University of Glasgow | Università di Pisa

CIKM 2021 1-5 November

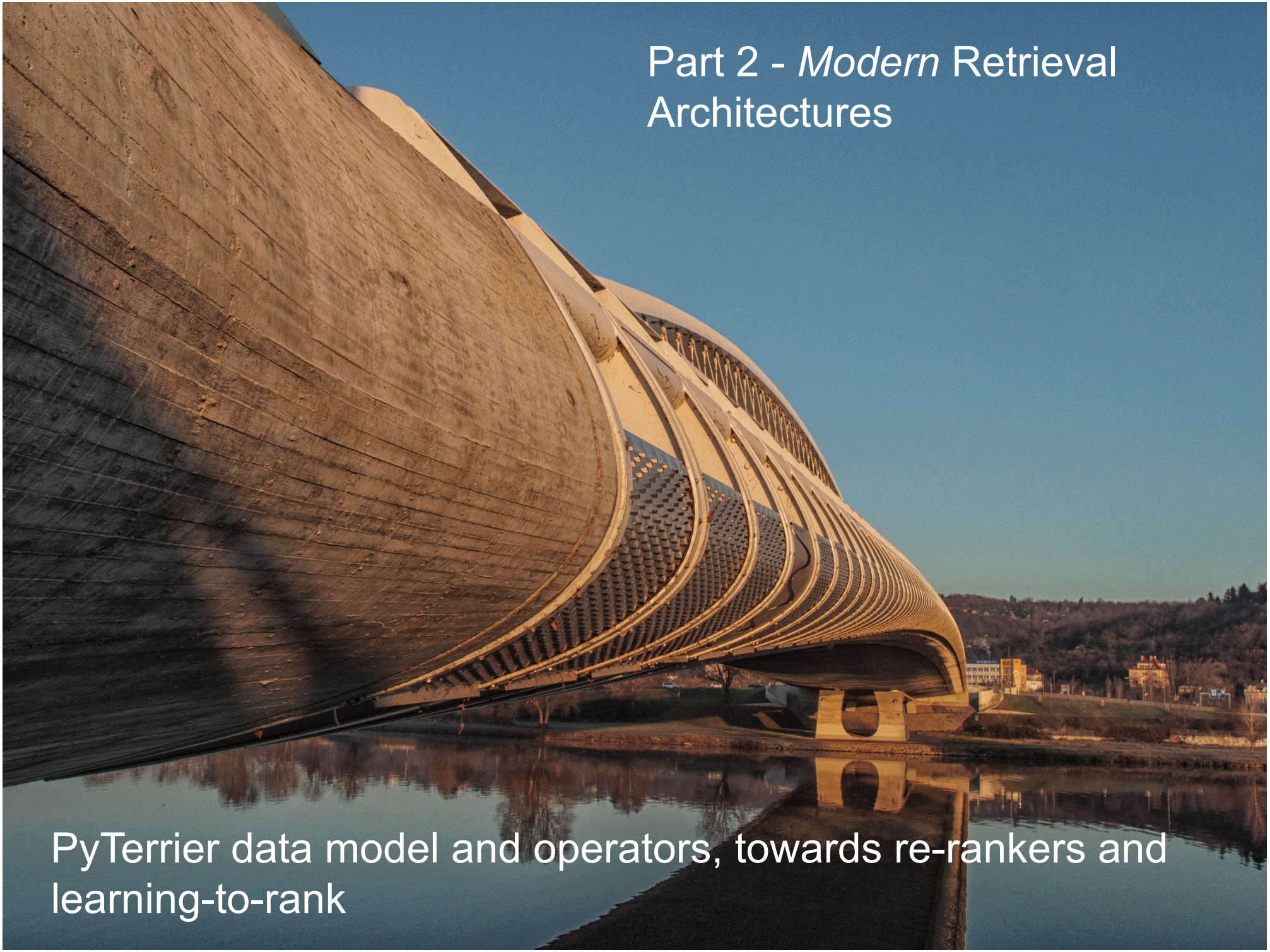
IR From Bag-of-words to BERT and Beyond through Practical Experiments

A CIKM 2021 tutorial with PyTerrier and OpenNIR

Sean MacAvaney*
Craig Macdonald*
Nicola Tonellotto*

Part 2 Starts at 1100 GMT (for Run 1)

*(*Alphabetical ordering)*

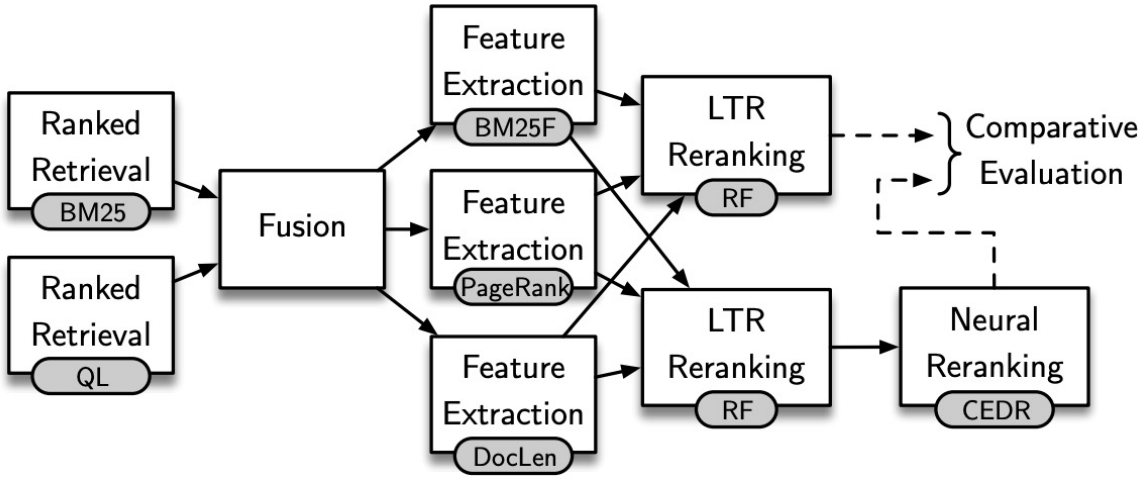

Part 2 - *Modern* Retrieval Architectures
PyTerrier data model and operators, towards re-rankers and learning-to-rank

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

Università di Pisa | University of Glasgow

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

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

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

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

Outline of Part 2

Università di Pisa | University of Glasgow

Part 2A: PyTerrier data model & transformers

Part 2B: Learning to Rank

Part 2C: Usefulness of PyTerrier for Teaching IR

Part 2D: Wrap up and move to practical session

Part 2A

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

Q

qid	query
0	gold coast temperature november
1	one way flight to gold coast price
…	…

D

docno	text	title
0	Science & Mathematics Physi…	The hot glowing…
1	School-Age Kids Growth &…	Developmental…
…	…	…

R

qid	query	docno	score
0	gold coast temperature november	15213	0.5134
0	gold coast temperature november	42635	0.3742
0	gold coast temperature november	26340	0.3223
…	…	…	…

$R \subset Q \times D$

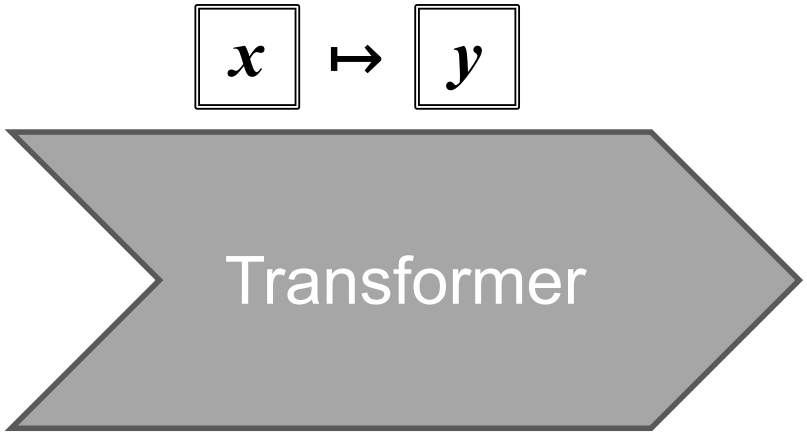
RA

qid	docno	label
0	15213	1
…	…	…

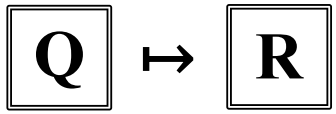
$RA \subset Q \times D$

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

Transformation Function Objects (Transformers)



