Agile Information Retrieval Experimentation with Terrier Notebooks*

Craig Macdonald, Richard McCreadie, Iadh Ounis
University of Glasgow
Glasgow, Scotland, UK
first.lastname@glasgow.ac.uk

ABSTRACT

Teaching modern information retrieval is greatly benefited by giving students hands-on experience with an open-source search engine that they can experiment with. As such, open source platforms such as Terrier are a valuable resource upon which learning exercises can be built. However, experimentation using such systems can be a laborious process when performed by hand; queries might be rewritten, executed, and model parameters tuned. Moreover, the rise of learning-to-rank as the de-facto standard for state-of-the-art retrieval complicates this further, with the introduction of training, validation and testing (likely over multiple folded datasets representing different query types). Currently, students resort to shell scripting to make experimentation easier, however this is far from ideal. On the other hand, the introduction of experimental pipelines in platforms like scikit-learn and Apache Spark in conjunction with notebook environments such as Jupyter have been shown to markedly reduce to barriers to non-experts setting up and running experiments. In this paper, we discuss how next generation information retrieval experimental pipelines can be combined in an agile manner using notebook-style interaction mechanisms. Building upon the Terrier IR platform, we describe how this is achieved using a recently released Terrier-Spark module and other recent changes in Terrier 5.0. Overall, this paper demonstrates the advantages of the agile nature of notebooks to experimental IR environments, from the classroom environment, through academic and industry research labs.

1 INTRODUCTION

Information retrieval (IR) is an important field in computing science, and is taught to undergraduate students at universities worldwide. IR is a dense topic to teach, as it encompasses 30 years of intensive research and development from academia and industry. Moreover, IR is constantly evolving as a field, as new techniques are introduced, tested and adopted by commercial search engines. Indeed, supervised machine learning techniques [9] have systematically replaced traditional theoretically-founded term weighting models (e.g. BM25, language models) over the last decade. As such, taught IR courses need to be flexible in the face of rapid changes in the broader field, so that students are prepared for the challenges they will face in industry.

As with many computing science subjects, hands-on coding experience in IR is very beneficial for grasping the concepts being taught. However, commercial search engines are not publicly available for students to experiment with. For this reason, academic research groups have devoted resources to developing open source search engines [14, 16] that can be used to support learning and teaching in IR, such as the Terrier IR platform [14], while keeping them up-to-date with state-of-the-art techniques.

However, while these platforms provide the core functionality of a search engine and are valuable for supporting hands-on exercises, significant time and effort is required for students to learn the basics of experimenting with such a platform. Furthermore, as the complexity of these platforms grow, the barriers to entry for using these platforms is increasing. For instance, the rise of learning-to-rank as the de-facto standard for state-of-the-art retrieval now requires students to understand and build command pipelines for document and query feature extraction (along with configuration and possibly optimisation); dataset folding; and the subsequent the training, validation and testing of the learned models. As a result, students either spend significant time manually running commands or resort to shell scripting to make experimentation easier. In either case, this is an undesirable burden on the students that wastes valuable tuition time.

On the other hand, agile experimentation, particularly for non-IR machine learning applications, is increasingly being facilitated by the use of experimental pipelines in toolkits like scikit-learn and Apache Spark. These toolkits break-down the steps involved in machine learning into small discrete operations, which can be dynamically chained together - forming a reusable and customisable pipeline. For example, in the scikit-learn toolkit from Python, each supervised technique exposes a `fit()` method for training, and a `transform()` method for applying the trained model. Apache Spark's MLib API similarly defines `fit()` and `transform()` methods for the same purpose. These experimental pipelines are very powerful when combined with recent 'notebook' applications, such as Jupyter, which enable developers to store, edit and re-run portions of their pipelines on-demand.

In this paper, we argue that similar experimental pipelines are the next step in enhancing the teaching of IR in the classroom. In particular, experimental pipelines encapsulated as notebooks:

1. Provide an agile experimental platform that students/researchers/practictioners can easily edit.
2. Enable reproducibility in information retrieval experiments.
3. Centralise configuration such that issues are more easily identified.
4. Allow for more complex examples released as pre-configured notebooks.

* This is an extended version of a demonstration paper that was published at SIGIR 2018.

DESIRES'18, 2018, Bertinoro, Italy
© 2018 Copyright held by the author(s).

CEUR-WS.org/Vol-2167/paper12.pdf

Furthermore, we discuss recent advances in the Terrier IR platform with the Terrier-Spark module and additions to Terrier 5.0 that enables experimental pipelines moving forward.

