DL-Learner Manual

Jens Lehmann

November 4, 2024

DL-Learner is a machine learning framework for OWL and description logics. It includes several learning algorithms and is easy to extend. DL-Learner widens the scope of Inductive Logic Programming to description logics and the Semantic Web. This manual provides the entry point to using DL-Learner and explains its basic concepts.

Please refer to the following publication (BibTeX) when citing DL-Learner:

```
@Article{dllearner_jws,
 author = "Lorenz B\"uhmann and Jens Lehmann and Patrick Westphal",
 title = "{DL-Learner} -- A framework for inductive learning on the Semantic Web",
 journal = "Journal of Web Semantics",
 year = "2016",
 volume = "39",
 pages = "15-24"
}
```

Contents

1 What is DL-Learner?

DL-Learner is an open source framework for (supervised) machine learning in OWL and Description Logics (from instance data). We further detail what this means:

OWL stands for "Web Ontology Language". In 2004, it became the W3C[1] standard ontology language[2]. As such it is one of the fundamental building blocks in the Semantic Web and has been used in several scenarios on and off the web. OWL is based on *description logics* (DLs), which are a family of knowledge representation languages. We refer to [?] for an introduction to description logics. Since OWL formally builds on description logics, we can apply DL-Learner to knowledge bases in OWL or a variety of description languages.

Machine Learning is a subfield of Artificial Intelligence, which focuses on detecting patterns, rules, models etc. in data. Often, this involves a training process on the input data. In *Supervised* learning, this data is labelled, i.e. we are given a number of input-output mappings. Those mappings are also called *examples*. If the output is binary, then we distinguish positive and negative examples. DL-Learner as a framework is not restricted to supervised learning, but all algorithms currently build into it, are supervised.

In the most common scenario we consider, we have a background knowledge base in OWL/DLs and additionally, we are given positive and negative examples. Each example is an individual in our knowledge base. The goal is to find an OWL *class expression*[3] such that all/many of the positive examples are *instances* of this expression and none/few of the negative examples are instances of it. The primary purpose of learning is to find a class expression, which can classify unseen individuals (i.e. not belonging to the examples) correctly. It is also important that the obtained class expression is easy to understand for a domain expert. We call these criteria *accuracy* and *readability*.

As an example, consider the problem to find out whether a chemical compound can cause cancer[4]. In this case, the background knowledge contains information about chemical compounds in general and certain concrete compounds we are interested in. The positive examples are those compounds causing cancer, whereas the negative examples are those compounds not causing cancer. The prediction for the examples has been obtained from experiments and long-term research trials in this case. Of course, all examples have to be described in the considered background knowledge. A learning algorithm can now derive a class expression from examples and background knowledge, e.g. such a class expression in natural language could be "chemical compounds containing a phosphorus atom". (Of course, in practice the expression will be more complex to obtain a reasonable accuracy.) Using this class expression, we can now classify unseen chemical compounds.

Please note that the latest versions of DL-Learner are not limited to OWL class expressions anymore. There is also preliminary support for learning simple SPARQL

---

[1] http://www.w3.org
[2] http://www.w3.org/2004/OWL/
[3] http://www.w3.org/TR/owl2-syntax/#Class_Expressions
[4] see http://dl-learner.org/community/Carcinogenesis for a more detailed description

queries [?]. Preliminary support for fuzzy OWL class expressions [?] is also included, but requires setting up a fuzzy OWL reasoner. Please contact us via the DL-Learner discussion list if you plan to do this.