Transformation Function Objects (Transformers)



qid	query
0	gold coast temperature…
1	one way flight to gold…
…	…

BM25

Inverted Index

k1=1.2 b=0.75

qid	query	docno	score
0	gold coast…	15213	0.5134
0	gold coast…	42635	0.3742
0	gold coast…	26340	0.3223
…	…	…	…

Transformation Function Objects (Transformers)

Università di Pisa | University of Glasgow

$$Q \mapsto R$$

qid	query
0	gold coast temperature…
1	one way flight to gold…
…	…

Your latest retrieval approach…

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)

Transformer Class	Examples
$\boxed{\mathbf{Q}} \mapsto \boxed{\mathbf{R}}$ Retrieval	BM25, ColBERT, ANCE (dense retrieval)

Transformation Function Objects (Transformers)

Transformer Class	Examples
$\boxed{\mathbf{Q}} \mapsto \boxed{\mathbf{R}}$ Retrieval	BM25, ColBERT, ANCE (dense retrieval)
$\boxed{\mathbf{R}} \mapsto \boxed{\mathbf{Q}}$ Query Expansion	RM3, Bo1, etc.

Transformation Function Objects (Transformers)

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

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

     bm25(topics)
```

	qid	docid	docno	rank	score	query
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...

R

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)))
```

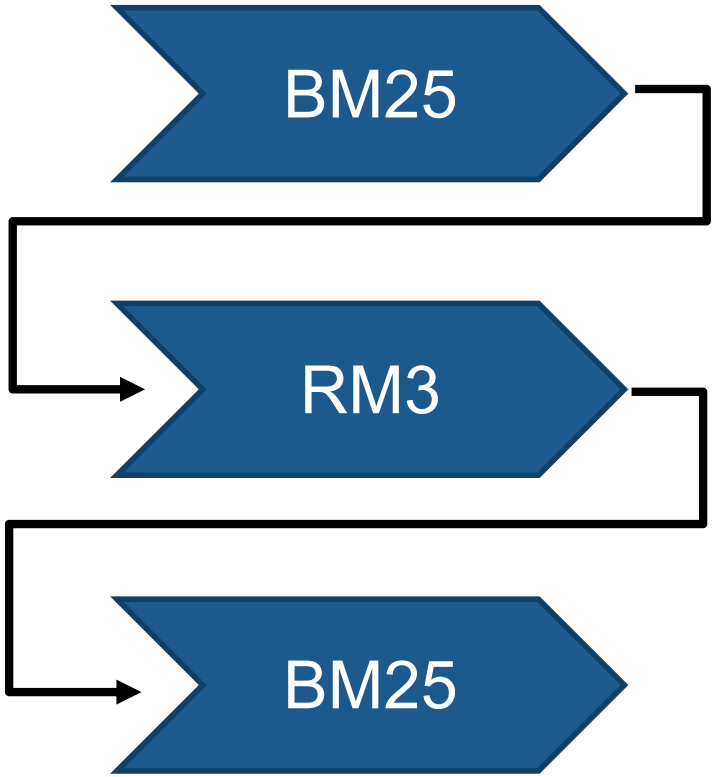
	qid	docid	docno	rank	score	query	query_0
0	1	109719	kgifmjvb	0	19.316403	19^0.200000018 origin^0.325867474 covid^0.2000...	what is the origin of covid 19
1	1	135553	wmfcey6f	1	19.281917	19^0.200000018 origin^0.325867474 covid^0.2000...	what is the origin of covid 19
2	1	83032	dv9m19yk	2	11.101417	19^0.200000018 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

