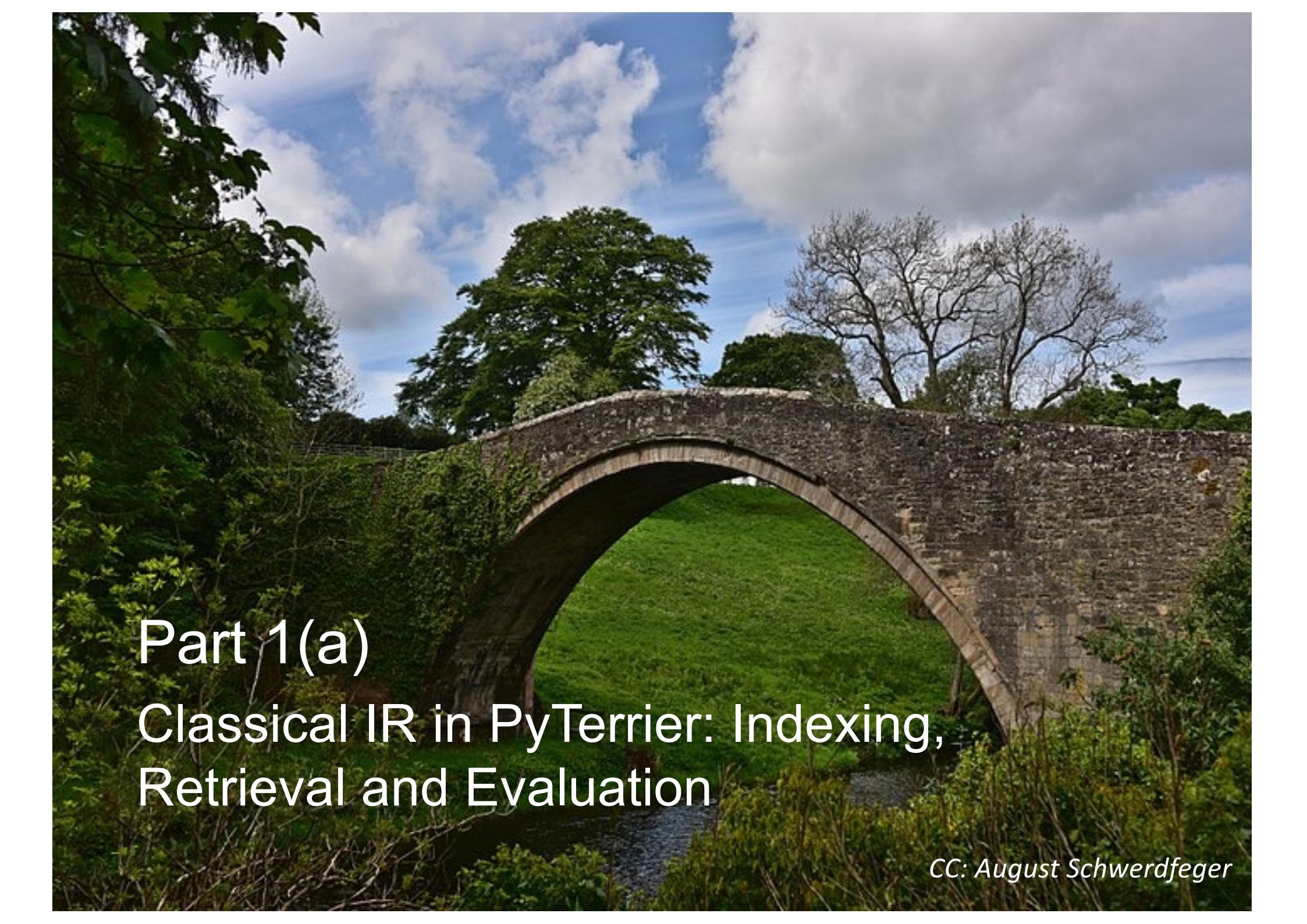
# **Intended Learning Outcomes (4/4)**



## **Part 3 – Recent Advances beyond the classical inverted index: learned sparse retrieval & dense retrieval**

ILO 3A. 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 3B. Experience DeeplImpact sparse retrieval approaches, as well as ANCE and ColBERT dense retrieval approaches

A photograph of a large, ancient stone arch bridge made of dark grey stones. The bridge spans a green grassy hillside. In the background, there are several trees, including a large green tree on the left and two bare trees on the right. The sky is blue with scattered white clouds.

# Part 1(a)

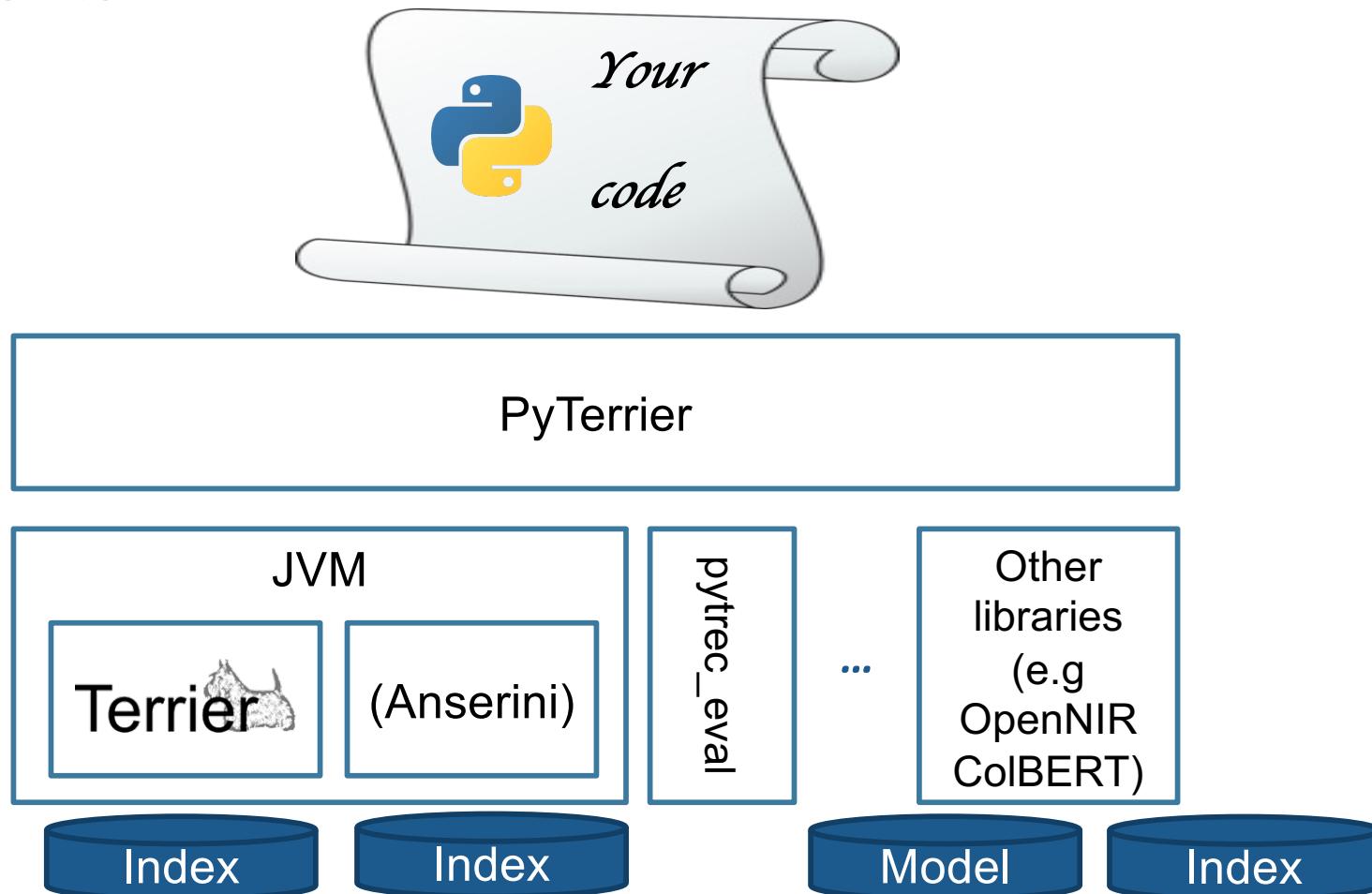
## Classical IR in PyTerrier: Indexing, Retrieval and Evaluation

*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 1b]**

Easy integration for neural re-rankers **[part 2]**

## 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**



***“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

Lexicon

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

DocumentIndex

| id | len | p |
|----|-----|---|
|----|-----|---|

MetaIndex

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

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

PostingIndex (inverted)

|    |    |    |    |    |    |
|----|----|----|----|----|----|
| id | tf | id | tf | id | tf |
|----|----|----|----|----|----|

*One posting list for each term: each posting represents an occurrence*

PostingIndex (direct)

|    |    |    |    |    |    |
|----|----|----|----|----|----|
| id | tf | id | tf | id | tf |
|----|----|----|----|----|----|

*One posting list for each document: each posting represents an occurrence*

# 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  
*(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 |
|--|-------|---------|-------|------------|----------|
| <a href="#">antique</a>                      | ✓     |         |       |            |          |
| <a href="#">antique/test</a>                 | ✓     | ✓       | ✓     |            |          |
| <a href="#">antique/test/non-offensive</a>   | ✓     | ✓       | ✓     |            |          |
| <a href="#">antique/train</a>                | ✓     | ✓       | ✓     |            |          |
| <a href="#">antique/train/split200-train</a> | ✓     | ✓       | ✓     |            |          |
| <a href="#">antique/train/split200-valid</a> | ✓     | ✓       | ✓     |            |          |
| <br><a href="#">aquaint</a>                  | <br>⚠ |         |       |            |          |
| <a href="#">aquaint/trec-robust-2005</a>     | ⚠     | ✓       | ✓     |            |          |

# 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 23<sup>rd</sup> 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 ... |

```
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 the 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 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<br>+ | map<br>- | map p-<br>value | map<br>reject | map p-value<br>corrected | ndcg_cut_10<br>+ | ndcg_cut_10<br>- | ndcg_cut_10<br>p-value | ndcg_cut_10<br>reject | ndcg_cut_10<br>p-value<br>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    | 0.299991 | 0.444409    | 46.0     | 45.0     | 0.238105        | 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

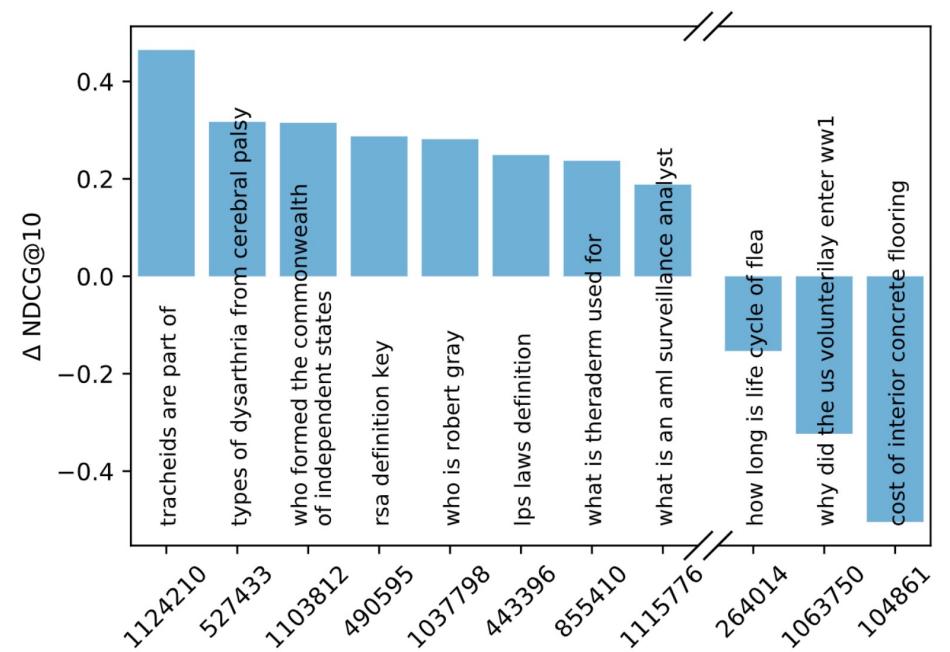
# Experiment Variations (4)

## 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"]))
```



*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.