The structure of this paper is as follows: Section 2 details recent feedback in an empirical information retrieval course; Section 3 highlights the main requirements for an experimental IR platform; Section 4 summarises the current Terrier platform; Section 5 introduces Terrier-Spark and how to conduct IR experiment using it; Section 6 highlights other relecant changes in Terrier 5.0; Section 7 discusses the advantages of combining Terrier-Spark with Jupyter notebooks. Concluding remarks follow in Section 8.

2 RECENT EXPERIENCES OF AN INFORMATION RETRIEVAL COURSE USING TERRIER

Information retrieval has been taught in Glasgow as an undergraduate and postgraduate elective since the mid-1980s. Its current incarnation consists of 20 hours of lectures, along with supplementary tutorials and laboratory sessions, allowing students to gain hands-on experience in developing IR technologies and critically assessing their performance. To address this latter point, we set students with two assessed exercises (aka courseworks), of approximately 20 hours in length total. These allow the student experience with empirical, experimental IR, both from core concepts (TF.IDF, document length normalisation) through to learning-to-rank.

Coursework 1 [8 hours]. Create an index of 50% of the .GOV test collection using Terrier; Perform retrieval for a variety of standard weighting models, with and without query expansion, on three tasks of the TREC 2004 Web track (homepage finding, named-page finding, topic distillation). They are also asked to compare and contrast with a "simple" TF.IDF they have implemented themselves. Students then analyse the results, including per-topic analysis, precision recall graphs, etc. Overall, Coursework 1 is designed to familiarise the students with the core workings of a (web) search engine, and performing IR experiments, as well as analysing their results, critically analysing the attributes of different retrieval techniques, and how these affect performance on different topic sets.

Coursework 2 [12 hours]. Use a provided index for the .GOV test collection that contains field and positional information, along with a number of standard Web features (PageRank, DFR proximity, BM25 on the title). The students are asked to implement two proximity features from a number described in [5], and combine these within a LambdaMART learning-to-rank model. Their analysis must use techniques learned through Coursework 1. e.g. identifying which queries were most benefitted by further proximity features, etc. Overall, this coursework allows student hands-on experience with deploying a learning-to-rank pipeline (training/validation/testing and evaluation), as well as the notion of positional information and posting list iterators necessary to implement their additional proximity features.

Student feedback on the current courseworks. The positive feedback on the current coursework exercises is that these form an effective vehicle for achieving the intended learning outcomes of the course. In particular, they encompass more than programming implementations, and that they force the students to understand the IR concepts already presented in the lectures. Students also value the reinforcement of the empirical nature of IR as a field, and the necessity of experimental evaluation.

However, students also note the difficulties in configuring Terrier (long commandline incantations, and/or awkward editing of the terrier.properties files). Indeed, not all students are familiar with commandline scripting technologies on their chosen operating system. Some students contrasted this with other courses, such as our recent Text-as-Data course on text mining/classification/information extraction, which uses Python notebooks, and lamented the lack of a Juypter environment for Terrier.

Overall, from the feedback described above, it is clear that moving towards providing a notebook environment for performing IR experiments would aid students. To this end, we have considered how to modernise the experimental environment for conducting experiments using Terrier. The next few sections detail the underlying essential requirements for an experimental IR platform (Section 3), the current Terrier status (Section 4), as well as how adapt Terrier to support a notebook paradigm leveraging Apache Spark (Section 5).

3 IR PLATFORM REQUIREMENTS FOR CONDUCTING EMPIRICAL EXPERIMENTS

Below, we argue for, in our experience, the main attributes of an experimental IR platform. These are described in terms of required functionalities - in practice, there are non-functional requirements such as running experiments efficiently on large corpora such as ClueWeb09.

- R1 Perform an "untrained" *run* for a weighting model over a set of query topics, retrieving and ranking results from an index.
- R2 Evaluate a run over a set of topics, based on relevance labels.
- R3 Train the parameters of a run, which may require repetitive execution of queries from an index and evaluation.
- R4 Extract a run with multiple features that can be used as input to a learning-to-rank technique.
- R5 Re-rank results based on multiple features and a pre-trained learning-to-rank technique.

R1 concerns the ability of the IR system to be executed in an offline *batch* mode - to produce the results of a set of query topics. Academic-based platforms such as Terrier, Indri [16], Galago [3] offer such functionality out of the box. R2 concerns the provision of evaluation tools that permit a run to be evaluated. Standard tools exist such as the C-based `trec_eval` library, but integration in the native language of the system may provide advantages for R3. Other systems such as Lucene/Solr/Elastic may need some scripting or external tools (Azzopardi et al. [1] highlight the lack of empirical tools for IR experimentation and teaching on Lucene, and have made some inroads into addressing this need).

Indeed, R3 represents the early advent of machine learning into the IR platform, where gradient ascent/descent algorithms were used to optimise the parameters of systems by (relatively expensive) repeated querying and evaluation of different parameter settings. Effective techniques such as BM25F [20] & PL2F [10] were facilitated by common use of such optimisation techniques.

