Combining Terrier with Apache Spark to create Agile Experimental Information Retrieval Pipelines

Craig Macdonald
University of Glasgow
Glasgow, Scotland, UK
craig.macdonald@glasgow.ac.uk

ABSTRACT

Experimentation using information retrieval (IR) systems has traditionally been a procedural and laborious process. Queries must be run on an index, with any parameters of the retrieval models suitably tuned. With the advent of learning-to-rank, such experimental processes (including the appropriate folding of queries to achieve cross-fold validation) have resulted in complicated experimental designs and hence scripting. At the same time, machine learning platforms such as Scikit Learn and Apache Spark have pioneered the notion of an experimental *pipeline*, which naturally allows a supervised classification experiment to be expressed as a series of stages, which can be learned or transformed. In this demonstration, we detail Terrier-Spark, a recent adaptation to the Terrier IR platform which permits it to be used within the experimental pipelines of Spark. We argue that this (1) provides an agile experimental platform for information retrieval, comparable to that enjoyed by other branches of data science; (2) aids research reproducibility in information retrieval by facilitating easily-distributable notebooks containing conducted experiments; and (3) facilitates the teaching of information retrieval experiments in educational environments.

ACM Reference Format:
Craig Macdonald. 2018. Combining Terrier with Apache Spark to create Agile Experimental Information Retrieval Pipelines. In *SIGIR '18: The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, July 8–12, 2018, Ann Arbor, MI, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3209978.3210174

1 INTRODUCTION

Information retrieval (IR) has evolved through the systemic introduction of supervised machine learning within commercial search engines. Traditional theoretically-founded term weighting models (e.g. BM25, language models, PL2) that could be combined with other sources of evidence by learning parameters (circa 2004) have been replaced with advanced rewritten query formulations (e.g. Sequential Dependence [9]), and learning-to-rank techniques [6] that combine evidence from many features to re-rank an initial set of retrieved documents. Learning-to-rank has been integrated within Terrier (from version 4.0, 2015), Solr (from version 6.0, 2017), while for Elastic, learning-to-rank remains an experimental plugin.

While the techniques to perform effective ranking have therefore improved and been integrated within platforms, the tools and platforms to allow agile *experimentation* for information retrieval researchers and practitioner have not kept up. Indeed, in this paper, we draw distinctions between the deployers of a search engine, who need to build and deploy an effective end-to-end search product, and a researcher, who may be developing techniques, trying out features, perhaps in an offline manner, before publishing, or deployment into a search engine. The teaching of IR experimentation concepts to graduate students also falls into this latter category.

Agile experimentation within other branches of supervised learning has been facilitated in recent years by the adoption of programming languages such as Python and Scala, both having clean syntax, and extensible libraries. We draw attention in particular to the Scikit Learn toolkit from Python, where each supervised technique expose a `fit()` method for training and a `transform()` method for applying the trained model. Inspired by Scikit Learn, Apache Spark's MLib API similarly defines `fit()` and `transform()` methods, which can be combined in series into *Pipelines*. In this paper, we build upon Apache Spark MLib APIs using the Terrier IR platform, and demonstrate how such Pipeline primitives allow the easy expression of complex IR experiments.

We argue that the resulting Terrier-Spark, used within a Jupyter notebook UI (1) provides an agile experimental platform for information retrieval, comparable to that enjoyed by other branches of data science; (2) aid research reproducibility in information retrieval by facilitating easily-distributable notebooks containing conducted experiments; and (3) facilitates the teaching of information retrieval experiments in universities.

The structure of this paper is as follows: Section 2 highlights the main requirements for an experimental IR platform; Section 3 summarises the current Terrier platform; Section 4 introduces Terrier-Spark and how to conduct IR experiment using it; Section 5 discusses the advantages of combining Terrier-Spark with Jupyter notebooks. Concluding remarks follow in Section 6.

2 IR PLATFORM REQUIREMENTS FOR CONDUCTING EMPIRICAL EXPERIMENTS

Below, we argue for, in our experience, the main attributes of an experimental IR platform, described in terms of required functionalities. In addition, there exist 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.