2 Getting Started

DL-Learner is written in Java, i.e. it can be used on almost all platforms. Currently, Java 8 or higher is required. To install the latest release, please visit the download page[5] and extract the file on your harddisk. In the `bin` directory, you will notice several executables. Those files ending with `bat` are Windows executables, whereas the corresponding files without file extension are the Non-Windows (e.g. Linux, Mac) executables. To test whether DL-Learner works, please run the following on the command line depending on your operating system:

```
bin/cli examples/father.conf        (Non-Windows Operating System)
bin/cli.bat examples\father.conf     (Windows Operating System)
```

Conf files, e.g. `examples/father.conf` in this case, describe the learning problem and specify which algorithm you want to use to solve it. In the simplest case they just say where to find the background knowledge to use (in the OWL file `examples/father.owl` in this case) and the positive and negative examples. When running the above command, you should get something similar to the following:

```
1   DL-Learner command line interface
2   Initializing Component "OWL File"... OK (1ms)
3   Initializing Component "closed world reasoner"... OK (325ms)
4   Initializing Component "PosNegLPStandard"... OK (0ms)
5   Initializing Component "CELOE"... OK (9ms)
6   Running algorithm instance "alg" (CELOE)
7   more accurate (50.00%) class expression found: Thing
8   more accurate (83.33%) class expression found: http://example.com/father#male
9   more accurate (100.00%) class expression found: (http://example.com/father#male and ↵
        http://example.com/father#hasChild some Thing)
10  Algorithm terminated successfully (time: 1s 0ms, 13380 descriptions tested, 7727 ↵
        nodes in the search tree).
11
12  number of retrievals: 6
13  retrieval reasoning time: 0ms ( 0ms per retrieval)
14  number of instance checks: 56442 (0 multiple)
15  instance check reasoning time: 163ms ( 0ms per instance check)
16  (complex) subsumption checks: 4 (0 multiple)
17  subsumption reasoning time: 9ms ( 2ms per subsumption check)
18  overall reasoning time: 172ms
19
20  solutions:
21  1: (http://example.com/father#male and http://example.com/father#hasChild some Thing) ↵
        (pred. acc.: 100.00%, F-measure: 100.00%)
22  2: ((not http://example.com/father#female) and http://example.com/father#hasChild ↵
        some Thing) (pred. acc.: 100.00%, F-measure: 100.00%)
23  3: (http://example.com/father#male and (http://example.com/father#female or ↵
        http://example.com/father#hasChild some Thing)) (pred. acc.: 100.00%, F-measure: ↵
        100.00%)
```

---

[5]`https://github.com/AKSW/DL-Learner/releases`

```
24  4: (http://example.com/father#male and http://example.com/father#hasChild some ↵
       (http://example.com/father#female or http://example.com/father#male)) (pred. ↵
       acc.: 100.00%, F-measure: 100.00%)
25  5: (http://example.com/father#male and (not http://example.com/father#female) and ↵
       http://example.com/father#hasChild some Thing) (pred. acc.: 100.00%, F-measure: ↵
       100.00%)
26  6: (http://example.com/father#hasChild some Thing and (http://example.com/father#male ↵
       or (not http://example.com/father#female))) (pred. acc.: 100.00%, F-measure: ↵
       100.00%)
27  7: ((not http://example.com/father#female) and (http://example.com/father#female or ↵
       http://example.com/father#hasChild some Thing)) (pred. acc.: 100.00%, F-measure: ↵
       100.00%)
28  8: ((not http://example.com/father#female) and http://example.com/father#hasChild ↵
       some (http://example.com/father#female or http://example.com/father#male)) (pred. ↵
       acc.: 100.00%, F-measure: 100.00%)
29  9: (http://example.com/father#male and http://example.com/father#hasChild some ↵
       (http://example.com/father#male or (not http://example.com/father#male))) (pred. ↵
       acc.: 100.00%, F-measure: 100.00%)
30  10: (http://example.com/father#male and http://example.com/father#hasChild some ↵
       (http://example.com/father#female or (not http://example.com/father#female))) ↵
       (pred. acc.: 100.00%, F-measure: 100.00%)
```

The first part of the output (line 1-5) tells you which components are used (more on this in Section 4). In the second part (line 6-10) you see output coming from the used learning algorithm, i.e. it can print information while running ("more accurate (83,33%) class expression found"). When the algorithm finished some overall runtime statistics are presented (line 12-18) and the final results are displayed in Manchester OWL Syntax[6]. There are several solutions ordered with the most promising one in the first position (`male and hasChild some Thing`).

