IRevalOO: An Object Oriented Framework for Retrieval Evaluation

Kevin Roitero
University of Udine
Udine, Italy
roitero.kevin@spes.uniud.it

Eddy Maddalena
University of Southampton
Southampton, U.K.
e.maddalena@soton.ac.uk

Yannick Ponte
University of Udine
Udine, Italy
ponte.yannick@spes.uniud.it

Stefano Mizzaro
University of Udine
Udine, Italy
mizzaro@uniud.it

ABSTRACT

We propose IRevalOO, a flexible Object Oriented framework that (i) can be used as-is as a replacement of the widely adopted trec_eval software, and (ii) can be easily extended (or "instantiated", in framework terminology) to implement different scenarios of test collection based retrieval evaluation. Instances of IRevalOO can provide a usable and convenient alternative to the state-of-the-art software commonly used by different initiatives (TREC, NTCIR, CLEF, FIRE, etc.). Also, those instances can be easily adapted to satisfy future customization needs of researchers, as: implementing and experimenting with new metrics, even based on new notions of relevance; using different formats for system output and "qrels"; and in general visualizing, comparing, and managing retrieval evaluation results.

CCS CONCEPTS

• **Information systems → Test collections**;

KEYWORDS

TREC, evaluation, test collections, trec_eval

ACM Reference Format:
Kevin Roitero, Eddy Maddalena, Yannick Ponte, and Stefano Mizzaro. 2018. IRevalOO: An Object Oriented Framework for Retrieval Evaluation. In *SIGIR '18: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, July 8–12, 2018, Ann Arbor, MI, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3209978.3210084

1 INTRODUCTION

In Information Retrieval (IR), effectiveness evaluation is an essential process. A widely-adopted methodology is evaluation by means of a test collection, which consists of: a set of documents, a test suite of information needs descriptions (called queries or topics), and the ground-truth of a set of relevance judgements made by experts for a subset of the topic-document pairs. Campaigns such as TREC, NTCIR, FIRE, CLEF, INEX, etc. evaluate effectiveness of systems by comparing their output with the ground-truth, using standard or custom evaluation metrics, often several of them working in different configurations. To facilitate the entire evaluation process, ad-hoc software tools have been created. trec_eval is probably the most common: it is used to evaluate the results of the participants to TREC competitions, as well as in other initiatives. trec_eval serves well its purpose but it is not free from limitations.

In this paper, we present a more general framework that aims to extend it, as well as similar IR evaluation software and tool-kits. Our system, called IRevalOO, is conceived according to the Object-Oriented (OO) programming paradigm and, more precisely, it is an OO framework. Besides allowing system evaluations as trec_eval, by exploiting the advantages of OO frameworks, IRevalOO offers a set of useful features: the easy implementation of new custom evaluation metrics, the management of multiple types of measurement scales, the handling of different input formats, and the customization of measurement sessions and results visualisation. The paper is structured as follows: Section 2 describes trec_eval and other evaluation tools, as well as object oriented frameworks, Section 3 presents IRevalOO and its evaluation, Section 4 concludes the paper.

2 BACKGROUND

Trec_eval, and Other Evaluation Tools. The need for evaluating the performance of systems has led to the creations of many toolkits and software for facilitating the measurement process. Trec_eval is probably a sort of de facto standard. Born in the early 90s, maintained by NIST, it is considered a milestone of the evaluation software and it inspired most of the other toolkits used in the different initiatives. trec_eval evaluates system effectiveness by comparing the systems results with a ground truth consisting of a set of relevance judgements expressed by human experts. trec_eval takes in input the *qrels* files, which are the systems output in the TREC format, the and a set of *runs* where each run consists of an execution of a system over a topic. trec_eval returns as output various measures obtained using different metrics, like Mean Average Precision (MAP), R-Precision, and many others. trec_eval allows to: specify the metric(s) to use; specify the format for the input and for the output files; and compute the evaluation for each topic. It provides some useful parameters to customize the evaluation process (as described in the README file), although these parameters are sometimes difficult to use and configure by the users.