R

Results of BM25 over RM3

Operators

```
bm25(rm3(bm25(topics)))
```

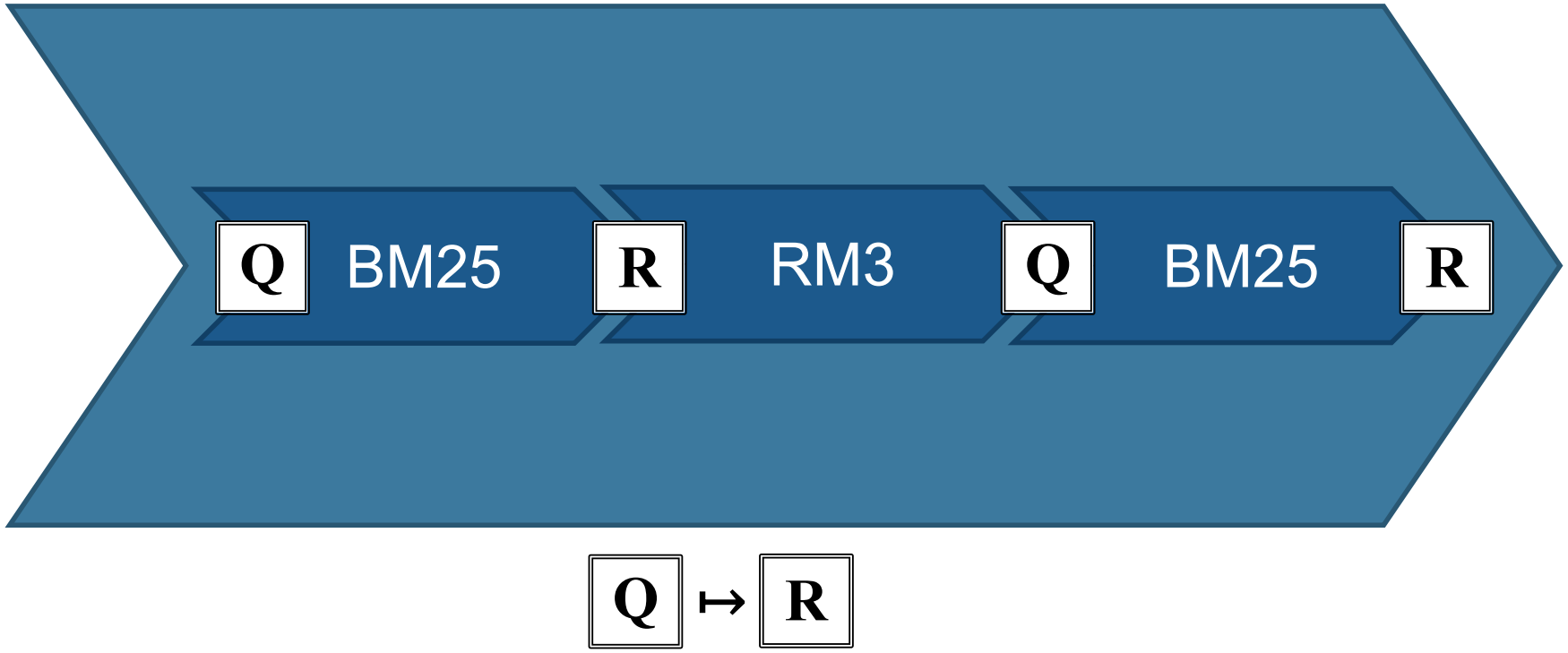


Operators

A better way: the “then” operator

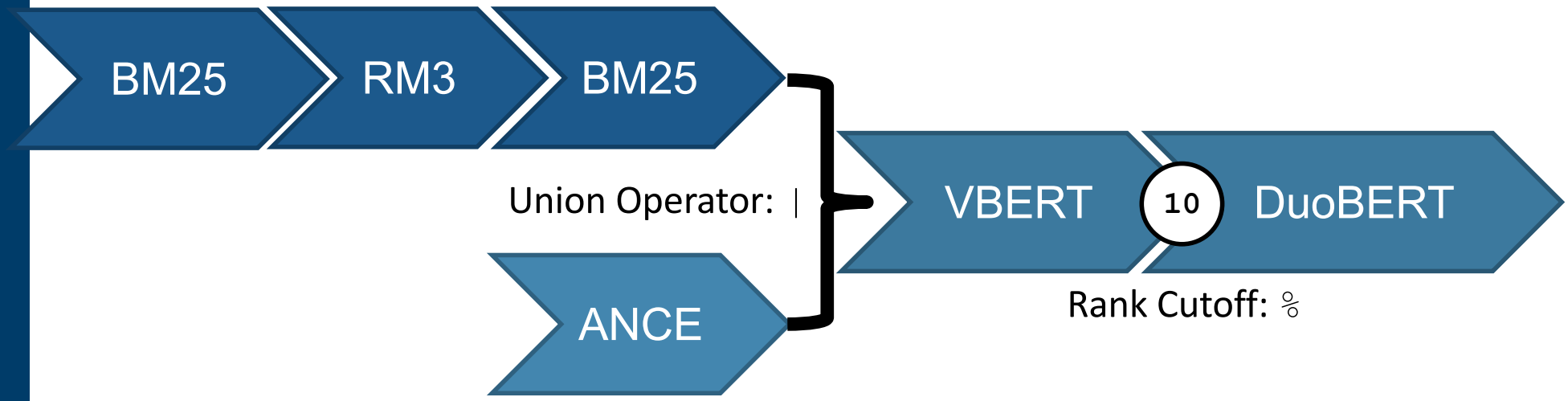
```
A >> B (A then B)
```

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



Operators

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



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

Transformer Operators

PyTerrier has a suite of operators on transformers

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

Output Type: Q

Output Type: R

	qid	query
0	1	chemical reactions

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

Lets look at some retrieval use cases

Università di Pisa | University of Glasgow

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

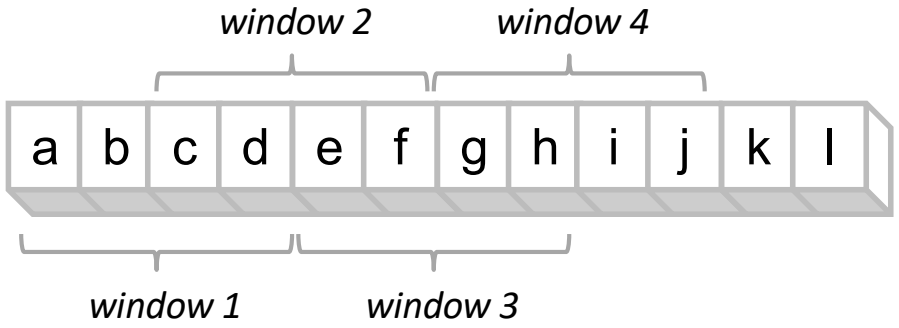
Output Type: R

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
- Its 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

[Dai & Callan, SIGIR 2018]

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()
```

BR output (R w/ text)

qid	docno	text
q1	d1	a b c d

sliding("text", 2, 1) 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

Università di Pisa | University of Glasgow

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)

Università di Pisa | University of Glasgow

All transformers implement a base class called TransformerBase

- 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 return 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 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

Summary: PyTerrier Standard Transformers

Università di Pisa | University of Glasgow

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)`
- `pt.apply.<ANY COLUMN NAME>(drop=True)`
- `pt.apply.rename(columns={“old”: “new”})`

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

Reflections (1)

Università di Pisa | University of Glasgow

Transformers build on PyTerrier's extensible data model

- defined on IR-level objects: queries and documents
- One could imagine more data model columns for, e.g., ConvQA

Operators, defined with clear relational algebra 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)

Università di Pisa | University of Glasgow

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_pipe5 has the following transformations

[qid, query] (Q) >> *[qid, query, query_embs]* (Q+) >> *[qid, query, query_embs, docno]* (R+) >> *[qid, query, query_embs, docno, rank, score]* (R+)

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 3 & 4:

- pyt_t5, pyt_colbert, pyt_ance, pyt_doc2query, pyt_deepimpact

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

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

Indexers also can be supported

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

Then its plain sailing ⛵ ⛵ ⛵!

Reasoning on Transformer Pipelines (1/2)

Università di Pisa | University of Glasgow

Validation *(Coming Soon!)* *pipe.validate()*

- Lots of transformers, lots of operators. Is my pipeline expression valid? Do I have the correct columns?

- We can know the desired input & actual output columns of each transformer
  - E.g. BatchRetrieve needs qid and query columns, sliding() needs text column etc
- Similarly, we know actual inputs and desired outputs of an entire pipeline
  - E.g. We know that queries have qid and query columns
  - E.g. We know that we need qid, docno, score to evaluate a ranking

- Therefore, it is possible to *validate* (type check) a pipeline before execution – check if each transformer stage emits the necessary columns for the next stage
  - Q + Q would throw an error – we cannot make a linear combination of Q
- This would be transparent to most users; Some "generic" pt.apply operations would need to specify their input and output columns to ensure validation

Reasoning on Transformer Pipelines (2/2)

Università di Pisa | University of Glasgow

Optimisation *pipe = pipe.compile()*

- Pipelines with identical semantics can be written in different ways

```
br1 = pt.BatchRetrieve(index) % 10
br2 = pt.BatchRetrieve(index, num_results=10)
```

- In our ICTIR 2020 paper, we showed that pipelines can be rewritten into equivalent but faster equivalent pipelines as a form of graph rewriting

Parallelisation *pipe = pipe.parallel(2)*

- Transformers can be parallelized using different processes, or even to different machines
- This is because the primary key attributes of each transformer are known, so we can partition a Q or R dataframe by qid; a D dataframe by docno

Tuning: GridScan & GridSearch

Università di Pisa | University of Glasgow

Often, we have transformers with parameters, e.g.

```
BM25 = pt.BatchRetrieve(index, wmodel="BM25",
    controls={"c" : 0.75, "bm25.k_1": 0.75, "bm25.k_3": 0.75})
```

Which we need to tune… `pt.GridSearch` and `pt.GridScan`