SIGIR '18, July 8–12, 2018, Ann Arbor, MI, USA
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5657-2/18/07.
https://doi.org/10.1145/3209978.3210174

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, Galago etc. 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 gap).

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 [16] & PL2F [7] 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 [8]. 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 3), and how it can be adopted within a Spark environment to meet these in a more agile fashion (Section 4).

3 BACKGROUND ON TERRIER

Terrier [11] 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.

However, we have increasingly found that a commandline API was not suited for all purposes. For instance, the chaining of the outcomes 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, 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.

4 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.

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, which 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 [15], which are often the most effective [2, 6]).

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 data frame. As Terrier is written in Java, and Scala and Java both are JVM-based languages, Terrier can run "in-process". In the future, we will investigate accessing remotely hosted Terrier servers, similar to that performed by Elastic's[3] and Solr's[4] Spark tools.

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

4.1 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 instead 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

[1] http://pandas.pydata.org/ [2] http://scikit-learn.org/
[3] https://www.elastic.co/guide/en/elasticsearch/hadoop/current/spark.html
[4] https://github.com/lucidworks/spark-solr

Component	Inputs	Output	Parameters
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.

```
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.

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

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

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

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 meanNDCG = new NDCGEvaluator(20).evaluate(r2)
```

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

```
//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.

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 permits knowledge of the relevant and non-relevant documents to be added to the dataframe by joining with the contents of the qrels file, before evaluation.

While clearly more verbose than the simpler commandline API, Listing 2 demonstrates equivalent functionality, and clearly highlights the needed data for 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.

4.2 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 the use of a gradient descent algorithm to tune the c parameters. However, at this stage we do not yet have a paralleised algorithm implemented that would make best use of a clustered Spark environment.

4.3 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 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 [8]. The resulting 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 [13]. 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 [15]; Similarly, Transformers for adapting the query representation, for example by applying query-log based expansions [5] or proximity-query rewriting such as Sequential Dependence models [9]. 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.

5 CONDUCTING IR EXPERIMENTS WITHIN A JUPYTER NOTEBOOK ENVIRONMENT

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 the corresponding analysis tables or visualisations.

Jupyter notebooks are increasingly used to share the algorithms and analysis conducted in machine learning research papers, significantly aiding reproducibility [12]. Indeed, in their report on the Daghstuhl workshop on reproducibility of IR experiments [4], Ferro, Fuhr *et al.* note that sharing of code and experimental methods would aid reproducibility in IR, but omitted any mention of notebooks.

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 [3, 14], as well as a plethora of data science or machine learning courses [12]. O'Hara et al. [10] 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. In particular, our experience in teaching graduate-level IR using Terrier suggests that reliance on a procedural commandline mechanism for interacting with the IR platform is limiting the types of experiments achievable by the students within the constraints of a graduate coursework. In contrast, our recent experience in teaching a text analytics/machine learning course to a similar student cohort suggests that the students are able to easily test and experiment with techniques to aid their understanding through the agile nature of a notebook environment. We believe that Terrier-Spark can bring the same advantages to conducting modern (e.g. learning-to-rank) IR experiments.

6 CONCLUSIONS

In this demonstration paper, we have introduced a toolkit to perform IR experimentation within the Spark distributed computing engine, by building upon and integrating the Terrier IR platform. As argued above, we believe this will aid in (1) providing an agile experimental platform for IR, comparable to that enjoyed by other branches of data science; (2) aid research reproducibility in information retrieval by facilitating easily-distributable notebooks that demonstrate the conducted experiments; and (3) facilitate the teaching of information retrieval experiments in educational environments. Terrier-Spark has been released as open source, and is available from

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

along with example Jupyter notebooks.

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] 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/

[4] 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).

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

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

[7] 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.*

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

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

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

[11] 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.*

[12] 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

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

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

[15] 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.

[16] 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.*

[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.