Trec_eval is a valuable tool for the IR research community, but it is not free from limitations: it is written in C, using legacy technologies, and it is not truly cross platform; researchers face some difficulties during its installation, configuration, and customization; customization is far from being simple and agile, and a user who needs to implement new features, or modify those already implemented, has to modify and even re-write several software modules. This can be a time consuming activity which also requires advanced programming skills. These limitations are probably the reasons to develop other evaluation tools. Most of them though adopt a

SIGIR '18, July 8–12, 2018, Ann Arbor, MI, USA

ACM ISBN 978-1-4503-5657-2/18/07.
https://doi.org/10.1145/3209978.3210084

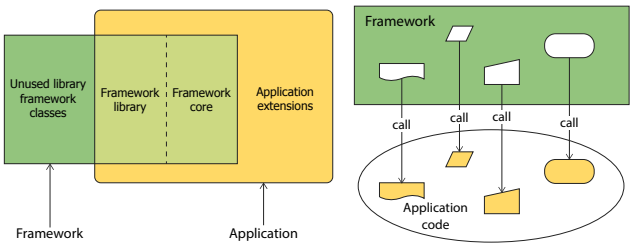

Figure 1: Framework architecture and Inversion of control.

sort of re-implementation approach, like the TREC_Files Evaluator [2], Pytrec eval [10], sometimes adding some functionalities, like Terrier [8], MATTERS [6], or adapting to a different domain like NTCIREVAL [9]. We propose a different approach: to implement an evaluation tool as an Object Oriented (OO) framework.

Object-Oriented Frameworks. An OO framework is a reusable, semi-complete application that can be specialized to produce custom applications, and consists in an abstract core representation of a specific domain [3, 4, 7]. A framework is designed to be specialised by the developer, who can extend its core to his/her specific application, which is called *instance* or *extension*. Figure 1 (left) shows the general structure of a framework. Differently from the traditional software libraries, which are invoked by the developer's code, the modules written by the developer will be invoked by the framework: this is called "inversion of control" (see Figure 1, right).

OO frameworks provide a set of advantages to both developers (i.e., users of the framework) and end-users (i.e., users of the instances of the framework) like modularity, reusability, extensibility [3]. OO frameworks define generic and high-level components that can be re-used by developers to create new applications; this avoids to waste resources in re-implementing the same solutions to similar problems, which is a quite common practice in many domains.

3 IREVALOO

We now turn to presenting our IRevalOO software. We discuss motivations and aims, usage scenarios, requirements, design, examples of use, and evaluation. IRevalOO (currently in beta release and still undergoing some refactoring, tuning, and optimizations) is implemented in Java 8 and it can be freely downloaded at https://github.com/IRevalOO/IRevalOO_Software.

Motivations and Aims. Both individual researchers and organizers of IR evaluation initiatives can obtain multiple benefits by the redesign of their evaluation software as an OO framework. The missing functionalities can be easily implemented by instances of the framework. To extend and customize an OO framework is much easier than modifying the original evaluation software, which is often written in a low-level language that is not as abstract as the OO paradigm. trec_eval is a concrete example that exemplifies these remarks: its limitations, such as the customization of the file loader, the results manager, the possibility to visually compare different metrics, the possibility to define and test new metrics, relevance, etc. can be easily implemented by IRevalOO instances. trec_eval is written in C and, due to the lack of an adaptive design, it is not naturally suitable to be extended to specific evaluation applications.

We propose IRevalOO, an OO framework that: (i) can be used as-is as a replacement of trec_eval, and (ii) can be easily extended/instantiated to implement different scenarios of test collection based retrieval evaluation. IRevalOO, by exploiting several advantages of OO paradigm like design patterns and OO frameworks in general, aims to be flexible by allowing its own extensions which let users to define new items, such as new metrics, relevance kinds, input and output formats, etc. We also aim at an easily configurable and customizable tool, in terms of, e.g., selecting output visualization, summarization, and export formats, and/or which metric, which topic or run sets should be included in the evaluation.