Finally, R4 & R5 are concerned with successful integration of learning-to-rank into the IR system. As with new technologies,

there can be a lag between research-fresh developments and how they are bled into production-ready systems. Of these, for the purposes of experimentation, R4 is the more important - the ability to efficiently extract multiple query dependent features has received some coverage in the literature [11]. R5 is concerned with taking this a stage further, and applying a learned model to re-rank the results.

In the following, we will describe how Terrier currently meets requirements R1-R5 (Section 4), and how it can be adopted within a Spark environment to meet these in a more agile fashion (Section 5).

4 BACKGROUND ON TERRIER

Terrier [14] is a retrieval platform dating back to 2001 with an experimental focus. First released as open source in 2004, it has been downloaded >50,000 times since. While Terrier portrays a Java API that allows extension and/or integration into a number of applications, the typical execution of Terrier is based upon procedural command invocations from the commandline. Listing 1 provides the commandline invocations necessary to fulfil requirements R1 & R2 using Terrier. All requirements R1-R5 listed above are supported by the commandline. Moreover, the use of a rich commandline scripting language (GNU Bash, for instance) permits infinite combinations of different configurations to be evaluated automatically. Moreover, with appropriate cluster management software, such runs can be conducted efficiently in a distributed fashion. This commandline API is also the main methods that students learn to interact with the IR system.

However, we have increasingly found that a commandline API was not suited for all purposes. For instance, the chaining of the outcomes of between invocations requires complicated scripting. For instance, consider, for each fold of a 5-fold cross validation: training the b length normalisation parameter of BM25, saving the optimal value, and using that for input to a learning-to-rank run, distributed among a cluster environment. Such an example would require creating tedious amounts of shell scripting, for little subsequent empirical benefit. In short, this paper argues that IR experimentation has now reached the stage where we should not be limited by the confines of a shell-scripting environment.

5 TERRIER-SPARK

To address the perceived limitations in the procedural commandline use of Terrier, we have developed a new experimental interface for the Terrier platform, building upon Apache Spark, and called Terrier-Spark. Apache Spark is a fast and general engine for large-scale data processing. While Spark can be invoked in Java, Scala and Python, we focus on the Scala environment, which allows for code that is more succinct than the equivalent Java (for instance, through the use of functional progamming constructs, and automatic type inference). Spark allows relational algebra operations on dataframes (relations) to be easily expressed as function calls, which are then compiled to a query plan that is distributed and executed on machines within the cluster.

Apache Spark borrows the notions of dataframes from Pandas[1] (a Python data analysis library), and similarly the notion of machine learning pipeline constructs and interfaces (e.g. `fit` and `transform` methods) from scikit-learn[2] (Python machine learning library), namely:

- DataFrame: a relation containing structured data.
- Transformer: an object that can transform a data instance from a DataFrame.
- Estimator: an algorithm that can be fitted to data in a DataFrame. The outcome of an Estimator can be a Transformer - for instance, a machine-learned model obtained from an Estimator will be a Transformer.
- Pipeline: A series of Transformer and Estimators chained together to create a workflow.
- Parameter: A configuration option for an Estimator.

In our adaptation of Terrier to the Spark environment, Terrier-Spark, we have implemented a number of Estimators and Transformers. These allow the natural stages of an IR system to be combined in various ways, while also leveraging the existing supervised ML techniques within Spark to permit the learning of ranking models (e.g. Spark contains logistic regression, random forests, gradient boosted regression trees, but notably no support for listwise based learning techniques such as LambdaMART [19], which are often the most effective [2, 9]).

Table 1 summarises the main components developed to support the integration of Terrier into Apache Spark, along with their inputs, outputs and key parameters. In particular, QueryingTransformer is the key Transformer, in that this internally invokes Terrier to retrieve the docids and scores of each retrieved document for the queries in the input dataframe. As Terrier is written in Java, and Scala and Java both are JVM-based languages, Terrier can run "in-process". Furthermore, as discussed in Section 6 below, further changes in Terrier 5.0 permit accessing indices on remotely hosted Terrier servers.

In the following, we provide examples of retrieval experimental listings using Spark through Scala.

```
bin/trec_terrier.sh -r -Dtrec.topics=/path/to/topics \
    -Dtrec.model=BM25
bin/trec_terrier.sh -e -Dtrec.qrels=/path/to/qrels
```

Listing 1: A simple retrieval run and evaluation using Terrier's commandline interface - c.f. requirements R1 & R2.

5.1 Performing an untrained retrieval run

Listing 1 shows how a simple retrieval run can be made using Terrier's commandline API. The location of the topics and qrels files as well as the weighting model, are set on the commandline, although defaults could be set in a configuration file.