3 DL-Learner Architecture

DL-Learner (see also [?]) consists of core functionality, which provides Machine Learning algorithms for solving the learning problem, support for different knowledge base formats, an OWL library, and reasoner interfaces. There are several interfaces for accessing this functionality, a couple of tools which use the DL-Learner algorithms, and a set of convenience scripts.

To be flexible in integrating new learning algorithms, new kinds of learning problems, new knowledge bases, and new reasoner implementations, DL-Learner uses a component based model. Adding a component can be done by implementing the appropriate Java interface and adding appropriate annotations (more on that in Section 7).

There are five common types of components (knowledge source, reasoning service, learning problem, learning algorithm, refinement operator). DL-Learner is not restricted to those types, i.e. others can easily be added, but we limit ourselves to those five to make this manual easier to read. For each type, there are several implemented components and each component can have its own configuration options as illustrated in Figure 1. Configuration options can be used to change parameters/settings of a component. In Section 4, we describe the components in DL-Learner and their configuration options.

---

[6]http://www.w3.org/2007/OWL/wiki/ManchesterSyntax
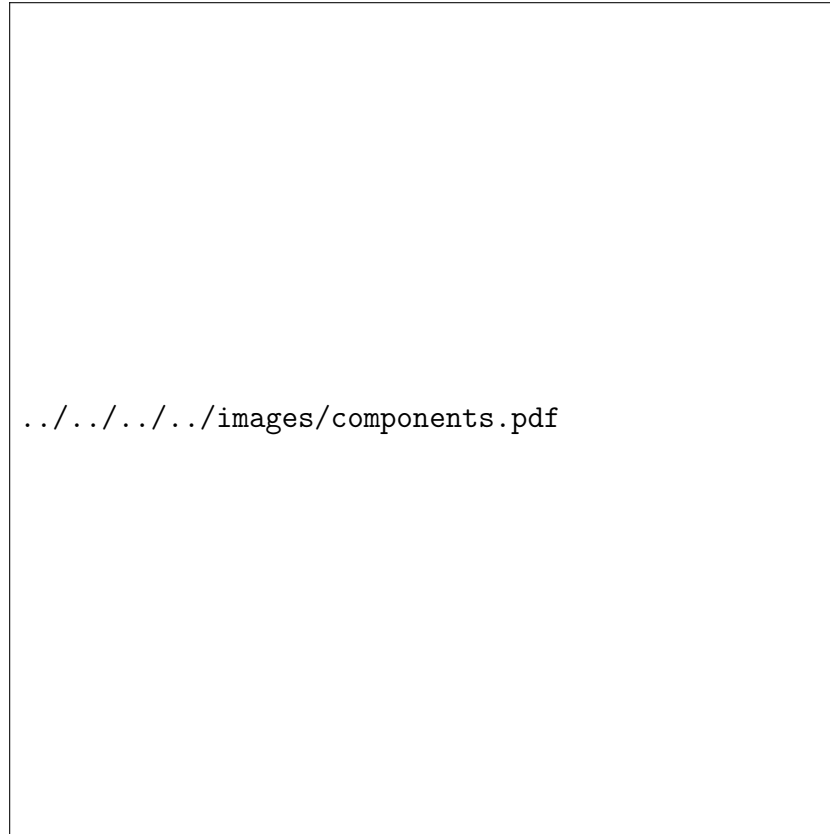
```
../../../../images/components.pdf
```

Figure 1: The architecture of DL-Learner is based on different component types, which can each have their own configuration options.

4 DL-Learner Components

In this part, we describe concrete components currently implemented in DL-Learner. Each of the subsections contains a list of components according to the type specified in the subsection heading. Note that this does not constitute a full description, i.e. we omit some components and many configuration options. The purpose of the manual is to obtain a general understanding of the implemented components. A full list, which is generated automatically from the source code, can be found in `interfaces/doc/configOptions.html` including the default values for all options and their usage in conf files. The file is also available online at `http://htmlpreview.github.io/?https://github.com/AKSW/DL-Learner/blob/master/interfaces/doc/configOptions.html`.