Three Scenarios. A first typical scenario of IRevalOO involves a user who wants to evaluate his/her own system on a test collection of a previous TREC edition by using the standard evaluation metrics. This need can be satisfied by both trec_eval and IRevalOO. The situation changes considering a second more complex scenarios. If the user has more specific needs, as the definition of a new input format, a new summary representation, or a different set of metrics, maybe even based on a new kind of relevance, trec_eval as-is would not be adequate. On the contrary, IRevalOO can easily be adapted to the user needs. A third interesting scenario can involve a user who needs to implement and test a novel metric called Uncertain MAP (UMAP), consisting in a variation of MAP that takes into account uncertainty in the relevance judgements.

Requirements and Design. On the basis of the aims and scenarios, we can list the following functional requirements of IRevalOO: import a test collection (i.e., documents, topics, runs), selecting the desired source and the input format; import more than one run, with the option of consider more than one system; allow the creation of new metrics and/or new metric sets; allow a customized management of results visualization and export format; create a summary report and provide a verbose log of the evaluation process; guarantee the compatibility with the original format of TREC commands; and create and manage a different kind of relevance.

The UML diagram shown in Figure 2 summarizes the structure of IRevalOO. It shows the main packages and classes of the software (about 50% of IRevalOO classes are in the diagram). IRevalOO is made up of five main packages (plus an `exceptions` package not shown). The `control` package is the controller of the framework. It contains the `EvaluationManager` class, that embodies the overall workflow: it uses the classes and methods in the other packages and manages the data flow as well. The other packages usually contain abstract classes to be implemented by the specific application. For example, the package `testcollection` contains the abstract classes `Topic` and `Document` that, with `Qrel`, form the `Collection`.

The package `relevance` contains `RelevanceType`, an abstract class that models the abstract concept for different kinds of relevance, in this case "Binary", "Numeric", and "Category". The relevance defined in the instance of the framework can be modelled as one of these categories. The package `run` models the concept of a run, or rather an evaluation of a system over a set of topics. The package `metric` contains two sub-packages: `metrics.definition`, containing the classes to define new metrics, and `metrics.results`, that uses the defined metrics to compute the metric values over the runs and export the results of the computation.

The packages `run` and `metrics.results` use some abstract classes (e.g., `ResultExporter`) to provide a set of methods which can be used to customize both the loader and the exporter of the data, adapting them to new formats. For example in this way a developer can easily import the run files from an XML file, a database, etc.

Examples of Use. We briefly describe how IRevalOO can be used by application developers. To use the framework, the developer has