In contrast, Listing 2 shows how the exact same run might be achieved from Scala in a Spark environment. Once the topics files are loaded into a two-column dataframe (keyed by "qid", the topic number), these are transformed into a dataframe of result sets, obtained from Terrier (keyed by "⟨qid,docno⟩"). Then a second transformer record the relevant and non-relevant documents within the dataframe, by joining with the contents of the qrels file, before evaluation.

[1] http://pandas.pydata.org/

[2] http://scikit-learn.org/

Component	Inputs	Output	Parameters
QueryStringTransformer	Queries	Queries	Lambda function to transform query
QueryingTransformer	Queries	docids, scores for each query	number of docs; weighting model
FeaturedQueryingTransformer	Queries	docids, scores of each feature for each query	+ feature set
QrelTransformer	results with docids	results with docids and labels	qrel file
NDCGEvaluator	results with docids and labels	Mean NDCG@K	cutoff K

Table 1: Summary of the primary user-facing components available Terrier-Spark.

```
val props = Map("terrier.home" -> "/path/to/Terrier")
TopicSource.configureTerrier(props)
val topics = "/path/to/topics.401-450"
val qrels = "/path/to/qrels.trec8"

val topics = TopicSource.extractTRECTopics(topics)
    .toList.toDF("qid", "query")

val queryTransform = new QueryingTransformer()
    .setTerrierProperties(props)
    .sampleModel.set("BM25")

val r1 = queryTransform.transform(topics)
//r1 is a dataframe with results for queries in topics
val qrelTransform = new QrelTransformer()
    .setQrelsFile(qrels)

val r2 = qrelTransform.transform(r1)
//r2 is a dataframe as r1, but also includes a label column

val meanNDCG = new NDCGEvaluator().evaluate(r2)
```

Listing 2: A retrieval run in Scala - c.f. requirements R1 & R2.

While clearly more verbose than the simpler commandline API, Listing 2 demonstrates equivalent functionality, and clearly highlights the needed data and the steps involved in the experiment. Moreover, the use of objects suitable to be built into a Spark pipeline offers the possibility to build and automate pipelines. As we show below, this functionality permits the powerful features of a functional language to allow more complex experimental pipelines.

5.2 Training weighting models

Listing 3 demonstrates the use of Spark's Pipeline and CrossValidator components to create a pipeline that applies a grid-search to determine the most effective weighting model and its corresponding document length normalisation c parameter. Such a grid-search can be parallelised across many Spark worker machines in a cluster. We note that while grid-search is one possibility, it is feasible to consider use of a gradient descent algorithm to tune the c parameter. However, at this stage we do not yet have a parallelised algorithm implemented that would make best use of a clustered Spark environment.

5.3 Training learning-to-rank models

Finally, Listing 4 demonstrates the use of Spark's in-built machine learning Random Forest regression technique to learn a learning-to-rank model. In this example, the initial ranking of documents is performed by the InL2 weighting model, with an additional three query

```
//assuming various variables as per Listing 2.
val pipeline = new Pipeline()
  .setStages(Array(queryTransform, qrelTransform))

val paramGrid = new ParamGridBuilder()
  .addGrid(queryTransform.localTerrierProperties,
       Array(Map["c"->"1"], Map["c"->"10"], Map["c"->"100"]))
  .addGrid(queryTransform.sampleModel,
       Array("InL2", "PL2"))
  .build()

val cv = new CrossValidator()
  .setEstimator(pipeline)
  .setEvaluator(new NDCGEvaluator)
  .setEstimatorParamMaps(paramGrid)
  .setNumFolds(5)
val cvModel = cv.fit(topics)
```

Listing 3: Grid searching the weighting model and document length normalisation c parameters using Spark's CrossValidator - c.f. requirement R3.

```
val queryTransform = new FeaturesQueryingTransformer()
  .setTerrierProperties(props)
  .setMaxResults(5000)
  .setRetrievalFeatures(List(
    "WMODEL:BM25",
    "WMODEL:PL2",
    "DSM:org.terrier.matching.dsms.DFRDependenceScoreModifier"))
  .setSampleModel("InL2")
val r1 = queryTransform.transform(topics)
//r1 is as per Listing 2, but now also has a column of 3
//feature values for each retrieved document
val qrelTransform = new QrelTransformer()
    .setQrelsFile(qrels)
val r2 = qrelTransform.transform(r1)

//learn a Random Forest model
val rf = new RandomForestRegressor()
   .setLabelCol("label")
   .setFeaturesCol("features")
   .setPredictionCol("newscore")
rf.fit(r2)
```

Listing 4: Training a Random Forests based learning-to-rank model - c.f. requirements R4 & R5.

dependent features being calculated for the top 5000 ranked documents for each query. Internally, this uses Terrier's Fat framework

for implementing the efficient calculation of additional query dependent features [11]. The received random forests model can be trivially applied to a further set of unseen topics (not shown). The resulting Scala code is markedly more comprehensible to the equivalent complex commandline invocations necessary for Terrier 4.2 [17]. Moreover, we highlight the uniqueness of our offering - while other platforms such as Solr and Elastic have Spark tools, none offer the ability to export a multi-feature representation suitable for conducting learning-to-rank experiments within Spark (c.f. R4 & R5).