### 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:

```
#!pip install git+https://github.com/Georgetown-IR-Lab/OpenNIR.git

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-ba
bm25_bert_terrier_stemmed_text = (
    pt.BatchRetrieve.from_dataset('msmarco_document', 'terrier_stemmed_text', wmodel='BM25', metadata=['docn
    >> 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



University  
of Glasgow

## 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

# ***Not just Terrier – Anserini & Pisa too***



UNIVERSITÀ DI PISA



University  
of Glasgow

**In particular, Pisa is a fast in-memory index retrieval backend, with state-of-the-art efficiency (e.g. BlockMaxWAND)**

## **Indexing**

```
# from a dataset
dataset = pt.get_dataset('irds:msmarco-passage')
index = PisaIndex('./msmarco-passage-pisa')
index.index(dataset.get_corpus_iter())
```

## **Retrieval**

```
dph = index.dph()
bm25 = index.bm25(k1=1.2, b=0.4)
pl2 = index.pl2(c=1.0)
qld = index.qld(mu=1000.)
```

**A complex toolkit is made more accessible through PyTerrier**

[https://github.com/terrierteam/pyterrier\\_pisa](https://github.com/terrierteam/pyterrier_pisa)

# **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

CIKM  
2021  
1-5 NOVEMBER

# QUESTIONS?

# Part 1 (cont) -*Modern Retrieval Architectures*



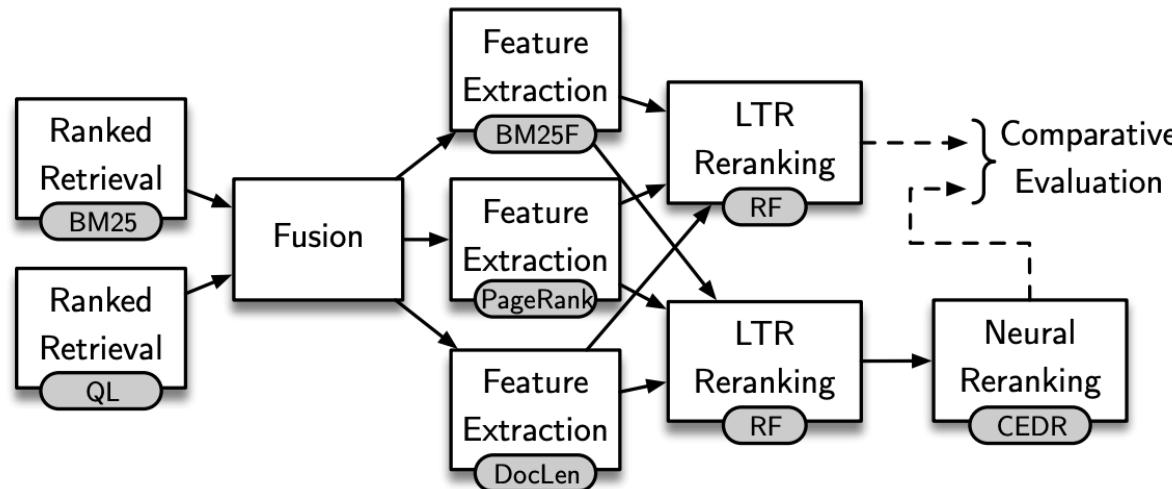
PyTerrier data model and operators, towards re-rankers and learning-to-rank

# Bridging to Modern Retrieval Architectures

In Part 1, we saw how to perform an experiment in a declarative manner, and perform simple retrievals

Actually, in IR, we have many retrieval and re-ranking components

Working with and combining result files to allow comparing complex ranking *pipelines* is tricky:



In Part 2, we generalise the architecture that has been seen into the PyTerrier data model and operators, to make it easy to build complex ranking pipelines

# **Modern Ranking Pipelines**

By using PyTerrier transformers and operators, we show that can easily express complex ranking pipelines in a **conceptual & declarative way**

Furthermore, we review **modern techniques that fit into such ranking pipelines – such as...**

- Query rewriting & pseudo-relevance feedback
- Learning to Rank

**... and demonstrate how they can be implemented and extended within the PyTerrier framework**

**Finally, we also discuss the usefulness of PyTerrier for teaching IR**

## **ILOs for this Session**

ILO 1D. Understand modern retrieval architectures, such as re-rankers and ranking pipelines incorporating learning-to-rank strategies.

ILO 1E. 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 1F. Perform retrieval experiments within a Python notebook, including use of different features and learning-to-rank

# *Outline of Remainder of Morning*



UNIVERSITÀ DI PISA



University  
of Glasgow

**PyTerrier data model & transformers**

**Coffee break**

**Learning to Rank**

**Wrap up and move to practical session**



University  
of Glasgow



UNIVERSITÀ DI PISA

CIKM  
2021  
1-5 NOVEMBER

# PYTERRIER DATA MODEL AND TRANSFORMERS

# **PyTerrier makes it easy to construct complex IR pipelines**

**Key components:**

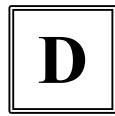
→ **A common data model**

- **A rich library of transformation functions**  
including classical retrieval, dense retrieval, LTR, neural re-ranking
- **Operators to combine transformers**

# Common Data Model



| <b>qid</b> | <b>query</b>                       |
|------------|------------------------------------|
| 0          | gold coast temperature november    |
| 1          | one way flight to gold coast price |
| ...        | ...                                |



| <b>docno</b> | <b>text</b>                    | <b>title</b>       |
|--------------|--------------------------------|--------------------|
| 0            | Science & Mathematics Physi... | The hot glowing... |
| 1            | School-Age Kids Growth &...    | Developmental...   |
| ...          | ...                            | ...                |



| <b>qid</b> | <b>query</b>                    | <b>docno</b> | <b>score</b> |
|------------|---------------------------------|--------------|--------------|
| 0          | gold coast temperature november | 15213        | 0.5134       |
| 0          | gold coast temperature november | 42635        | 0.3742       |
| 0          | gold coast temperature november | 26340        | 0.3223       |
| ...        | ...                             | ...          | ...          |

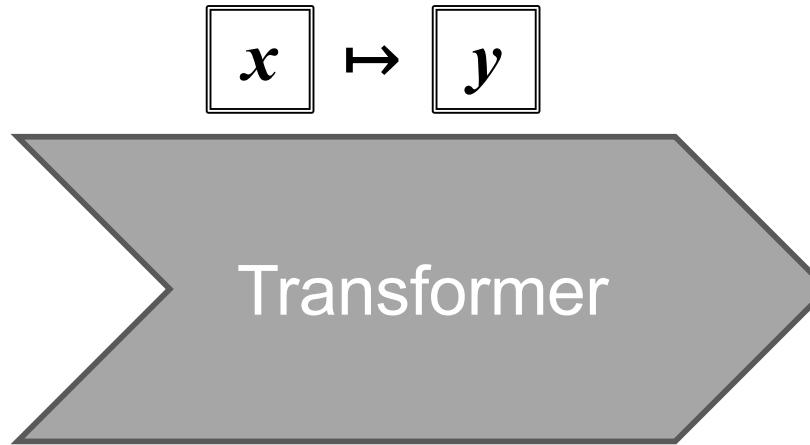


| <b>qid</b> | <b>docno</b> | <b>label</b> |
|------------|--------------|--------------|
| 0          | 15213        | 1            |
| ...        | ...          | ...          |

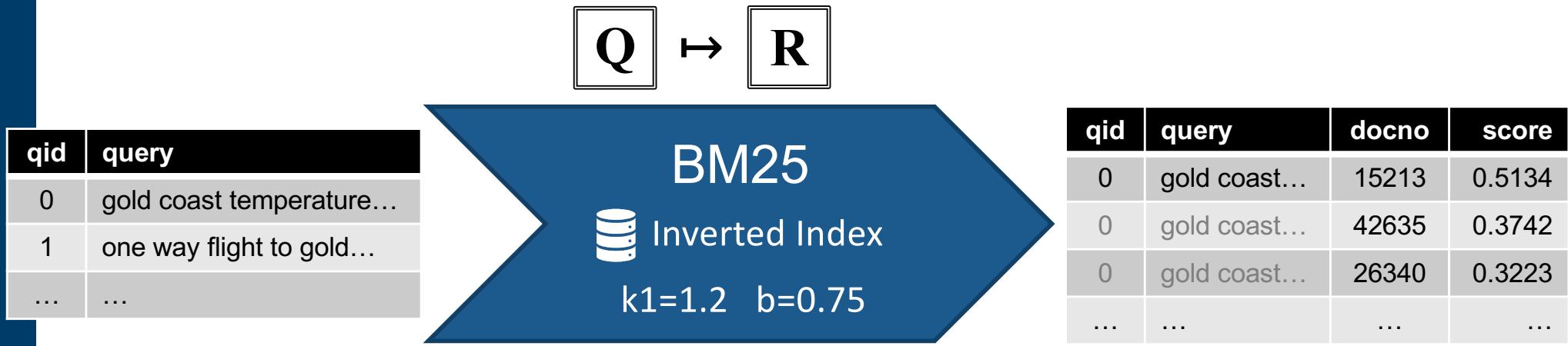
$RA \subset Q \times D$

(All of these are Pandas  
dataframes, i.e. relations)

# Transformation Function Objects (Transformers)



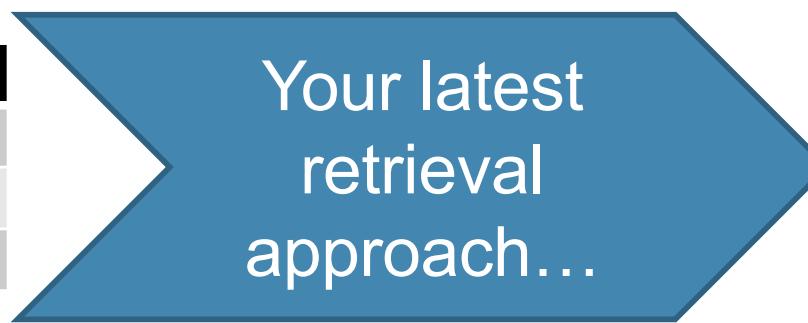
# Transformation Function Objects (Transformers)



# Transformation Function Objects (Transformers)

$$Q \mapsto R$$

| qid | query                     |
|-----|---------------------------|
| 0   | gold coast temperature... |
| 1   | one way flight to gold... |
| ... | ...                       |



| qid | query         | docno | score  |
|-----|---------------|-------|--------|
| 0   | gold coast... | 43242 | 0.1252 |
| 0   | gold coast... | 13746 | 0.1196 |
| 0   | gold coast... | 86454 | 0.0934 |
| ... | ...           | ...   | ...    |

# Transformation Function Objects (Transformers)



UNIVERSITÀ DI PISA



University  
of Glasgow

| Transformer Class           | Examples                              |
|-----------------------------|---------------------------------------|
| $Q \rightarrow R$ Retrieval | BM25, ColBERT, ANCE (dense retrieval) |

# Transformation Function Objects (Transformers)



| Transformer Class                 | Examples                              |
|-----------------------------------|---------------------------------------|
| $Q \rightarrow R$ Retrieval       | BM25, ColBERT, ANCE (dense retrieval) |
| $R \rightarrow Q$ Query Expansion | RM3, Bo1, etc.                        |

# Transformation Function Objects (Transformers)

| Transformer Class                  | Examples                              |
|------------------------------------|---------------------------------------|
| $Q \rightarrow R$ Retrieval        | BM25, ColBERT, ANCE (dense retrieval) |
| $R \rightarrow Q$ Query Expansion  | RM3, Bo1, etc.                        |
| $R \rightarrow R$ Re-ranking       | Vanilla BERT, monoT5, etc.            |
| $Q \rightarrow Q$ Query Re-writing | SDM, IntenT5, etc.                    |