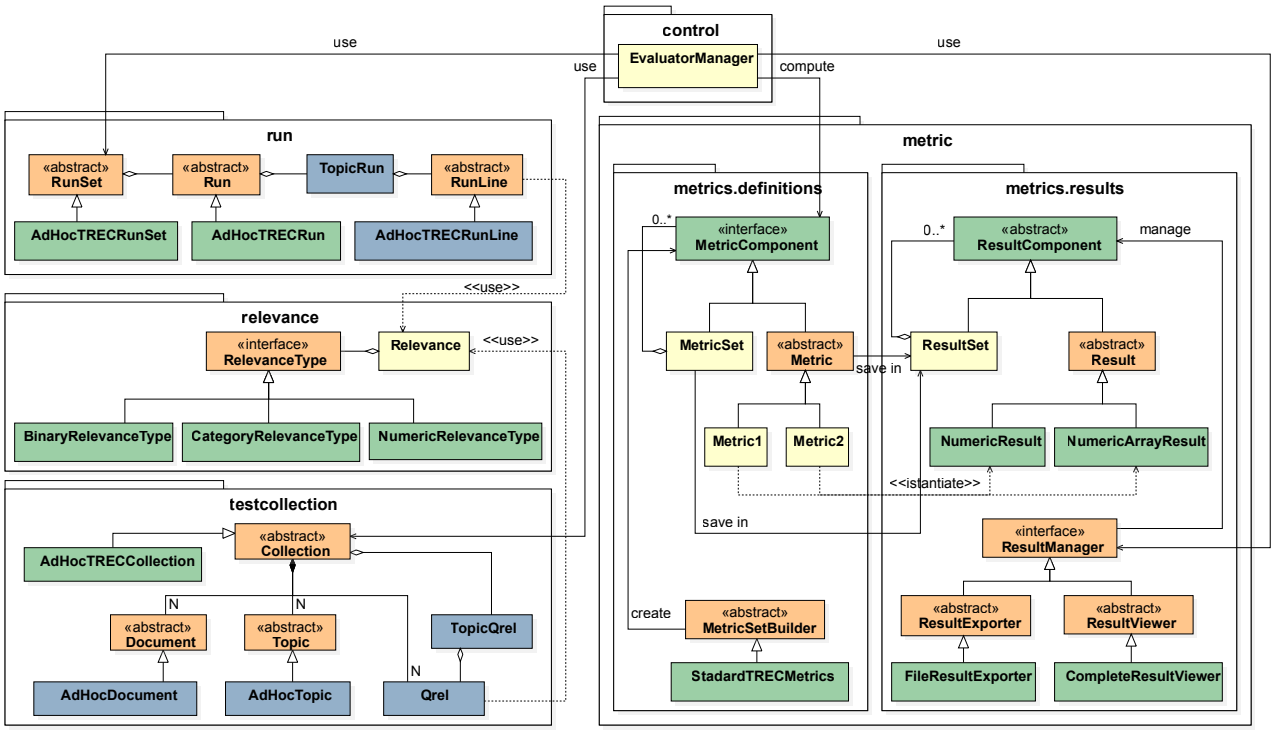

Figure 2: UML diagram of the main components of IRevalOO.

```
1  public static void main(String[] args) {
2    String qrel = ".../qrels.txt";
3    String run = ".../run";
4    Collection c = new  AdHocTRECCollection(new
5      NumericCategoryRelevanceType(7,2), "","",qrel);
6    RunSet r = new
7      AdHocTRECRunSet(new NumericRelevanceType(),run);
8    MetricSet m = new MetricSet();
9    m.add(new BPref());
10   m.add(new PatN(10));
11   m.add(new R());
12   EvaluatorManager em = new EvaluatorManager(c,r,m);
13   em.evaluate();
14   em.showResults(new OverAllResultViewer());
15   String outFile = ".../example.out";
16   em.exportResults(new FileResultExporter(outFile));
17 }
```

Figure 3: The Java code for a typical usage of IRevalOO.

to create instances of it. An instance which reflects the first typical scenario described above follows the schema detailed in Figure 3:

1. instantiate the collection (line 4);
2. instantiate the run set (line 6);
3. instantiate the metric set (line 8);
4. instantiate an Evaluation manager, which takes in input the test collection, the metrics, and the runs (line 12);
5. (optional) set/customize the options for the Evaluation manager;
6. start the retrieval evaluation (line 13);
7. manage the results according to the user preferences (line 15).

IRevalOO is also adequate for the other two scenarios previously described: Figure 4 shows an example of a new category of relevance definition, where the binary relevance judgement is enriched with a degree of uncertainty; and Figure 5 shows the skeleton of a class implementing the possible new metric UMAP.