Of course, the pipeline framework of Estimators and Transformers is generic, and one can easily imagine further implementations of both to increase the diversity of possible experiments: For instance, new Estimators for increased coverage of learning-to-rank techniques, such as LambdaMART [19]; Similarly, Transformers for adapting the query representation, for example by applying query-log based expansions [7] or proximity-query rewriting such as Sequential Dependence models [12]. Once a suitable Pipeline is configured, conducting experiments such as learning-to-rank feature ablations can be conducted in only a few lines of Scala.

6 OTHER CHANGES TO TERRIER 5.0

We have also made a number of other changes to Terrier, which have been incorporated into the recently released version 5.0, which aid in the expanding the possible retrieval concepts that can be easily implemented using Terrier-Spark, while increasing the flexibility offered by the platform.

6.1 Indri-esque matching query language

Terrier 5.0 implements a subset of the Indri/Galago query language [3, Ch. 5], including complex operators such as #syn and #uwN. In particular, Terrier 5.0 provides:

- #syn(t1 t2): groups a set of terms into a single term for the purposes of matching. This can be used for implementing query-time stemming.
- #uwN(t1 t2): counts the number of occurrences of t1 & t2 within unordered windows of size N.
- #1(t1 t2): counts the number of exact matches of the bigram t1 & t2.
- #combine(t1 t2): allows the weighting of t1 & t2 to be adjusted.
- #tag(NAME t1 t2): this allows to assign a name to a set of terms, which can be then be formed as a set of features by the later Fat layer. In doing so, such tagged expression allows various sub-queries to form separate features during learning-to-rank. This functionality is not present in Indri or Galago.

For example, Metzler and Croft's sequential dependence proximity model [12] can be formulated using combinations of #uwN and #owN query operators. Such a query rewriting technique can be easily implemented within Terrier-Spark by applying a QueryStringTransformer that applies a lambda function upon each query, allow users to build upon the new complex query operators implemented in Terrier 5.0. Figure 1 demonstrates applying sequential dependence to a dataframe of queries, within a Jupyter notebook. This is achieved by instantiating a QueryStringTransformer upon a dataframe of queries, using a lambda function that appropriately rewrites the queries.

6.2 Remote Querying

As discussed in Section 5 above, Terrier-Spark has initially been designed for in-process querying. However, concurrent changes to Terrier for version 5.0 have abstracted the notion of all retrieval access being within the current process or accessible from the same machine. Indeed, a reference to an index may refer to an index hosted on another server, and made accessible over a RESTful interface. While this is a conventional facility offered by some other search engine products (and made available through their Spark tools, such as for Elastic's[3] and Solr's[4]), this offers a number of advantages for teaching. Indeed, often IR test collection can be too large to provide as downloads - allowing a remote index accessible over a (secured) RESTful HTTP connection would negate the need to provide students with the raw contents of the documents for indexing. Moreover, unlike conventional Spark tools for Elastic and Solr, the results returned can have various features pre-calculated for applying and evaluating learning-to-rank models.

To make this more concrete, consider the TREC Microblog track which used an "evaluation-as-a-service" methodology [8]. In this, the evaluation track organisers provided a search API based upon Lucene, through which the collection can be accessed for completing the evaluation task. The advancements described here would allow a Terrier index to be provided for a particular TREC collection, easily accessible through either the conventional Terrier commandline tools, or through Terrier-Spark. A run in that track could then be crafted and submitted to TREC wholly within a Jupyter notebook, facilitating easy run reproducibility.

7 CONDUCTING IR EXPERIMENTS WITHIN A JUPYTER NOTEBOOK ENVIRONMENT

Spark can be used in a number of manners: by compiling Scala source files into executable (Jar) files, which are submitted to a Spark cluster, or through line-by-line execution in spark-shell (a *Read-Eval-Print-Loop* or REPL tool). However, each has its disadvantages: the former only permits slow development iterations through the necessity to recompile at each iteration; on the other hand, the REPL spark-shell environment does not easily record the developed code, nor allow parts of the code to be re-executed.

Instead, we note that the use of a Spark environment naturally fits with the use of Scala Jupyter notebooks[5][6]. Jupyter is an open-source web application that allows the creation and sharing of documents that contain code, equations, visualisations and narrative text. Increasingly entire technical report documents, slides and books are being written as Jupyter notebooks, due to the easy integration of text, code and resulting analysis tables or visualisations.