4.1 Knowledge Sources

The following contains some knowledge sources implemented in DL-Learner. To give an example, this is how a local OWL file can be declared as knowlege source in a conf file:

```
ks.type = "OWL File"
```

```
ks.fileName = "father.owl"
```

OWL File DL-Learner supports OWL files in different formats, e.g. RDF/XML or N-Triples. If there is a standard OWL format, you want to use, but is not supported by DL-Learner please let us know. We use the OWL API for parsing, so all formats supported by it can be used[7].

KB File KB files are an internal non-standardised knowledge representation format, which corresponds to description logic syntax except that the special symbols have been replaced by ASCII strings, e.g. `AND` instead of $\sqcap$. You can find several KB files in the examples folder. A description of the syntax is available online[8]. If in doubt, please use the standard OWL syntax formats.

SPARQL endpoint fragment DL-Learner allows to use SPARQL endpoints as background knowledge source, which enables the incorporation of parts from very large knowledge bases, e.g. DBpedia[?], in DL-Learner. This works by using a set of start instances, which usually correspond to the examples in a learning problem, and then retrieving knowledge about these instances via SPARQL queries. The obtained knowledge base fragment can be converted to OWL and consumed by a reasoner later since it is now sufficiently small to be processed in reasonable time. Please see [?] for details about the knowledge fragment extraction algorithm. Some options of the SPARQL component are:

- instances: Set of individuals to use for starting the knowledge fragment extraction. Example use in conf file:

  ```
  sparql.type = "SPARQL endpoint fragment"
  sparql.instances = {"http://dbpedia.org/resource/Matt_Stone",
   "http://dbpedia.org/resource/Sarah_Silverman"}
  ```

- recursionDepth: Maximum distance of an extracted individual from a start individual. This influences the size of the extracted fragment and depends on the maximum property depth you want the learned class expression to have. Example use in conf file: `sparql.recursionDepth = 2`.

- saveExtractedFragment: Specifies whether the extracted ontology is written to a file or not. If set to true, then the OWL file is written to the cache dir. Example usage: `sparql.saveExtractedFragment = true`

Many further options allow to modify the extracted fragment on the fly or fine-tune the extraction process. The extraction can be started separately by running and modifying `org.dllearner.test.SparqlExtractionTest`. The collected ontology will be saved in the DL-Learner directory.

---

[7]for a list see `http://owlapi.sourceforge.net`
[8]`https://raw.githubusercontent.com/AKSW/DL-Learner/master/interfaces/doc/kbFileSyntax.txt`

SPARQL endpoint Algorithms which work directly on SPARQL endpoints (without fragment extraction) can make use of the SPARQL endpoint component. The endpoint will then be queried for each algorithm task as required.

4.2 Reasoner Components

Several reasoner components are implemented, which can be interfaces to concrete reasoner implementations. Note that OWLlink reasoners can be attached via the OWL API interface.

OWL API The OWL API reasoner interface can be used in conjunction with the Pellet, FaCT++, HermiT and OWLlink reasoners. The only option allows to switch between them:

- reasonerImplementation: Selects the desired reasoner. By default, Pellet is used. Usage: `owlAPIReasoner.reasonerImplementation = "fact"`. Pellet, FaCT++ and HermiT are already included in DL-Learner. Note that for FaCT++, you need to add -Djava.library.path=lib/fact/64bit (or 32bit) to the Java command. You can also use an external OWLlink reasoner by setting the reasoner type to `owllink`. You can then use the option `owlLinkURL` to specify the URL of the OWLlink reasoner (http://localhost:8080/ by default).

SPARQL Reasoner Combined with the SPARQL endpoint or an OWL file as knowledge sources the SPARQL reasoner [?] answers reasoning requests directly via generated SPARQL queries. This allows handling huge knowledge sources which do not fit into memory but requires that entailments are already materialized.