Evaluation: Correctness and Efficiency. Evaluation of a framework is not simple and can usually be done with precise results only at the end of life cycle of the framework, when it has no practical usefulness [1, 3–5]. However, we can provide an evaluation

```
1  public class UncertaintyRelevanceType
2                    implements RelevanceType {
3    public UncertaintyRelevanceType() {}
4    public double readValue(Object obj)
5                    throws IRevalOOException {
6      if (obj.value.equals("-")) {return -1;}
7      else if (obj.value.equals("NOT RELEVANT")){
8        return 0;
9      } else if(obj.value.equals("RELEVANT")){
10       return 1;
11     } else {
12       throw new IRevalOORelevanceException(
13         "unreadable category relevance " + obj.value);
14     }
15   }
16   public double readUncertainty(Object obj)
17                   throws IRevalOOException {
18     if(obj.confidence >= 0 && obj.confidence <= 1)
19       return obj.confidence;
20     else {
21       throw new IRevalOORelevanceException(
22         "unexpected confidence value " +
23         obj.confidence);
24     }
25   }
26   public String toString(Object obj) {
27     return "The object is " + obj.value +
28       " with a confidence of " +obj.confidence;
29   }
30 }
```

Figure 4: The code to create a new kind of relevance.

of IRevalOO in terms of some software metrics, correctness and efficiency; as well as exploiting trec_eval as a comparison baseline.

Table 1 shows some software metrics of trec_eval, both complete (in the second column) and the part of it corresponding to IRevalOO implementation (in the third column), and IRevalOO (fourth column). IRevalOO is not very complex (80 classes in total), and it features fewer lines of code than trec_eval, both when considering trec_eval full implementation and when not taking into account the part of trec_eval that has not been implemented in IRevalOO yet. This is mainly due to the fact that Java is a slightly higher level

```
public class UMAP extends Metric {
  public UMAP() {
    acronym = "UMAP";
    completeName = "Uncertain Mean Average Precision";
    ...
  }
  public Result computeTopicResult(TopicRun topicRun,
                                   Collection c) {
    for (int i = 0; i < retrieved; i++) {
      ... // Iterate over retrieved docs
      RunLine rl = topicRun.getRun().get(i);
      Qrel q =
        c.getQrel(rl.getIdDoc(),rl. getIdTopic());
      ... // The core part of the metric
    }
    umap = ... // UMAP computation
    return new NumericResult(topicRun.getIdTopic(),
                             this, umap);
  }
  public Result computeOverallResult(ResultSet res,
                                     Collection c) {
    return NumericResult.arithmeticMeanResults(res,
                                          this, c);
  }
}
```

Figure 5: The skeleton of the code to create a new metric.

Table 1: Statistical data

	trec_eval (C)	trec_eval (impl.)	IRevalOO
Lines of code	8030	5087	2475
Lines of comment	1796	1158	988
Total lines	9826	5993	3463
Classes / Methods	- / -	- / -	80 / 260

language than C and allows to create a more compact source code — that is more easy to understand and maintain as well.

Concerning correctness, IRevalOO has been tested on all trec_eval examples and on many real TREC test collections: the adHoc tracks of TREC2, 3, 5, 6, 7, 8, and TREC2001, the Robust tracks of 2004 and 2005, the Terabyte tracks of 2004–2006, the Web tracks of 2011–2014, and the Million Query tracks of 2007–2009. It has been tested with different relevance levels (binary, three-level, etc.), and it always provides exactly the same results as trec_eval. Concerning efficiency, IRevalOO has been compared to trec_eval using data from two real TREC tracks: TREC8 AdHoc (AH99), and TREC2007 Million Query (MQ07). These two datasets are complementary: one features a high number of runs and the other one a high number of topics. Then, to run a sort of stress test, we created an artificial collection featuring 2000 topics and 200 runs. It has been created considering 1000 documents retrieved by each system, and a set of relevance judgements allocated considering a plausible distribution; although artificial, this collection is realistic, and it represents a sort of "worst case" scenario. Datasets details are shown in Table 2.