```
[12] index = pt.IndexRef.of('./cord19-index')
    bm25 = pt.TerrierRetrieve(index, wmodel='BM25')

    bm25(topics)
```

|       | <b>qid</b> | <b>docid</b> | <b>docno</b> | <b>rank</b> | <b>score</b> | <b>query</b>                                      |
|-------|------------|--------------|--------------|-------------|--------------|---|
| 0     | 1          | 83032        | dv9m19yk     | 0           | 11.865104    | what is the origin of covid 19                    |
| 1     | 1          | 109719       | kgifmjvb     | 1           | 11.412908    | what is the origin of covid 19                    |
| 2     | 1          | 135553       | wmfcey6f     | 2           | 11.397281    | what is the origin of covid 19                    |
| 3     | 1          | 68472        | 4dtk1kyh     | 3           | 10.805808    | what is the origin of covid 19                    |
| 4     | 1          | 114244       | cniyembt     | 4           | 10.763978    | what is the origin of covid 19                    |
| ...   | ...        | ...          | ...          | ...         | ...          | ...   |
| 49995 | 50         | 146581       | 2o8u4eny     | 995         | 17.073736    | what is known about an mrna vaccine for the sa... |
| 49996 | 50         | 76091        | fayr38c7     | 996         | 17.071367    | what is known about an mrna vaccine for the sa... |
| 49997 | 50         | 102986       | jlrk8fg0     | 997         | 17.070633    | what is known about an mrna vaccine for the sa... |
| 49998 | 50         | 183502       | imwa033f     | 998         | 17.070633    | what is known about an mrna vaccine for the sa... |
| 49999 | 50         | 95511        | j5j3hxxw     | 999         | 17.070486    | what is known about an mrna vaccine for the sa... |

## BM25 Results

[22] rm3 = pt.rewrite.RM3(index)  
 rm3(bm25(topics))

|   | qid | query   | query_0   |
|---|-----|---|---|
| 0 | 1   | 19^0.200000018 origin^0.325867474 covid^0.2000... | what is the origin of covid 19                    |
| 1 | 10  | 19^0.085714296 across^0.019867204 epidem^0.029... | has social distancing had an impact on slowing... |
| 2 | 11  | walk^0.030252097 urolog^0.042577028 optim^0.04... | what are the guidelines for triaging patients ... |
| 3 | 12  | older^0.041744966 best^0.100000009 quarantin^0... | what are best practices in hospitals and at ho... |
| 4 | 13  | contact^0.026820807 possibl^0.021502888 outbre... | what are the transmission routes of coronavirus   |
| 5 | 14  | centuri^0.020000001 manag^0.020000001 contain^... | what evidence is there related to covid 19 sup... |
| 6 | 15  | bodi^0.167176828 can^0.120000005 cold^0.034101... | how long can the coronavirus live outside the ... |
| 7 | 16  | stabl^0.152924821 dai^0.042988449 surfac^0.218... | how long does coronavirus remain stable on sur... |

**Q**

Expanded query

Original query

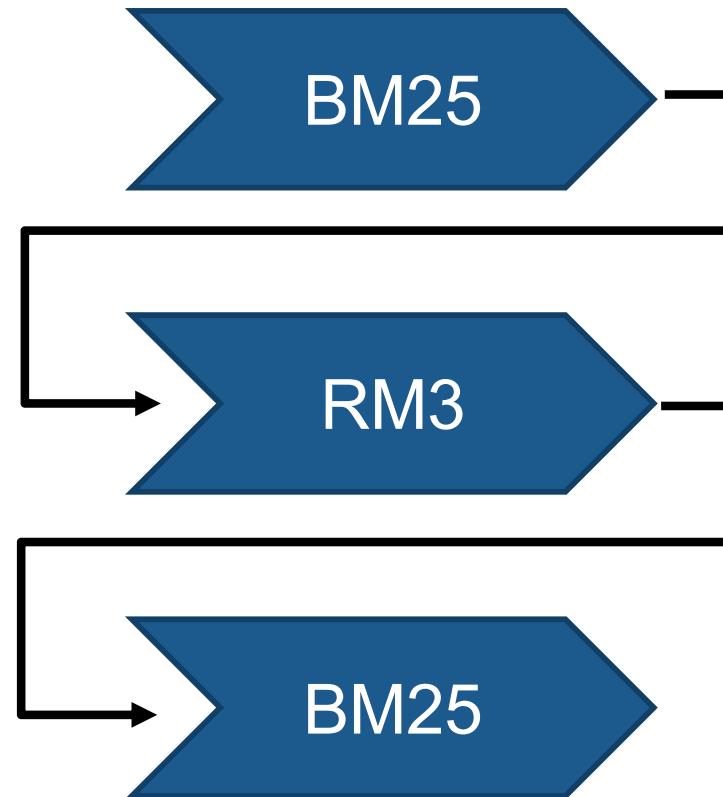
[23] `bm25(rm3(bm25(topics)))`

|       | <b>qid</b> | <b>docid</b> | <b>docno</b> | <b>rank</b> | <b>score</b> |   | <b>query</b>                     | <b>query_0</b> |
|-------|------------|--------------|--------------|-------------|--------------|---|----------------------------------|----------------|
| 0     | 1          | 109719       | kgifmjvb     | 0           | 19.316403    | 19^0.200000018 origin^0.325867474 covid^0.2000... | what is the origin of covid 19   |                |
| 1     | 1          | 135553       | wmfcey6f     | 1           | 19.281917    | 19^0.200000018 origin^0.325867474 covid^0.2000... | what is the origin of covid 19   |                |
| 2     | 1          | 83032        | dv9m19yk     | 2           | 11.101417    | 19^0.200000018 origin^0.325867474 covid^0.2000... | what is the origin of covid 19   |                |
| 3     | 1          | 103317       | ec8lpgl3     | 3           | 11.044733    | 19^0.200000018 origin^0.325867474 covid^0.2000... | what is the origin of covid 19   |                |
| 4     | 1          | 114244       | cniyembt     | 4           | 10.865901    | 19^0.200000018 origin^0.325867474 covid^0.2000... | what is the origin of covid 19   |                |
| ...   | ...        | ...          | ...          | ...         | ...          | ...   | ...                              | ...            |
| 49995 | 9          | 164232       | 6djfpvz7     | 995         | 6.987903     | 19^0.150000006 covid^0.150000006 mp^0.05102380... | how has covid 19 affected canada |                |
| 49996 | 9          | 68116        | 6k8vedy3     | 996         | 6.983706     | 19^0.150000006 covid^0.150000006 mp^0.05102380... | how has covid 19 affected canada |                |
| 49997 | 9          | 132455       | 9z0azq8y     | 997         | 6.983706     | 19^0.150000006 covid^0.150000006 mp^0.05102380... | how has covid 19 affected canada |                |
| 49998 | 9          | 132456       | wfss2j7v     | 998         | 6.983706     | 19^0.150000006 covid^0.150000006 mp^0.05102380... | how has covid 19 affected canada |                |
| 49999 | 9          | 163423       | n76892ur     | 999         | 6.981468     | 19^0.150000006 covid^0.150000006 mp^0.05102380... | how has covid 19 affected canada |                |

## Results of BM25 over RM3

# Operators

bm25 (rm3 (bm25 (topics) ) )

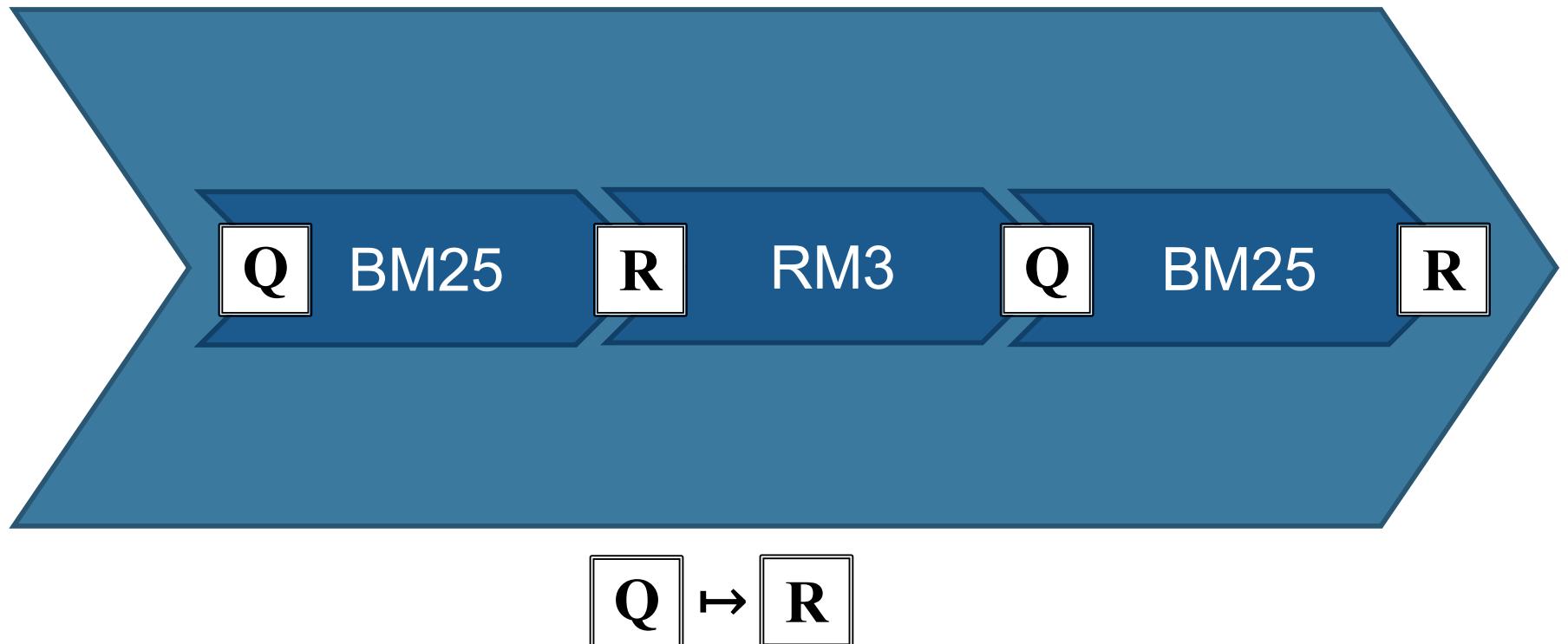


# Operators

A better way: the “then” operator

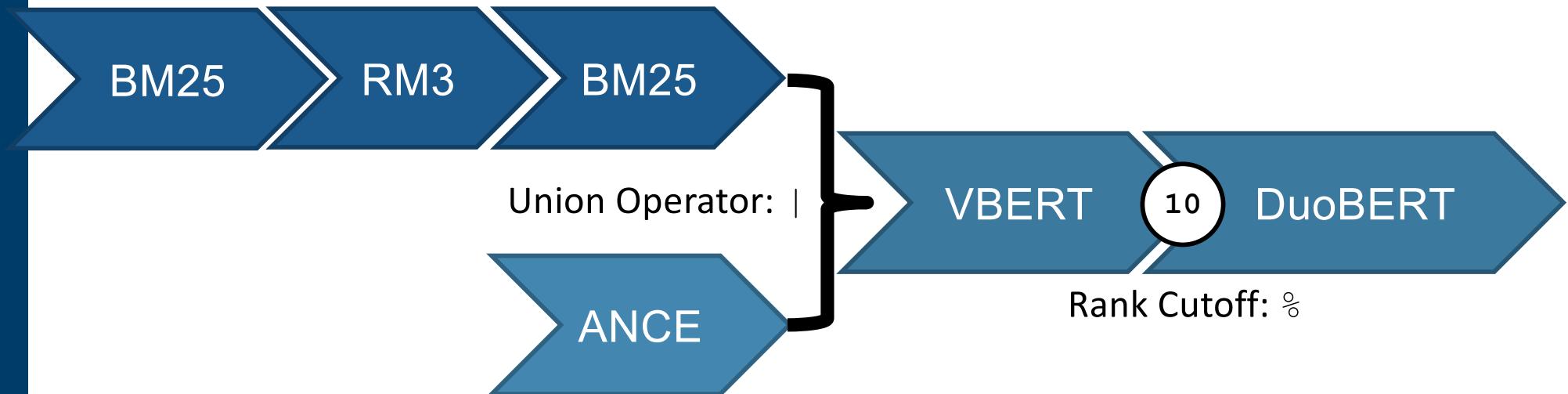
A >> B (A then B)

pipeline = bm25 >> rm3 >> bm25



# Operators