Jupyter notebooks are increasingly used to share the algorithms and analysis conducted in machine learning research papers, significantly aiding reproducibility [15]. Indeed, in their report on the Daghstuhl workshop on reproducibility of IR experiments [6], Ferro, Fuhr et al. note that sharing of code and experimental methods would aid reproducibility in IR, but do not recognise the ability of notebooks to aid in this process.

[3] https://www.elastic.co/guide/en/elasticsearch/hadoop/current/spark.html
[4] https://github.com/lucidworks/spark-solr [5] http://jupyter.org/ [6] We note that Jupyter notebooks are extensible through plugins to Scala and other languages, i.e. not limited to Python.

```
//function that constructs a sequential dependence query for a String query
def doMRF_SD(q : String) = {
    val terms = q.split("\\s+")
    val pairs = terms.sliding(2) //get all pairs of query terms
    // construct the MRF SD query form
    terms.length match {
        case 0 | 1 => q
        //easier to queries for length 2
        case 2 => ("#combine:0=0.85:1=0.15:2=0.05("
            +" #combine(" + q + ") "
            +  pairs.map(x => "#uw8("+ x.mkString(" ") + ")" ).mkString(" ")
            +" #uw12(" + terms.mkString(" ") + ") )")
        case _ => ("#combine:0=0.85:1=0.15:2=0.05("
            +" #combine(" + q + ")"
            +" #combine(" + pairs.map(x => "#uw8("+x.mkString(" ") +")" ).mkString(" ") + ")"
            +" #uw12(" + terms.mkString(" ") + ") )")
    }
}

val testDF = Seq(
   (1, "two terms")
 ).toDF("qid", "query")

new QueryStringTransformer(doMRF_SD).transform(testDF).collect

testDF = [qid: int, query: string]

doMRF_SD: (q: String)String
```

1	#combine:0=0.85:1=0.15:2=0.05(#combine(two terms) #uw8(two terms) #uw12(two terms))

Figure 1: Example of applying a Transformer to apply sequential dependency proximity to a dataframe of queries.

Jupyter notebooks are interactive in manner, in that a code block in a single cell can be run independently of all other cells in the notebook. As a result, Jupyter is also increasingly used for educational purposes - for example, teaching programming within undergraduate degree courses [4, 18], as well as a plethora of data science or machine learning courses [15]. O'Hara et al. [13] described four uses for notebooks in classroom situations, including lectures, flipped-classrooms, home/lab work and exams. For instance, the use of notebooks within a lecturing situation easily permits the students to replicate the analysis demonstrated by the lecturer.

We argue that these general advantages of notebooks can be applied to experimental information retrieval education, through the use of a Spark-integrated IR platform, such as that described in this paper. Indeed, we believe that the changes described in Sections 5 & 6 should address these these feedbacks, allow students to more easily configure the retrieval platform (all configuration of Terrier is presented on the screen), make more powerful experimentation available to a wider and less experienced audience not wishing to engage in complicated shell-scripting.

We believe that Terrier-Spark can bring these same advantages to conducting modern (e.g. learning-to-rank) IR experiments, combined with the accessible and agile nature of a notebook environment. Moreover, an integrated Juypter environment also facilitates, for instance in the IR teaching environment, the creation and presentation of inline figures (e.g. created using the Scala vegas-viz library[7]), such as per-query analyses and interpolated precision-recall graphs. Figures 2 - 4 provide screenshots from such a notebook[8][9] In particular: Figure 2 demonstrates the querying of the Terrier for two different retrieval models; Figure 3 shows analysis on the results, by ranking queries based on the difference of their evaluation performance between rankings; Finally, Figure 4 demonstrates the same information as a per-query analysis figure.

[7] https://github.com/vegas-viz/. [8] We use the Apache Toree kernel for Jupyter, which allows notebooks written in Scala and which automatically interfaces with Apache Spark. [9] The original notebook can be found in the Terrier-Spark repository, see https://github.com/terrier-org/terrier-spark/tree/master/example_notebooks/toree.

8 CONCLUSIONS

In this paper, we have described the challenge of teaching a modern undergraduate- and postgraduate-level elective course on information retrieval. We highlight the main requirements of an experimental IR platform, then further describe Terrier-Spark, an extension of Terrier IR platform to perform IR experimentation within the Spark distributed computing engine, which not only addresses these requirements, but can allow complex experiments to be easily defined within a few lines of Scala code. Terrier-Spark and Terrier 5.0 have been released as open source. In addition, we also argue that Jupyter notebooks for IR can aid not only agile IR experimentation by students, but also in research reproducibility in information retrieval by facilitating easily-distributable notebooks that demonstrate the conducted experiments.