```
pt.GridSearch(
    BM25,
    {BM25: {
        "c" : [0, 0.2, 0.4, 0.6, 0.8, 1 ],
        "bm25.k_1": [ 0.3, 0.6, 0.9, 1.2, 1.4, 1.6, 2],
        "bm25.k_3": [ 0.5, 2, 4, 6, 8, 10, 12, 14, 20]
    }},
    train_topics,
    train_qrels,
    "map")
```

Transformer pipeline to use

Dictionary mapping transformer to parameter to value range

Topics, qrels, measure

- `GridSearch` – find the best setting; `GridScan` – evaluate all settings
- Also a `KFoldGridSearch` facilitates cross-validation settings

Part 2B

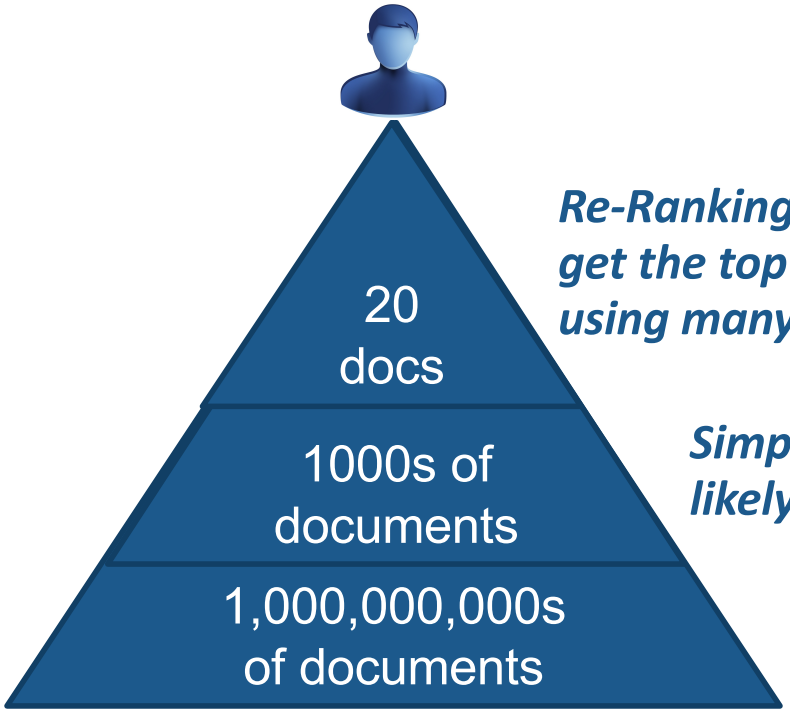
LEARNING TO RANK

Ranking Cascades

Università di Pisa | University of Glasgow

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***

[1] J Pederson. Query understanding at Bing. SIGIR 2010 Industry Day.

Motivations for Learning

Università di Pisa | University of Glasgow

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?

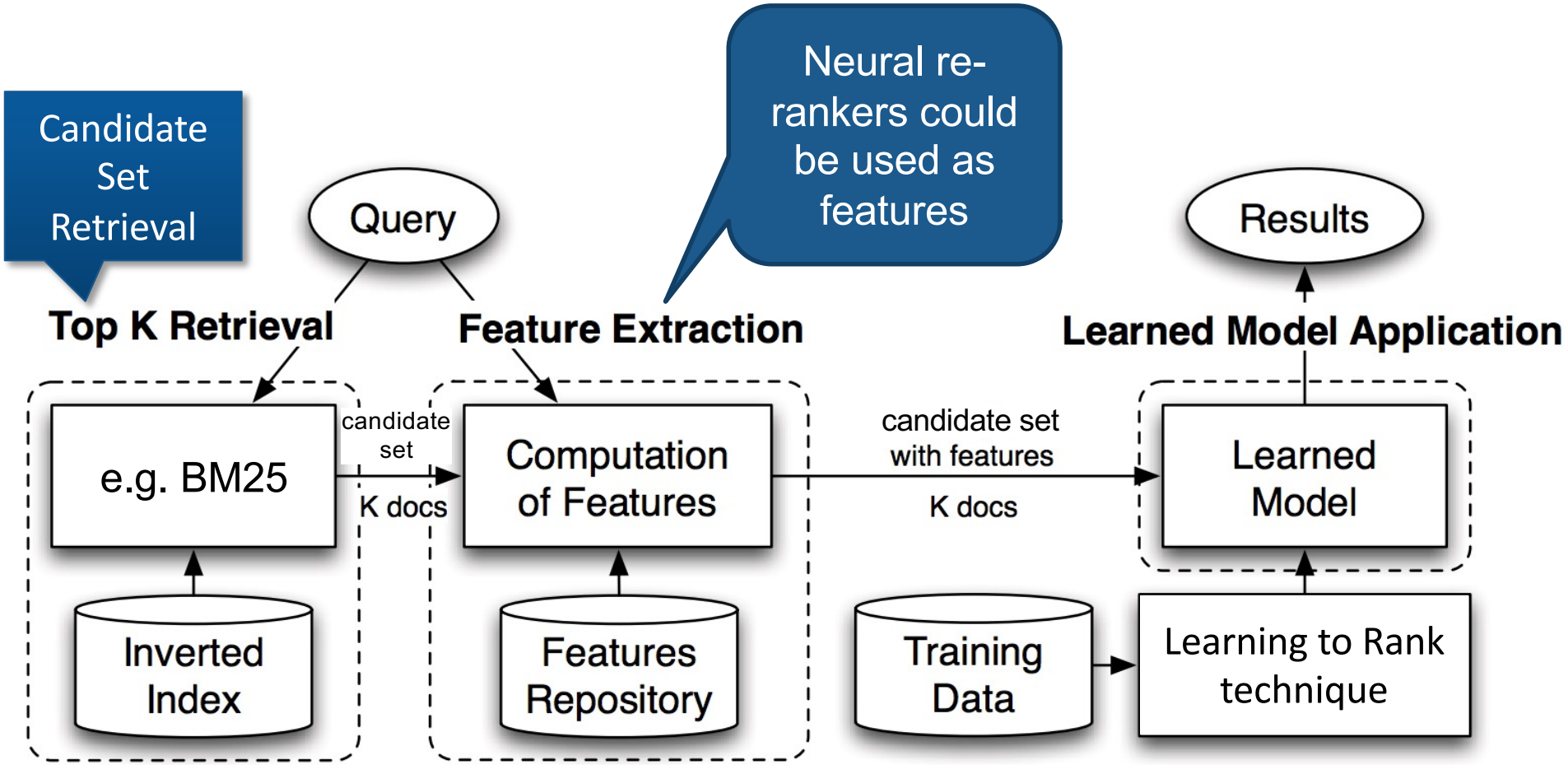
Learning to Rank definition

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…



N. Tonellotto, C. Macdonald, I. Ounis. (2013). Efficient and effective retrieval using selective pruning. WSDM'13.

Types of Features

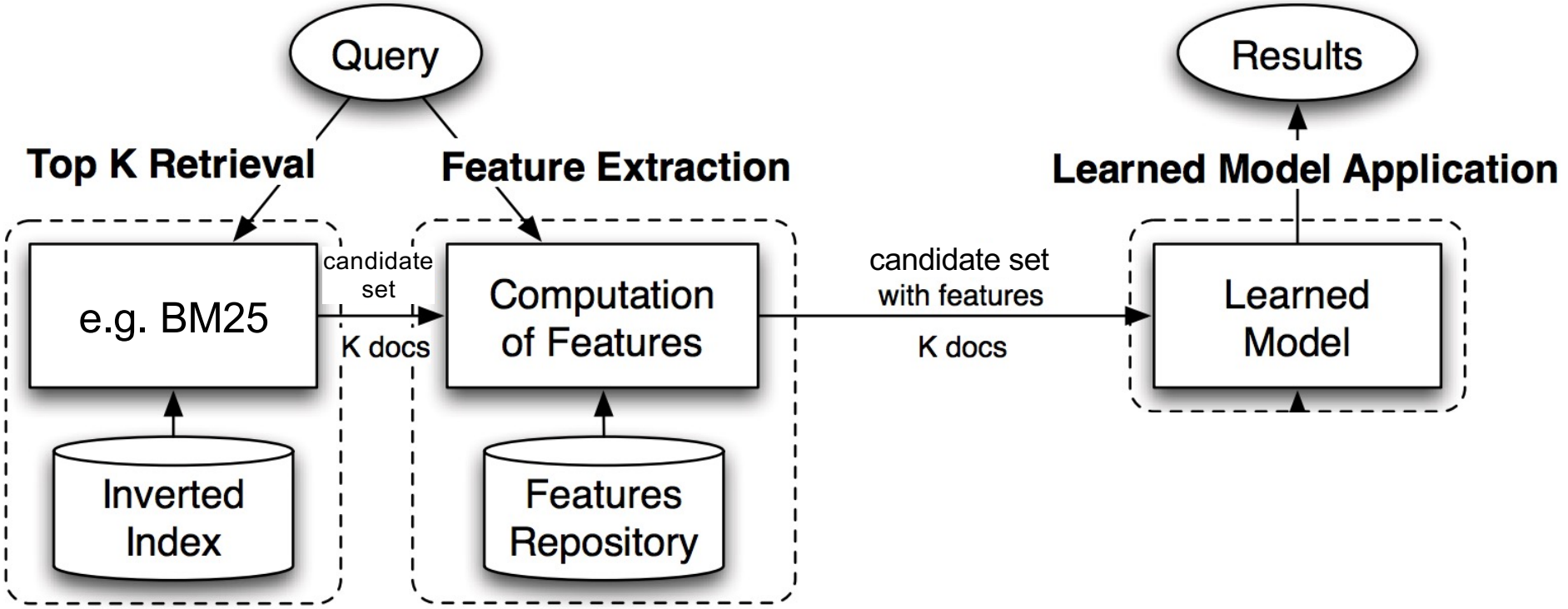
Università di Pisa | University of Glasgow

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 Proximity models, e.g. Markov Random Fields Field-based weighting models, e.g. PL2F Neural re-rankers [Part 3]
Query Independent Features	X	✓	PageRank, number of inlinks Spamminess
Query Features	✓	X	Query length, Query performance predictors Presence of entities

C. Macdonald et al. On the Usefulness of Query Features for Learning-to-Rank, CIKM 2012

Matching LTR concepts to PyTerrier…



In PyTerrier:

```
                                  (
                                    pt.BatchRetrieve()
                                  **
pt.BatchRetrieve() % K >> pt.apply.doc_score()   >>   pt.ltr.apply_learned_model()
                                  **
                                  ..
                                  )
```

"Featured" Data Model

Università di Pisa | University of Glasgow

Retrieved documents with features, R_f:

qid	query	docno	score	rank	*features*

- 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") )
```

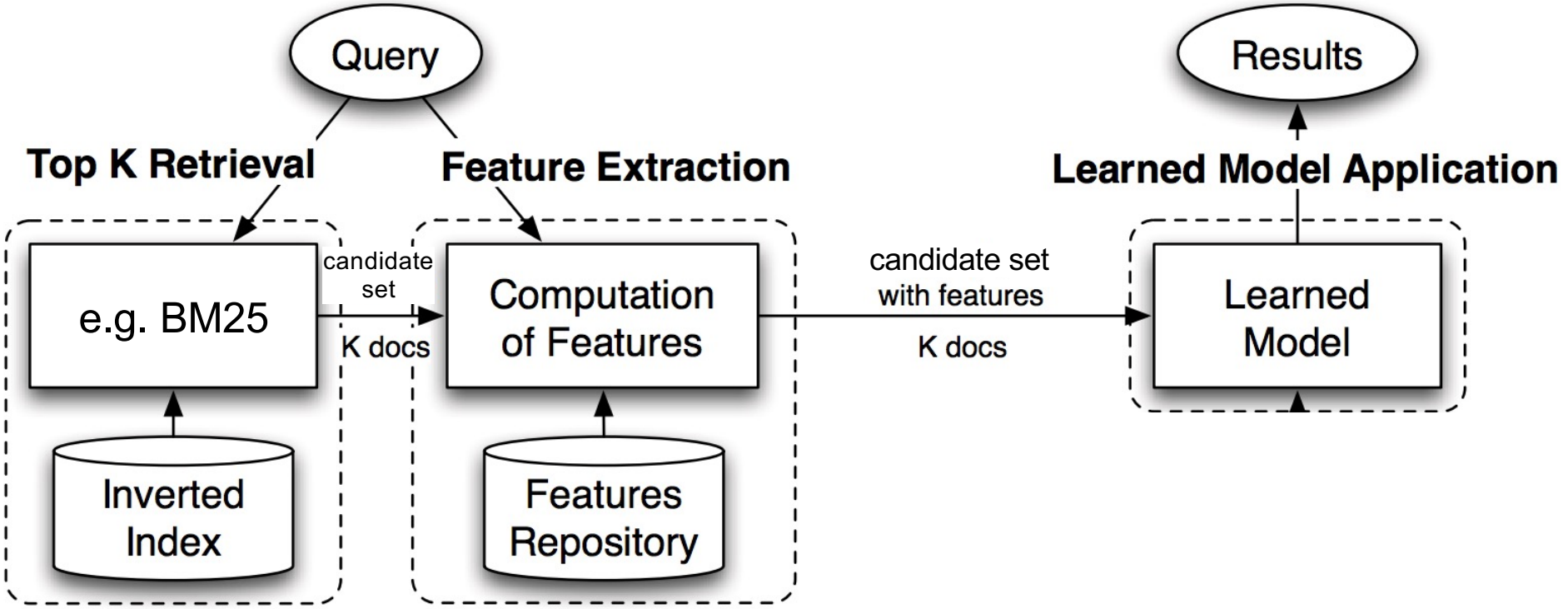
R

qid	docno	score
q1	d5	(bm25 score)

R_f

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

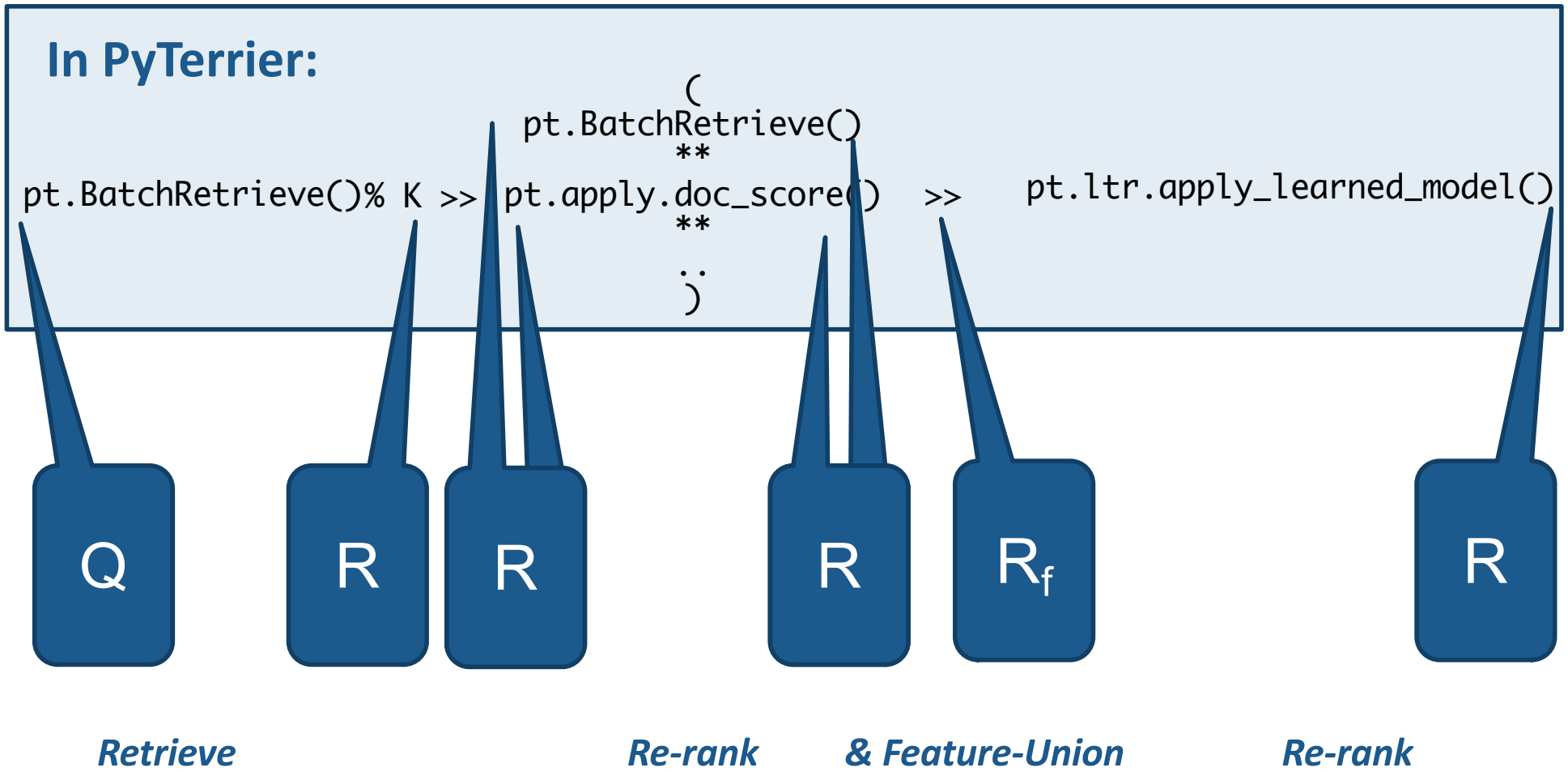
Matching LTR concepts to PyTerrier…



In PyTerrier:

```
                                  (
                                    pt.BatchRetrieve()
                                  **
pt.BatchRetrieve() % K >> pt.apply.doc_score()   >>   pt.ltr.apply_learned_model()
                                  **
                                  ..
                                  )
```

Data Model Transformations



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

C Macdonald et al. (2012). About Learning Models with Multiple Query Dependent Features. TOIS 31(3).