```
pipeline = bm25 >> rm3 >> bm25
```



Other operators available, including caching, linear score combination, etc.

# Transformer Operators

## PyTerrier defines a suite of operators on transformers

| Op. | Name                                | Description  |
|-----|-------------------------------------|--|
| >>  | <i>then</i><br><i>(composition)</i> | Pass the output from one transformer to the next transformer                   |
| +   | <i>linear combine</i>               | Sum the query-document scores of the two retrieved results lists               |
| *   | <i>scalar product</i>               | Multiply the query-document scores of a retrieved results list by a scalar     |
| **  | <i>feature union</i>                | Combine two retrieved results lists as features                                |
|     | <i>set union</i>                    | Make the set union of documents from the two retrieved results lists           |
| &   | <i>set intersection</i>             | Make the set intersection of the two retrieved results lists                   |
| %   | <i>rank cutoff</i>                  | Shorten a retrieved results list to the first $K$ elements                     |
| ^   | <i>concatenate</i>                  | Add the retrieved results list from one transformer to the bottom of the other |

Our ICTIR 2020 paper on PyTerrier provides relational algebra semantics for each operator

- Let's look at a few others...

# Lets look at some retrieval use cases

## 1. Linear Combinations

```
linear = 2 * pt.BatchRetrieve(index, wmodel="BM25")  
+ 3 * pt.BatchRetrieve(index, wmodel="DPH")
```

Final Output Type: R

Output Types: R

## 2. Sequential Dependence Model (proximity)

```
sdm = pt.rewrite.SequentialDependence() >> bm25
```

Input Type: Q

| qid | query                |
|-----|----------------------|
| 0   | 1 chemical reactions |

Output Type: Q

Output Type: R

| qid | query  |
|-----|--|
| 0   | 1 #combine:0=0.1:wmodel=org.terrier.matching.models.dependence.pBiL(#1(chemical reactions))<br>#combine:0=0.1:wmodel=org.terrier.matching.models.dependence.pBiL(#uw8(chemical reactions))<br>#combine:0=0.1:wmodel=org.terrier.matching.models.dependence.pBiL(#uw12(chemical reactions)) |

# Lets look at some retrieval use cases

## 1. Linear Combinations

```
linear = 2 * pt.BatchRetrieve(index, wmodel="BM25")  
+ 3 * pt.BatchRetrieve(index, wmodel="DPH")
```

Final Output Type: R

Output Types: R

## 2. Sequential Dependence Model (proximity)

```
sdm = pt.rewrite.SequentialDependence() >> bm25
```

Input Type: Q

Output Type: Q

Output Type: R

## 3. Pseudo-relevance feedback (QE)

```
rm3 = bm25 >> pt.rewrite.RM3(index) >> bm25
```

Output Type: R

Output Type: Q

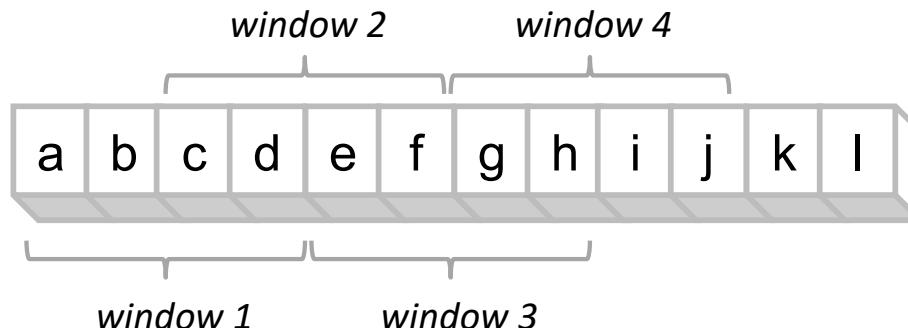
Output Type: R

# **Retrieval Use Case: Passaging**

**Some long documents may only contain a relevant passage about the query**

- In such cases, **passage-based retrieval** can benefit effectiveness
- It's also useful for neural models that do not scale efficiently to long documents

**For instance, applying a **sliding window** over the text of the document to get passages, and scoring each passage individually**



Window size 4  
Window stride 2

- Some settings benefit from prepending the title to each passage

# Retrieval Use Case: Passaging

Lets try this out:

```
Bm25maxp = pt.BatchRetrieve(index, wmodel="BM25", metadata=["docno", "text"])
>> pt.text.sliding("text", length=150, stride=75)
>> pt.text.scorer(wmodel="BM25")
>> pt.text.max_passage()
```

*sliding("text", 2, 1)*

*BR output (R w/ text)*

| qid | docno | text    |
|-----|-------|---------|
| q1  | d1    | a b c d |

*output*

| qid | docno | text |
|-----|-------|------|
| q1  | d1%p1 | a b  |
| q1  | d1%p2 | b c  |
| q1  | d1%p3 | c d  |

*textscorer output (R)*

| qid | docno | rank | score |
|-----|-------|------|-------|
| q1  | d1%p5 | 0    | 5.0   |
| q1  | d2%p4 | 1    | 4.0   |
| q1  | d1%p3 | 2    | 3.0   |
| q1  | d1%p1 | 3    | 1.0   |

*max\_passage output*

| qid | docno | rank | score |
|-----|-------|------|-------|
| q1  | d1    | 0    | 5.0   |
| q1  | d2    | 1    | 4.0   |

*(Some attributes omitted for brevity, e.g. query)*

Of course, we could change `pt.text.scorer()` to be a more advanced approach, e.g. BERT-based neural re-ranker [Part 3]

# *Indexing Pipelines*

## Applying passaging during retrieval can be expensive

- Can it be applied at indexing time?
- There are also other techniques that we will see in Parts 3 & 4 (e.g. DeepCT, doc2query) that can be applied at indexing time

The `>>` operator can also be applied to create **indexing pipelines**

Hence, a MaxPassage retrieval can also be achieved as follows

```
indexer = pt.text.sliding("text", length=150, stride=75)
          >> pt.IterDictIndexer()
indexref = indexer.index(dataset.get_corpus_iter())
MaxP = pt.BatchRetrieve(indexref, wmodel="BM25")
       >> pt.text.max_passage()
```

Note that while transformers operate on a dataframes, we might not fit an entire corpus into a dataframe

- Hence indexing pipelines perform transparent batching of documents

# Transformer Implementations (1/2)



All transformers implement a base class called `pt.Transformer`

- This has all the methods to support operator overloading, e.g. `__add__()`, `__rshift__()`

However, creating your own transformer is easy using `pt.apply` functions. These create a new transformer from your own function

- For instance, to rewrite a query – `pt.apply.query()`

```
# this will remove pre-defined stopwords from the query
stops=set(["and", "the"])

# a naieve function to remove stopwords
def _remove_stops(q):
    terms = q["query"].split(" ")
    terms = [t for t in terms if not t in stops ]
    return " ".join(terms)

# a query rewriting transformer applying _remove_stops
p1 = pt.apply.query(_remove_stops) >> pt.BatchRetrieve(index, wmodel="DPH")
```

- `pt.apply.query()` returns a transformer, which applies `_remove_stops` to each query. The previous query is saved in the `query_0` attribute

# *Transformer Implementations*

## (2/2): *pt.apply.doc\_score()*

**pt.apply.doc\_score() uses a function that returns a score**

```
# this transformer will subtract the number of character in the query  
# from the score of each retrieved document
```

```
p = pt.BatchRetrieve(index, wmodel="DPH") >>  
    pt.apply.doc_score(lambda doc : doc["score"] - len(doc["query"]))
```

**In this way, ANY function that can returns a number can be used for retrieval**

- i.e. any (SOTA) approach that can be expressed as a function of the **text of the document** and the **text of the query** can easily be used as a re-ranker in PyTerrier

**pt.apply.doc\_score() & pt.apply.query() have known semantics**

- There are also advanced “generic” apply functions that can apply an arbitrary change of a dataframe as a transformer

# Worked Example – Reranking using SentenceTransformers

## Let's compare two BERT models for reranking BM25:

```
import pandas as pd
from sentence_transformers import CrossEncoder, SentenceTransformer
crossmodel = CrossEncoder('cross-encoder/ms-marco-MiniLM-L-12-v2', max_length=512)
bimodel = SentenceTransformer('paraphrase-MiniLM-L6-v2')

def _crossencoder_apply(df : pd.DataFrame):
    return crossmodel.predict(list(zip(df['query'].values, df['text'].values)))

cross_encT = pt.apply.doc_score(_crossencoder_apply, batch_size=128)

def _biencoder_apply(df : pd.DataFrame):
    from sentence_transformers.util import cos_sim
    query_embs = bimodel.encode(df['query'].values)
    doc_embs = bimodel.encode(df['text'].values)
    scores = cos_sim(query_embs, doc_embs)
    return scores[0]

bi_encT = pt.apply.doc_score(_biencoder_apply, batch_size=128)

pt.Experiment(
    [ bm25, bm25 >> bi_encT, bm25 >> cross_encT ],
    dataset.get_topics(),
    dataset.get_qrels(),
    ["map"],
    names=["BM25", "BM25 >> BiEncoder", "BM25 >> CrossEncoder"]
)
```

- } Load CE & BE models
- } CE Apply function scores 128 queries and texts at once using CE
- } BE apply function scores 128 queries, and 128 texts, then computes Cos scores
- } Combine each with BM25, and evaluate

# Worked Example – Reranking using SentenceTransformers

Let's compare two BERT models for reranking BM25:

```
import pandas as pd
from sentence_transformers import CrossEncoder, SentenceTransformer
crossmodel = CrossEncoder('cross-encoder/ms-marco-MiniLM-L-12-v2', max_length=512)
bimodel = SentenceTransformer('paraphrase-MiniLM-L6-v2')
```

```
def _crossencoder_apply(df : pd.DataFrame):
    return crossmodel.predict(list(zip(df['query'].values,
```

```
df['text'].values)))
```

|   |  | name                 | RR(rel=2) | nDCG@10  | nDCG@100 | AP(rel=2) |
|---|--|----------------------|-----------|----------|----------|-----------|
| 0 |  | BM25                 | 0.641565  | 0.479540 | 0.487416 | 0.286448  |
| 1 |  | BM25 >> BiEncoder    | 0.793350  | 0.605352 | 0.549940 | 0.362597  |
| 2 |  | BM25 >> CrossEncoder | 0.914729  | 0.738378 | 0.669271 | 0.488546  |

```
dataset.get_qrels(),
["map"],
names=["BM25", "BM25 >> BiEncoder", "BM25 >> CrossEncoder"])
```

# ***Summary: PyTerrier Standard Transformers***

## **Retrieval**

- `pt.BatchRetrieve()`

## **Query Rewriting/Expansion**

- `pt.rewrite.SequentialDependence()`
- `pt.rewrite.RM3()`
- `pt.rewrite.Bo1QueryExpansion()`
- `pt.rewrite.KLQueryExpansion()`

## **Text**

- `pt.text.get_text()`
- `pt.text.scorer()`
- `pt.text.sliding()`
- `pt.text.max_passage()` `first_passage()` `mean_passage()` `kmaxavg_passage()`

In Part 2B, we'll see learning-to-rank transformers

In Parts 3 & 4, we'll see more add-on transformers for neural re-ranking and end-to-end

# ***Summary: PyTerrier Utility Transformers***



UNIVERSITÀ DI PISA



University  
of Glasgow

## **Apply functions as pipelines**

- `pt.apply.doc_score(_fn)`
- `pt.apply.query(_fn)`
- `pt.apply.generic(_fn)`
- `pt.apply.by_query(_fn)`
- `pt.apply.<ANY COLUMN NAME>(_fn)`

User-defined functions to  
change the pipeline in  
arbitrary ways

## **Inspect within a transformer pipeline**

- `pt.debug.print_rows()`
- `pt.debug.print_columns()`
- `pt.debug.print_num_rows()`

# Summary: PyTerrier Transformer Operators



| Op. | Name                                | Description  |
|-----|-------------------------------------|--|
| >>  | <i>then</i><br><i>(composition)</i> | Pass the output from one transformer to the next transformer                   |
| +   | <i>linear combine</i>               | Sum the query-document scores of the two retrieved results lists               |
| *   | <i>scalar product</i>               | Multiply the query-document scores of a retrieved results list by a scalar     |
| **  | <i>feature union</i>                | Combine two retrieved results lists as features <i>(Part 2B)</i>               |
|     | <i>set union</i>                    | Make the set union of documents from the two retrieved results lists           |
| &   | <i>set intersection</i>             | Make the set intersection of the two retrieved results lists                   |
| %   | <i>rank cutoff</i>                  | Shorten a retrieved results list to the first $K$ elements                     |
| ^   | <i>concatenate</i>                  | Add the retrieved results list from one transformer to the bottom of the other |

# Reflections (1)

**Transformers build on PyTerrier's extensible data model**

- defined on IR-level objects: queries and documents
- We can add more data model columns for, e.g., ConvQA

**Operators, defined with clear semantics, allow different transformers to be easily combined**

- The structure of the pipeline implementations are close to their conceptual design
- This is all declarative in nature - we aren't **writing code** that iterates over any queries or documents (no for loops ☺!)

**New techniques can be used and evaluated within a unified framework**

- `pt.apply.doc_score()` can be easily applied to integrate new state-of-the-art approaches

# Reflections (2)

Thinking in this manner allows many techniques to be easily instantiated... and new techniques to be designed.

- We have created pipelines for integrating PRF into dense retrieval
- Both of our CIKM papers making advances on ColBERT are designed and implemented as variants of pipelines

```
import pyterrier_colbert.pruning

#CIKM 2021 Approximate Scoring paper: only retrieve 200 candidates for exact re-ranking
ann_pipe = (factory.ann_retrieve_score() % 200) >> factory.index_scorer(query_encoded=True)