Closed World Reasoner Instance checks, i.e. testing whether an individual is instance of a class, is the major reasoner task in many learning algorithms. This reasoner is a self-development of the DL-Learner project. It remedies some problems related to Machine Learning and the Open World Assumption in OWL and therefore is not correct w.r.t. OWL semantics. (See [?] Section 4 for an explanation.) Furthermore, it provides an improved performance for instance checks by precomputing some inferences and keeping them in memory. The closed world reasoner is build on top of any other reasoner component in DL-Learner.

4.3 Learning Problems

In the introductory Sections 1 and 2, we described a specific learning problem where positive and negative examples are given. In practice different variations of similar problems occur.

Positive and Negative Examples Let the name of the background ontology be $\mathcal{O}$. The goal in this learning problem is to find an OWL class expression C such that all/many positive examples are instances of C w.r.t. $\mathcal{O}$ and none/few negative examples are instances of C w.r.t. $\mathcal{O}$. As explained previously, C should be learned

such that it generalises to unseen individuals and is readable. The important configuration options of this component are obviously the positive and negative examples, which you can specify via, e.g. `posNegLPStandard.positiveExamples = {...}`.

Positive Examples This learning problem is similar to the one before, but without negative examples. In this case, it is desirable to find a class expression which closely fits the positive examples while still generalising sufficiently well. For instance, you usually do not want to have `owl:Thing` as a solution for this problem, but neither do you want to have an enumeration of all examples.

Class Learning In class learning, you are given an existing class A within your ontology $\mathcal{O}$ and want to describe it. It is similar to the previous problem in that you can use the instances of the class as positive examples. However, there are some differences, e.g. you do not want to have A itself as a proposed solution of the problem, and since this is an ontology engineering task, the focus on short and readable class expressions is stronger than for the two problems mentioned before. The learner can also take advantage of existing knowledge about the class to describe.

4.4 Learning Algorithms

The implemented algorithms vary from very simple (and usually inappropriate) algorithms to sophisticated ones. Learning algorithms can make use of a refinement operator (see Sec. 4.5) to traverse the search space.

OWL Class Expression Learner (OCEL) The general idea of the OCEL algorithm is to build a search tree based on a refinement operator and making use of a heuristic guiding the traversal of the search space. While the refinement operator defines how new candidate concepts may be derived from concepts already in the search tree, the heuristic can be configured to find *'good'* candidates to look at (e.g. w.r.t. the concept length, its specificity and the number of covered examples).

The OCEL algorithm supports double datatypes and hasValue restrictions (which again can be turned on or off as desired). It also includes explicit noise handling through the `noisePercentage` option. More than 30 options can be set to control its behaviour.

Class Expression Learning for Ontology Engineering (CELOE) Currently CELOE is the best class learning algorithm available within DL-Learner. It uses the same refinement operator as OCEL, but a completely different heuristics. Furthermore, it guarantees that the returned class expressions are minimal in the sense that one cannot remove parts of them without getting an inequivalent expression. Furthermore, it makes use of existing background knowledge in coverage checks. Statistical methods are used to improve the efficiency of the algorithm such that it scales to large knowledge bases. While it was originally designed for ontology engineering, it can also be used for other learning problems and might even be superior to the

other algorithms in many cases. Note that many configuration options of OCEL were dropped for the sake of simplicity, but may be introduced if needed.

EL Tree Learner (ELTL) This algorithm has EL as its target language, i.e. is specialized for this relatively simple language. There are two algorithms: `el` learns EL concepts and `disjunctiveEL` learns disjunctions of EL concepts.

Inductive Statistical Learning of Expressions (ISLE) The ISLE approach [?] is an extension of the ELTL algorithm, which can take standard knowledge sources (see Sec. 4.1) and textual evidence into account. An input corpus should contain textual descriptions of classes which support the generation on an ontology in an *ontology learning* setting.