Overall, we believe that notebooks are an important aspect of data science, and that we as an IR community should not fall behind other branches of data science in using notebooks for empirical IR experimentation. The frameworks described here might be extended to other languages (e.g. a Python wrapper for Terrier's RESTful interface), or even to other IR platforms. In doing so, we bring more powerful and agile experimental IR tools into the hands of researchers and students alike. Terrier-Spark has been released as open source, and is available from

https://github.com/terrier-org/terrier-spark

along with example Jupyter notebooks.

This is our simple function to obtain a Terrier run. It returns a Dataset with three columns: the qid, the query, and the resulting nDCG@20

In [4]:
```
import org.terrier.spark.ml._
import org.terrier.spark._
import org.terrier.querying.IndexRef

import org.apache.spark.sql.Row
import org.apache.spark.sql.types.{StructField,StructType,IntegerType, DoubleType}

def ndcgForModel(model : String) = {

    val indexref = IndexRef.of(wt2gIndex)

    val props = Map(
    "terrier.home" -> terrierHome,
    "termpipelines"->"")

    TopicSource.configureTerrier(props)
    val topics = TopicSource.extractTRECTopics(topicsFile)
        .toList.toDF("qid", "query")

    val queryTransform = new QueryingTransformer()
        .setTerrierProperties(props)
        .setIndexReference(indexref)
        .setSampleModel(model)

    val r1 = queryTransform.transform(topics)
    //r1 is a dataframe with results for queries in topics
    val qrelTransform = new QrelTransformer()
        .setQrelsFile(qrelsFile)

    val r2 = qrelTransform.transform(r1)
    //r2 is a dataframe as r1, but also includes a label column
    val ndcg = new RankingEvaluator(Measure.NDCG, 20).evaluateByQuery(r2).toList


    val newSchema = StructType(topics.schema.fields
                            ++ Array(StructField("ndcg", DoubleType, false)))
    spark.createDataFrame(topics.rdd.zipWithIndex.map{
        case (row, index) => Row.fromSeq(row.toSeq ++ Array(ndcg(index.toInt)))
    }, newSchema)
}
```

```
ndcgForModel: (model: String)org.apache.spark.sql.DataFrame
```

Lets get the results for two weighting models, TF_IDF and BM25. What is returned is a Dataset with three columns ("qid", "query", "ndcg")

In [5]:
```
val TFIDF = ndcgForModel("TF_IDF")
val BM25 = ndcgForModel("BM25")
```

Figure 2: Conducting two different retrieval runs within a Jupyter notebook using a function defined in Terrier-Spark.

REFERENCES

[1] Leif Azzopardi et al. 2017. Lucene4IR: Developing Information Retrieval Evaluation Resources Using Lucene. *SIGIR Forum* 50, 2 (2017), 18.

[2] Olivier Chapelle and Yi Chang. 2011. Yahoo! Learning to Rank Challenge Overview. *Proceedings of Machine Learning Research* 14 (2011).

[3] Bruce Croft, Donald Metzler, and Trevor Strohman. 2009. *Search Engines: Information Retrieval in Practice* (1st ed.). Addison-Wesley Publishing Company, USA.

[4] Lynn Cullimore. 2016. Using Jupyter Notebooks to teach computational literacy. (2016). http://www.elearning.eps.manchester.ac.uk/blog/2016/using-jupyter-notebooks-to-teach-computational-literacy/

[5] Ronan Cummins and Colm O'Riordan. 2009. Learning in a Pairwise Term-term Proximity Framework for Information Retrieval. In *Proceedings of SIGIR*.

[6] Nicola Ferro, Norbert Fuhr, et al. 2016. Increasing Reproducibility in IR: Findings from the Dagstuhl Seminar on "Reproducibility of Data-Oriented Experiments in e-Science". *SIGIR Forum* 50, 1 (2016).

[7] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. 2006. Generating Query Substitutions. In *Proceedings of WWW*.

[8] Jimmy Lin and Miles Efron. 2013. Evaluation As a Service for Information Retrieval. *SIGIR Forum* 47, 2 (Jan. 2013), 8–14.

[9] Tie-Yan Liu. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009).

[10] Craig Macdonald, Vassilis Plachouras, Ben He, Christina Lioma, and Iadh Ounis. 2006. University of Glasgow at WebCLEF 2005: Experiments in per-field normlisation and language specific stemming. In *Proceedings of CLEF*.

[11] Craig Macdonald, Rodrygo L.T. Santos, Iadh Ounis, and Ben He. 2013. About Learning Models with Multiple Query-dependent Features. *ToIS* 31, 3 (2013).

[12] Donald Metzler and W. Bruce Croft. 2005. A Markov random field model for term dependencies. In *Proceedings of SIGIR*.