#CIKM 2021 query embeddings paper: only keep the 9 tokens with highest ICF
qep_pipe5 = (factory.query_encoder()
              >> pyterrier_colbert.pruning.query_embedding_pruning(factory, 5)
              >> factory.set_retrieve(query_encoded=True)
              >> factory.index_scorer(query_encoded=False)
)
qep_pipe9 = (factory.query_encoder()
              >> pyterrier_colbert.pruning.query_embedding_pruning(factory, 9)
              >> factory.set_retrieve(query_encoded=True)
              >> factory.index_scorer(query_encoded=False)
)
```

Even if we have a single coherent end-to-end retrieval model, for deployment, we may need things like query-biased snippets, which are pipelined after retrieval...

# **Building your own**



**We have been extending PyTerrier with “plugins” for various models - some you will see in Parts 2 & 3:**

- pyt\_t5, pyt\_colbert, pyt\_ance, pyt\_doc2query, pyt\_splade

**Its easy to make a retrieval function into a PyTerrier transformer:**

- Make a class that extends pt.Transformer
- Define a transform() method that takes as input a dataframe and returns a dataframe
- pt.apply functions can reduce the boilerplate even further.

**Indexers also can be supported**

- Make a class that extends pt.Indexer
- Define an index() method that takes iterable of dicts each containing a document.

**Then its plain sailing!**



University  
of Glasgow



UNIVERSITÀ DI PISA

CIKM  
2021  
1-5 NOVEMBER

11:00-11:15

## COFFEE BREAK



University  
of Glasgow



UNIVERSITÀ DI PISA

CIKM  
2021  
1-5 NOVEMBER

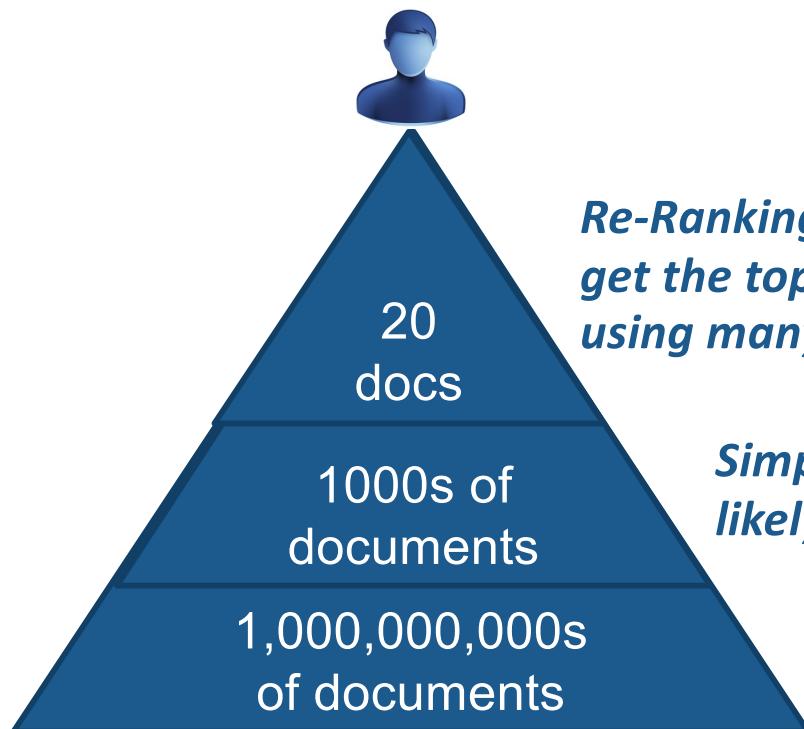
**Part 1B**

# **LEARNING TO RANK**

# Ranking Cascades

In many search scenarios, the ranking process can be seen as a series of cascades [1]

- Rank some documents
- Pass top-ranked onto next cascade for refined re-ranking



*Re-Ranking: Try really hard to get the top of the ranking correct, using many signals (features)*



**LEARNING TO RANK Features & Models**

*Simple Ranking: Identify a set most likely to contain relevant documents*



**e.g. BM25**

*Boolean: Do query terms occur?*



**e.g. AND/OR /phrase**

## How to choose term weighting models?

- Term weighting models have different **assumptions** about how relevant documents should be retrieved

Also:

- **Proximity-models:** close co-occurrences matter more
- **Priors:** documents with particular lengths or URL/inlink distributions matter more
- **Field-based models:** term occurrences in different fields matter differently (e.g. BM25F, PL2F)
- ***Query Features:*** Long queries, difficult queries, query type

How to combine all these easily and appropriately?

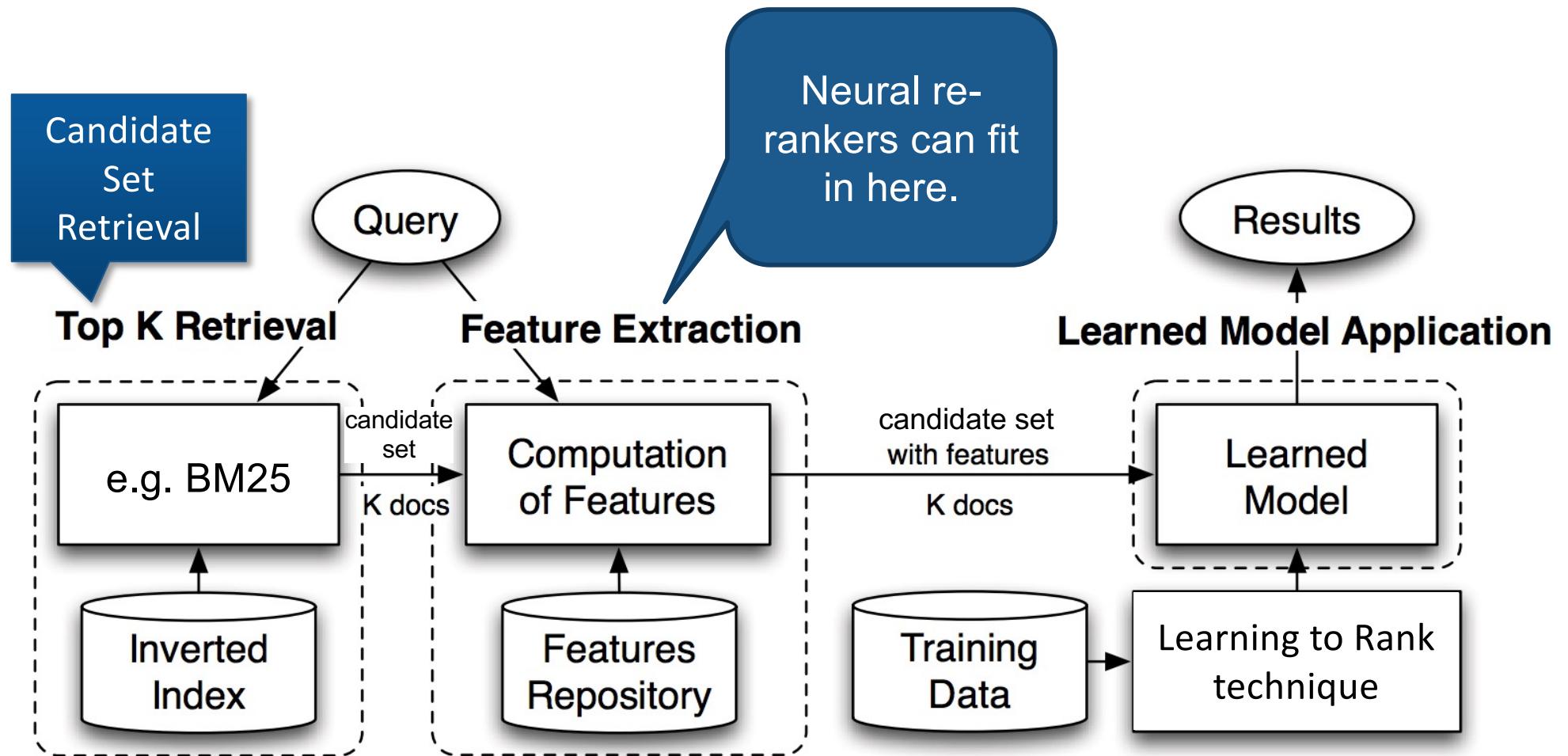
# **Learning to Rank definition**

**Learning to rank describes the application of tailored machine learning techniques, applied to re-rank a candidate set of retrieved documents**

- This requires hand-engineered features (incl. weighting models) as relevance signals
- LTR techniques should focus on getting the top-ranked documents ranked correctly, using adapted loss functions:
  - Pointwise: e.g. regression – just tries to predict document label
  - Pairwise: tries to get pairs of documents with differing label correctly ranked
  - Listwise: tries to get most relevant document at top of ranking

**Different model forms are popular: linear, trees, neural nets**

# LTR, Schematically...

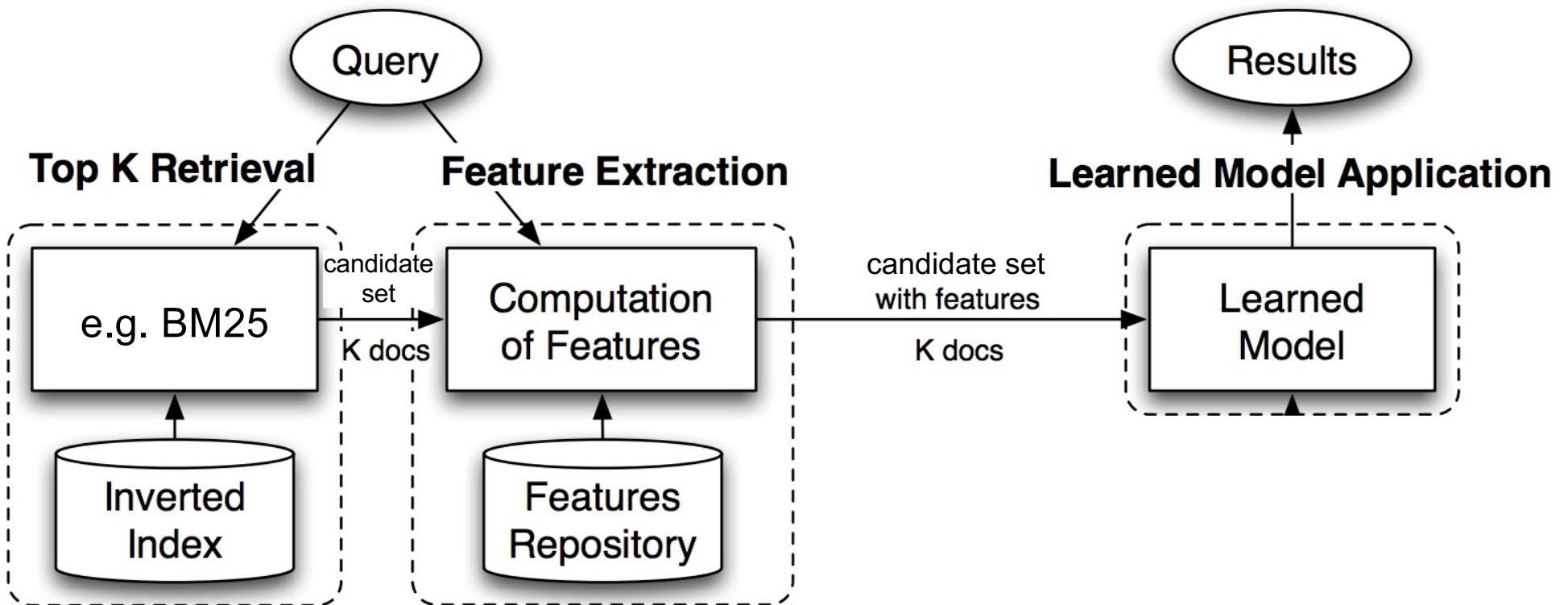


# Types of Features

Typically, commercial search engines use hundreds of features for ranking documents, usually categorised as follows:

| Name                       | Varies depending on... |          | Examples   |
|----------------------------|------------------------|----------|--|
|                            | Query                  | Document |  |
| Query Dependent Features   | ✓                      | ✓        | Weighting models, e.g. BM25, PL2<br>Proximity models, e.g. Markov Random Fields<br>Field-based weighting models, e.g. PL2F<br>Neural re-rankers [Part 3] |
| Query Independent Features | X                      | ✓        | PageRank, number of inlinks<br>Spamminess  |
| Query Features             | ✓                      | X        | Query length, Query performance predictors<br>Presence of entities   |

# Matching LTR concepts to PyTerrier...



In PyTerrier:

BM25

%K

>>

PL2F

\*\*

>>

pt.ltr.  
apply\_learned\_model()

pt.apply.doc\_score()

# ***“Featured” Data Model***

## Retrieved documents with features, $R_f$ :

| <u>qid</u> | query | <u>docno</u> | score | rank | <i>features</i> |
|------------|-------|--------------|-------|------|-----------------|
|            |       |              |       |      |                 |
|            |       |              |       |      |                 |

- The features column contains, for each document, an array of additional feature scores.
- These can be used for learning and applying LTR models
- Applying an LTR model would override the original score of the document, causing a re-ranking

# Feature Union Operator

## Feature Union (\*\*)

- Say we want to take multiple retrieval approaches as different features. We use  $**$  to denote, that given a set of input documents, use these transformers to obtain additional feature scores

- Formally:  $Rf = (T_1 \ ** \ T_2)(R) :=$

$$R_1 = T_1(R), \quad R_2 = T_2(R)$$

$$Rf = (R_1 \bowtie R_2)[[f_1, f_2] \rightarrow f]$$

- E.g.

BatchRetrieve("BM25") >>

( BatchRetrieve("Tf") \*\* BatchRetrieve("PL2") )

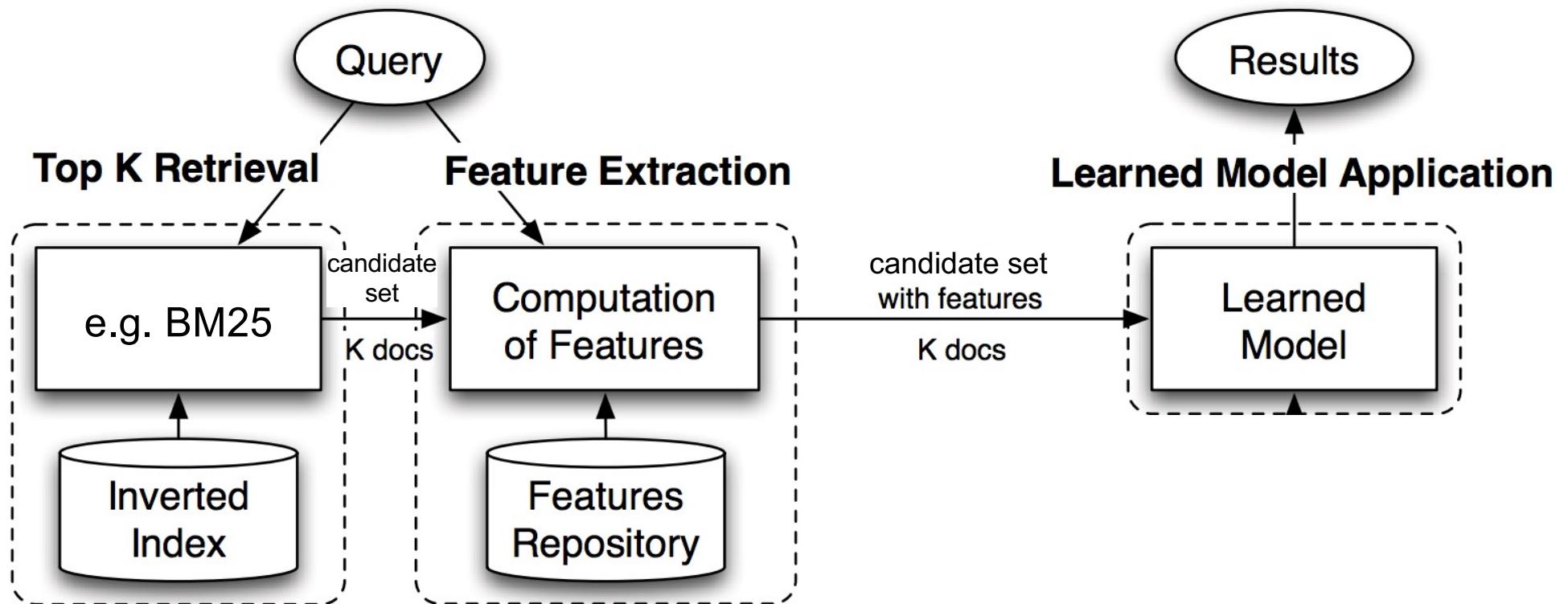
$R$

|   | qid | docno | score        |
|---|-----|-------|--------------|
| 1 | q1  | d5    | (bm25 score) |

$R_f$

|   | qid | docno | score        | features              |
|---|-----|-------|--------------|-----------------------|
| 1 | q1  | d5    | (bm25 score) | [tf score, pl2 score] |

# Matching LTR concepts to PyTerrier...



In PyTerrier:

BM25

%K

>>

PL2F

\*\*

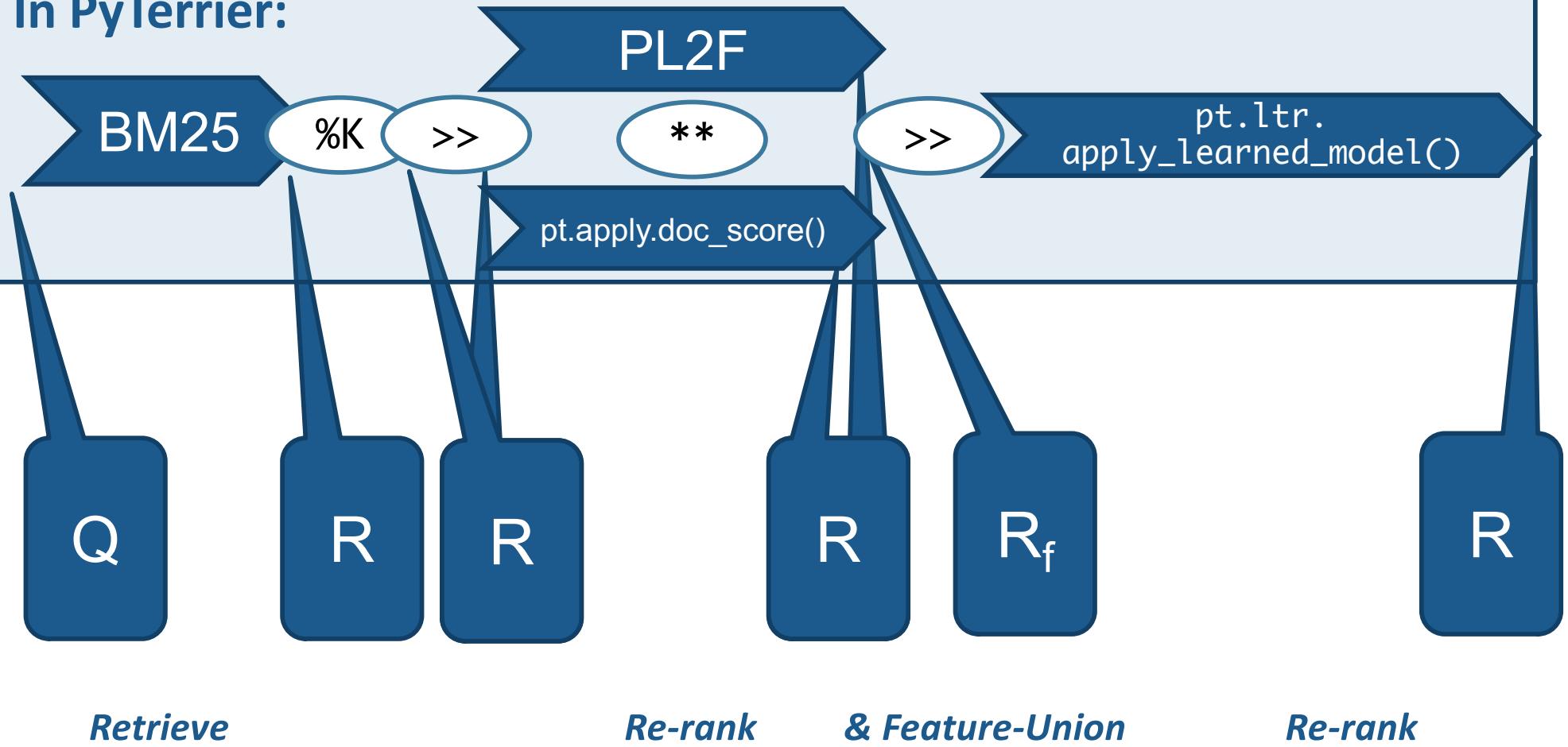
>>

pt.ltr.  
apply\_learned\_model()

pt.apply.doc\_score()

# Data Model Transformations

In PyTerrier:



# Query-Dependent Feature Extraction



BatchRetrieve("BM25") >> ( BatchRetrieve("Tf") \*\* BatchRetrieve("PL2") )

## Here we are using BatchRetrieve as a **re-ranker**

- Feature union is only defined on an input set of documents
- RHS BatchRetrieves will re-score the BM25 retrieved documents using the other weighting models

## We might deploy a number of query dependent features

- Such as additional weighting models, e.g. fields/proximity, that are calculated based on information in the inverted index
- Each BatchRetrieve causes another access to the inverted index posting lists

## FeatureBatchRetrieve is an alternative for Terrier