4.5 Refinement Operators

Refinement operators define a mapping from a given input concept description to a set of derived, or *refined* concept descriptions. In general there are to kinds of refinement operators: 1) *upward refinement operators*, returning refined concept descriptions that are more general than the input concept description, and 2) *downward refinement operators* returning concept descriptions that are more specific than the input concept description. The specificity/generality here refers to the concept hierarchy in terms of subclass-of relations of concept descriptions.

Besides this, refinement operators differ in their expressiveness, i.e. the types of DL constructs they support. As an example, the `rho` *Rho (downward) refinement operator* can be configured to create refinements that contain cardinality restrictions, which is not supported by the `eldown` *EL Downward refinement operator*.

A list of available refinement operators is given in the components and configuration settings document.

5 Enrichment using DL-Learner

Enrichment is usually a semi-automatic process, which adds information to an existing knowledge base. Enrichment suggestions generated by an algorithm should be reviewed by a knowledge engineer who can then decide to accept or reject it. Because of this, there is a need for serialising enrichment suggestions such that the generation of them is independent of the process of accepting or rejecting them. Since all enrichment suggestions are OWL axioms, they could simply be written in an RDF or OWL file. However, this might be insufficient, because we lose a lot of metadata this way, which could be relevant for the knowledge engineer. For this reason, DL-Learner uses an enrichment ontology, which is partially building on related efforts in [?] and `http://vocab.org/changeset/schema.html`. Such an interchange format is also relevant, because the process of generating enrichments for all schema elements in very large knowledge bases will often take several hours to complete. Furthermore, it may be

desirable to include sufficient metadata to be able to reproduce algorithm runs for creating enrichment suggestions. We briefly describe the main elements of the enrichment ontology.

Classes

Suggestion Base class for enrichment suggestions.

> **AddSuggestion** Contains suggestions for adding axioms to an ontology.

> **RemoveSuggestion** Contains suggestions for removing axioms from an ontology.

SuggestionSet This class is used to group several suggestions with similar characteristics, e.g. those generated by the same run of an algorithm.

Algorithm The class containing all algorithms.

Creation Contains all processes of creating a suggestion.

> **Manual** Manual enrichment suggestion processes, e.g. a person suggesting a particular change to an ontology.

> **Automatic** Automatic and semi-automatic enrichment suggestions.

>> **AlgorithmRun** Contains runs of a particular algorithm.

Parameter Contains parameters of an algorithm.

Change Actually performed changes in an ontology, e.g. an accepted suggestion could become an instance of Change. This can be used to track which changes have already been performed as result of the enrichment process.

ChangeSet A set of instances of Change.

Object Properties

hasSuggestion Links a set of suggestions to its elements.

> **Domain:** SuggestionSet

> **Range:** Suggestion

hasAxiom Connects a suggestion to the axiom contained in it. Currently, this axiom is stored as Manchester OWL Syntax text string.

> **Domain:** Suggestion

hasParameter Links to a parameter of an algorithm.

> **Domain:** AlgorithmRun

> **Range:** Parameter

creator Specifies who or what has created a set of enrichment suggestions.

> **Domain:** SuggestionSet
>
> **Range:** Creation

usedAlgorithm Specifies the used algorithm

> **Domain:** AlgorithmRun
>
> **Range:** Algorithm

hasInput Can be used specify input resources of an algorithm, for instance SPARQL endpoints.

> **Domain:** AlgorithmRun

Data Properties

confidence The confidence with which a Creator made a suggestion. The confidence is value between 0 and 1 with 0 indicating no confidence and 1 indicating absolute confidence.

> **Domain:** Suggestion
>
> **Range:** xsd:double

explanation A textual explanation why a suggestion was given. This could be a remark made by a person or a summary of statistical analysis results of an algorithm.

> **Domain:** Suggestion

parameterName The name of a parameter of an algorithm.

> **Domain:** Parameter

parameterValue The value of a parameter of an algorithm.

> **Domain:** Parameter

timestamp Timestamp of the start of automatic process.

> **Domain:** Automatic

version The version of the used algorithm.