[13] Keith J. O'Hara, Douglas Blank, and James Marshall. 2015. Computational Notebooks for AI Education. In *Proceedings of FLAIRS*.

[14] Iadh Ounis, Gianni Amati, Vassilis Plachouras, Ben He, Craig Macdonald, and Christina Lioma. 2006. Terrier: A High Performance and Scalable Information Retrieval Platform. In *Proceedings of OSIR*.

[15] Fernando Perez and Brian E Granger. 2015. *Project Jupyter: Computational narratives as the engine of collaborative data science.* Technical Report. http://archive.ipython.org/JupyterGrantNarrative-2015.pdf

[16] Trevor Strohman, Donald Metzler, Howard Turtle, and W Bruce Croft. 2005. Indri: A language model-based search engine for complex queries. In *Proceedings of the International Conference on Intelligent Analysis*, Vol. 2. Citeseer, 2–6.

[17] Terrier.org. 2016. Learning to Rank with Terrier. (2016). http://terrier.org/docs/v4.2/learning.html

[18] John Williamson. 2017. CS1P: Running Jupyter from the command line. (2017). https://www.youtube.com/watch?v=hqpuC0YLbpM

[19] Qiang Wu, Chris J. C. Burges, Krysta M. Svore, and Jianfeng Gao. 2008. *Ranking, Boosting, and Model Adaptation.* Technical Report MSR-TR-2008-109. Microsoft.

[20] Hugo Zaragoza, Nick Craswell, Michael Taylor, Suchi Saria, and Stephen Robertson. 2004. Microsoft Cambridge at TREC-13: Web and HARD tracks. In *Proceedings of TREC*.

In [7]:

```
import org.apache.spark.sql.functions.col

val joined = TFIDF
    .join(BM25, TFIDF.col("qid") === BM25.col("qid"))
    .toDF("qid", "query", "TFIDF", "qidBM25", "queryBM25", "BM25")
    .select($"qid", $"query", $"BM25" - $"TFIDF")
    .orderBy($"BM25" - $"TFIDF")

joined.filter(row => row.get(2) != 0.0).collect
```

```
[Stage 33:=================================================>      (178 + 2) / 200]

joined = [qid: string, query: string ... 1 more field]
```

Out[7]:

427	uv damage eyes	-0.05857478287366785
424	suicides	-0.04259925243625749
445	women clergy	-0.0420222350661259
408	tropical storms	-0.010144724844642017
401	foreign minorities germany	-0.003744521822722807
421	industrial waste disposal	-0.003600929080181753
422	art stolen forged	-0.0033479627985089078
411	salvaging shipwreck treasure	-0.002034006395408086
403	osteoporosis	-0.0016673813718870445
404	ireland peace talks	-9.332068582119102E-4
436	railway accidents	-8.390389246867025E-4
412	airport security	-5.916538929868764E-4
446	tourists violence	1.7459024478422291E-4
405	cosmic events	0.002750823278844128
433	greek philosophy stoicism	0.007707636798678946
443	u s investment africa	0.011348999308453525
418	quilts income	0.012175629147371914
448	ship losses	0.01879636692115455
419	recycle automobile tires	0.06216480659093851

Figure 3: Comparing the results of two different retrieval runs.

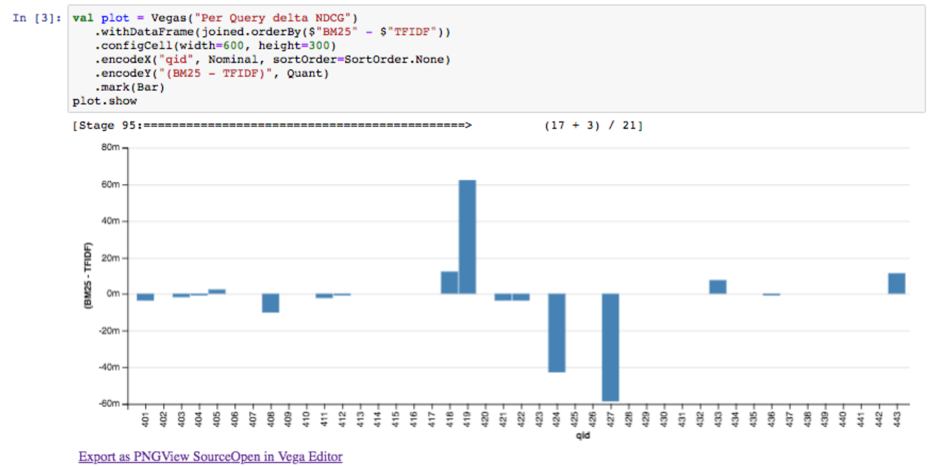

Figure 4: Graphically displaying the per-query differences between different retrieval runs.