```
pt.FeatureBatchRetrieve(index, wmodel="BM25",
                        features=[“WMODEL:Tf”, “WMODEL:PL2”])
```

- Multiple features scored in a single pass of the inverted index
- PyTerrier can perform this optimisation when .compile() is called



University  
of Glasgow



UNIVERSITÀ DI PISA

CIKM  
2021  
1-5 NOVEMBER

# TYPES OF MODELS FOR LTR

# **Types of Learned Models (1)**

## **Linear Model**

### **Linear Models (the most intuitive to comprehend)**

- Many learning to rank techniques generate a linear combination of feature values:

$$score(d, Q) = \sum_f w_f \cdot value_f(d)$$

- Linear Models make some assumptions:
  - **Feature Usage**: They assume that the same features are needed by all queries
  - **Model Form**: The model is only a linear combination of feature values.
    - Contrast this with genetic algorithms, which can learn functional combinations of features, by randomly introducing operators (e.g. try divide feature  $a$  by feature  $b$ ), but are unpractical to learn

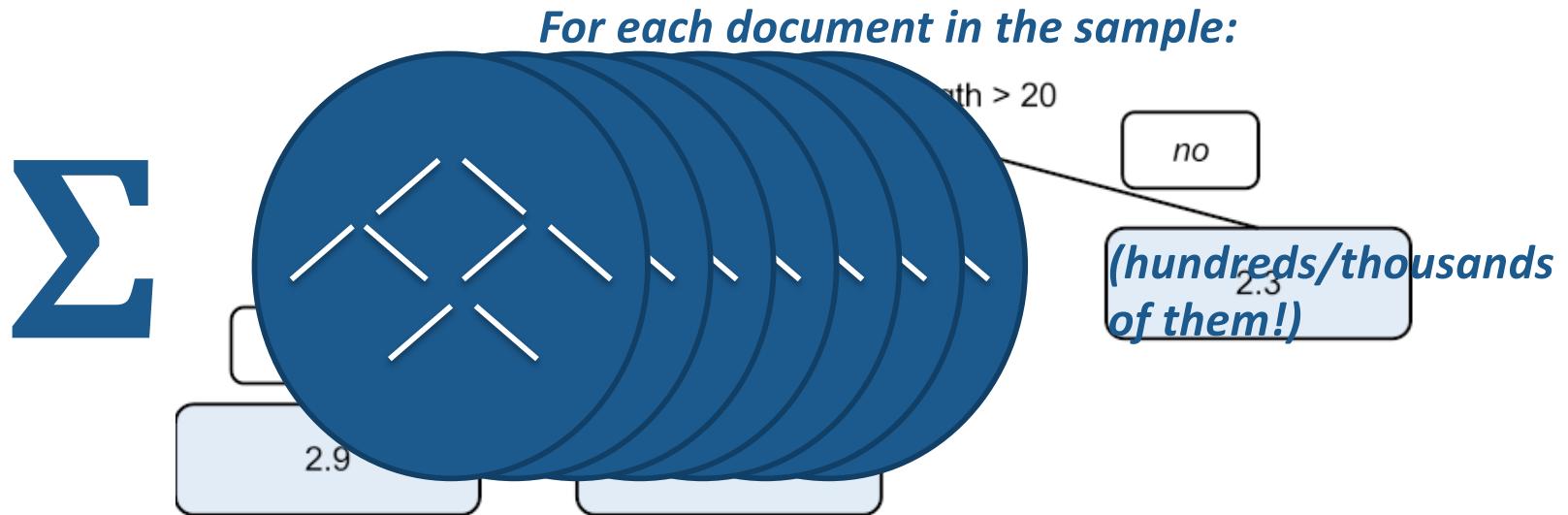
**It is difficult to find  $w_f$  values that maximise the performance of an IR evaluation metric, as they are non-smooth and non-differentiable**

- Typically, techniques such as simulated annealing or stochastic gradient descent are used to empirically obtain  $w_f$  values

# Type of Learned Models (2)

## Regression Trees

- A *regression tree* is series of decisions, leading to a partial score output
- The outcome of the learner is a “forest” of many such trees, used to calculate the final score of a document for a query
- Their ability to customise branches makes them more effective than linear models
- Regression trees are **pointwise** in nature, but several major search engines have created adapted regression tree techniques that are **listwise**
  - E.g. Microsoft’s **LambdaMART** is at the heart of the Bing search engine

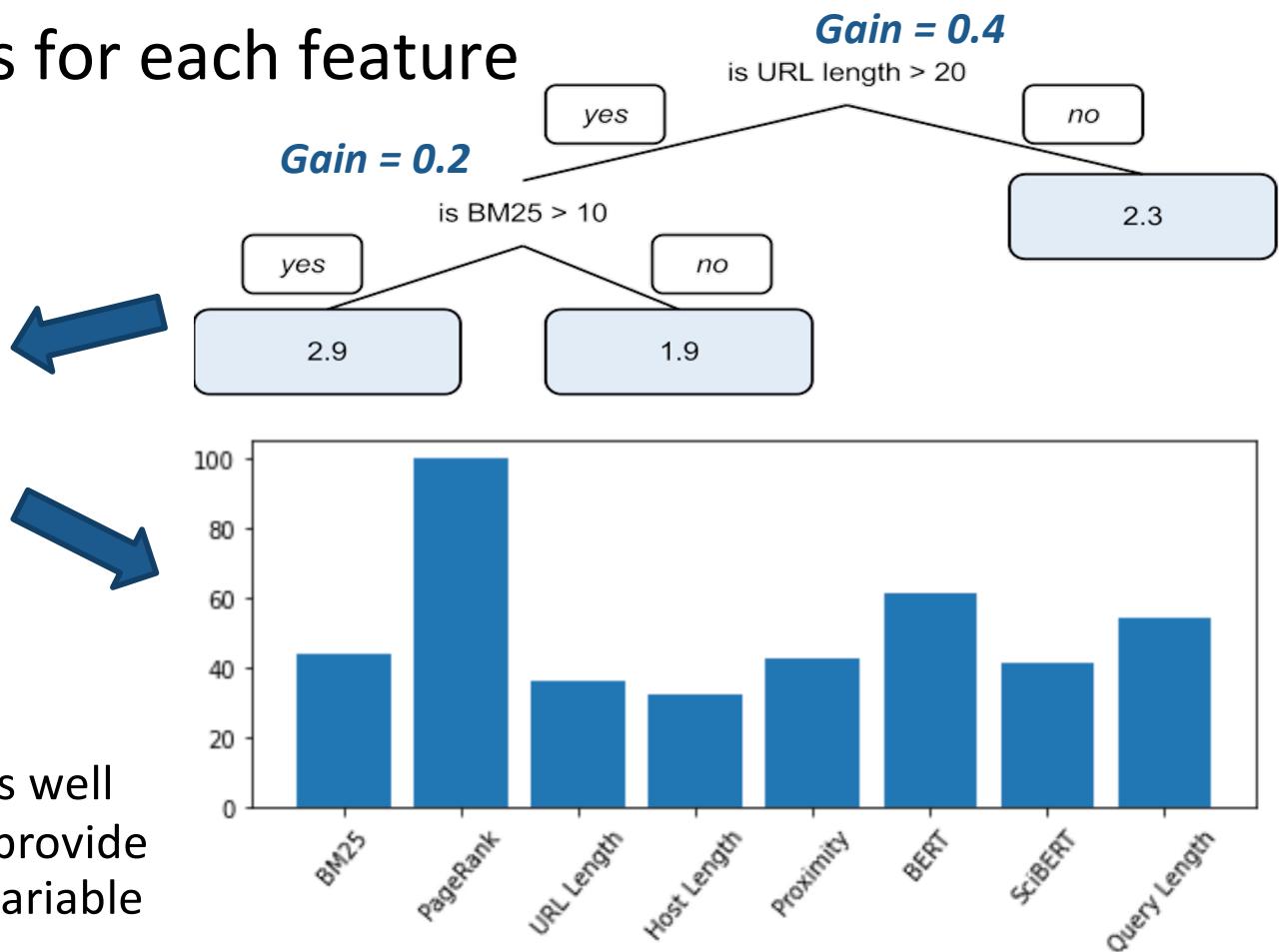


# Feature Importance

## Regression Trees have an in-built measure of feature importance

- By recording the “gain” at each node during training time
- Summing the gains for each feature

| Feature    | Norm.<br>Total<br>Gain |
|------------|------------------------|
| BM25       | 45                     |
| PageRank   | 100                    |
| URL Length | 39                     |
| ...        | ...                    |



NB: sklearn's RandomForest, as well as XGBoost and LightGBM, all provide a `feature_importances_` variable

# **Learning and Applying a Learned Model**

## **PyTerrier has transformers that accept standard learners**

- Accessed via `pt.ltr.apply_learned_model()`
- E.g. sci-kit learn has a random forest (pointwise) learner

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators=400)
rf_pipe = pipeline >> pt.ltr.apply_learned_model(rf)
rf_pipe.fit(train_topics, qrels)
```
- Also for XGBoost and LightGBM, which have LambdaMART (listwise) implementations

```
import xgboost as xgb
lmart = xgb.sklearn.XGBRanker(objective='rank:ndcg')
lmart_pipe = pipeline >> pt.ltr.apply_learned_model(lmart_x, form="ltr")
lmart_pipe.fit(train_topics, train_qrels,
               validation_topics, validation_qrels)
```