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

Difficulty in Training

IR evaluation measures are not continuous, and hence are not differentiable wrt. a feature weight

- i.e. they are not smooth as we vary a feature weight *w*

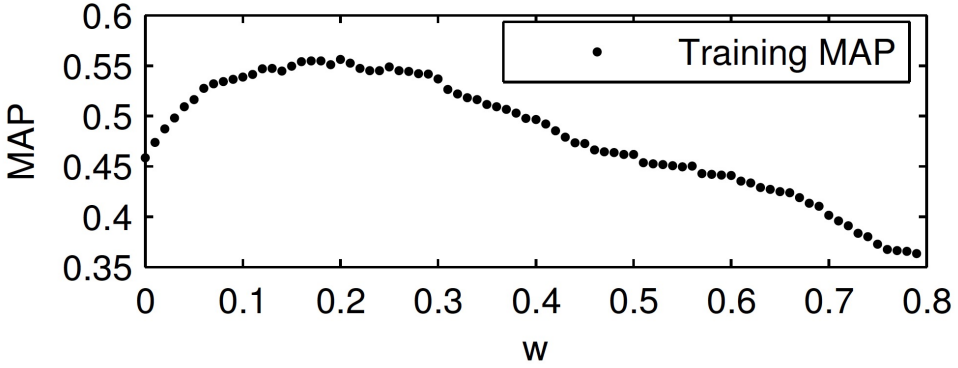
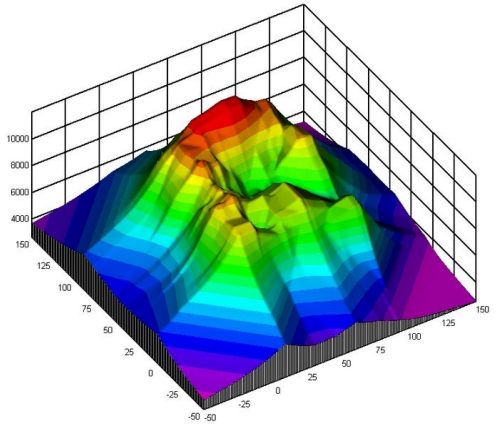


Figure 2: Tuning w, for *log* PageRank combination.

Why?



Metrics don't respond to changes in the scores of documents, but only when a swap between the ordering of two documents takes place

- We need advanced gradient descent/hill-climbing/simulated annealing
- Many of the advances in learning to rank try to address this

Figures from: (a) Craswell et al., SIGIR 2005 (b) Hongning Wang

(Greedy) Automatic Feature Selection (AFS)

Università di Pisa | University of Glasgow

Consider a linear model:

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

AFS: Select a $w_f > 0$ for the feature f that best improves an evaluation measure M

	f_1	f_2	f_3	f_4
d_1	0.2	0.1	0.5	0.8
d_2	0.5	0.7	0.0	0.2
d_3	0.3	0.2	0.5	0.0

Step 1: Select the most effective single feature according to M

f1	0.30
f2	0.35
f3	0.05
f4	0.10

→ *(f2, 1)*

w_1	w_2	w_3	w_4

Current Best M = 0

D. Metzler. Automatic Feature Selection for the Markov Random Field Model for IR. CIKM'07.

(Greedy) Automatic Feature Selection (AFS)

Università di Pisa | University of Glasgow

Consider a linear model:

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

AFS: Select a $w_f > 0$ for the feature f that best improves an evaluation measure M

Step 2: Select the feature and corresponding w_f that most improves M (e.g. using sim. annealing)

	f_1	f_2	f_3	f_4
d_1	0.2	0.1	0.5	0.8
d_2	0.5	0.7	0.0	0.2
d_3	0.3	0.2	0.5	0.0

f2 + ? x f1	
f2 + ? x f3	
f2 + ? x f4	

w_1	w_2	w_3	w_4
	1		

Current Best M = 0.35

D. Metzler. Automatic Feature Selection for the Markov Random Field Model for IR. CIKM'07.

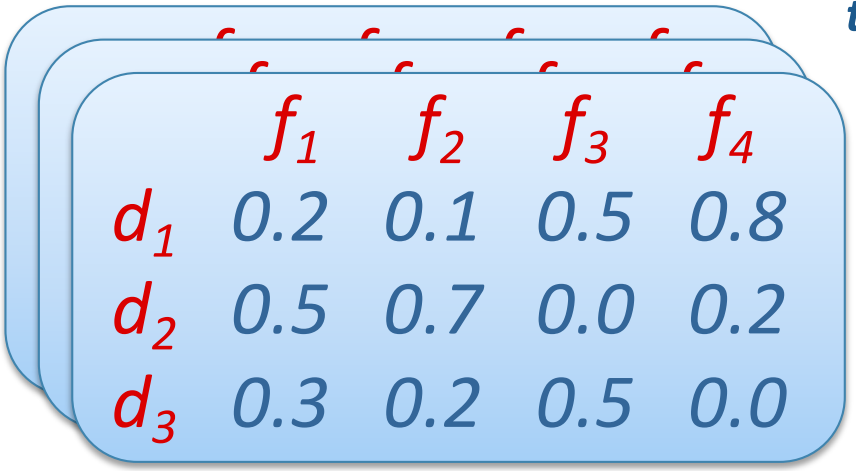
(Greedy) Automatic Feature Selection (AFS)

Università di Pisa | University of Glasgow

Consider a linear model:

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

AFS: Select a $w_f > 0$ for the feature f that best improves an evaluation measure M



Step 2: Select the feature and corresponding w_f that most improves M (e.g. using sim. annealing)

f2 + 1.3 x f1	0.36
f2 + 0.3 x f3	0.40
f2 + 0.9 x f4	0.34

→ *(f3, 0.3)*

w_1	w_2	w_3	w_4
	1	0.3	

Current Best M = 0.40

D. Metzler. Automatic Feature Selection for the Markov Random Field Model for IR. CIKM'07.

(Greedy) Automatic Feature Selection (AFS)

Università di Pisa | University of Glasgow

Consider a linear model:

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

AFS: Select a $w_f > 0$ for the feature f that best improves an evaluation measure M

	f_1	f_2	f_3	f_4
d_1	0.2	0.1	0.5	0.8
d_2	0.5	0.7	0.0	0.2
d_3	0.3	0.2	0.5	0.0

Step 2: Select the feature and corresponding w_f that most improves M (e.g. using sim. annealing)

f2 + 1.3 x f1	0.36
f2 + 0.3 x f3	0.40
f2 + 0.9 x f4	0.34

→ *(f3, 0.3)*

Step 3: Goto step 2, until we observe no more improvements in M, or no more features