We measured the execution time on an ordinary laptop: a MacBook Pro, Retina, 13-inch, Mid 2014, 3 GHz Intel Core i7 processor, 16 GB of 1600 MHz DDR3 RAM, SSD drive). IRevalOO turns out to be always slower than trec_eval. However, trec_eval speed-up is much smaller for large datasets: it goes from around 1.7 times for AH99 to around 1.2 times (1 would mean no speed-up) for the larger artificial collection. When the amount of data and the overall execution time grow, trec_eval and IRevalOO time performance are of the same order of magnitude. IRevalOO time performance seems reasonable: the user will not be too harmed when going from 11s to 19s. Indeed, the difference is not too large. Also, it is important to remark that trec_eval has probably undergone multiple efficiency tuning and improvements during its more than twenty years lifetime, whereas IRevalOO has not been carefully optimized yet (although we did pay some attention to efficiency by, for example, adopting Hash Maps and relying on memoization techniques). Furthermore, the small lack of efficiency is of course balanced by the new added functionalities offered by IRevalOO, that should hopefully save the, probably more precious, researcher's time.

Table 2: Efficiency test: datasets and results.

Dataset	no. topics	no. systems	trec_eval	IRevalOO
AH99	50	129	11s	19s
MQ07	≈1,000	29	235s	490s
Artificial	2,000	200	709s	848s

4 CONCLUSIONS AND FUTURE WORK

We presented IRevalOO, an Object Oriented framework that can replace and extend the well known trec_eval software as well as other evaluation tools. IRevalOO allows the users to easily implement instances and define, when needed, both new main components as relevance or metrics and useful features for evaluation analysis. Instances of the framework can also be easily created to replace the software used in other initiatives of retrieval evaluation. In the future we plan to provide different instances which will emulate and extend the software used in other initiatives (like, e.g., NTCIREVAL). For the future, after defining a proper set of test cases using a unit test suite like JUnit, we plan to do some refactoring and optimization, as well as to extend the framework by modelling other domain aspects like the abstract concepts of "document" and "topic" which will allow the user to customize test collection components, to experiment with new test collections, and to adapt the framework to specific needs of other initiatives. Furthermore, we intend to provide a graphic user interface that will allow less expert users to interact with the framework and its instances. Finally, going back to the efficiency issue, we plan to use profiling and various optimization techniques to fine-tune IRevalOO and improve its efficiency if needed. Given the public and free release of the software we expect feedback and improvements from the community.

REFERENCES

[1] Jan Bosch, Peter Molin, Michael Mattsson, and PerOlof Bengtsson. 2000. Object-oriented Framework-based Software Development: Problems and Experiences. *ACM Comput. Surv.* 32, 1es, Article 3 (March 2000).

[2] Savvas A. Chatzichristofis, Konstantinos Zagoris, and Avi Arampatzis. 2011. The TREC Files: The (Ground) Truth is out There. In *Proc. of the 34th ACM SIGIR*. ACM, New York, NY, USA, 1289–1290. https://doi.org/10.1145/2009916.2010164

[3] Mohamed Fayad and Douglas C Schmidt. 1997. Object-oriented application frameworks. *Commun. ACM* 40, 10 (1997), 32–38.

[4] Garry Froehlich, James Hoover, Ling Liu, and Paul Sorenson. 1998. Designing object-oriented frameworks. *University of Alberta, Canada* (1998).

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, Boston, MA, USA.

[6] Information Management Systems Research Group. 2017. MATTERS. http://matters.dei.unipd.it. Last access: 2017-01-22.

[7] Ralph E Johnson and Brian Foote. 1988. Designing reusable classes. *Journal of object-oriented programming* 1, 2 (1988), 22–35.

[8] University of Glasgow. 2017. TERRIER homepage. http://terrier.org. Last access: 2017-01-22.

[9] Tetsuya Sakai. 2017. NTCIREVAL home page. http://research.nii.ac.jp/ntcir/tools/ntcireval-en.html. Last access: 2017-01-08.

[10] Alberto Tonon. 2017. Pytrec Eval download. https://github.com/eXascaleInfolab/pytrec_eval. Last access: 2017-01-08.