- And for FastRank, which has coordinate ascent (linear, listwise) and random forest (pointwise) learners

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



# The Importance of Candidate Set Size

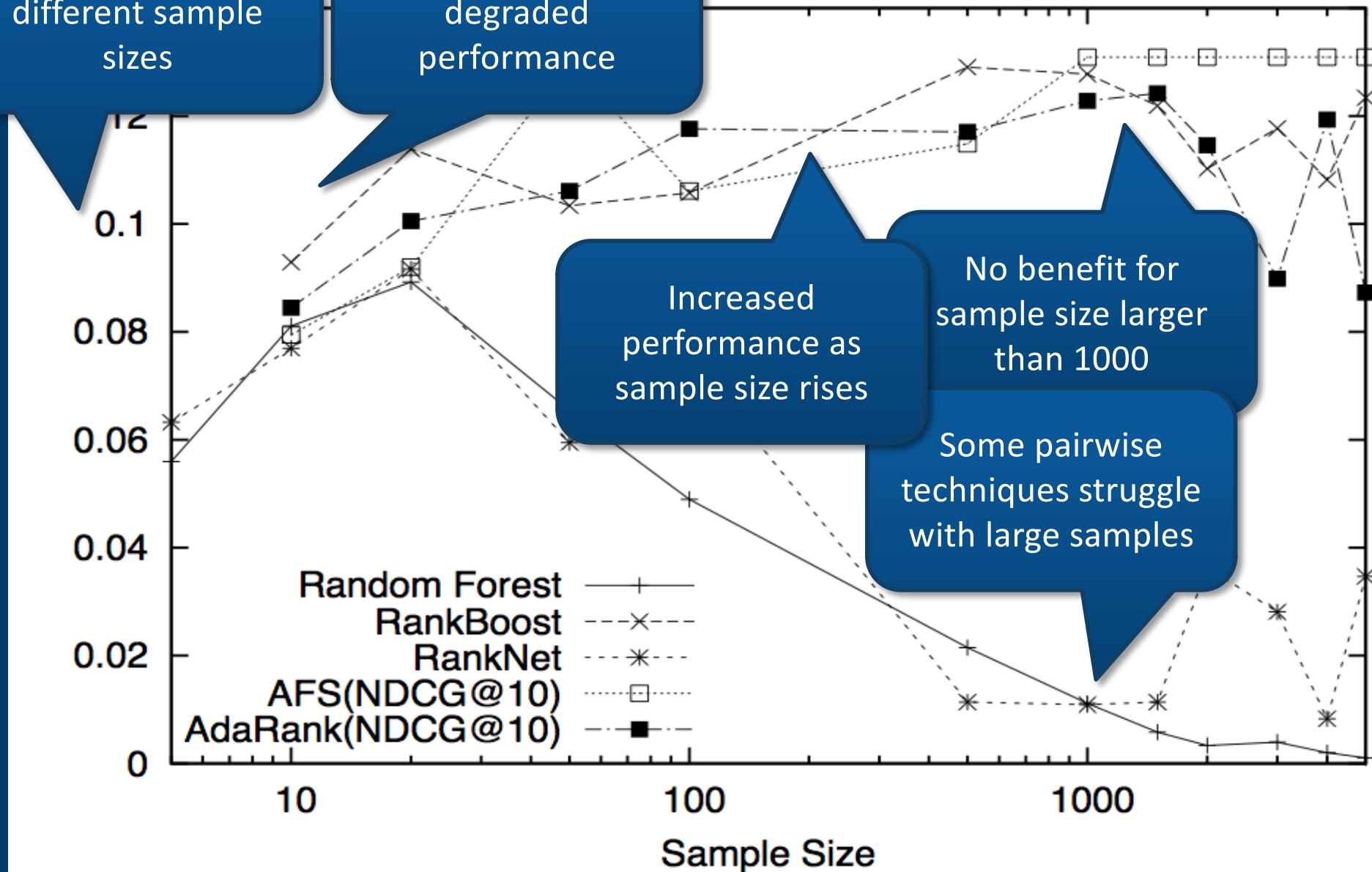
NDCG@20 for different sample sizes

Small sample size, degraded performance

Increased performance as sample size rises

No benefit for sample size larger than 1000

Some pairwise techniques struggle with large samples



# *Analysing a Model*



## Many tree-based learners provide a feature\_importances\_ variable

- This allows the researcher to see the importance of various features during *training*
- (How much each feature was used to reduce the loss during tree construction)

Alternatively, utility on the *test* set can be measured by ranking by each feature alone, or ablating (removing) a feature before learning

- E.g. `pt.ltr.ablate_features(int)` is a transformer that zeros out one or more features.
- This is useful when your pipeline is already defined.

```
# learn a model for 3 features, removing one each time
for fid in range(numf):
    ablated = pipeline >> pt.ltr.ablate_features(fid) >> pt.ltr.apply_learned_model(RandomForestRegressor(n_estimators=400))
    ablated.fit(trainTopics, trainQrels, validTopics, validQrels)
    rankers.append(full)
```

- You will see another analysis in the Part 2 notebook

# **PyTerrier Learning to Rank Summary**



UNIVERSITÀ DI PISA



University  
of Glasgow

**Key Operators:** `>>`, `**`

**Key Transformers:**

- Retrieval: `pt.BatchRetrieve`
- Features: `pt.apply.doc_score()`, `pt.FeaturesBatchRetrieve`, `pt.BatchRetrieve`
- Learning to Rank: `pt.ltr.apply_learned_model()`
- Analyse: `pt.ltr.ablate_features()`, `pt.ltr.feature_to_score()`

**LTR Integrations:**

- sklearn: Regression (esp. Random Forests)
- LightGBM: Regression trees/LambdaMART
- xgBoost: Regression trees/LambdaMART
- FastRank: Coordinate Ascent (linear), Random Forests

# **LTR Summary**

**Learning to rank is still an important method of integrating many features within a machine-learned search architecture**

- Neural re-ranking models can also fit in this architecture
- Indeed, we found them to be the most important feature for LTR models learned on MSMARCO

**PyTerrier aims to make it as easy as possible to express new features, and analyse the generated models**

- We include integrations of various LTR libraries
- Tensorflow Ranking is one notable possible future integration, which blurs the lines between LTR and neural re-ranking

# References

- [Burges2010] Christopher J.C. Burges. 2010. From RankNet to LambdaRank to LambdaMART: An Overview. Technical Report MSR-TR-2010-82.
- [Chen2016] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In Proceedings of SIGKDD.
- [Foley2019] FastRank alpha, John Foley <https://jjfoley.me/2019/10/11/fastrank-alpha.html>
- [Ke2017] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. Proceedings of NeurIPS.
- [Liu2009] Tie-Yan Liu (2009), "Learning to Rank for Information Retrieval", Foundations and Trends® in Information Retrieval: Vol. 3: No. 3, pp 225-331.
- [Macdonald2012] The Whens and Hows of Learning to Rank. Craig Macdonald, Rodrygo Santos and Iadh Ounis. Information Retrieval 16(5):584-628.
- [Macdonald2012a] On the Usefulness of Query Features for Learning to Rank. Craig Macdonald, Rodrygo Santos and Iadh Ounis. In Proceedings of CIKM 2012.
- [Macdonald2013] About Learning Models with Multiple Query Dependent Features. Craig Macdonald, Rodrygo L.T. Santos, Iadh Ounis and Ben He. *Transactions on Information Systems*. 31(3).
- [Metzler2005] Donald Metzler and W Bruce Croft. 2005. A Markov random field model for term dependencies. In Proceedings of SIGIR.
- [Tonellotto2018] Efficient Query Processing for Scalable Web Search. Nicola Tonellotto, Craig Macdonald and Iadh Ounis. In *Foundations and Trends® in Information Retrieval*. Vol. 12: No. 4-5, pp 319-500, 2018.



University  
of Glasgow



UNIVERSITÀ DI PISA

CIKM  
2021  
1-5 NOVEMBER

**Part 2D**

**WRAPUP**

# **Achievements in Parts 1(b)**

We have **reviewed modern IR architectures**, namely...

- ...how existing IR techniques fit into the PyTerrier datamodel and transformers
- ...how transformer pipelines can be created using operators
- the learning-to-rank architecture, as well as common learners, and how to easily create features using LTR

In the practical session, you will have a chance to **experience** these techniques in practice on TREC Covid test collection.

In Part 2, we will **review** contemporary IR re-ranking architectures based on neural networks, as well allowing you to **experience** these through PyTerrier



University  
of Glasgow



UNIVERSITÀ DI PISA

CIKM  
2021  
1-5 NOVEMBER

# QUESTIONS?

# *Practical Time*



UNIVERSITÀ DI PISA



University  
of Glasgow

## The tutorial Github repo has links to the notebook for **Part 2**

- <https://github.com/terrier-org/searchsolutions2022-tutorial>
- Press the  Open in Colab link for each notebook to start a Colab session

### Timings:

- Practical – in breakout rooms – until 1230
- Lunch break 1230-1330
- Part 3 resumes at 1330