> **Domain:** Algorithm

To run the above enrichment algorithms on a SPARQL endpoint, you can use the provided enrichment script of DL-Learner. It is a commandline interface, which you can start with "./enrichment" in Unix systems and "enrichment.bat" Windows systems. Figure 2 shows the help screen, which is printed when running `enrichment -?`. We will briefly explain the options:

`-e` and `-g` are used to specify the used endpoint and optionally a graph in this endpoint.

-r is used to specify the resource (property or class), which should be enriched. The system automatically determines whether this resource is a class, object property or data property and runs the corresponding algorithms. If this parameter is omitted, enrichment for the complete knowledge base is performed, i.e. the system loops over all classes and properties in the SPARQL endpoint and generates enrichments.

The -f switch can be used to control the format of the output. By default, the suggestions are just printed to the console, but they can also be saved in a file in combination with the -o option, e.g. using the previously described enrichment ontology. This is useful for decoupling the enrichment suggestion generation process from the actual presentation of those suggestions to a knowledge engineer.

The -i switch allows to turn inference on or off. If it is turned on, schema knowledge is loaded into a reasoner in the first step. Powerful reasoning capabilities may improve the quality of suggestions, in particular for those axioms, which rely on knowing the class hierarchy of the knowledge base, e.g. domain and range axioms.

-t allows to specify a threshold for enrichment suggestions, i.e. suggestions with a lower score will be omitted.

Examples

Obtain enrichment suggestions for the `currency` property in DBpedia:

```
-e http://dbpedia.org/sparql -g http://dbpedia.org
-r http://dbpedia.org/ontology/currency
```

Write the enrichments in Turtle syntax in a file using the enrichment ontology:

```
-e http://dbpedia.org/sparql -g http://dbpedia.org
-r http://dbpedia.org/ontology/currency -o results.ttl
-f turtle
```

Do the same task with an increased threshold and without inference

```
-e http://dbpedia.org/sparql -g http://dbpedia.org
-r http://dbpedia.org/ontology/currency -o results.ttl
-f turtle -t 0.9 -i false
```

Generate all enrichments for DBpedia (will take several hours to complete):

```
-e http://dbpedia.org/sparql -g http://dbpedia.org
```

6 DL-Learner Interfaces

One interface you have already used in Section 2 is the command line. The main executable, which can be used for starting DL-Learner on the commandline is `cli` which

```
Option                         Description
------                         -----------
-?, -h, --help                 Show help.
-e, --endpoint <URL>           SPARQL endpoint URL to be used.
-f, --format                   Format of the generated output (plain,
                                  rdf/xml, turtle, n-triples).
                                  (default: plain)
-g, --graph [URI]              URI of default graph for queries on
                                  SPARQL endpoint.
-i, --inference [Boolean]      Specifies whether to use inference. If
                                  yes, the schema will be loaded into
                                  a reasoner and used for computing
                                  the scores. (default: true)
-o, --output [File]            Specify a file where the output can be
                                  written.
-r, --resource [URI]           The resource for which enrichment
                                  axioms should be suggested.
-t, --threshold [Double]       Confidence threshold for suggestions.
                                  Set it to a value between 0 and 1.
                                  (default: 0.7)


Additional explanations: The resource specified should be a class,
object property or data property. DL-Learner will try to automatically
detect its type. If no resource is specified, DL-Learner will generate
enrichment suggestions for all detected classes and properties in the
given endpoint and graph. This can take several hours.
```

Figure 2: Depiction of the help screen of the enrichment script.

takes a conf file as argument. There are a lot of conf files available in the `/examples` directory, which you can use a base for your own experiments.

Another means to access DL-Learner, in particular for ontology engineering, is to use the OntoWiki and Protégé plugins. The OntoWiki plugin is not officially released yet, but can be used in the SVN version of OntoWiki. The Protégé 5 plugin can be installed either by downloading it from the DL-Learner download page or directly within Protégé 4 by clicking on "File", "Preferences", "Plugins", "Check for Downloads" now and selecting the DL-Learner plugin. For more information and a screencast see the Protégé plugin wiki page [9].