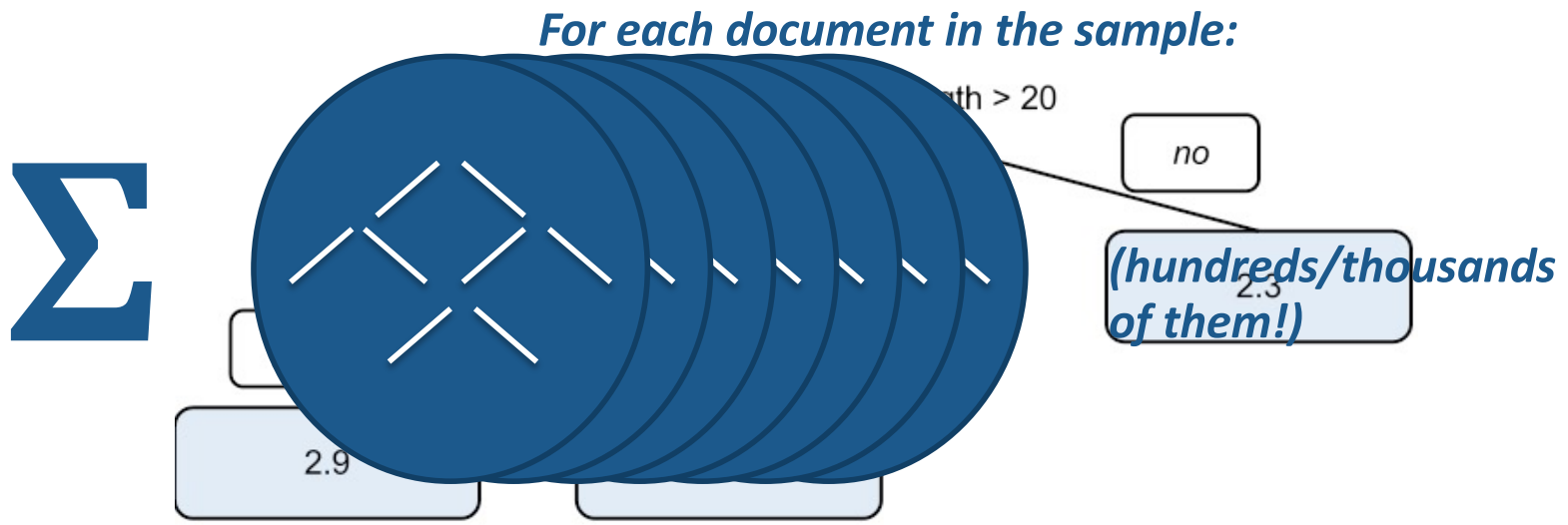
w_1	w_2	w_3	w_4
	1	0.3	

Current Best M = 0.40

D. Metzler. Automatic Feature Selection for the Markov Random Field Model for IR. CIKM'07.

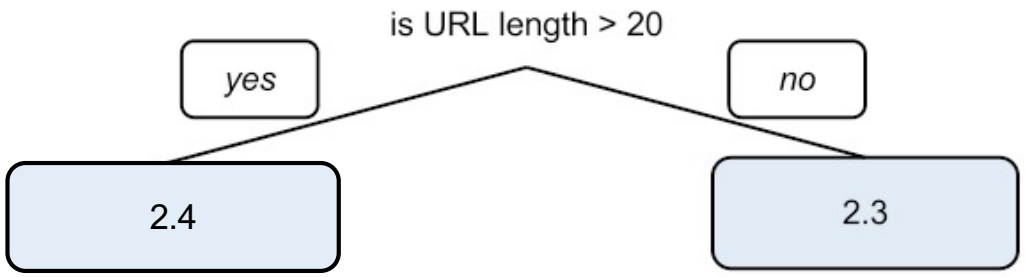
Type of Learned Models (2) Regression Trees

- 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



S. Tyree, K. Weinberger, K. Agrawal, J. Paykin. (2011). Parallel Boosted Regression Trees for Web Search Ranking. WWW'11.

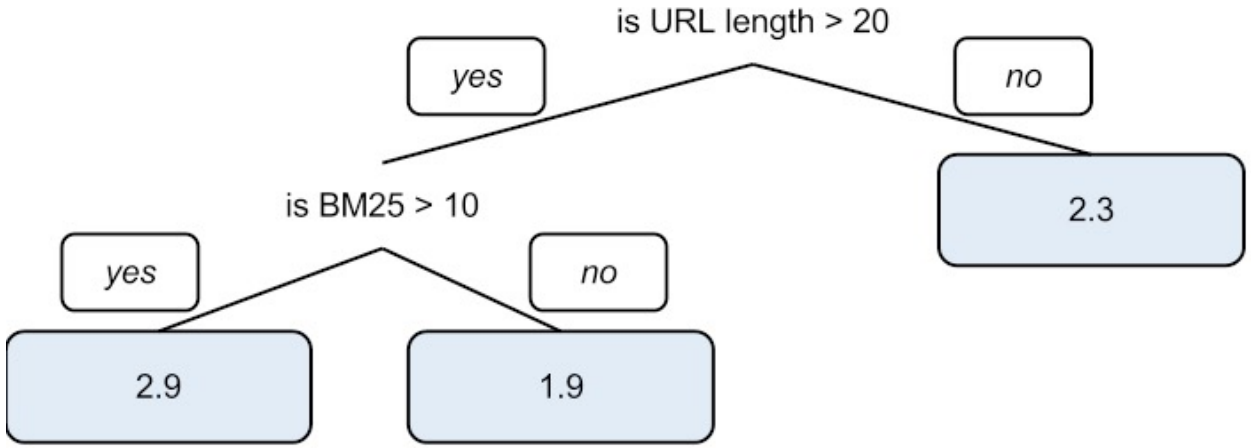
Growing a Tree



	f_1	f_2	f_3	f_4	L
d_1	0.2	0.1	0.5	0.8	1
d_2	0.5	0.7	0.0	0.2	2
d_3	0.3	0.2	0.5	0.0	0

Loss = 1.4

Q: Can we add another decision point?



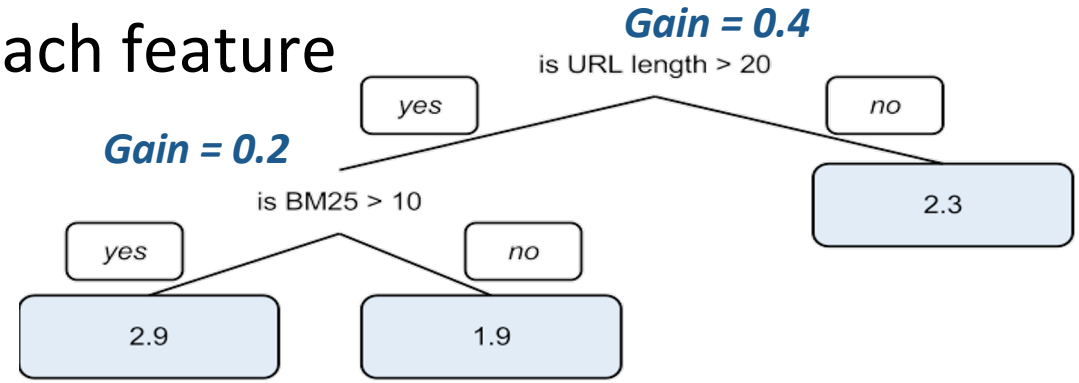
Loss = 1.2 $\Longrightarrow$ ***Gain = 0.2***

