A Lightweight Framework for Reproducible Parameter Sweeping in Information Retrieval

Richard Eckart de Castilho
Iryna Gurevych
Motivation

Problems with experiments
What happens if the original researcher leaves?
How can we preserve the experiment in a repeatable way?

Experimentation Framework

Requirements
Must get out of the way
Must facilitate repeatable experiments
Must preserve experiment results
Our Goal

Conduct our Information Retrieval experiments

1. with a **lightweight declarative** set up

2. with **parameter sweeping**

3. in a **reproducible** manner

Generic core framework for arbitrary experiments
Extensions for application domains (e.g. IR)
A typical IR experiment

(1) Document Preprocessing Task

(2) Topic Preprocessing Task

(3) Retrieval Evaluation Task

Parameters
- Data set
- Stop word list
- IR model
- Fields
- Pruning
- …
DKPro Lab Framework

- High-level experimentation framework
- Modular architecture
- Lightweight implementation

- Declarative all-in-one style for reproducible experiments
- Model tasks and data dependencies
- Parameter sweeping

- Preserve results
- Generate reports

| Java | Groovy | or |

Oct 28, 2011 | DESIRE 2011 | Technische Universität Darmstadt UKP Lab - Prof. Dr. Iryna Gurevych | Richard Eckart de Castilho |
Task

- Configurable unit of work
- Can be executed multiple times with different configurations
- Adapts between the Lab and the low-level experiment code

```java
class DocumentIndexingTask extends ExecutableTaskBase {
    @Discriminator File documentsPath;
    @Discriminator String language;
    @Discriminator DiscriminableClosure termSelector;

    public void execute(TaskContext aContext) throws Exception {
        // read files from documents
        // apply language-specific pre-processing
        // select terms to index from document
        // write document terms to the index
    }
}
```

Parameters

Experiment Logic
Parameter

- Supported are all types with \( v1.toString() == v2.toString \Leftrightarrow v1.equals(v2) \)
  - String, Integer, Boolean, …, File, URL, …

- Collection types need to have a stable element order
  - Good: ArrayList
  - Bad: HashSet

- Closures can be parameters
  - Inject pieces of code for maximum flexibility

Dimension \( dimInnerProduct = new ClosureDimension(\"innerProduct\", [\"
  \"Cosine\": { vec1, vec2 -> cosine(vec1, vec2) },
  \"Lesk\": { vec1, vec2 -> overlap(vec1, vec2) }, ]);\)

\[\text{def score = } innerProduct(\text{queryVector, documentVector})\]
Parameter Space

- Space of all possible parameter combinations

```java
ParameterSpace pSpace = [
    dimDataSet, dimIrModel, dimTermSelector ];

def dimDataSet = Dimension.createBundle(
    "dataSet", dataSetEn, dataSetDe);

def dataSetEn = [
    __bundleId: 'dataSetEn',
    language: 'en',
    documentsPath: 'src/test/resources/en/documents' as File,
    topicsPath: 'src/test/resources/en/topics' as File,
    judgementFile: 'src/test/resources/en/judgements/judgements.qrels' as File];

def dataSetDe = [
    __bundleId: 'dataSetDe',
    language: 'de',
    documentsPath: 'src/test/resources/de/documents' as File,
    topicsPath: 'src/test/resources/de/topics' as File,
    judgementFile: 'src/test/resources/de/judgements/judgements.qrels' as File];
```

---

### Document Preprocessing Task

1. **Topic Preprocessing Task**

2. **Retrieval & Evaluation Task**

3. **Evaluation Results**

---

@Discriminator String language;
@Discriminator File documentsPath;
Data dependencies

- Lab provides new context for each task execution (think directory)
- Data read/stored by keys (think files and directories)
- Data can be imported from other tasks via these keys

```
addImport(JUDGEMENT_FILE_KEY, judgementFile);
addImportLatest(KEY_TOPIC_XMI, KEY_XMI, topicPreprocessingTask);
addImportLatest(KEY_DOC_INDEX, KEY_INDEX, docIndexingTask);
```

Diagram:
- (1) Document Preprocessing Task
  - Docs
  - Preprocess & Index
  - Indexed Documents
- (2) Topic Preprocessing Task
  - Topics
  - Preprocess
  - Processed Topics
- (3) Retrieval & Evaluation Task
  - CollectionReaderDescription.xml
  - CollectionReaderDescription.xml.html
  - DISCRIMINATORS.txt
  - JudgementFile.txt
  - METADATA.txt
  - PROPERTIES.txt
  - Queries.txt
  - SearchResults.txt.gz
  - TrecTool-byQuery.csv
  - TrecTool.gz
Batch Task

- Parameter sweep over a set of tasks
- Data dependencies control task execution order
- Tasks executed only when affected by a parameter change

BatchTask batchTask = [parameterSpace: [dimDataSet, dimIrModel, dimTermSelector],
tasks: [docIndexingTask, topicPreprocessingTask, evaluationTask],
reports: [TrecBatchReport, IrBatchReport, PairedTTestReport]]
Probes

- Probes record information about a task
- Design pattern, not an API
- Built directly into the tasks
UIMA Task with Probes

DocumentIndexingTask

AnalysisEngineDescription getAnalysisEngineDescription(TaskContext ctx) {
  File output = ctx.getStorageLocation(KEY_XMI, READWRITE);
  File outLucene = ctx.getStorageLocation(KEY_INDEX_LUCENE, READWRITE);

  return createEngine(
    createEngine(BreakIteratorSegmenter),
    createEngine(TreeTaggerPosLemmaTT4J),
    createEngine(StopWordRemover, ...),
    createEngine(SnowballStemmer, ...),
    createEngine(IndexTermAnnotator, ...),
    createEngine(LuceneIndexWriter, PARAM_INDEX_PATH, outLucene.path),
    createEngine(DocumentProbe),
    createEngine(ShallowFeaturesProbe))
};
Post-process task results and probe output

```java
BatchTask batchTask = {
    parameterSpace: [dimDataSet, dimIrModel, dimTermSelector],
    tasks: [docIndexingTask, topicPreprocessingTask, evaluationTask],
    reports: [TrecBatchReport, IrBatchReport, PairedTTestReport]
};
```
DKPro Lab IR Extension Package

Data models
- Judgment set
- Document set
- Topic set
- Evaluation result set

Probes
- Document text
- Raw and processed topics
- Search results
- Type/Token ratio

Reports
- Result type report
  - True/false positive/negative stats
- TRECeval report
  - Precision, recall, etc.
- Accuracy report
  - Accuracy by score threshold
- Correlation report
  - Pearson/Spearman correlation
- Significance report
  - Paired-T test
Anatomy of a Lab Experiment

- **Basic setup**
  - 1 file describing the experimental setup and parameters
  - 1 file per task (recommended)

- **Optional setup (shared between experiments)**
  - 1 file per report
  - 1 file per data model

- **DKPro Lab provides convenience classes for**
  - Scalable PDF charts (based on JFreeChart)
  - Flexible data tables (render as Excel, CSV, TWiki)
Summary and Outlook

Summary
- DKPro Lab is a generic parameter sweep experiment framework
- Lightweight and very extensible
- Facilitates repeatable experiments
- Demonstrated application for Information Retrieval

Outlook
- Deploy and run experiments on a compute cluster, e.g. using Apache Hadoop
- Add extensions for other areas, e.g. Machine Learning

Open Source
- Apache License
- Currently the core framework and an ML example
Thank you!

Questions?

Thanks to all my colleagues at UKP!

Used templates from www.presentationmagazine.com