---

[9]`http://dl-learner.org/community/protege-plugin/`

7 Extending DL-Learner

DL-Learner is open source and component based. If you want to develop a specific part or extension of a class expression learning algorithm for OWL, then you are invited to use DL-Learner as a base. This allows you to focus on the part you want to implement while being able to use DL-Learner as a library and access it through one of the interfaces.

If you want to create a new component, then you first have to decide on the type of your component. To implement a concrete component, you can implement one of the following interfaces (list is incomplete):

- `org.dllearner.core.KnowledgeSource`

- `org.dllearner.core.ReasonerComponent`

- `org.dllearner.core.LearningProblem`

- `org.dllearner.core.LearningAlgorithm`

That is sufficient for using your component programmatically in combination with existing DL-Learner components. If your class name is `org.example.TestAlgorithm`, then you can instantiate your class in a conf file via:

```
c.type = "org.example.TestAlgorithm"
```

As you have probably seen by now in various conf files, DL-Learner allows to configure components. This is done via standard Java Beans. If you want to create a conf option `testOption`, you just need to create a variable with getters and setters in your code:

```
public class TestAlgorithm implements LearningAlgorithm {

    private double testOption = 0.0;

    [...]

    public double getTestOption() {
        return testOption;
    }

    public void setTestOption(double testOption) {
        this.testOption = testOption;
    }

}
```

That would be sufficient to include your components in conf files:

```
c.type = "org.example.TestAlgorithm"
c.testOption = 0.3
```

In your code, you should have an empty default constructor and an `init()` method (as required by the Component interface). The default constructor will be called first, followed by setter methods and then the `init()` method. This is a standard Java Beans approach. In summary, you need to the following:

- implement an appropriate DL-Learner interface for what you want to do

- add variables for all config options as well as getters and setters for them

- if you implement a constructor, please also add a default constructor with an empty set of arguments

By only requiring those few steps, we want to make adding further components to DL-Learner as lightweight as possible. If you are familiar with the Spring framework[10], then it is helpful to know that conf files are just an abbreviated syntax for Spring XML configurations. You can use all powerful features of Spring in your code, which we do not describe in full detail here.

If you are a DL-Learner developer and want to properly document your component, you should do some further steps:

- add an annotation for your class

- add annotations for all configuration options

An example of an annotated class could look as follows:

```
@ComponentAnn(name="Test Algorithm", shortName="ta", version=0.1,
  description="My first experiment.")
public class TestAlgorithm implements LearningAlgorithm { ...

    @ConfigOption(defaultValue="0.0",
      description="The option allows to control xyz.")
    private double testOption = 0.0;

    [...]

    public double getTestOption() {
        return testOption;
    }

    public void setTestOption(double testOption) {
        this.testOption = testOption;
```

---

[10]http://www.springsource.org

```
    }

}
```

The `@ComponentAnn` annotation allow to mark classes as DL-Learner components. Similarly, the `@ConfigOption` annotations marks variables as configuration options. That should be those variables, which you want the user to be able to configure and play with. A benefit of adding the extra metadata provided by the annotations is that the component will appear in documentation pages[11]. In general, they provide users of your component with useful information.

This quick introduction only serves as an entry point to get you started. For more detailed questions about how to extend DL-Learner, please drop us a message in the DL-Learner mailing list.

8 General Information

- Homepage: `http://dl-learner.org`

- GitHub project page: `https://github.com/AKSW/DL-Learner`

- Tracker (bugs, features): `https://github.com/AKSW/DL-Learner/issues`

- Mailing Lists: `http://sourceforge.net/p/dl-learner/mailman/`

- Contact: jens.lehmann@cs.uni-bonn.de (please use the mailing list if possible)

- Latest Release: `https://github.com/AKSW/DL-Learner/releases/tag/1.3.0`

References

---

[11]such as `http://htmlpreview.github.io/?https://github.com/AKSW/DL-Learner/blob/master/interfaces/doc/configOptions.html`