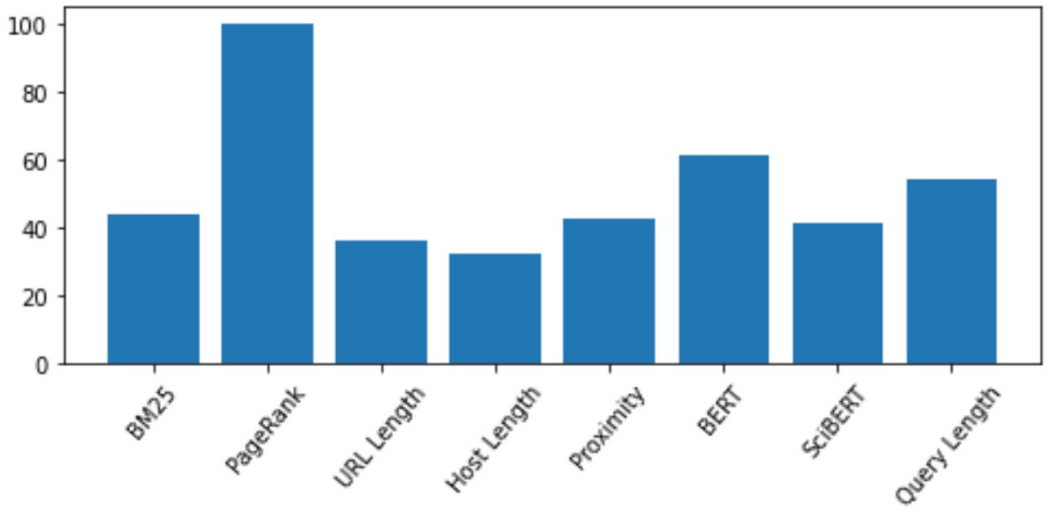
Feature Importance

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. Total 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)
```

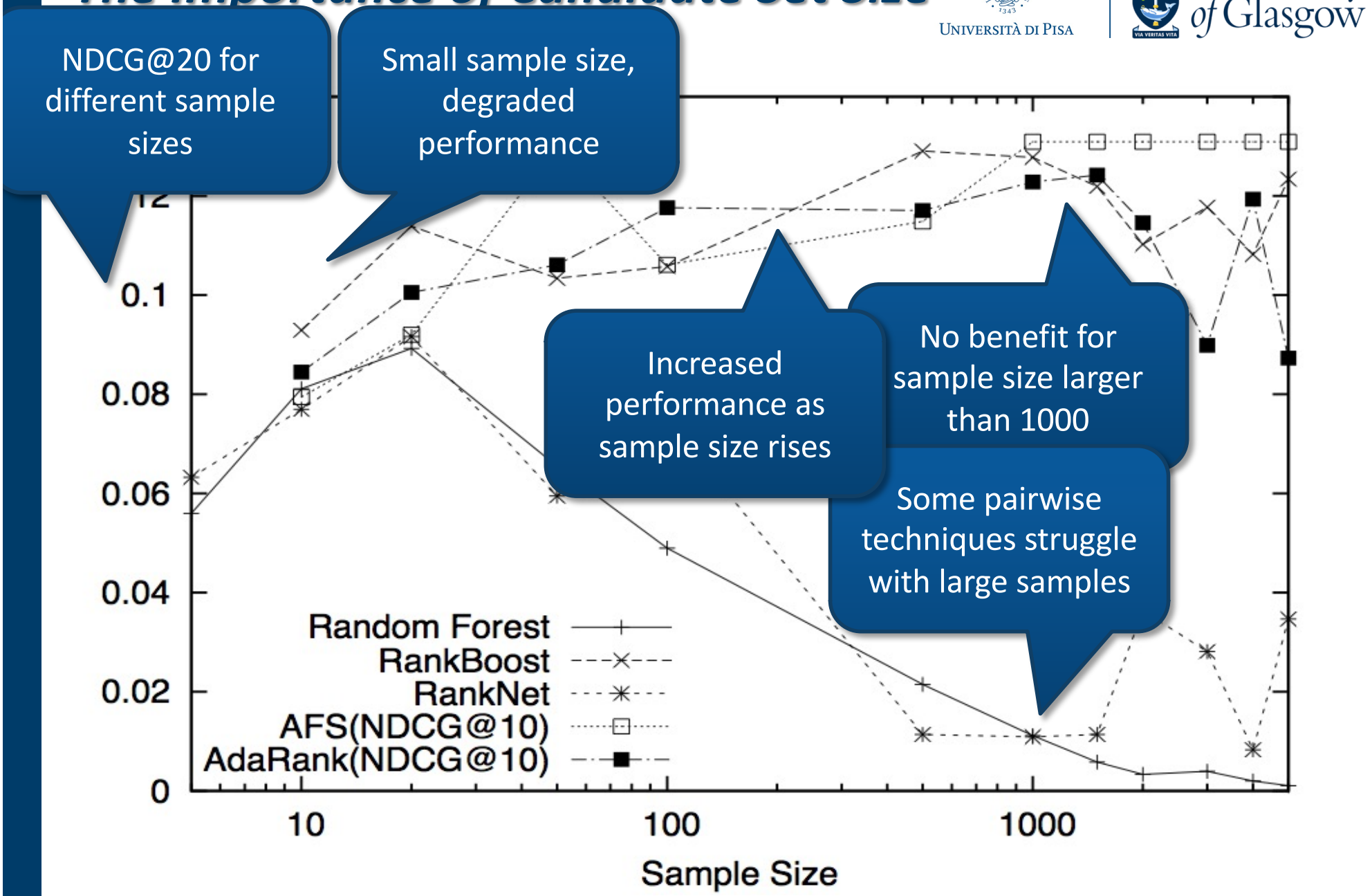
- Also for XGBoost and LightGBM, which have LambdaMART (listwise) implementations

```
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



C. Macdonald, R. Santos and I. Ounis. (2012). The Whens and Hows of Learning to Rank. IR Journal. DOI: 1007/s10791-012-9209-9

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

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 (aka AFS, linear), Random Forests

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

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 2C

USEFULNESS FOR TEACHING IR

Using PyTerrier for Teaching (during a Pandemic)

Università di Pisa | University of Glasgow

We (Iadh Ounis, Sean & I) used PyTerrier for teaching IR 2x this year

- All exercises were conducted in Google Colab, and hence suitable for online teaching during the pandemic
- Made use of PyTerrier's Dataset API to provide index/topics/qrels (with authentication for licensed datasets)
- Students prefer notebooks to commandline interfaces (c.f. Terrier)

Pedagogical appeal

- Full pipeline easily available – indexing, retrieval etc.
- Concepts taught in the course can be easily connected to Python (transformer) classes w/ clear inputs & outputs
  - e.g., a pseudo-relevance feedback component operates after an initial retrieval, and before another ranking component is deployed (final ranking of documents).

Practical Courseworks (1/2)

Università di Pisa | University of Glasgow

Our practical assessments balance 'programming independence' with easily markable assessments that scale to large classes

Coursework 1:

- Perform indexing & retrieval; PyTerrier operators — `pt.DFIndexer()` `pt.BatchRetrieve()`
- Conduct experiments on several query sets, using several classical and probabilistic IR weighting models, e.g., TF-IDF,BM25, PL2, with and without PRF, including applying significance testing, drawing interpolated precision-recall graphs & per-query performance analysis — `pt.BatchRetrieve()` `Bo1QueryExpansion()` `pt.Experiment()`
- Implement a word2vec based QE technique and compare to classical QE — `pt.apply.query()`

Most of the exercise has known correct answers can be easily automatically marked in a quiz

Practical Courseworks (2/2)

Coursework 2:

- Train a learning to rank model for web search
- Analyse feature importance

`pt.ltr.*`

- Define and implement features for URL length and query term proximity

`pt.apply.doc_score()`

- Conduct experiments, perform significance testing & failure analysis

`pt.Experiment()`

Overall feedback:

- PyTerrier builds on familiar technologies: Python, Pandas
- Easy to design and implement new techniques / features
- Students who embrace declarative thinking will write elegant, clear code
- Providing Colab template notebooks helped students to get going
- Increased student performance viz. comparative exercises on commandline/Java

Teaching Users

Università di Pisa

University of Glasgow

TU Delft

Delft University of Technology

The University of Queensland

Australia

University of Glasgow

VIA VERITAS VITA

University of Washington

McGill University

University of Glasgow | Università di Pisa

CIKM 2021 1-5 NOVEMBER

Part 2D

WRAPUP

Achievements in Part 2

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

Finally, we also discussed the usefulness of PyTerrier for teaching

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

In Part 3, we will review contemporary IR re-ranking architectures based on neural networks, as well allowing you to experience these through PyTerrier

QUESTIONS?

A Tutorial Scheduled over 22 hours

Run 1 – Live Lectures, Live Labs

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

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

Run 2 – Recorded Lectures, Live Labs

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

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

Breakout Lab Facilitators: Sean, Nic, Craig

Breakout Lab Facilitators: Eugene, Luca

Practical Time

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

- https://github.com/terrier-org/cikm2021tutorial
- Press the 

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

Run 1 times, GMT

If you are leaving us here, please complete our feedback quiz https://forms.office.com/r/RiYSAxAKhk [link also on repo]